UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Polsearchine: Implementierung einer Policy-basierten Suchmaschine zur Internetregulierung

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informatik

vorgelegt von

Michael Ruster

Erstgutachter:	Prof. Dr. Rüdiger Grimm Institut für Wirtschafts- und Verwaltungsinformatik
Zweitgutachter:	Dipl.-Inform. Andreas Kasten Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im September 2013

Erklärung

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

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Michael Ruster)

Präambel In dieser Arbeit wird die nach der Grammatik männliche Form in einem neutralen Sinne verwendet. Dies bedeutet, dass gänzlich auf das Binnen-I oder den Gender Gap verzichtet wird. Der einzige Grund für diese Entscheidung ist eine bessere Lesbarkeit. Frauen, Männer, Transgender und intersexuelle Menschen werden weiterhin gleichermaßen angesprochen.

Des Weiteren finden sich in dieser Arbeit URLs, die keine Verweise oder Quellen darstellen. Sie werden stattdessen lediglich zur Erklärung einzelner Sachverhalte genutzt. Eine Befürwortung der Inhalte hinter diesen URLs oder ihrer Autoren ist nicht beabsichtigt.

Zusammenfassung. Viele Suchmaschinen betreiben Formen der Internetregulierung. Diese ist für den Endbenutzer teils schwer ersichtlich und leicht umgehbar. Weiter ist es oft schwierig, Hintergrundinformationen zu Regulierungen zu erfahren. Um diese Schwachstellen zu beheben, wird die Entwicklung der prototypischen Meta-Suchmaschine „Polsearchine“ beschrieben. Ihre Regulierung erfolgt mittels der von Kasten und Scherp für Internetregulierung entwickelten Ontologie InFO. Dabei wird die konkrete Erweiterung SEFCO zur Anwendung auf Suchmaschinenebene verwendet. Zur Beschaffung der Suchergebnisse wird eine externe Suchmaschinen-API genutzt. Um nicht von einer bestimmten API abhängig zu sein, kann die API leicht ausgetauscht werden.

Abstract. Many search engines regulate Internet communication in some way. It is often difficult for end users to notice such regulation, as well as obtaining background information for it. Additionally, the regulation can usually be circumvented easily. This bachelor thesis presents the prototypical metasearch engine „Polsearchine“ for addressing these weaknesses. Its regulation is established through InFO, a model for regulating information flows developed by Kasten and Scherp. More precisely, the extension for regulating search engines SEFCO is being used. For retrieving search results, Polsearchine uses an external search engine API. The API can be interchanged easily to make this metasearch engine independent from one specific API.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Tests des OpenNet Initiative-Projekts aus dem Jahre 2006 konnten bei 26 von 40 untersuchten Staaten Formen angewandter Internetregulierung nachweisen [FV08]. Zudem zeigen Beobachtungen dieses Projekts aus den Jahren 2001 bis 2006 wachsenden Einsatz von Internetregulierung [ZP08]. Die betroffenen Inhalte sind dabei vielzählig. Nach Deibert et al. [DR08] reichen sie unter anderem von Kinderpornographie, Anonymisierungsdiensten, Urheberrechtsverletzung, bis hin zu Menschenrechten. Dementsprechend unterschiedlich sind auch die Gründe für die Anwendung von Internetregulierung. So handelt es sich bei Kinderpornographie um den Schutz von Minderjährigen. Eine Filterung von Inhalten mit menschenrechtlicher Thematik wird hingegen häufig von Staaten eingesetzt, welche um die Legitimation der eigenen Regierung besorgt sind [DR08]. Internetregulierung ist folglich weder zwingend positiv noch negativ. Oftmals wird sie allerdings nicht gestützt auf Gesetzen ausgeübt [DR10]. In diesen Fällen wird der Zugriff auf ganze Internetauftritte oder einzelne Webseiten ohne rechtliche Grundlage erschwert oder gänzlich verhindert. Dies kann ungewollt aufgrund technischer Einschränkungen geschehen oder bewusst, um vermeintliche Zensur zu verschleiern [DR10]. Durchgeführt wird Internetregulierung auf verschiedenen Ebenen und Arten. Dabei wird hauptsächlich DNS-, IP- und URL-Regulierung eingesetzt [ZP08].

Im Kontext von Suchmaschinen wird eine Regulierung auf Applikationsschicht angewandt. Eine solche Regulierung ist aufgrund der Position von Suchmaschinen zwischen Nutzer und Inhalt [Dia08] sinnig. So ergab eine Studie der GVU [Geo13], dass bereits 1998 etwa 86% der Studienteilnehmer eine Suchmaschine

nutzen, um neue Webseiten und Internetauftritte zu entdecken [KT00]. Die Ergebnisse, die Suchmaschinennutzer erhalten, sind dabei maßgeblich für deren Wahrnehmung des im Internet verfügbaren Inhalts. Bereits heute regulieren einige Suchmaschinen ihre Ergebnisse. Als Beispiel sei eine Google-Suche mit dem Suchbegriff „`http://stormfront.org`“ angeführt. `http://stormfront.org` ist ein Forum für Vertreter neonazistischer Ideologien [ZRQ+05]. Bei der Suche nach diesem Begriff mit Google Search Frankreich[1] oder Google Search Deutschland[2] werden nur Ergebnisse über diesen Internetauftritt zurückgegeben. Darunter befinden sich beispielsweise inhaltskritische Artikel, nicht jedoch ein Link auf den Internetauftritt selbst (vgl. [FV08]). Eine Suche mit Google Search UK[3] hingegen liefert direkte Links auf `http://stormfront.org`[4].

Solch ein Verhalten führt zu drei kritischen Punkten. Erstens erhält der Nutzer keine ausreichenden Hinweise und Erklärungen bezüglich der Regulierung der ihm ausgelieferten Ergebnisse. In diesem Beispiel wird auf eine Filterung hingewiesen[5]. Es ist allerdings beispielsweise unklar, wer die Filterung bewirkt hat. Zweitens ist es für einen Nutzer einfach eine Filterung zu umgehen, solange diese nicht zusätzlich gleichermaßen auf tieferer Ebene durchgeführt wird. Erfolgt keine weitere Regulierung auf beispielsweise DNS- und IP-Ebene, werden dem Nutzer bei einer Suche nur Ergebnisse vorenthalten. In dem zuvor beschriebenen Fall würde er keine Links zu `http://stormfront.org` erhalten. Es ist ihm jedoch möglich, diese Webseite aufzurufen, solange er die Adresse kennt. Damit verliert eine Regulierung allein auf Suchmaschinenebene an Wirksamkeit. Der dritte Punkt beschreibt den umgekehrten Fall bei dem einzig auf tieferer Ebene gefiltert wird. Dadurch können Ergebnisse in Suchmaschinen auftauchen, die für den Nutzer nicht abrufbar sind.

Im Rahmen dieser Bachelorarbeit wird eine Suchmaschine entwickelt, die durch ihren Aufbau auf InFO Lösungsansätze bieten soll. InFO [KS12] steht für Information Flow Ontology und bezeichnet eine von Kasten und Scherp entwickelte Ontologie. Einerseits bietet sie Ansätze, um eine Internetregulierung auf allen aktuell relevanten Ebenen durchzuführen. Dazu zählen Router, Nameserver

[1] `https://www.google.fr/`
[2] `https://www.google.de/`
[3] `https://www.google.co.uk/`
[4] Stand aller drei Tests: 09.09.2013, 19:30 Uhr
[5] `http://www.chillingeffects.org/notice.cgi?sID=945` letzter Zugriff 09.09.2013, 20:00 Uhr

und Proxyserver auf der Anwendungsebene [KS12]. Andererseits ist eines der Hauptziele der Entwicklung eine möglichst transparente Form der Internetregulierung gewesen. Eingesetzt wird die Ontologie als spezielle Policys, welche Zugriffe erlauben bzw. verbieten.

Diese Bachelorarbeit beginnt mit der Vermittlung von Grundlagen über die Arbeitsweise von Suchmaschinen und die für diese Suchmaschine genutzte InFO-Implementierung in Kapitel 2. In Kapitel 3 werden Anforderungen für die in dieser Bachelorarbeit entwickelte Suchmaschine aufgestellt. Kapitel 4 beschreibt die konzeptionelle Arbeitsweise der Suchmaschine. In Kapitel 5 wird die konkrete Architektur und Umsetzung der Suchmaschine behandelt. Daraufhin werden in Kapitel 6 die Ergebnisse der Implementierung zusammengefasst. Zusätzlich werden mögliche Weiterentwicklungen aufgezeigt. Das abschließende Kapitel 7 zieht ein Fazit aus dieser Arbeit.

Kapitel 2

Grundlagen

Dieses Kapitel thematisiert die Grundlagen zu dieser Arbeit. Zuerst werden Suchmaschinenarten charakterisiert und im Anschluss ausführlicher beschrieben. Dabei wird auf den Aufbau und die Arbeitsweise von Suchmaschinen und Meta-Suchmaschinen eingegangen. Darauf folgt eine grundlegende Beschreibung der Information Flow Ontology. Diese ist wichtig für das Verständnis der abschließend erläuterten Suchmaschinen-Domänenontologie Search Engine-based Flow Control Ontology. Hierbei werden ihr konkreter Aufbau und grundlegende Konzepte ausgeführt. Beides ist für die in Abschnitt 4.2 vorgestellten Algorithmen zur Policy-Verarbeitung essentiell.

2.1 Grundlagen zu Suchmaschinen

Webbasierte Suchmaschinen lassen sich in verschiedene Arten unterscheiden. Für diese Charakterisierung ist der Begriff des *Suchmaschinenindexes* entscheidend. Ein Suchmaschinenindex ist eine Datenstruktur. Sie beinhaltet unter anderem die Links auf Webseiten, die eine Suchmaschine als Ergebnisse einem Nutzer präsentieren kann. Suchmaschinenindexe werden im anschließenden Abschnitt 2.1.1 noch ausführlicher beschrieben. Suchmaschinen können anhand der Art der Entstehung ihres Indexinhalts unterschieden werden. Dieser kann vollkommen maschinell, rein menschlich oder aus einer Kombination von beidem hinzugefügt worden sein [Sar03]. Das Open Directory Project[1] ist ein Beispiel für einen von Menschenhand

[1] `http://www.dmoz.org/` letzter Zugriff 16.03.2013, 21:00 Uhr

gepflegten Indexinhalt [MJ08]. Ein solcher zeichnet sich durch meist weniger, dafür aber potentiell qualitativ höherwertige Ergebnisse aus (vgl. [PD05, Lew05]). Dies bedeutet unter anderem, dass *Spamdexing*, auch Suchmaschinen-Spamming genannt, unwahrscheinlicher ist [Lew05]. Spamdexing bezeichnet den Versuch, scheinbar ungerechtfertigter Weise eine Webseite als möglichst ersten Treffer in vielen Suchergebnissen auftauchen zu lassen [Nat98]. Dazu werden beispielsweise Stichworte auf einer Webseite eingefügt, die mit dem eigentlichen Inhalt nichts zu tun haben [Nat98]. Ein Beispiel für eine Suchmaschine, die ihren Indexinhalt maschinell hinzufügt, ist nach Brin et al. [BP98] Google[2]. Dahingegen kann Ask.com[3] als Beispiel für eine hybride (Meta-) Suchmaschine gesehen werden [Lew05, Yan06]. Demnach gibt sie sowohl maschinell hinzugefügte Ergebnisse als auch Ergebnisse aus externen, menschlich gepflegten Indexen aus.

Suchmaschinen unterscheiden sich weiter darin, ob sie einen eigenen Index besitzen, den einer anderen Suchmaschine nutzen oder beides [JSK07, DH97]. Solche, die den Index einer anderen Suchmaschine benutzen, werden *Meta-Suchmaschinen* genannt [DH97]. Dabei steht es Suchmaschinen frei, Zugriff auf ihren Index via Programmierschnittstellen zu geben. Bing, Google oder Yahoo bieten beispielsweise solche APIs an [Mic12a, Goo12a, Yah13]. Meta-Suchmaschinen können Indexe unterschiedlicher Suchmaschinen gleichzeitig nutzen [LLG98]. Dadurch kann eine Meta-Suchmaschine umfassendere und aktuellere Ergebnisse ausliefern als eine einzelne Suchmaschine [SE97]. Der nachfolgende Abschnitt 2.1.1 beschreibt den Aufbau von Suchmaschinen mit ausschließlich maschinell erstelltem, absolut eigenem Index. Die Beschreibung bezieht sich dabei auf eine Suchmaschine, die Webseiten anhand von Stichworten sucht. Suchmaschinen, die beispielsweise Bilder oder Videos suchen, sind nicht Thema dieses Grundlagenabschnitts.

2.1.1 Aufbau und Arbeitsweise einer Suchmaschine

Aufbau und Arbeitsweise sind analog zu der nachfolgenden Beschreibung auch in Abbildung 2.1 dargestellt. Sofern nicht weiter angegeben, stammen die Informationen aus Arasu et al. [ACGM+01]. Die Autoren unterscheiden die Hauptbestandteile einer Suchmaschine in *Crawler*, *Crawl Control*, *Indexer*, *Collection Analysis*, *Repository*, *Query Engine*, *Ranking* und ihre *Indexe*. Da sich Inhalte im Internet stetig

[2] `https://www.google.com/` letzter Zugriff 26.03.2013, 20:45 Uhr
[3] `http://ask.com` letzter Zugriff 25.03.2013, 20:00 Uhr

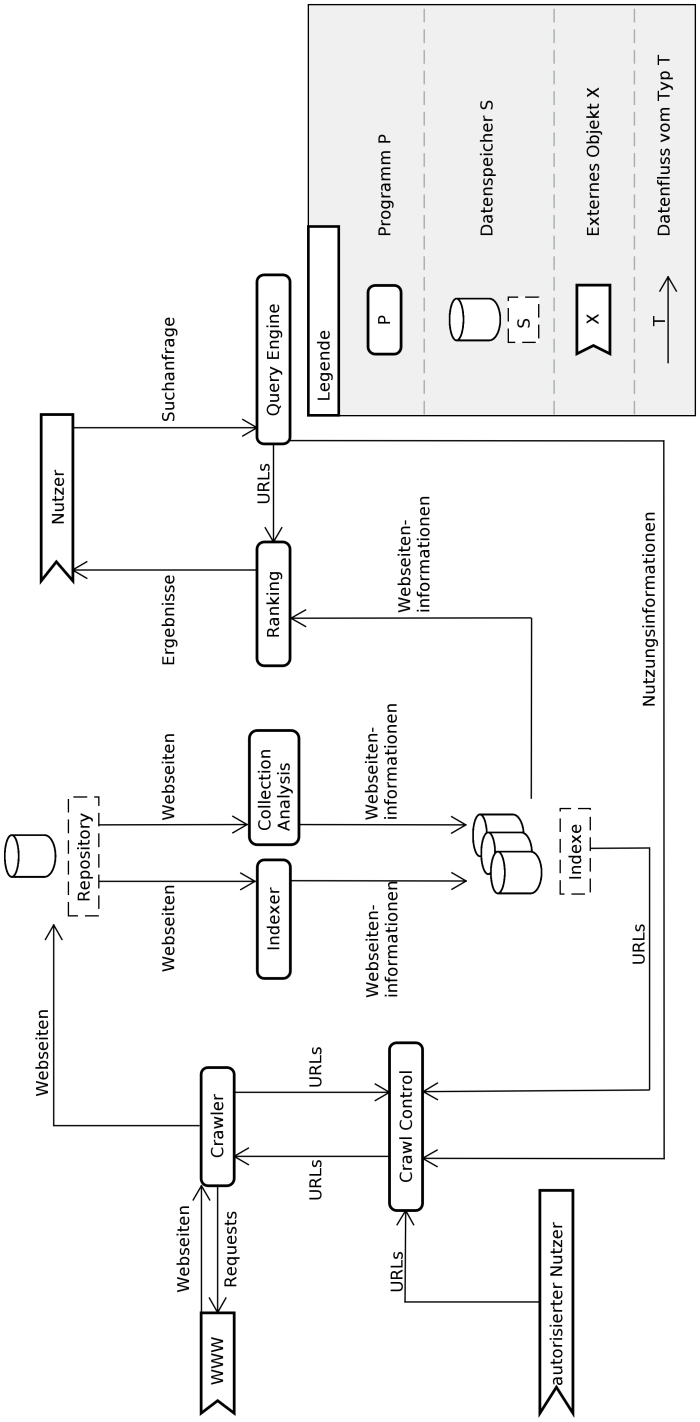

Abbildung 2.1: Diese Abbildung zeigt den grundlegenden Aufbau einer Suchmaschine (angepasste Grafik nach [ACGM+01] Abbildung 1).

verändern [TK06], müssen Suchmaschinen ihre gesammelten Daten ständig aktualisieren und erweitern. Dies geschieht unter anderem durch Crawler und der Crawl Control. Crawler sind Programme, welche Webseiten anhand vorgegebener URLs laden. Sie extrahieren aus den analysierten Webseiten weitere URLs. Im Anschluss übergeben die Crawler zum einen die geladenen Webseiten an das Repository. Zum anderen übermitteln sie die URLs an die Crawl Control.

Die Crawl Control ist ein Programm. Sie entscheidet, welche Webseiten als nächstes analysiert werden sollen. Dabei trifft sie ihre Wahl anhand der für die Suchmaschine gesetzten Parameter [KR09], dem Inhalt der analysierten Seiten und Nutzungsstatistiken. So beschreiben Kassim et al. [KR09], wie ein autorisierter Nutzer Zugriff auf eine spezielle Benutzeroberfläche der Suchmaschine hat. Über diese kann er URLs eintragen, die von Crawlern bevorzugt werden sollen. Ebenso können URLs aus dem Structure-Index geladen werden, welcher später näher erläutert wird. Zudem kann das häufige Besuchen von Webseiten durch Suchmaschinennutzer zu einer Priorisierung dieser Webseiten führen (vgl. [JHW07]). Eine solche Bevorzugung äußert sich darin, dass die Crawler bestimmte Webseiten früher oder öfter besuchen.

Das Repository ist ein Datenspeicher und beinhaltet die von einem Crawler besuchten Webseiten. Sie werden, je nach Anwendungskontext, unterschiedlich lange gespeichert. Die Internet Archive WaybackMachine[4] ist ein Beispiel für eine äußerst lange Speicherung. Ihr Ziel ist es, Webseiten und deren Veränderungen für die Öffentlichkeit zu archivieren. Aus diesem Grund wird versucht, Repositoryeinträge möglichst nicht zu löschen. In anderen Anwendungskontexten werden aber die Inhalte im Repository aus Platzgründen häufig nur vorübergehend gespeichert. Die Inhalte müssen jedoch mindestens so lange vorbehalten werden, bis der Indexer sie vearbeitet hat.

Der Indexer und die Collection Analysis sind Programme, die die Webseiten aus dem Repository verarbeiten, um Suchmaschinenindexinhalte zu erstellen. Welche und wie viele Indexe eine Suchmaschine besitzt, ist abhängig von den Funktionen, die sie anbietet. Arasu et al. [ACGM+01] unterscheiden zwischen *Text-Index*, *Structure-Index* und *Utility-Index*. Der Indexer parst einzelne Webseiten, um Text- und Structure-Index zu erstellen. Die Collection Analysis erzeugt den Utility-Index-Inhalt, indem sie Webseiten exzessiver analysiert. Sobald ein Nutzer mittels Stichworten suchen können soll, wird ein Text-Index benötigt. Dieser

[4] `http://archive.org/web/web.php` letzter Zugriff 20.02.2013, 17:30 Uhr

enthält Wörter bereits geparster Webseiten und einen Verweis auf die Webseiten, die diese Wörter beinhalten [DH97]. Zudem können in einem zusätzlichen Feld zu jedem Indexeintrag Gewichtungen gespeichert werden. Diese ergeben sich aus der Verwendung des Wortes auf der Webseite. So können beispielsweise Wörter, die in HTML-Tags wie `h1` oder `b` vorkommen, stärker gewichtet werden. Solche Tags werden zur Markierung von Überschriften und Hervorhebung von Worten genutzt. Es wird dadurch davon ausgegangen, dass solche Worte bezeichnend für den Inhalt der Webseite sind. Mit einem Structure-Index bezeichnen Arasu et al. [ACGM+01] die Abbildung von Linkstrukturen auf einen Index. Verweist eine Webseite durch einen Link auf eine andere Webseite, so kann dies im Structure-Index vermerkt werden. Neben diesen beiden Indexen können weitere Indexe erstellt werden. Sie werden zusammengefasst als Utility-Index bezeichnet. Ein konkreter Utility-Index kann genutzt werden, um zum Beispiel Ähnlichkeiten zwischen Webseiten zu modellieren. In einem solchen Index würden Webseiten miteinander verknüpft werden, die thematisch ähnlich zueinander sind. Für die Grundfunktionalität einer Suchmaschine, die anhand von Stichworten sucht, wird ein solcher Index allerdings nicht benötigt.

Die letzten beiden wichtigen Komponenten einer Suchmaschine sind die Query Engine und das Ranking. Die Query Engine verarbeitet die Eingaben des Nutzers und liest passende Ergebnisse aus dem Text-Index. Bevor sie die Ergebnisse ausgibt, erfolgt allerdings eine Sortierung. Diese wird vom Ranking-Programm durchgeführt. Mit dem Sortieren wird versucht, für den Nutzer interessante und nützliche Ergebnisse möglichst zu Beginn der Ergebnisausgabe zu präsentieren. Je besser somit das Ranking funktioniert, desto zufriedener sind die Nutzer mit den Suchergebnissen [Vau04]. Dafür wird auf die zuvor beschriebene Gewichtung im Text-Index, sowie gegebenenfalls auf die Structure- und Utility-Indexe zurückgegriffen. Dreilinger und Howe [DH97] beschreiben zudem, wie durch das Beobachten der Interaktionen der Nutzer mit der Suchmaschine die Ergebnisse weiter verbessert werden können. Klickt ein Nutzer beispielsweise einen Link aus den Ergebnissen an, wird angenommen, dass es sich für den Nutzer um ein nützliches Ergebnis handelte. Somit wird die Priorität des Links in diesem Suchkontext erhöht. Hierfür wird in dieser Realisierung ein zusätzlicher Index verwendet. Die Sortierung wird in dem Falle mithilfe dieses Utility-Indexes und anderen Parametern berechnet.

2.1.2 Aufbau und Arbeitsweise einer Meta-Suchmaschine

Jansen et al. [JSK07] beschreiben eine Meta-Suchmaschine wie folgt. In einem ersten Schritt leitet sie die Suchanfragen an andere Suchmaschinen weiter. Anschließend nimmt die Meta-Suchmaschine deren Ergebnisse entgegen, um sie zu verarbeiten. Die verschiedenen Ergebnisse werden zusammengeführt und beispielsweise umsortiert. Abschließend werden sie dem Benutzer einheitlich präsentiert. Die in dieser Bachelorarbeit entwickelte Suchmaschine ist eine reine Meta-Suchmaschine. Aus diesem Grund wird nun die Arbeitsweise einer solchen anhand eines typischen Nutzungsszenarios ausführlicher erklärt. Dieses ist zusätzlich in Abbildung 2.2 abgebildet. Soweit nicht weiter angegeben, stammen die nachfolgenden Informationen aus der von Dreilinger und Howe [DH97] beschriebenen Architektur einer Meta-Suchmaschine. Diese Architektur unterteilen sie in die drei Komponenten *Dispatch Mechanism*, *Interface Agents* und *Display Mechanism*.

Zunächst muss ein Nutzer seinen Suchbegriff formulieren und diesen an die Meta-Suchmaschine übertragen. Dazu benutzt er die bereitgestellte Oberfläche der Meta-Suchmaschine. Anhand der Nutzereingabe entscheidet der Dispatch Mechanism, an welche der Meta-Suchmaschine bekannten Suchmaschinen diese Eingabe weitergeleitet werden soll. Es handelt sich dabei um einen Algorithmus, der eine einzige, mehrere oder alle Suchmaschinen auswählt. Für jede Suchmaschine gibt es ein spezielles Programm, welches als Interface Agent bezeichnet wird. Da die Eingabensyntax von Suchmaschinen unterschiedlich sein kann [HUHN01], vorverarbeitet ein Interface Agent die Nutzereingabe für die ihm zugeteilte Suchmaschine. Nach dieser Anpassung wird die unter Umständen veränderte Eingabe an die Suchmaschine weitergeleitet. Die übergibt die erhaltene Eingabe ihrer Query Engine, welche anhand dieser aus dem Index passende Ergebnisse extrahiert. Im Anschluss bewertet das Ranking die Relevanz der Ergebnisse und sortiert sie gegebenenfalls um. Die Rückgabe der Suchmaschine wird von den zugehörigen Interface Agents auf ein einheitliches Format gebracht. Diese nachträgliche Bearbeitung ist nötig, da die Programmierschnittstellen der Suchmaschinen das Rückgabeformat selbst bestimmen [Wil10]. Der Display Mechanism wartet, bis er die Ergebnisse aller genutzten Interface Agents erhalten hat. Die Suchmaschinenergebnisse werden anschließend von dem Display-Mechanism-Algorithmus zusammengeführt. Dieser kann darauf versuchen, Duplikate zu entdecken und sie zu entfernen. Wie auch eine reine Suchmaschine kann eine Meta-Suchmaschine

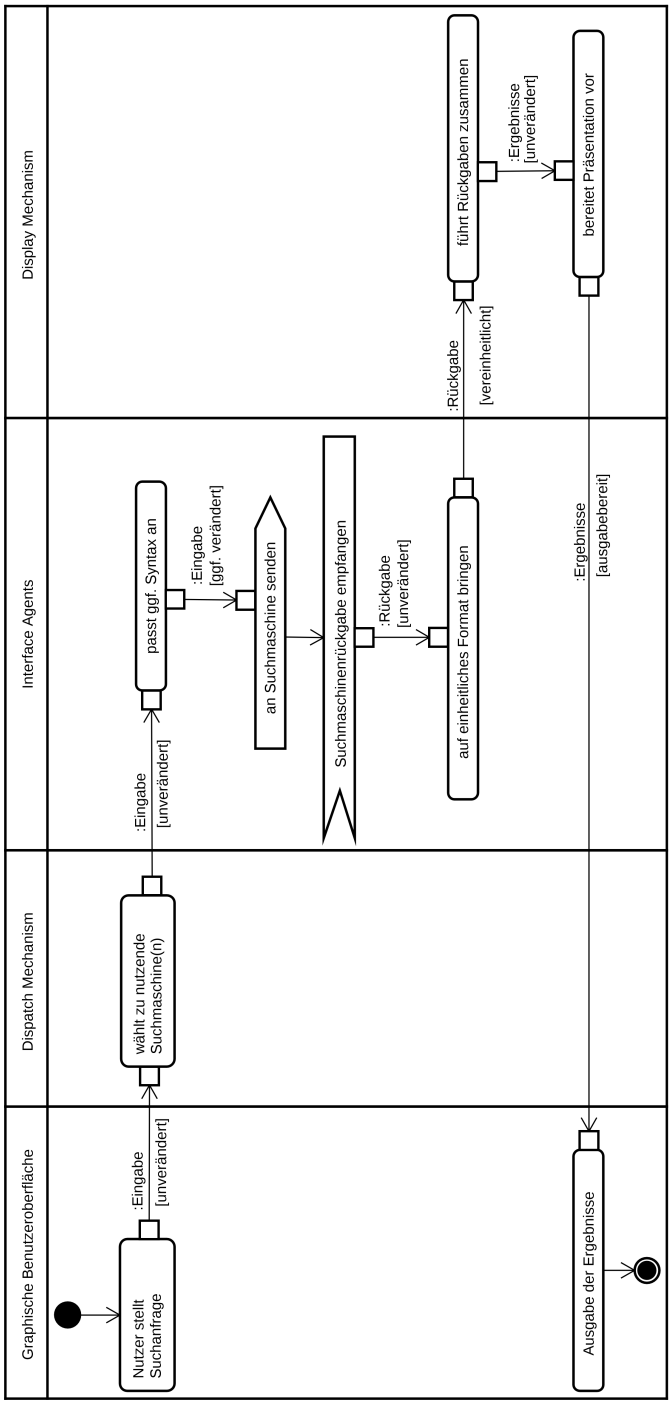

Abbildung 2.2: Diese Abbildung zeigt die Arbeitsweise und den grundlegenden Aufbau einer Meta-Suchmaschine mittels eines Aktivitätsdiagramm.

die Ergebnisse erneut umsortieren. Um allerdings eine komplexere Sortierung durchführen zu können, ist ein eigener Index nötig, wovon hier nicht ausgegangen wird. Bevor die Ergebnisse dem Nutzer präsentiert werden können, muss sie der Display Mechanism für die Darstellung im Webbrowser aufbereiten.

2.2 Information Flow Ontology

In dem nachfolgenden Abschnitt 2.2.1 wird der Aufbau und die für diese Bachelorarbeit wichtigen Funktionen der Information Flow Ontology (*InFO*) [KS12] erklärt. Der Abschnitt soll nur ein grundlegendes Verständnis für die Nutzung von InFO schaffen. Deshalb wird in Abschnitt 2.2.1 beispielsweise nicht auf die konkreten Bezeichner einzelner Sprachelemente eingegangen. Ebenso wird auf eine detaillierte Beschreibung des Aufbaus und der Abhängigkeiten verzichtet. Der darauffolgende Abschnitt 2.2.2 beschreibt hingegen die technische Umsetzung von InFO im Detail. Abschließend wird in Abschnitt 2.2.3 die *Search Engine-based Flow Control Ontology* (nachfolgend abgekürzt mit *SEFCO*) vorgestellt. Diese ist die Domänenontologie, welche das InFO-Framework um Suchmaschinen-spezifische Sprachelemente erweitert.

2.2.1 Grundlagen zur Information Flow Ontology

InFO [KS12] ist eine in der Web Ontology Language [G+09] (kurz: *OWL*) umgesetzte Ontologie. Sie bietet die Möglichkeit, Zugriffe auf Informationen im Internet wie zum Beispiel Webseiten zu regulieren. Aktuell liegt InFO in Version 0.1[5] vor. Version 0.2 befindet sich in Entwicklung. Diese Bachelorarbeit nutzt deshalb InFO 0.1. In [KS12] beschreiben die Autoren die funktionalen Anforderungen an die Ontologie. Anhand dieser Anforderungen werden nachfolgend Funktionen für ein grundlegendes Verständnis von InFO erläutert.

InFO kann einen konkreten Informationsfluss mittels Regeln entweder erlauben oder verbieten. Was konkret bei einem verbotenen Zugriff geschehen soll, ist abhängig von der umsetzenden Regel. Es kann zum Beispiel stattdessen ein anderer Inhalt oder auch kein Inhalt übertragen werden. InFO bietet weiter die Möglichkeit eine allgemeine Rückfallregel zu wählen. Diese wird angewandt,

[5] `http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/0.1/` letzter Zugriff 14.09.2013, 13:00 Uhr

wenn ein Informationsfluss reguliert werden kann, aber keine Regel genau diesen beschreibt. Falls in dem Kontext zum Beispiel nur wenige den Zugriff verbietende Regeln existieren, könnte eine Rückfallregel den Zugriff auf Inhalte pauschal erlauben. Eine Verbotsregel kann einen gesamten Internetauftritt wie `http://stormfront.org` umfassen. Zeitgleich kann eine Erlaubensregel eine darunter liegende Webseite wie `http://stormfront.org/forum/sendmessage.php` erlauben. Für solche Fälle stellt InFO Konfliktlösungs- und Priorisierungsalgorithmen bereit. Diese kommen auch zum Einsatz, wenn sich beispielsweise mehrere Verbotsregeln auf denselben Inhalt beziehen. Generell besteht ein Konflikt, wenn trotz der Priorisierung einzelner Regeln keine eindeutige Situation entsteht. Dies bedeutet, dass weiterhin Regeln mit gleicher Priorität existieren, die unterschiedliche Regulierungen über dieselben Adressen beschreiben. Demzufolge besteht ein Konflikt aus mindestens zwei Regeln. Durch Konfliktlösungs- und Priorisierungsalgorithmen wird sichergestellt, dass in diesem Szenario stets nur eine Regel angewandt wird. Mehrere Algorithmen desselben Typs werden immer mit einer Ausführungsreihenfolge spezifiziert. Somit wird verhindert, dass zwischen Konfliktlösungs- oder Priorisierungsalgorithmen untereinander Konflikte entstehen. Abschnitt 2.2.2 beschreibt unter anderem die unterschiedlichen Aufgaben der Algorithmen genauer.

Eine Policy ist eine Sammlung von Regeln, die einen gemeinsamen Zweck verfolgen. Der Zweck der Policy wird durch rechtliche Grundlagen und/oder organisatorische Richtlinien beschrieben. InFO verknüpft zudem Policys mit den verantwortlichen Institutionen. Es wird angegeben, welcher Betreiber diese Policy auf welchem Umsetzungssystem technisch durchsetzt. Zusätzlich können Policys lokale Konfliktlösungsalgorithmen definieren. Ein lokaler Konfliktlösungsalgorithmus löst Konflikte zwischen Regeln, deren Policy mit der des Algorithmusses übereinstimmt. Meta Policys sind Sammlungen von Policys. Sie beinhalten zudem die Priorisierungs- und globalen Konfliktlösungsalgorithmen. Bei Letzteren handelt es sich um Algorithmen, die auf Konflikten ungeachtet der Policy-Zugehörigkeit der Regeln arbeiten. Neben diesen Algorithmen besitzen Meta Policys die Rückfallregel. Ein Umsetzungssystem reguliert immer anhand genau einer Meta Policy. Demzufolge sind Meta Policys wie Policys auf ein Umsetzungssystem und einen Betreiber eingeschränkt.

2.2.2 Details zur Umsetzung in OWL

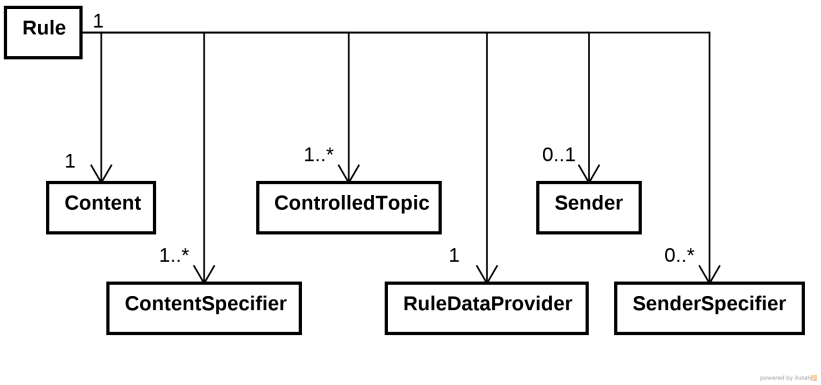

Abbildung 2.3: Diese Abbildung zeigt den vereinfachten Aufbau einer Regel anhand des Flow Control Rule Patterns.

Nachfolgend wird auf die technische Umsetzung von InFO eingegangen. Sofern nicht anders angegeben, entstammen die Informationen dem Eintrag zur Flow Control Ontology [Kas12] des offiziellen InFO-Wikis[6]. Abbildung 2.3 zeigt den vereinfachten Aufbau einer Regel anhand des *Flow Control Rule Pattern*s. Regeln werden nach diesem Pattern als `Rule` modelliert. Die `Rule` besitzt einen `Content`, dessen Zugriff reguliert wird. Dies kann zum Beispiel eine Webseite oder ein Bild sein. Durch einen oder mehrere `ContentSpecifier` wird dieser Inhalt eindeutig durch Angabe einer `Region` identifizierbar gemacht. Im Fall der Implementierung einer Suchmaschine werden durch `URLRegion`s beispielsweise konkret Adressen wie `http://stormfront.org` spezifiziert. Mittels `ControlledTopic` wird in einer `Rule` menschenlesbar beschrieben, um welche Art von Inhalt es sich bei dem durch die `Rule` regulierten `Content` handelt. Bei `http://stormfront.org` ist die Art des Inhalts beispielsweise „neonazistische Ideologien“. Der Ersteller der `Rule` wird mittels `RuleDataProvider` abgebildet. Zusätzlich kann die `Rule` auf konkrete `Sender` eingeschränkt werden. Bei `Sender`n handelt es sich im Falle einer Suchmaschine um die Nutzer, die die Suchanfragen formulieren. Es kann etwa eine Regulierung gezielt für ein Computernetzwerk erfolgen. Dieses wird durch `SenderSpecifier` auf konkrete

[6] `http://icp.it-risk.iwvi.uni-koblenz.de` letzter Zugriff 17.07.2013, 17:00 Uhr

Adressen abgebildet. Somit können zum Beispiel Einschränkungen für bestimmte IPv4-Adressen modelliert werden. Durch diese Realisierung ist es möglich, dass sich eine `Rule` nur auf eine eingeschränkte Nutzerschaft bezieht. Das ist nützlich, wenn `Rules` beispielsweise länderspezifisch sein sollen. SEFCO erweitert die Information Flow Ontology durch *Typen* von `Rules`. Dabei handelt es sich um erlaubende und verbietende `Rules`, die sich entweder auf URLs oder Hashwerte von Webseiten beziehen. Anhand dieser wird auch entschieden, welche `ContentSpecifier`-Werte berücksichtigt werden sollen. Die Erweiterungen durch SEFCO werden in Abschnitt 2.2.3 detaillierter behandelt.

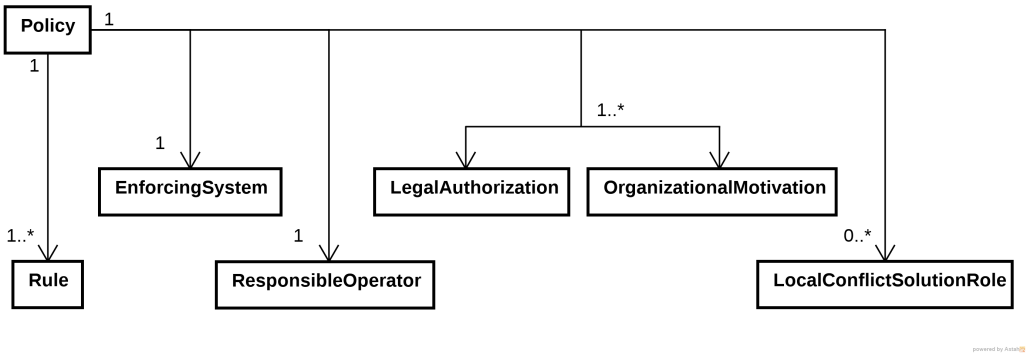

Abbildung 2.4: Diese Abbildung zeigt den vereinfachten Aufbau einer Policy anhand des Flow Control Policy Patterns.

Das *Flow Control Policy Pattern* beschreibt den Aufbau von Policys. Dieser ist in Abbildung 2.4 dargestellt. Eine `Policy` besitzt demnach eine bis mehrere `Rules`. Ob und wie `Rules` Zugriffe erlauben oder verbieten, spielt dabei keine Rolle. `Policys` werden auf Umsetzungssysteme eingeschränkt, auf denen sie einsetzbar sind. Dazu wird mit `EnforcingSystem` zum Beispiel angegeben, dass eine `Policy` auf einer konkreten Suchmaschine benutzbar ist. Diese Angaben sind wichtig, da einzelne InFO-Domänenontologien unterschiedliche Ausprägungen haben. `Rules` hingegen brauchen dies in InFO nicht, da sie stets mit `Policys` verknüpft sein müssen. Das hat auch den Vorteil, dass `Rules` für verschiedene Umsetzungssysteme wiederverwendet werden können. Mit genau einem `ResponsibleOperator` wird ausgedrückt, wer eine `Policy` technisch durchsetzt. Es handelt sich dabei um den Betreiber des Systems. Eine `Policy` kann mehrere `LocalConflictSolutionRoles` besitzen. Diese können bei `Rule`-Konflikten

zur Lösung solcher genutzt werden, bevor die globalen Konfliktlösungsalgorithmen der `MetaPolicy` greifen würden. Abschließend besitzt eine `Policy` noch mindestens eine `LegalAuthorization` oder mindestens eine `OrganizationalMotivation`. Beide können auch zusammen und mehrfach in einer `Policy` auftauchen. Die `LegalAuthorization` gibt den rechtlichen Hintergrund an [KS12]. Das kann zum Beispiel ein konkreter Paragraph eines Gesetzes sein. Durch die `OrganizationalMotivation` wird der Beweggrund des `ResponsibleOperator`s zum Einsatz dieser `Policy` ausgeführt [KS12]. Bei vielen Firmen ist das der sogenannte Verhaltenskodex (Englisch: „Code of conduct“).

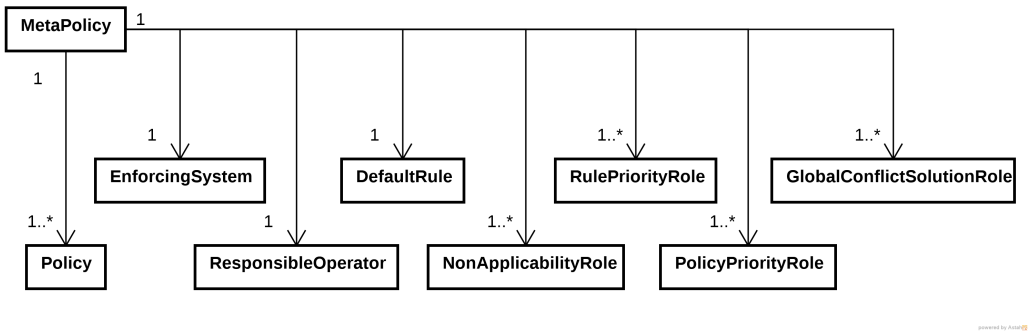

Abbildung 2.5: Diese Abbildung zeigt einen vereinfachten Aufbau einer Meta Policy anhand des Flow Control Meta Policy Patterns.

Das *Flow Control Meta Policy Pattern* beschreibt den Aufbau einer Meta Policy. Jede `Policy` ist dabei genau einer `MetaPolicy` (siehe Abbildung 2.5) zugeordnet. Diese `Policy`s müssen das gleiche `EnforcingSystem` und den gleichen `ResponsibleOperator` wie die `MetaPolicy` besitzen. Ein `EnforcingSystem` wie zum Beispiel eine Suchmaschine kann immer nur eine `MetaPolicy` zur selben Zeit einsetzen. `MetaPolicy`s besitzen genau eine `DefaultRule`. Diese wurde im vorherigen Abschnitt 2.2.1 als Rückfallregel bezeichnet und ist eine `Rule`. InFO erlaubt, dass eine `Rule` ein Sprachelement verwendet, das das einsetzende System nicht kennt oder nicht umsetzen kann. Für diesen Fall besitzt eine `MetaPolicy` mindestens eine `NonApplicabilityRole`. Damit werden Algorithmen spezifiziert, die bei unbekannten oder fehlenden Sprachelementen eingesetzt werden. Beispielsweise weist der `DiscardNonApplicableRuleAlgorithm` das `EnforcingSystem` an, `Rule`s mit nicht unterstützten Sprachelementen zu verwerfen. `MetaPolicy`s spezifizieren weiter `RulePriorityRole`s

und PolicyPriorityRoles. Erneut müssen MetaPolicys von beiden jeweils mindestens einen Algorithmus vorhalten. Im vorangegangen Abschnitt 2.2.1 wurden diese Algorithmen Priorisierungsalgorithmen genannt. Sie werden genutzt, noch bevor Konfliktlösungsalgorithmen eingesetzt werden. Durch diese Algorithmen werden Rules beziehungsweise Policys nach ihrer Priorität sortiert. Die Sortierung bestimmt, welcher Rule eine höhere Priorität als einer anderen zugewiesen wird. Erst wenn so keine strikte Ordnung entsteht, werden Konfliktlösungsalgorithmen benötigt. Dazu besitzt eine MetaPolicy mit GlobalConflictSolutionRole mindestens einen globalen Konfliktlösungsalgorithmus. Diese Algorithmen werden bei Konflikten zwischen Rules angewandt. Konflikte zwischen Rules unterschiedlicher Policys können auch nach dem Ausführen der, durch LocalConflictSolutionRole beschriebenen Algorithmen, existieren. Der Grund dafür ist, dass diese Algorithmen nur innerhalb ihrer assoziierten Policy eine konfliktfreie Situation erzeugen. Abschnitt 4.2.3 und Abschnitt 4.2.4 gehen detailliert auf die Anwendung der Algorithmen ein.

2.2.3 Search Engine-based Flow Control Ontology

Die in dieser Bachelorarbeit entwickelte Suchmaschine nutzt als Ontologie die Search Engine-based Flow Control Ontology. SEFCO wird zusammen mit InFO in ihrer aktuellen Version 0.1[7] verwendet. Die nachfolgenden Informationen entstammen dem zu SEFCO gehörigen Eintrag [Kas13] des offiziellen InFO-Wikis. Sie erweitert InFO um verschiedene Regeln und Algorithmen zur Priorisierung von Regeln. Die Regeln können durch den Anwendungsbereich ihrer Regulierung unterschieden werden. So stellt SEFCO Regeln bereit, die Informationsflüsse anhand von URLs oder Hashwerten regulieren. Die URL-basierten Regeln teilen sich in URLAllowingRule und URLBlockingRule auf. Erstere erlaubt einen Zugriff auf die in der Regel spezifizierten URLs und die andere verbietet ihn. Die Hashwert-basierten Regeln heißen analog HashValueAllowingRule und HashValueBlockingRule. Zu vergleichende Hashwerte werden innerhalb der Regel spezifiziert. Sie können über eine gesamte Webseite oder über vorgegebene Teile einer Webseite berechnet werden.

[7] http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/0.1/
letzter Zugriff 14.09.2013, 13:00 Uhr

SEFCO stellt acht weitere Priorisierungsalgorithmen bereit. Diese beziehen sich allesamt auf URL-basierte Regeln. Die Algorithmen `PreferLongestDomainNameAlgorithm` und `PreferShortestDomainNameAlgorithm` weisen eine umsetzende Suchmaschine an, Regeln mit mehr Labels oder mit weniger Labels zu bevorzugen. Labels in Domainnamen werden durch Punkte getrennt [BLFM+05]. Die Webseite `https://www.example.org` hat beispielsweise die drei Labels „`www`“, „`example`“ und „`org`“. Sie ist damit länger als etwa `https://example.org`, wodurch `PreferLongestDomainNameAlgorithm` ihr eine höhere Priorität zuweisen würde. SEFCO erweitert InFO ebenfalls um die beiden Algorithmen `PreferLongestPathAlgorithm` und `PreferShortestPathAlgorithm`. Diese regulieren anhand der Länge des URL-Pfades. Die Länge entspricht der Anzahl an Verzeichnissen und Dateien in einer URL. So besitzt `https://example.org/on` ein Verzeichnis „`on`“ und `https://example.org/on/se.html` ein Verzeichnis „`on`“ und eine Datei „`se.html`“. Durch die beiden Algorithmen `PreferLongestQueryStringAlgorithm` und `PreferShortestQueryStringAlgorithm` werden Regeln anhand der Länge des Query-Strings ihrer URL priorisiert. Der Query-Teil einer URL wird mit einem Fragezeichen markiert [BLFM+05]. Die zwei Algorithmen bestimmen die Länge des Query-Strings über die Anzahl der Parameter. Sie tauchen häufig als Parameter-Wert-Paare auf. Diese Parameter-Wert-Paare oder einzelne Parameter werden meist durch Et-Zeichen oder Semikolons voneinander getrennt [BLC95]. Eine verpflichtende Konvention über das Zeichen zur Trennung existiert allerdings nicht (vgl. [BLFM+05]). Auch die Trennung der Parameter von ihren Werten ist nicht standardisiert (vgl. [BLFM+05]). Nach [BLFM+05] wird hierfür häufig das Gleichheitszeichen verwendet. Es werde angenommen, das Et-Zeichen würde als Zeichen zur Trennung der Parameter beziehungsweise Parameter-Wert-Paare genutzt. Weiter werde angenommen, das Gleichheitszeichen würde zur Trennung der Parameter und Werte verwendet. In dem Fall wäre beispielsweise eine URL mit dem Query-String „`a=b&c=d`“ länger als eine URL mit dem Query-String „`a=b`“. Die erste URL würde dadurch eine höhere Priorität erhalten, wenn `PreferLongestQueryStringAlgorithm` angewandt wird. Mit `PreferSingleFileToWebSiteAlgorithm` werden Regeln, die sich auf einzelne Dateien beziehen, vom Umsetzungssystem höher priorisiert als Regeln, die Internetauftritte beschreiben. Als einzelne Dateien gelten beispielsweise Webseiten oder Bilder. Ebenso stellt SEFCO eine umgekehrte Sortierung mit `PreferWebSiteToSingleFileAlgorithm` bereit, die Internet-

auftritte bevorzugt. Wie diese Algorithmen im Detail umgesetzt werden, wird in Abschnitt 4.2.3 beschrieben.

Kapitel 3

Anforderungsanalyse

Die Aufgabenstellung dieser Bachelorarbeit ist die Implementierung einer Policy-basierten Suchmaschine zur Internetregulierung. Als Name wurde für diese Suchmaschine Polsearchine [ˈpɒlsɜː(ɹ)t͡ʃɪn] gewählt. Es handelt sich dabei um ein Kofferwort, das sich aus einer freien englischen Übersetzung des Titels dieser Arbeit ableitet. Polsearchine ist damit kurz für **Pol**icy-based regulating **Search** Eng**ine**. Dadurch beinhaltet der Name eine Kernbeschreibung dieser Suchmaschine reduziert auf drei Silben. Es gab zudem während der Recherche keinen Hinweis darauf, dass dieser Name bereits für ein Produkt genutzt wird.

In diesem Kapitel werden die Anforderungen an Polsearchine vorgestellt. Dabei beginnt jeder Abschnitt mit einem Text, in dem die Anforderungen beschrieben und begründet werden. Die Relevanz der Anforderungen wird durch die Worte „muss“, „soll“ und „kann“ ausgedrückt. Die Gewichtung der Worte ist angelehnt an RFC 2119 [Bra97]. Eine Muss-Anforderung drückt dabei aus, dass diese Anforderung essentiell ist und nicht vernachlässigt werden darf. Soll-Anforderungen beschreiben Anforderungen, deren Umsetzung empfohlen ist. Sie dürfen deshalb nur bei triftigen Gründen vernachlässigt werden. Kann-Anforderungen besitzen die geringste Gewichtung und können vernachlässigt werden. Sie beschreiben mögliche Erweiterungen, die in manchen Szenarien als umsetzenswert erachtet werden können. Abschnitt 3.1 behandelt die grundsätzlichen Anforderungen an Polsearchine. Anschließend werden in Abschnitt 3.2 Anforderungen an die Regulierung gestellt. Der darauffolgende Abschnitt 3.3 beschreibt Anforderung an die Verarbeitung der InFO-Policys. In Abschnitt 3.4 werden Benutzerkonten eingeführt und diesbezüglich Anforderungen aufgestellt. Abschnitt 3.5 beschreibt Anforde-

rungen an die Verwaltung und Speicherung von Policy-Dateien. Abschließend werden in Abschnitt 3.6 Anforderungen an die Administration von Polsearchine gestellt.

3.1 Grundlegende Anforderungen

Eine eigene Suchmaschine von Grund auf zu entwickeln, wäre außerhalb der Möglichkeiten einer in einer Bachelorarbeit entwickelten Software. Deshalb muss Polsearchine als Meta-Suchmaschine entwickelt werden (Anforderung A1). Viele Suchmaschinenanbieter stellen Schnittstellen zur Nutzung ihrer Suchmaschine bereit. Diese Schnittstellen werden als *API* bezeichnet. API steht für Application Programming Interface (Deutsch: „Anwendungsprogrammierschnittstelle“). Polsearchine muss Mithilfe mindestens einer solchen API entwickelt werden (Anforderung A1.1). Die Suchmaschinen der APIs werden fortan zur besseren Unterscheidung von Polsearchine als *Backend-Suchmaschine* bezeichnet. Meta-Suchmaschinen können grundsätzlich Informationen unterschiedlicher Suchmaschinen gleichzeitig darstellen. Dies erfordert allerdings eine Vermischung und Umsortierung der Ergebnismengen der unterschiedlichen Suchmaschinen. Hinzu kommt dabei die Notwendigkeit, doppelte Ergebnisse unterschiedlicher Suchmaschinen zu erkennen, um keine Duplikate anzuzeigen. Da es sich bei Polsearchine allerdings um einen Machbarkeitsnachweis handelt, soll die Komplexität diesbezüglich gering gehalten werden. Deshalb ist die gleichzeitige Nutzung mehrere Backend-Suchmaschinen nur eine Kann-Anforderung (Anforderung A1.2). In jedem Fall muss die Backend-Suchmaschine austauschbar sein (Anforderung A1.3), um nicht von einem Anbieter allein abhängig zu sein. Polsearchine muss anhand von Stichworten nach Webseiten suchen können (Anforderung A1.4). Eine Erweiterung um eine Suchmöglichkeit nach beispielsweise Videos (Anforderung A1.5) oder Bildern (Anforderung A1.6) kann umgesetzt werden. Sie ist aber für Polsearchine als Machbarkeitsnachweis nicht zwingend erforderlich und deshalb nur eine Kann-Anforderung. Polsearchine muss eine Webanwendung sein, mit der der Endbenutzer mittels seines Webbrowsers interagieren kann (Anforderung A1.7). Dies ermöglicht eine dezentrale Nutzung der Meta-Suchmaschine. Zusätzlich benötigt der Endbenutzer so keine separate Anwendung zur Nutzung, sondern kann einen vorhandenen Webbrowser nutzen. Es soll eine eingeschränkte Suche nach Dateitypen möglich sein, die von der Backend-Suchmaschine unterstützt werden

(Anforderung A1.8). Wenn der Nutzer seine Suchanfrage mit „`filetype:PDF`“ beginnt, soll beispielsweise nur nach Dateien des Typs PDF gesucht werden. Dabei soll allerdings nur nach den Dateitypen gesucht werden können, die die aktuell genutzte Backend-Suchmaschine unterstützt.

Es soll ein IP-Adressen-Muster konfigurierbar sein, um die Suche auf IP-Adressen einzuschränken (Anforderung A2). Dadurch ist es möglich, die Benutzung der Suchfunktion auf Gruppen von Endbenutzern anhand ihrer IP-Adressen zu limitieren. Eine denkbare Realisierung würde den Beginn der Endbenutzer-IP-Adresse mit dem Muster vergleichen. Wäre das Muster etwa „`141.26.`“ so hätte ein Endbenutzer mit der IP-Adresse 141.26.69.83 Zugriff auf die Suchfunktion. Polsearchine muss nach §§5, 13 Telemediengesetz[1] ein Impressum und eine Datenschutzerklärung besitzen (Anforderung A3 und Anforderung A4). Die Datenschutzerklärung soll zusätzlich, abhängig von der aktuell gewählten Backend-Suchmaschine, dynamisch erweitert werden können (Anforderung A4.1). Dies ist wichtig, da es Betreibern von Suchmaschinen-APIs möglich ist, von der nutzenden Anwendung angepasste Datenschutzerklärungen zu verlangen (siehe bspw. [Yah12a]). Dazu soll ein Dateipfad existieren, in dem die für die Backend-Suchmaschinen spezifische Erweiterungen der Datenschutzerklärung gespeichert werden (Anforderung A4.1.1). Zusammengefasst ergeben sich die folgende Anforderungen:

A1 Polsearchine muss eine Meta-Suchmaschine sein.

- A1.1 Polsearchine muss mindestens eine Backend-Suchmaschine nutzen.
- A1.2 Polsearchine kann mehrere Backend-Suchmaschinen parallel verwenden.
- A1.3 Die genutzte Backend-Suchmaschine muss austauschbar sein.
- A1.4 Polsearchine muss nach Webseiten anhand von Stichwörtern suchen können.
- A1.5 Polsearchine kann nach Videos suchen.
- A1.6 Polsearchine kann nach Bildern suchen.
- A1.7 Polsearchine muss eine Webanwendung sein.

[1] Telemediengesetz vom 26. Februar 2007 (BGBl. I S. 179), das zuletzt durch Artikel 1 des Gesetzes vom 31. Mai 2010 (BGBl. I S. 692) geändert worden ist

A1.8 Polsearchine soll eine Suche nach, von der Backend-Suchmaschine unterstützten, Dateitypen ermöglichen.

A2 Es soll ein IP-Adressen-Muster für selektiven Zugriff konfigurierbar sein.

A3 Polsearchine muss ein Impressum besitzen.

A4 Polsearchine muss eine Datenschutzerklärung besitzen.

A4.1 Die Datenschutzerklärung muss für die aktuell aktive Backend-Suchmaschine dynamisch erweitert werden können.

A4.1.1 Es soll ein Dateipfad konfigurierbar sein, der für die Backend-Suchmaschinen spezifische Erweiterungen der Datenschutzerklärung beinhaltet.

3.2 Regulierung

Wie bereits das Kofferwort „Polsearchine“ ausdrücken soll, muss diese Meta-Suchmaschine Suchergebnisse anhand von Policys regulieren können. Genauer müssen InFO-Policy-Dateien dazu genutzt werden (Anforderung A5). Die aktuelle Version 0.1 von SEFCO sieht eine Regulierung sowohl anhand von URLs als auch anhand von Hashwerten vor (siehe Abschintt 2.2.3). Allerdings existieren für eine Regulierung mittels Hashwerten noch keine konkrete Regel-Priorisierungsalgorithmen. Ebenso ist undefiniert, über welche Teile einer Webseite eine Berechnung der Hashwerte durchgeführt wird. Aus diesem Grund wird eine Regulierung anhand von Hashwerten als Kann-Anforderung betrachtet (Anforderung A5.1). Unabhängig davon muss eine Regulierung mittels URLs möglich sein (Anforderung A5.2). Falls Policy-Dateien hinzugefügt oder gelöscht werden, kann Polsearchine die Änderungen zur Laufzeit übernehmen (Anforderung A5.3). Es handelt sich dabei aufgrund der damit verbundenen Komplexität nur um eine Kann-Anforderung.

Wenn ein Suchergebnis von einer Regulierung betroffen ist, die das Anzeigen unterbindet, muss der Endbenutzer die dazugehörigen Hintergrundinformationen einsehen können (Anforderung A6). So erhält er einen Eindruck über das Ausmaß der Regulierung seiner Suche und kann die Gründe hierfür nachvollziehen. Dem Nutzer muss angezeigt werden, wer die Regel erstellt hat (Anforderung A6.1). Dazu wird in InFO das Sprachelement `RuleDataProvider` genutzt. Weiter muss

die Art des regulierten Inhalts angezeigt werden (Anforderung A6.2). InFO drückt dies durch `ControlledTopic` aus. Zusätzlich muss der Nutzer die rechtliche Grundlage der Regel und/oder den Beweggrund des Betreibers des Umsetzungssystems einsehen können (Anforderung A6.3). Es muss dabei nur mindestens eine der beiden Informationen vorhanden sein. Dies geht auf den Aufbau von InFO zurück, der mindestens eine `LegalAuthorization` oder mindestens eine `OrganizationalMotivation` fordert (siehe Abschnitt 2.2.2). Hieraus ergeben sich die folgenden Anforderungen:

A5 Polsearchine muss Suchergebnisse anhand von InFO-Policy-Dateien regulieren können.

- A5.1 Suchergebnisse können anhand von Hashwerten reguliert werden.
- A5.2 Suchergebnisse müssen anhand von URLs reguliert werden.
- A5.3 Polsearchine kann zur Laufzeit Veränderungen bezüglich der Policy-Dateien umsetzen.

A6 Zu jedem gefilterten Ergebnis muss der Nutzer die Hintergrundinformationen einsehen können.

- A6.1 Es muss der Ersteller der Regel angezeigt werden.
- A6.2 Es muss das Thema der Regel angezeigt werden.
- A6.3 Es muss die rechtliche Grundlage und/oder der organisatorische Beweggrund angezeigt werden.

3.3 InFO-Policyverarbeitung

Um anhand von InFO-Policys regulieren zu können, muss Polsearchine Policys verarbeiten können (Anforderung A7). Während dieser Verarbeitung müssen die Informationen in interne Formate umgewandelt werden, die nur Informationen enthalten, die Polsearchine zwingend zur korrekten Umsetzung der Policy-Dateien benötigt (Anforderung A7.1). Nachfolgend werden diese Formate als *Entitys* bezeichnet. Um nachträglich auf die Ergebnisse der initialen Policy-Verarbeitung Zugriff zu haben, müssen die Informationen der Policy-Dateien als Entitys in

einem persistenten Speicher gespeichert werden (Anforderung A7.2). Die Policy-Verarbeitung muss soweit ausgeführt werden, bis für jede URL eindeutig entscheidbar ist, ob und wie sie reguliert werden muss (Anforderung A7.3). Dadurch wird die Komplexität der Feststellung einer Regulierung anhand einer URL gesenkt. Zusätzlich reduziert die eindeutige Entscheidbarkeit einer Regulierung die Menge an Informationen, die im persistenten Speicher gespeichert werden. Regulierungen müssen zur Laufzeit anhand der Entitys festgestellt und durchgeführt werden (Anforderung A7.4). Da Entitys die Informationen der Policy-Dateien beinhalten, werden die Regulierungen weiterhin durch die Policys festgelegt. Eine Regulierung zur Laufzeit anhand der Entitys zu fordern, verhindert lediglich, dass bei jeder Suchanfrage der aufwändige Arbeitsschritt der Policy-Auswertung wiederholt werden muss. Es lassen sich die Anforderungen dieses Abschnitts wie folgt zusammenfassen:

A7 Polsearchine muss Policys verarbeiten können.

- A7.1 Die Policy-Dateien müssen in Entitys umgewandelt werden, die nur die für Polsearchine zwingend benötigten Informationen enthalten.
- A7.2 Entitys müssen persistent gespeichert werden.
- A7.3 Für jede URL muss eindeutig feststellbar sein, ob und wie sie reguliert wird.
- A7.4 Regulierungen müssen zur Laufzeit anhand der Entitys festgestellt und umgesetzt werden.

3.4 Benutzerkonten

Polsearchine muss Benutzerkonten unterstützen (Anforderung A8). Ein Benutzer muss sich bei Polsearchine durch seinen Webbrowser anmelden können (Anforderung A8.1). Die Übertragung der Anmeldedaten soll über eine verschlüsselte Verbindung erfolgen (Anforderung A8.1.1). Dies ist wichtig, um das Abfangen der Anmdeldaten im Klartext durch Dritte zu verhindern. Falls eine andere Person Zugriff auf den Webbrowser eines angemeldeten Benutzers besitzt, erhält er unberechtigten Zugriff auf Polsearchine. Um dem Benutzer eine größere Kontrolle über sein Benutzerkonto zu bieten, kann eine Abmeldung unterstützt werden (Anforderung A8.2). Es handelt sich dabei um eine Kann-Anforderung, da es dem

Endbenutzer möglich ist, den Zugriff auf seinen Webbrowser einzuschränken. Dahingegen hat er keinen direkten Einfluss auf eine verschlüsselte Übertragung, weshalb Anforderung A8.1.1 eine Soll-Anforderung ist. Ein Benutzerkonto muss durch einen eindeutigen Nutzernamen identifiziert werden (Anforderung A8.3). Der Name muss mit einem Passwort assoziiert werden (Anforderung A8.4). Zusätzlich soll ein Benutzername mit *Rechtegruppen* verknüpfbar sein (Anforderung A8.5). Als Rechtegruppe wird in dieser Bachelorarbeit eine Modellierung von Benutzerrechten bezeichnet, die mehrere Nutzer teilen können. Eine Assoziation mit einer konkreten Rechtegruppe kann beispielsweise die Voraussetzung für den Zugriff auf eine Webseite sein. Benutzerkonten müssen zusammen mit ihrem Passwort und ihren assoziierten Rechtegruppen persistent gespeichert werden (Anforderung A8.6). Eine persistente Speicherung ist beispielsweise in einer Datei oder einer Datenbank möglich. Dabei ist es wichtig, dass das Passwort nicht im Klartext gespeichert werden darf. Stattdessen muss ein Hashwert gewählt werden (Anforderung A8.7). Dadurch ist das Passwort nur dem Benutzer bekannt, selbst wenn eine Person direkten Zugriff auf die persistente Speicherung besitzt [SBEW01]. Es wird in dieser Bachelorarbeit lediglich eine Rechtegruppe benötigt. Die Anzahl der Benutzerkonten ist aufgrund der erweiterten Rechte auf Polsearchine gering. Deshalb wird eine graphische Benutzeroberfläche zur Erstellung und Verwaltung der Benutzerkonten nicht benötigt. Eine solche Benutzerkontenverwaltung kann dennoch erstellt werden (Anforderung A8.8). Diese Anforderungen werden wie folgt zusammengefasst:

A8 Polsearchine muss Benutzerkonten unterstützen.

- A8.1 Benutzer müssen sich im Webbrowser bei Polsearchine anmelden können.
  - A8.1.1 Die Übertragung der Anmeldedaten soll über eine verschlüsselte Verbindung erfolgen.
- A8.2 Polsearchine kann eine Funktion zur Abmeldung von Benutzern besitzen.
- A8.3 Der Nutzername muss einmalig sein.
- A8.4 Ein Benutzerkonto muss mittels einer Kombination aus Nutzername und Passwort modelliert werden.

A8.5 Benutzerkonten müssen mit unterschiedlichen Rechtegruppen verknüpft werden können.

A8.6 Benutzerkonten und ihre Assoziationen müssen persistent gespeichert werden.

A8.7 Das Passwort muss als Hashwert gespeichert werden.

A8.8 Es kann eine graphische Benutzeroberfläche zur Benutzerkontenverwaltung eingerichtet werden.

3.5 Verwaltung und Speicherung von Policy-Dateien

Policy-Dateien sollen über eine Webseite verwaltet werden können (Anforderung A9). Dadurch ist eine dezentrale Verwaltung möglich. Änderungen an den Policy-Dateien haben Einfluss auf die Arbeitsweise von Polsearchine. Deshalb muss die Verwaltung auf eine Rechtegruppe eingeschränkt sein (Anforderung A9.1). Wenn nachfolgend von Benutzern gesprochen wird, sind damit nicht alle Endbenutzer gemeint. Stattdessen bezieht sich der Begriff in diesem Abschnitt auf die Teilmenge an Benutzern, deren Benutzerkonto mit der erforderlichen Rechtegruppe verknüpft ist. In der Verwaltung müssen Benutzer einsehen können, welche Policy-Dateien zur Nutzung mit Polsearchine gespeichert sind (Anforderung A9.2). Dazu müssen den Benutzern die Namen der gespeicherten Policy-Dateien angezeigt werden. Es kann zudem eine menschenlesbare Kurzbeschreibung der Policys angezeigt werden (Anforderung A9.3). Da InFO kein eigenes Sprachelement dafür besitzt, handelt es sich nur um eine Kann-Anforderung. Weiter muss es Benutzern möglich sein, neue Policy-Dateien durch Hochladen bei Polsearchine hinzuzufügen (Anforderung A9.4). Außerdem muss es Benutzern möglich sein, Policy-Dateien zu löschen (Anforderung A9.5).

Die Policy-Dateien sollen auf dem Dateisystem des Servers gespeichert werden (Anforderung A10). Dadurch sind sie persistent gespeichert. Zusätzlich kann eine Person mit Zugriff auf das Dateisystem des Servers die Policy-Dateien verwalten, ohne die Webanwendung zuvor zu starten. Für die Speicherung soll ein Dateipfad spezifizierbar und konfigurierbar sein (Anforderung A10.1). Zudem soll eine Dateiendung für Policy-Dateien konfigurierbar sein (Anforderung A10.2).

Dies ist nützlich, um etwa Policy-Dateien mit einer einheitlichen Dateiendung zu speichern. Zusammengefasst ergeben sich die folgenden Anforderungen:

A9 Policy-Dateien sollen über eine Webseite verwaltet werden können.

- A9.1 Die Verwaltung von Policy-Dateien muss auf eine Rechtegruppe eingeschränkt sein.
- A9.2 Die Namen der Policy-Dateien müssen in der Policy-Verwaltung angezeigt werden.
- A9.3 Es kann eine menschenlesbare Kurzbeschreibung der Policys angezeigt werden.
- A9.4 Policy-Dateien müssen in der Policy-Verwaltung durch Hochladen hinzugefügt werden können.
- A9.5 Policy-Dateien müssen aus der Policy-Verwaltung heraus gelöscht werden können.

A10 Policy-Dateien sollen auf dem Dateisystem des Servers gespeichert werden.

- A10.1 Es soll ein Dateipfad zur Speicherung der Policy-Dateien konfigurierbar sein.
- A10.2 Es soll eine Dateiendung für Policy-Dateien konfigurierbar sein.

3.6 Anwendungsadministration

Polsearchine muss ein *Backend* besitzen (Anforderung A11). Dabei handelt es sich um eine zentrale Oberfläche zur externen Administration von Polsearchine. Extern bedeutet in diesem Sinne, dass der administrierende Nutzer keinen direkten Zugriff auf den Server benötigt. Da Polsearchine nach Anforderung A1.7 eine im Webbrowser benutzbare Webanwendung sein muss, soll das Backend eine Webseite sein (Anforderung A11.1). Der Zugriff auf das Backend muss auf Benutzerkonten eingeschränkt werden, die mit einer konkreten Rechtegruppe verknüpft sind. (Anforderung A11.2). Solche Benutzer werden nachfolgend als *Administratornutzer* und die dazugehörige Rechtegruppe als *Administratornutzer-Rechtegruppe* bezeichnet. Auf der Webseite des Backends können dem Administratornutzer Informationen über die aktuelle Konfiguration von Polsearchine gezeigt werden

(Anforderung A11.3). Diese können beispielsweise die gegenwärtig gewählte Backend-Suchmaschine oder der Pfad, in dem die Policy-Dateien gespeichert werden, sein. Ein Administratornutzer kann zudem die Konfiguration im Backend ändern (Anforderung A11.4). Änderungen können umgehend in Kraft treten (Anforderung A11.5). Dies ist etwa nützlich, um die Backend-Suchmaschine im laufenden Betrieb auszutauschen. Das Backend soll zudem die Möglichkeit bieten, Policy-Dateien zu verwalten (Anforderung A11.6). Die Anforderungen dieses Abschnitts werden wie folgt zusammengefügt:

A11 Polsearchine muss ein Backend besitzen.

- A11.1 Das Backend soll eine Webseite sein.
- A11.2 Das Backend muss den Zugriff nur Administratornutzern erlauben.
- A11.3 Dem Administratornutzer können Informationen zur aktuellen Konfiguration von Polsearchine angezeigt werden.
- A11.4 Ein Administratornutzer kann die Konfiguration im Backend verändern.
- A11.5 Änderungen an der Konfiguration können direkt umgesetzt werden.
- A11.6 Das Backend soll die Verwaltung von Policy-Dateien unterstützen.

Kapitel 4

Konzeption

Dieses Kapitel stellt die Konzeption zur Umsetzung von Polsearchine vor. Zusammen mit den Anforderungen aus Kapitel 3 ergibt sich so ein Verständnis für die konzeptionelle Arbeitsweise der Meta-Suchmaschine. Zuerst beschreibt Abschnitt 4.1 das Hinzufügen und Entfernen von Policys. Dabei wird insbesondere auf die Speicherung der Policy-Dateien eingegangen. Anschließend behandelt Abschnitt 4.2 den Anwendungsstart und die damit verbundene Policy-Verarbeitung. Abschnitt 4.3 beschreibt den Ablauf einer Suchanfrage und die dazu nötigen Algorithmen. Die Suchanfrage erstreckt sich von der Eingabe eines Suchbegriffs durch den Nutzer bis hin zur Präsentation der Ergebnisse.

4.1 Hinzufügen und Entfernen von Policys

Dieser Abschnitt beschreibt einen Algorithmus, der das Speichern von hochgeladenen Policy-Dateien nach einem einheitlichen Schema durchführt. Daraufhin wird ein weiterer Algorithmus vorgestellt, dessen Aufgabe das Anzeigen der Policy-Dateien nach Anforderung A9.2 ist. Abschließend werden mögliche Probleme aufgezeigt, die durch gleichzeitiges Hochladen und/oder Löschen von Policy-Dateien entstehen können.

Nach Anforderung A9 soll ein Administratornutzer in der Lage sein, Policy-Dateien zu verwalten. Diese Anforderung wird dadurch umgesetzt, dass die Policyverwaltung nach Anforderung A11.6 Teil des Backends ist. Neue Policys sollen nach Anforderung A9.4 durch das Hochladen von Dateien hinzugefügt werden. Dabei soll Polsearchine die Dateien auf dem Server-Dateisystem spei-

chern. Dies soll ein Algorithmus durchführen, der zusätzlich die Dateinamen der Policy-Dateien festlegt. Er soll dabei zuerst eine Validierung für die vom Nutzer hochgeladene Datei vornehmen. Diese Validierung soll anhand der Dateiendung geschehen. Die akzeptierten Dateiendungen basieren auf den akzeptierten Datenformaten. Sie werden von dem Datenspeicher bestimmt, der im Hintergrund verwendet wird. Diese Bachelorarbeit nutzt einen InFO-Parser [BGSS13], der auf Apache Jena[1] zurückgreift. Deshalb sollen vom Algorithmus nur Dateien akzeptiert werden, deren Dateiendung auf ein von Apache Jena unterstütztes Format[2] hindeutet. Dementsprechend soll eine Fehlermeldung zurückgegeben werden, wenn die Dateiendung nicht `ttl`, `nt`, `nq`, `trig`, `rdf` oder `owl` ist. Weist die Dateiendung allerdings auf ein unterstütztes Format hin, soll die Datei gespeichert werden. Als Zielverzeichnis soll das durch Anforderung A10.1 geforderte Verzeichnis zur Speicherung von Policy-Dateien genutzt werden. Die hochgeladenen Policy-Dateien sollen einem einheitlichen Schema in der Benennung der Dateien folgen. Da Dateien mit unterschiedlichen Dateiendungen akzeptiert werden, muss der Algorithmus die in Anforderung A10.2 definierte Dateiendung auslesen. Nachfolgend wird in Beispielen davon ausgegangen, dass die Dateiendung `owl` sein soll. Lädt ein Administratornutzer beispielsweise eine „`regulation.rdf`“ hoch, soll diese zu „`regulation.owl`“ umbenannt werden. Der Dateiname soll komplett kleingeschrieben sein. Lädt der Nutzer etwa eine Datei „`Regulation.owl`“ hoch, soll diese zu „`regulation.owl`“ geändert werden. Leerstellen sollen durch Unterstriche ersetzt werden. Eine „`weitere regulation.owl`“ würde somit als „`weitere_regulation.xml`“ weiterverarbeitet werden. Der Algorithmus soll außerdem verhindern, dass eine hochgeladene Datei eine bereits vorhandene überschreibt. Hierfür sollen die Dateien abschließend durchnummeriert werden. Das nachfolgende Beispiel soll dies illustrieren. Es wird angenommen, ein Nutzer lädt die Datei „`regulation.owl`“ mehrfach hoch. Beim ersten Hochladen wird eine Datei mit dem Namen „`regulation_0.owl`“ auf dem Server gespeichert. Die Zahl hinter dem Unterstrich soll dabei bei jeder weiteren Datei mit diesem Namen erhöht werden. Somit entstehen anschließend die Dateien „`regulation_1.owl`“, „`regulation_2.owl`“ und so weiter. Zusammen mit den zuvor beschriebenen

[1] `https://jena.apache.org/index.html` letzter Zugriff 14.09.2013, 12:00 Uhr

[2] `https://jena.apache.org/documentation/io/index.html` letzter Zugriff 14:09.2013, 12:00 Uhr

Verfahren wird, wenn der Nutzer im Anschluss die Datei „`weitere Regulation.owl`“ hochlädt, diese als „`weitere_regulation_0.owl`“ gespeichert. Zu Protokollierungszwecken soll in einer Logdatei vermerkt werden, dass eine Datei erfolgreich hochgeladen wurde. Hierzu soll der gewählte Dateiname gespeichert werden. Der Algorithmus soll immer erst versuchen, Dateien mit der Ergänzung „`_0`“ zu erstellen. Wenn diese Datei schon existiert, soll er bis zur nächsten freien Zahl inkrementieren. Dieses Vorgehen verhindert das Entstehen von Lücken in der Nummerierung der Dateinamen. Sie könnten sonst durch selektives Löschen einzelner Policy-Dateien entstehen.

Im Backend sollen alle Policy-Dateien angezeigt werden. Hierfür muss ein anderer Algorithmus das durch Anforderung A10.1 festgelegte Verzeichnis zur Speicherung der Policy-Dateien auslesen. Dieses Verzeichnis soll ausschließlich Policy-Dateien beinhalten. Es ist allerdings möglich, dass eine Person mit Zugriff auf den ausführenden Server dort zusätzlich andere Dateien speichert. Aus diesem Grund soll anhand der zuvor beschriebenen Restriktionen gefiltert werden. Diese bestimmen die Dateiendung und verbieten Großbuchstaben sowie Leerzeichen im Dateinamen. Stimmt die Endung nicht mit der in Anforderung A10.2 eingeführten, konfigurierbaren Dateiendung überein, so soll die Datei nicht im Backend angezeigt werden. Die Darstellung der Policy-Dateien soll anhand ihrer Dateinamen nach Anforderung A9.2 erfolgen. Eine Darstellung mittels einer Kurzbeschreibung pro Policy nach Anforderung A9.3 soll nicht umgesetzt werden. InFO bietet in Version 0.1 dazu kein eigenes Sprachelement an. Eine mögliche Lösung zur Erfüllung dieser Kann-Anforderung wird in Abschnitt 6.3 beschrieben. Die Dateinamen sollen in einer Liste dargestellt werden. Für jeden Listeneintrag muss durch das Klicken eines Knopfes die Möglichkeit bestehen, die zugehörige Datei zu löschen. Zusätzlich soll in einer Logdatei vermerkt werden, wenn eine Datei entfernt wurde. So kann nachträglich festgestellt werden, ob Dateien aus dem Backend heraus gelöscht wurden. Es können für Polsearchine mehrere Administratornutzer existieren. Sie können unter Umständen gleichzeitig neue Policy-Dateien mit demselben Namen hochladen oder löschen wollen. In dem Fall kann ein kritischer Wettlauf (Englisch: „Race condition“) [Ung95] entstehen. Um dies zu verhindern, muss sichergestellt werden, dass der Zugriff auf das Dateisystem beim Hochladen und Löschen exklusiv gesperrt wird. Dadurch kann zur selben Zeit nur maximal eine der beiden Operationen durchgeführt werden. Wie die Policy-Dateien verarbeitet und genutzt werden, wird in Abschnitt 4.2 behandelt.

4.2 Anwendungsstart und Initialisierung

Beim Starten von Polsearchine sollen unterschiedliche Aktionen ausgeführt werden. Sie sollen das korrekte Arbeiten der Meta-Suchmaschine ermöglichen. Die Aktionen sind in Abbildung 4.1 zu sehen. Sie werden analog dazu in den nachfolgenden Abschnitten beschrieben. Zuerst wird in Abschnitt 4.2.1 erläutert, wie die Konfiguration von Polsearchine geprüft werden sollen. Anschließend beginnt die Policy-Verarbeitung. In Abschnitt 4.2.2 wird dazu beschrieben, wie die Policy-Dateien eingelesen werden. Ihre Inhalte werden auf eine Nutzbarkeit in Polsearchine geprüft und aussortiert. Dabei werden unter anderem einsetzbare Regeln extrahiert. Abschnitt 4.2.3 beschreibt, wie die einsetzbaren Regeln daraufhin unterschiedlich priorisiert werden. Der nachfolgende Abschnitt 4.2.4 behandelt die Analyse der priorisierten Regeln und eine eventuelle Konfliktlösung. Daraufhin wird in Abschnitt 4.2.5 eine mögliche Entfernung ungenutzter Regeln beschrieben. Abschnitt 4.2.6 erläutert, wie die verarbeiteten Informationen der Policy-Dateien in Polsearchine-interne Formate gewandelt werden. Zusätzlich beschreibt dieser Abschnitt knapp die Vorkehrungen zur anschließenden persistenten Speicherung dieser Formate.

4.2.1 Überprüfung der Konfiguration

Die Initialisierung umfasst die Aktionen, die während des Anwendungsstarts von Polsearchine durchgeführt werden. Dieser Abschnitt geht dabei auf die Überprüfung der Konfiguration ein. Durch die Überprüfung sollen Fehler ausgeschlossen werden, die die korrekte Arbeitsweise von Polsearchine beeinflussen können. Sobald eine Prüfung negativ ausfällt, soll die Initialisierung abgebrochen werden. Die Realisierung der Konfiguration ist an dieser Stelle unerheblich. In der Implementierung wird zur Konfiguration eine XML-Datei genutzt (siehe Abschnitt 5.4.4).

Zuerst muss geprüft werden, dass mindestens eine Backend-Suchmaschine zur Nutzung angegeben wurde. Die durch Anforderung A10.2 eingeführte Dateiendung für Policy-Dateien soll auf ihre Gültigkeit geprüft werden. Hierzu soll festgestellt werden, ob sie einer der in Abschnitt 4.1 beschriebenen Dateiendungen entspricht. Diese sind `ttl`, `nt`, `nq`, `trig`, `rdf` und `owl`. Als Nächstes soll der durch Anforderung A4.1.1 beschriebene Dateipfad untersucht werden. In diesem sollen die an die Backend-Suchmaschinen angepassten Erweiterungen der Daten-

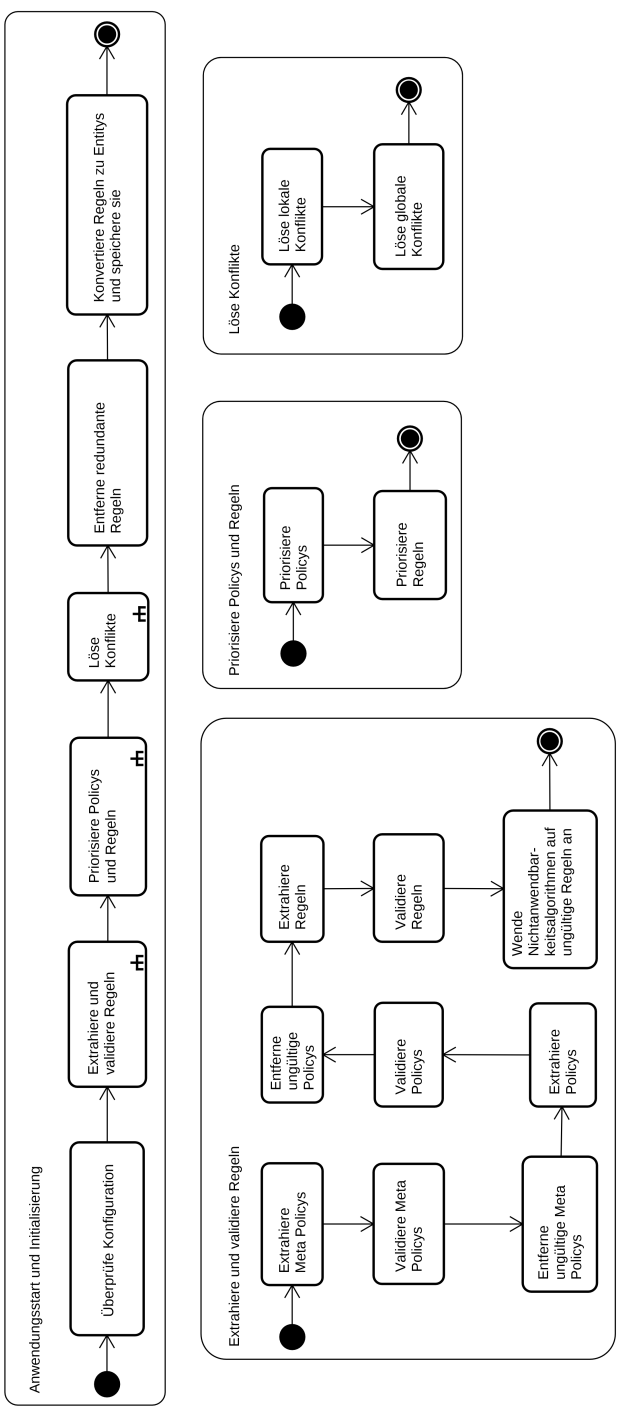

Abbildung 4.1: Diese Abbildung zeigt einen vereinfachten Ablauf der Initialisierung. Die Aktivitätsknoten im oberen Teil werden jeweils durch einen eigenen Abschnitt erklärt.

schutzerklärung gespeichert sein. Der Dateipfad muss auf dem Dateisystem des Servers existieren und Polsearchine muss Lesezugriff auf ihn besitzen. Dazu soll auch geprüft werden, ob Polsearchine die Erweiterung(en) der Datenschutzerklärung für die aktive(n) Backend-Suchmaschine(n) lesen kann. Der mittels Anforderung A10.1 spezifizierte Dateipfad zur Speicherung der Policy-Dateien muss analog dazu überprüft werden. Zusätzlich muss geprüft werden, dass Polsearchine Schreibzugriff auf den Pfad besitzt. Dies ist wichtig, da Anforderung A9.4 das Hochladen und anschließende Speichern von Policy-Dateien erfordert. Sollten diese Überprüfungen erfolgreich ausfallen, kann die Verarbeitung der Policy-Dateien erfolgen.

4.2.2 Extraktion und Validierung der Policy-Informationen

Dieser Abschnitt behandelt die Extraktion der Informationen aus den Policy-Dateien. Neben der Extraktion findet eine Überprüfung der Informationen statt. Zuerst sollen die Meta Policys aus den Dateien ausgelesen werden. Die Meta Policys sollen auf ihre generelle Gültig- und Anwendbarkeit geprüft werden. Einerseits soll verifiziert werden, dass die InFO-Kardinalitäten aus Abbildung 2.5 eingehalten wurden. Andererseits soll geprüft werden, ob das Umsetzungssystem der Meta Policy auf Polsearchine anwendbar ist. Dazu soll darauf geprüft werden, ob als Umsetzungssystem eine Suchmaschine angegeben ist. InFO bietet zusätzlich die Möglichkeit, das Umsetzungssystem durch ein `TechnicalSystem` zu konkretisieren. Mit diesem können etwa die IP-Adresse oder der Domänenname des Umsetzungssystem spezifiziert werden. In der Implementierung wird ein InFO-Parser verwendet, der diese konkrete Prüfung nicht unterstützt. Aus Zeitgründen wurde deshalb darauf verzichtet, sie zu unterstützen. Alle Meta Policys, die nach diesen beiden Prüfungen als ungültig identifiziert wurden, sollen ignoriert werden. Es sollte genau eine Meta Policy übrig bleiben, die anschließend weiterverarbeitet wird. Wird jedoch keine gültige Meta Policy oder mehrere gültige gefunden, muss die Initialisierung abgebrochen werden. Wenn auf genau einer Meta Policy gearbeitet werden kann, müssen die InFO-Policys aus ihr extrahiert werden. Diese werden ebenfalls auf ihre Gültig- und Anwendbarkeit geprüft. Die Policys müssen die InFO-Kardinalitäten aus Abbildung 2.4 berücksichtigen. Außerdem muss das Umsetzungssystem mit dem der Meta Policy übereinstimmen. Hinzu kommt die Prüfung, ob der `ResponsibleOperator` mit dem der

Meta Policy identisch ist. Nur wenn diese Tests erfolgreich sind, soll eine Policy beibehalten und weiter verarbeitet werden.

Aus den gültigen Policys sollen anschließend, die enthaltenen Regeln extrahiert werden. Die Regeln müssen auf Gültigkeit anhand der in Abbildung 2.3 dargestellten InFO-Kardinalitäten geprüft werden. Hierbei müssen auch die Regionen, die durch `ContentSpecifier` beschrieben werden, mit dem Regeltyp abgeglichen werden. Sollte eine Region wie etwa `URLRegion` nicht zu einem Regeltyp wie beispielsweise `HashValueBlockingRule` passen, muss die Region entfernt werden. Die Regel gilt in so einem Fall nicht als ungültig. Stattdessen werden dadurch Regionen entfernt, die sonst ignoriert würden. Wenn eine Regel den `ContentWebSite` besitzt, müssen ihre Regionen auf Pfaden enden. Dementsprechend dürfen die Regionen keinen Query-String, kein Fragment und keine Datei spezifizieren. Ist dies dennoch der Fall, muss die Regel als ungültig erkannt werden. Polsearchine unterstützt die `Sender`- und `SenderSpecifier`-Sprachelemente nicht. Aus diesem Grund müssen Regeln, die diese Sprachelemente benutzen, als ungültig erkannt werden. Im Gegensatz zu ungültigen Policys und Meta Policys dürfen ungültige Regeln nicht unmittelbar verworfen werden. Stattdessen müssen die Nichtanwendbarkeitalgorithmen der Meta Policy angewandt werden. InFO stellt in der aktuellen Version die beiden Algorithmen `DiscardNonApplicableRuleAlgorithm` und `DiscardNonApplicablePolicyAlgorithm` bereit. Der erste Algorithmus weist an, die ungültige Regel zu verwerfen. Durch den zweiten werden die gesamte Policy der Regel und alle mit ihr assoziierten Regeln verworfen. Beide Algorithmen garantieren durch das Entfernen von mindestens der betroffenen Regel, dass anschließend keine zusätzlichen Nichtanwendbarkeitsalgorithmen durchgeführt werden müssen.

In der verwendeten InFO-Version 0.1 können Regeln mehrere `ContentSpecifier` spezifizieren. Die in InFO spezifizierten Priorisierungsalgorithmen arbeiten allerdings immer nur auf genau einer URL (vgl. [Kas13]). Um die Regeln trotzdem korrekt anwenden zu können, sollen sie in einzelne Regeln mit jeweils einem `ContentSpecifier` aufgespalten werden. Dabei sollen die aufgeteilten Regeln bis auf den `ContentSpecifier` identisch sein. An dieser Stelle wurden die Regeln, Policys und eine Meta Policy extrahiert, auf denen nachträglich gearbeitet werden soll. Wurde dabei keine gültige Regel extrahiert, soll direkt mit der Konvertierung in Entitys in Abschnitt 4.2.6 fortgefahren werden. Anderenfalls folgt die Priorisierung der Regeln in Abschnitt 4.2.3.

4.2.3 Priorisierung von Policys und Regeln

In diesem Abschnitt wird die Priorisierung von Policys und Regeln beschrieben. Durch die Priorisierung erfolgt eine Sortierung. Mittels dieser ist feststellbar, ob eine Regel einer anderen vorzuziehen ist, wenn sie denselben Adressbereich beschreiben. Der Abschnitt beschreibt zunächst die Modellierung der Regeln, Policys und ihrer Priorisierungen. Anschließend wird der Ablauf der Priorisierung knapp erklärt. Zum Schluss wird die Arbeitsweise der Priorisierungsalgorithmen für Policys und danach für Regeln beschrieben.

Für die Priorisierung von Regeln sind konkrete Priorisierungswerte unerheblich. Es muss jedoch feststellbar sein, ob eine Regel eine höhere, niedrigere oder dieselbe Priorität wie eine andere hat. Zur Modellierung ungleicher Prioritäten kann deshalb beispielsweise eine Liste als Datentyp verwendet werden. Durch ihre interne Sortierung ist es möglich, eine Priorisierung abzubilden. Um auch Regeln gleicher Priorität zu modellieren, soll die Liste nicht direkt Regeln beinhalten. Stattdessen soll sie Mengen von Regeln enthalten. Der Pseudocode in Abschnitt 4.2.3, Abschnitt 4.2.5 und die Implementierung setzen eine interne Sortierung der Mengen voraus, um Regeln gezielt mittels Indizes zu extrahieren. Diese Sortierung besitzt allerdings keine Aussagekraft über eine Priorisierung einzelner Regeln. Mengen können somit etwa auch mittels eines Listen-Datentyps modelliert werden. Zur sprachlichen Unterscheidung wird allerdings nachfolgend der Begriff der Menge verwendet. Das erste Element der Liste soll die höchstpriorisierte Menge und das letzte Element die niedrigstpriorisierte Menge abbilden.

Zu Beginn des Policy-Priorisierungsalgorithmusses handelt es sich bei Policys um Mengen von Regeln. Nach der Anwendung der Policy-Priorisierungsalgorithmen müssen die Mengen zu einer Liste hinzugefügt werden, da sie unterschiedlich priorisiert wurden. Es ist möglich, dass nach Anwendung eines Policy-Priorisierungsalgorithmusses zwei oder mehrere Policys dieselbe Priorität besitzen. Um dies abzubilden, müssen die Mengen der betroffenen Policys zusammengeführt werden. Dabei muss die Assoziation der Regeln zu ihren Policys erhalten bleiben, um später lokale Konfliktlösungsalgorithmen anwenden zu können (siehe Abschnitt 4.2.4). Anschließend sollen die Regel-Priorisierungsalgorithmen die Regeln innerhalb der Mengen sortieren. Eine Unterscheidung in der Priorität soll durch Aufteilen in neue Mengen modelliert werden. Ebenso sollen Mengen zusammengeführt werden, die dieselbe Priorität besitzen.

Die Sortierung der Policys erfolgt anhand der `PolicyPriorityAlgorithms`. Diese Algorithmen befinden sich in der Meta Policy. In der aktuellen InFO-Version existieren drei Policy-Priorisierungsalgorithmen. Sie sind auf Sprachelemente angewiesen, die der in der Implementierung verwendete InFO-Parser nicht unterstützt. `PreferLatestPolicyAlgorithm` und `PreferOldestPolicyAlgorithm` sind etwa darauf angewiesen, den Erstellzeitpunkt von Policys auslesen zu können. Der Algorithmus `EvaluatePolicyOrderingAlgorithm` weist das Umsetzungssystem an, die Policys anhand einer internen Anordnung zu sortieren. Aus Zeitgründen wird darauf verzichtet, diese Sprachelemente eigens zu unterstützen. Stattdessen soll ein weiterer Algorithmus eingeführt werden. Dieser soll `HandleAllPoliciesWithEqualPriorityAlgorithm`[3] heißen. Er soll das Umsetzungssystem anweisen, jegliche Informationen der Policy-Priorisierung zu vernachlässigen. Dadurch sollen alle Regeln zu einer gemeinsamen Menge hinzugefügt werden.

Anschließend sollen die Regeln innerhalb ihrer Mengen neu sortiert werden. Dafür müssen die Regel-Priorisierungsalgorithmen der Meta Policy verwendet werden. Sie müssen nach der in den Policy-Dateien spezifizierten internen Reihenfolge durchgeführt werden. Die jeweilige Aufgabe der einzelnen Algorithmen wurde in Abschnitt 2.2.3 beschrieben. Für `PreferLongestDomainNameAlgorithm` und `PreferShortestDomainNameAlgorithm` müssen die Labels des Domainnamens gezählt und miteinander verglichen werden. Es soll dazu genügen, die Punkte im jeweiligen Domainnamen zu zählen. `PreferLongestPathAlgorithm` und `PreferShortestPathAlgorithm` sortieren anhand der Länge des URL-Pfades. Die Priorität der einzelnen Regeln soll durch die Anzahl der Schrägstriche in der URL berechnet werden. Dabei sollen die Schrägstriche, die das Protokoll von der eigentlichen Adresse trennen, ignoriert werden. Ist das letzte Zeichen des URL-Pfades kein Schrägstrich, so soll angenommen werden, dass die URL auf eine Datei und nicht auf ein Verzeichnis verweist. In dem Fall muss die Priorität um 1 inkrementiert werden. Die Algorithmen `PreferLongestQueryStringAlgorithm` und `PreferShortestQueryStringAlgorithm` berechnen die Priorität einzelner Regeln aus der Parameteranzahl des Query-Strings. Der Beginn des

[3] Der Algorithmus soll durch die folgende URI identifiziert werden können: `http://icp.it-risk.iwvi.uni-koblenz.de/ontologies/search_engine_flow_control.owl#HandleAllPoliciesWithEqualPriorityAlgorithm`

Query-Strings wird durch ein Fragezeichen gekennzeichnet [BLFM+05]. Er endet mit dem Ende der URL oder dem Rautesymbol [BLFM+05]. In Abschnitt 2.2.3 wurde die fehlende Standardisierung zur Unterscheidung von und innerhalb von Parametern beschrieben. Der Algorithmus soll möglichst dynamisch auf den Aufbau des Query-Strings reagieren können. Wenn dieser ein Gleichheitszeichen beinhaltet, soll angenommen werden, dass er Parameter-Wert-Paare beinhaltet. In diesem Fall bestimmt die Anzahl der Gleichheitszeichen innerhalb des Query-Strings die Priorität der Regel. Anderenfalls soll geprüft werden, ob der Query-String ein Semikolon beinhaltet. Die Priorität wird in dieser Situation durch Zählen der Semikolons errechnet. Wenn kein Semikolon im Query-String enthalten ist, soll die Priorität anhand der Anzahl der Et-Zeichen berechnet werden. Zuerst auf Semikolons statt auf Et-Zeichen zu prüfen, geht auf eine Empfehlung der W3C [RLHJ+99] zurück. Dort wird angemerkt, dass Et-Zeichen aufgrund von URI-Enkodierung zu Problemen führen können. Deshalb sind nach dieser Empfehlung Semikolons zur Parameter-Trennung beziehungsweise zur Parameter-Wert-Paar-Trennung vorzuziehen. Die letzten beiden Regel-Priorisierungsalgorithmen sind `PreferSingleFileToWebSiteAlgorithm` und `PreferWebSiteToSingleFileAlgorithm`. Zur Unterscheidung müssen die Regeln anhand ihres `Content`s untersucht werden. In der aktuellen InFO-Version 0.1 wird zwischen `WebSite` und `WebPage` als `Content` unterschieden. `WebPage` wird dabei wie auch etwa eine Bilddatei als eine einzelne Datei (Englisch: „Single file“) verstanden. Aus diesem Grund sollen Regeln, die eine `WebSite` beschreiben, in eine eigene Menge verschoben werden. Abhängig vom gewählten Algorithmus soll die neue Menge vor oder nach der alten Menge in der Liste eingefügt werden. Beim `PreferSingleFileToWebSiteAlgorithm` soll die Menge mit den Regeln, die `WebSite`s beschreiben, angehängt werden. Dementsprechend soll beim `PreferWebSiteToSingleFileAlgorithm` die Menge der `WebSite`-Regeln vorangestellt werden.

4.2.4 Konfliktlösung von Regeln

Dieser Abschnitt beschreibt knapp die durch InFO bereitgestellten lokalen und globalen Konfliktlösungsalgorithmen. Anschließend wird die Auswirkung des `Content`s einer Regel auf den Wirkungsbereich ihrer Region festgelegt. Abschließend wird das Vorgehen zur Konfliktlösung vorgestellt. Dabei wird eine Methode in Pseudcode beschrieben, die Konflikte erkennt und betroffene Regeln zurückgibt.

Nachdem die Priorisierungen abgeschlossen wurden, können Konflikte zwischen den einzelnen Regeln existieren. Falls Policys lokale Konfliktlösungsalgorithmen besitzen, sollen diese Algorithmen auf ihre in Konflikt stehenden Regeln angewandt werden. InFO stellt einen lokalen Konfliktlösungsalgorithmus bereit. Dieser heißt `DiscardConflictingRulesAlgorithm` und weist das Umsetzungssystem an, alle in Konflikt stehenden Algorithmen zu verwerfen. Die globale Konfliktlösung erfolgt nach der lokalen. Als globale Konfliktlösungsalgorithmen existieren `DiscardAffectedRulesAlgorithm` und `DiscardAffectedPolicysAlgorithm`. Ersterer verwirft alle in Konflikt stehenden Regeln. Der zweite Algorithmus verwirft die Policys der in Konflikt stehenden Regeln. Damit verbunden werden alle Regeln der betroffenen Policys verworfen (vgl. [KS12]). Alle drei Algorithmen lösen immer den Konflikt, indem sie mindestens alle vom Konflikt betroffenen Regeln entfernen.

Um in Konflikt stehende Regeln erkennen zu können, muss zuerst definiert werden, wie der `Content` den Umfang der zu regulierenden Region beeinflusst. `WebPage` beschreibt genau die Adresse, die im `ContentSpecifier` hinterlegt wurde. Soll etwa `https://eff.org/wp` reguliert werden, ist diese Regel auch nur auf exakt diese URL anwendbar. `WebSite` hingegen beschreibt einen erweiterten Bereich, der in der aktuellen InFO-Version 0.1 nicht exakt definiert wird. Grundsätzlich gibt es drei Möglichkeiten, welchen Bereich dieselbe URL als `WebSite` beschreiben kann:

(1) `https://eff.org/*`

(2) `https://eff.org/wp/*`

(3) `https://*.eff.org/wp/*`

Die „`*`“ symbolisieren, dass der zu regulierende Adressbereich an dieser Stelle erweitert werden kann. So umfasst Schema (1) alle Dateien und Pfade unter der angegebenen Domäne unabhängig von möglichen Query-Strings oder URL-Fragmenten. Das erschwert eine konkrete Regulierung einzelner Unterpfade. Somit wird eine ungewollte *Überregulierung* wahrscheinlicher. Eine Überregulierung bezeichnet eine Regulierung, die eine umfangreichere Region betrifft, als sie eigentlich umfassen sollte. Dies kann technische oder auch menschliche Ursachen haben. Eine ähnliche Problematik existiert bei einer Implementierung nach Schema (3). Das erste „`*`“ -Symbol weist dabei auf hinzufügbare Labels hin. Es können deshalb

einfacher als bei der Implementierung nach Schema (2) Subdomänen irrtümlich überreguliert werden. Für Polsearchine sollen Regeln, die `WebSites` abbilden, deshalb nach Schema (2) interpretiert werden.

Ein Algorithmus zur Konfliktlösung arbeitet auf der Liste, die gleichpriorisierte Regeln als Mengen enthält. Jede Menge muss dazu einzeln auf Konflikte geprüft und diese gegebenenfalls behoben werden. Die Suche nach Konflikten soll über eine Methode `findeErstenKonflikt()` geschehen. Sie wird nachfolgend aufgeteilt in die Listings 4.1 bis 4.4 in Pseudocode dargestellt. Unterstrichene Worte kennzeichnen globale Konstanten. Nach jedem Listing befindet sich eine Erläuterung.

```
1 Menge findeErstenKonflikt(Menge regelnSelberPrioritaet, ↩
    Bool ausSelberPolicy):

2   Menge ergebnisMenge = erstelleLeereMenge()

3   for (i from 1 to regelnSelberPrioritaet.laenge):
4     Regel vergleichsRegel = regelnSelberPrioritaet.hole(i)
5     String vergleichsURL = vergleichsRegel.url

7     for (j from i+1 to regelnSelberPrioritaet.laenge):
8       Regel aktuelleRegel = regelnSelberPrioritaet.hole(j)
9       String aktuelleURL = aktuelleRegel.url
```

Listing 4.1: Dieses Listing zeigt den Grundaufbau der `findeErstenKonflikt`-Methode.

Listing 4.1 zeigt die Grundstruktur der Methode. Der Parameterwert `ausSelberPolicy` legt fest, ob lokale oder globale Konflikte ermittelt werden sollen (Zeile 1). Zeile 2 zeigt die Initialisierung der `ergebnisMenge`, welche am Ende der Methode alle gesuchten Regeln enthält. Die äußere Schleife in Zeile 3 iteriert über alle Regeln, wohingegen die innere Schleife (Zeile 7) nur über alle noch nicht bearbeiteten Regeln iteriert. Dadurch wird verhindert, dass Regeln mehrfach in `ergebnisMenge` eingefügt werden. In beiden Schleifen wird eine Regel und ihre URL extrahiert (Zeilen 4, 5, 8 und 9).

```
10      if (aktuelleRegel.typ == vergleichsRegel.typ):
11          continue # innere Schleife

12      if (ausSelberPolicy):
```

```
13            if (aktuelleRegel.policy != vergleichsRegel.policy):
14                continue # innere Schleife
15        Bool istKonfliktfall = False
```

Listing 4.2: Dieses Listing zeigt die Fälle, bei denen der Vergleich frühzeitig abgebrochen werden soll.

In Listing 4.2 wird der aktuelle Vergleich abgebrochen, falls die vergleichenden Regeln nicht in Konflikt zueinander stehen können (Zeilen 10 und 11). Der Grund dafür ist, dass beide Regeln den Informationsfluss entweder erlauben oder verbieten (Zeile 11). Der aktuelle Vergleich wird auch abgebrochen, wenn die beiden Regeln nicht derselben Policy entstammen, aber lokale Konflikte gesucht werden sollen (Zeilen 12 bis 14). Die boolesche Variable `istKonfliktfall` in Zeile 15 ist solange `False` bis ein Konflikt zwischen `aktuelleRegel` und `vergleichsRegel` festgestellt wurde.

```
16        if (vergleichsRegel.content == WebSite):
17            if (aktuelleURL.beginntMit(vergleichsURL)):
18                istKonfliktfall = True

19        if (not istKonfliktfall):
20            if (aktuelleRegel.content == WebSite):
21                if (vergleichsURL.beginntMit(aktuelleURL)):
22                    istKonfliktfall = True

23            else: # aktuelleRegel beschreibt keine WebSite
24                if (aktuelleURL == vergleichsURL):
25                    istKonfliktfall = True
```

Listing 4.3: In diesem Listing wird gegebenfalls ein Konflikt festgestellt.

In Listing 4.3 werden Konflikte erkannt und entsprechend boolesche Variablen gesetzt. Sollte `vergleichsRegel` `WebSite` als `Content` besitzen (Zeile 16), muss auf eine mögliche Regions-Überschneidung geprüft werden (Zeile 17). Dies wird durch den zuvor festgelegten Wirkungsbereich einer Regel mit dem `Content` `WebSite` bedingt. Dadurch ist es möglich, dass `aktuelleRegel` eine URL besitzt, die auf derselben Domäne auf eine präzisere URL als `vergleichsRegel` verweist. Die Prüfung erfolgt durch eine Methode `beginntMit()`. Sie soll `True` zurückgeben, wenn der ausführende String vollständig mit dem Parameter beginnt. Ebenso soll die Methode `True` zurückgeben, wenn die Strings gleich sind.

Die Zeilen 20 bis 22 beschreiben den umgekehrten Fall. Sie werden evaluiert, falls `vergleichsRegel` noch nicht in Konflikt zu `aktuelleRegel` steht (Zeile 19). Im letzten Fall (Zeilen 23 bis 25) wird darauf geprüft, ob die beiden Regeln dieselbe Region beschreiben, aber nicht als `Content WebSite` besitzen. An dieser Stelle ist der Vergleich der Regeln miteinander abgeschlossen. Es ist nun bekannt, ob die Regeln in Konflikt zueinander stehen. Weiter ist bekannt, ob die `vergleichs-Regel` von einer anderen Regel umschlossen wird oder ob sie die allgemeinere Region beschreibt.

```
26          if (istKonfliktfall):
27              ergebnisMenge.hinzufuegen(aktuelleRegel)
28          # Ende innere Schleife

29      if (ergebnisMenge.istNichtLeer()):
30          ergebnisMenge.hinzufuegen(vergleichsRegel)
31          break # äußere Schleife
31      # Ende äußere Schleife

32  return ergebnisMenge
```

Listing 4.4: In diesem Listing werden der `ergebnisMenge` bei vorhandenen Konflikten Regeln hinzugefügt.

In Listing 4.4 werden Regeln zur `ergebnisMenge` hinzugefügt, wenn sie in Konflikt zueinander stehen. Gab es eine Überschneidung von Adressräumen (Zeile 26), muss die `aktuelleRegel` hinzugefügt werden (Zeile 27). Ein Konflikt kann sich aus mehr als zwei Regeln zusammensetzen. Dementsprechend muss auch wenn `istKonfliktfall` `True` ist, die innere Schleife vollständig durchlaufen werden. Erst im Anschluss wird im Konfliktfall auch die `vergleichsRegel` hinzugefügt (Zeilen 29 und 30). So wird sichergestellt, dass sie nicht mehrfach zur `ergebnisMenge` hinzugefügt wird. Im Konfliktfall wird zusätzlich die äußere Schleife beendet (Zeile 31). Wurde kein Konflikt gefunden, wird die äußere Schleife fortgesetzt. In Zeile 32 wird abschließend die `ergebnisMenge` zurückgegeben. Wenn die Rückgabe keine leere Menge ist, wurden in Konflikt stehende Regeln gefunden. Auf die soeben untersuchte Ausgangsmenge müssen anschließend die entsprechenden Konfliktlösungsalgorithmen für die betroffenen Regeln angewandt werden. Dadurch werden Regeln entfernt und die so neu entstandenen Mengen müssen erneut mit `findeErstenKonflikt()` auf Konflikte geprüft

werden. Erst wenn diese Methode für keine der Mengen mehr Konflikte feststellen kann, ist die Konfliktlösung abgeschlossen.

4.2.5 Entfernen redundanter Regeln

Dieser Abschnitt beschreibt die Grundstruktur zweier Algorithmen zur Entfernung redundanter Regeln. Redundante Regeln sind Regeln, die niemals angewandt werden können, da eine höher- oder gleichpriorisierte Regel ihren Adressbereich umschließt. Der zuerst vorgestellte Algorithmus soll redundante Regeln gleicher Priorität entfernen. Dadurch ist es möglich, für jede Adresse eindeutig die anzuwendende Regel zu bestimmen. Ein zweiter Algorithmus soll anschließend die restlichen redundanten Regeln entfernen, da sie nie angewandt werden können.

Der erste Algorithmus soll alle Regeln selber Priorität entfernen, deren Region in der Region einer anderen Regel vollständig enthalten ist. Durch die vorherigen Konfliktlösungsalgorithmen sind Überschneidungen von Adressräumen nur noch bei gleichpriorisierten Regeln desselben Typs möglich. Ein Algorithmus soll ähnlich vorgehen, wie der in Abschnitt 4.2.4 vorgestellte. Dabei sollen stets zwei Regeln gleicher Priorität verglichen werden. Die konkretere Regel soll entfernt werden. Der Algorithmus soll für jede Menge von Regeln angewandt werden. Er gibt so jeweils eine minimale Menge zwingend benötigter Regeln derselben Priorität zurück. Damit noch weitere ungenutzte Regeln entfernt werden können, soll ein zusätzlicher Algorithmus Regeln höherer Priorität mit Regeln niedriger Priorität vergleichen. Die Grundstruktur dieses Algorithmusses soll, wie nachfolgend in Listing 4.5 mittels Pseudocode dargestellt, aussehen.

```
1  for (i from 1 to mengenListe.laenge):
2    Menge vergleichsRegelMenge = mengenListe.hole(i)
3    for (j from 1 to vergleichsRegelMenge.laenge):
4      Regel vergleichsRegel = vergleichsRegelMenge.hole(j)

5      for (k from i+1 to mengenListe.laenge)
6        Menge aktuelleRegelMenge = mengenListe.hole(k)
7        for (l from 1 to aktuelleRegelMenge.laenge):
8          Regel aktuelleRegel = aktuelleRegelMenge.hole(l)
```

Listing 4.5: Dieses Listing zeigt den Grundaufbau zur Prüfung auf redundante Regeln.

Die Variable `mengenListe` ist die Liste, die die Mengen von Regeln enthält. Mittels der ersten Schleife (Zeile 1) werden die höher priorisierten Mengen von Regeln extrahiert (Zeile 2). Die ihr untergeordnete Schleife (Zeile 3) bestimmt anschließend die `vergleichsRegel` (Zeile 4). Durch die dritte Schleife (Zeile 5) werden alle Mengen von Regeln extrahiert, die eine niedrigere Priorität als die `vergleichsRegel` haben (Zeile 6). Über diese niedriger priorisierten Mengen von Regeln iteriert die vierte Schleife (Zeile 7). Durch sie wird eine `aktuelleRegel` zum Vergleich bestimmt. Der Algorithmus soll anschließend alle `aktuelleRegeln` aus `mengenListe` entfernen, deren Regionen gänzlich von `vergleichsRegel` abgedeckt werden. Ebenso sollen `vergleichsRegeln` entfernt werden, wenn eine `aktuelleRegel` die allgemeinere Regel ist. Eine konkrete Vorgehensweise wird hier aufgrund der Ähnlichkeit zur detailliert beschriebenen `findeErsten-Konflikt`-Methode (siehe Abschnitt 4.2.4) nicht bereitgestellt. Die Rückgabe soll Mengen von Regeln enthalten, deren Regionen keine Regionen niedriger priorisierter Regeln vollständig abdecken, solange sie vom selben Typ sind. Dadurch kann eine Regel mit dem `Content WebPage` nur noch existieren, wenn sie für ihren Adressbereich die anzuwendende Regel ist. Zusätzlich ist für jeden anderen Adressbereich stets eindeutig die anzuwendende Regel mit dem `Content WebSite` feststellbar. Dazu muss aus allen anwendbaren Regeln die Regel mit der höchsten Priorität gewählt werden.

4.2.6 Formatskonvertierung und persistente Speicherung

Dieser Abschnitt beschreibt die Konvertierung der Informationen der verarbeiteten Policy-Dateien in Polsearchine-eigene Formate. Es handelt sich dabei um die in Anforderung A7.1 geforderten Entitys. In diesem Abschnitt wird ihr Aufbau vorgestellt und begründet. Anschließend wird knapp auf Vorkehrungen zur persistenten Speicherung eingegangen. Die Entitys sollen nur die Informationen der Policy-Dateien speichern, die Polsearchine zwingend zur korrekten Umsetzung der Regulierung benötigt (Anforderung A7.1). An dieser Stelle sind die Regeln konfliktfrei und eindeutig. Deshalb müssen nur die Informationen der Regeln, ihre Policys und die extrahierte Rückfallregel in Entity-Gegenstücke übertragen werden. Abbildung 4.2 und Abbildung 4.3 zeigen die benötigten Entitys, deren Modellierung nachfolgend beschrieben wird.

Meta Policys und ihre Algorithmen werden nicht weiter benötigt. Einzig die Rückfallregel muss im persistenten Speicher als `DefaultRuleEntity` gespeichert werden. Dabei muss lediglich vermerkt werden, ob die Rückfallregel Informationsflüsse pauschal erlaubt oder verbietet. Die in den Regeln enthaltenen Informationen werden alle zur Regulierung benötigt. Sie sollen um ein Prioritätsattribut ergänzt werden, das die Gruppierung der Listen widerspiegelt. Befindet sich eine Regel etwa in der höchstpriorisierten Liste, soll sie die Priorität 1 erhalten. Eine Regel der zweithöchstpriorisierten Liste soll dementsprechend die Priorität 2 erhalten und so weiter. Obwohl Anforderung A5.2 nur die Regulierung via URLs fordert, soll eine abstrakte Klasse `AbstractRuleEntity` für Regel-Entitys genutzt werden. Sie soll den durch Anforderung A6.1 geforderten Ersteller und das durch Anforderung A6.2 geforderte Thema der Regel enthalten. Zusätzlich muss `AbstractRuleEntity` ein Attribut `content` bereitstellen, um etwa den Umfang der Region einer `URLAllowingRule` modellieren zu können. Es genügt dabei die URI des Objekts zu speichern, das der `Content` spezifiziert. Dies ist entweder `WebSite` oder `WebPage`, wobei beide in InFO über eine URI identifiziert werden. Da SEFCO-Regeln ungeachtet der zu regulierenden Regionen nur in erlaubend oder verbietend unterschieden werden (siehe Abschnitt 2.2.3), soll zusätzlich ein boolesches Attribut `isInformationFlowAllowed` für das Regel-Entity genutzt werden. `AbstractRuleEntity` wird durch die Klasse `URLRuleEntity` spezialisiert. Diese konkrete Klasse soll zusätzlich das Attribut `regionURL` besitzen, um die URL-Region der Regeln abbilden zu können. Bei Policys müssen nur die

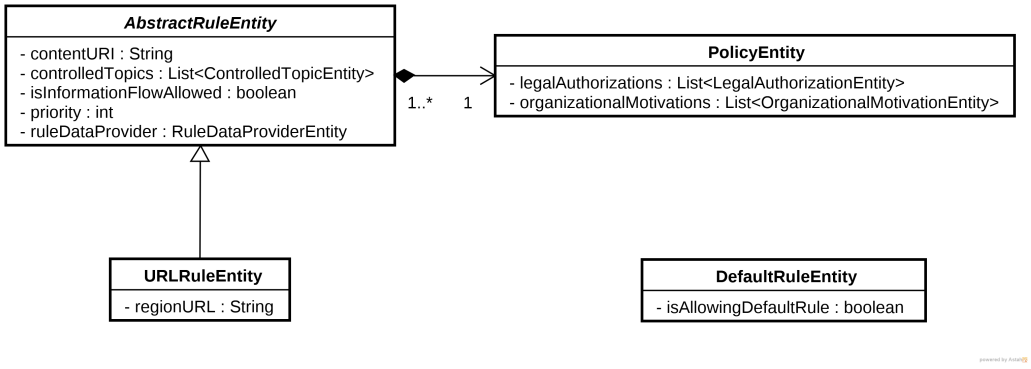

Abbildung 4.2: Diese Abbildung zeigt den Aufbau der Entitys, welche die Rückfallregel, eine Regel und deren Policy modellieren. Die als Attribute verwendeten Entitys sind in Abbildung 4.3 dargestellt.

durch Anforderung A6.3 bedingte rechtliche Grundlage und der organisatorische Beweggrund persistent gespeichert werden. Dabei besitzen diese zur Präsentation benötigten Sprachelemente in der aktuellen InFO-Version 0.1 keine weiteren Informationen als ihre URIs. Es ist denkbar, dass mittels InFO-kompatibler Ontologien beispielsweise `RuleDataProvider` um Adressinformationen des Betreibers ergänzt wird. Deshalb sollen die Sprachelemente, die auf die Hintergrundinformationen verweisen, als eigenständige Entitys modelliert werden. Dies erleichtert eine mögliche, spätere Erweiterung. Der Aufbau ist in Abbildung 4.3 dargestellt. Nach Anforderung A7.2 müssen die Entitys persistent gespeichert werden. Be-

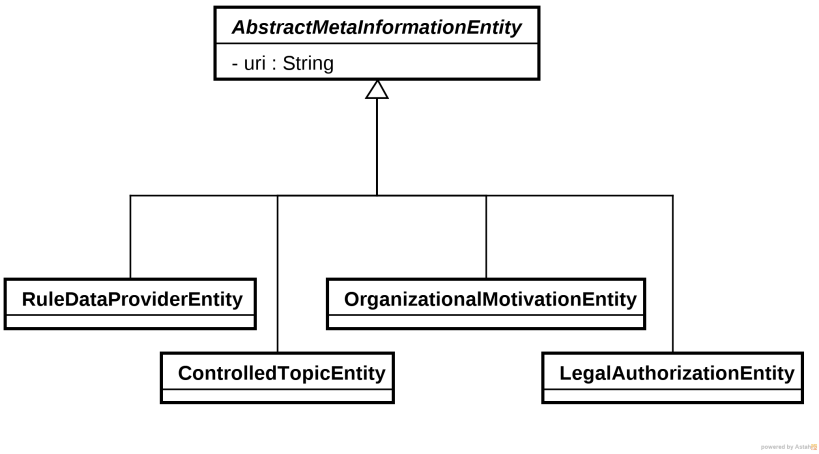

Abbildung 4.3: Diese Abbildung zeigt den Aufbau der durch Anforderung A6 geforderten InFO-Sprachelemente als Polsearchine-Entitys.

vor dies geschehen kann, müssen alle im persistenten Datenspeicher enthaltenen Entitys gelöscht werden. Auf diese Weise ist die aktuelle Policy-Verarbeitung unabhängig von vorangegangenen. Die Initialisierung von Polsearchine ist an dieser Stelle abgeschlossen. Der Server soll demzufolge anschließend auf Anfragen von Endbenutzer warten.

4.3 Suche

Nachfolgend wird beschrieben, wie eine Suchanfrage von Polsearchine bearbeitet werden soll. Aus Gründen der Komplexität wurde darauf verzichtet, eine parallele Nutzung mehrerer Backend-Suchmaschinen zu unterstützen. Anforderung A1.2

wird deshalb an dieser Stelle verletzt. Die nachfolgende Konzeption der Suche bezieht sich auf den Einsatz einer Backend-Suchmaschine (Anforderung A1.1), die austauschbar sein soll (Anforderung A1.3). Zuerst beschreibt Abschnitt 4.3.1, wie mit einer Nutzereingabe allgemein analysiert und vorverarbeitet werden soll. Der anschließende Abschnitt 4.3.2 behandelt die Beschaffung der Ergebnisse der Backend-Suchmaschine. Dort wird zudem beschrieben, wie ein Algorithmus die zu nutzende Backend-Suchmaschine bestimmen soll. Im darauffolgenden Abschnitt 4.3.3 wird der Ergebnisparser vorgestellt. Es wird beschrieben, wie die Rückgabe der Backend-Suchmaschine verarbeitet werden soll. Abschließend erläutert Abschnitt 4.3.4 die Präsentation der Ergebnisse an den Endbenutzer.

4.3.1 Eingabeverarbeitung

Dieser Abschnitt beschreibt den Ablauf der Eingabeverarbeitung. Die Verarbeitung beginnt nachdem ein Endbenutzer eine Suchanfrage gesendet hat. In diesem Abschnitt wird insbesondere auf die Erkennung von Dateitypen in der Suchanfrage eingegangen. Zu Beginn soll die IP-Adresse des Nutzers ausgewertet werden. Anhand dieser soll geprüft werden, ob er die Suchfunktion von Polsearchine benutzen darf. Dazu soll sie mit dem in Anforderung A2 eingeführten IP-Adressen-Muster abgeglichen werden. Beginnt die Adresse des Endbenutzers mit der des Musters oder ist sie identisch, soll die Eingabeverarbeitung und damit die Suche fortgesetzt werden. Anderenfalls soll dem Nutzer mitgeteilt werden, dass ihm aufgrund seiner IP-Adresse keine Suchergebnisse angezeigt werden können.

Nach Anforderung A1.8 soll eine Suche nach bekannten Dateitypen möglich sein. Wenn die Suchanfrage „`filetype:`“ enthält, soll von einer Suche nach Dateitypen ausgegangen werden. Wenn dies nicht der Fall ist, die Suchanfrage aber eine der Backend-Suchmaschine bekannte Dateiendung beinhaltet, soll dem Benutzer ein Hinweistext angezeigt werden. Dieser Text soll den Nutzer darüber informieren, dass Polsearchine eine spezielle Syntax anbietet, um seine Suche auf den enthaltenen Dateityp einzuschränken. Um bekannte Dateiendungen in Suchanfragen zu erkennen, muss jede in Polsearchine implementierte Backend-Suchmaschine dazu in einer Schnittstelle die Dateitypen angeben, die sie unterstützt. Die Rückgabe einer Implementierung der Schnittstelle soll auch leer sein dürfen, wenn eine solche Suche nicht unterstützt oder die Heuristik nicht genutzt

werden soll. Die Heuristik soll die Nutzereingabe auf unterstützte Dateitypen hin untersuchen, die als alleinstehendes Wort eingegeben wurden. Zwei Wörter sollen dabei anhand eines Leerzeichens zwischen ihnen erkannt werden. Als Beispiel soll geprüft werden, ob sich der Dateityp PDF in einer Nutzereingabe befindet. In dem Fall können beispielsweise die regulären Ausdrücke „`^PDF .*`“, „`.* PDF$`“ und „`.* PDF .*`“ genutzt werden. Ist einer von diesen erfolgreich anwendbar, so soll der Nutzer darauf hingewiesen werden. Dazu soll ihm mitgeteilt werden, dass er in seiner Suchanfrage auch „`filetype:PDF`“ verwenden kann, um nur nach PDF-Dateien zu suchen. Unabhängig davon, ob ein vermeintlich gesuchter Datentyp gefunden wurde, soll anschließend die Sucheingabe konform nach RFC 3986 [BLFM+05] kodiert werden. Dadurch kann sie später als Parameter in einer URL an eine Backend-Suchmaschine weitergeleitet werden. Da dieser Schritt für alle URL-basierten APIs benötigt wird, soll er bereits an dieser Stelle durchgeführt werden, statt erst im Interface Agent. Bei dem Interface Agent handelt es sich um ein Programm, das unter anderem Anpassungen an der Sucheingabe vornimmt, die von der jeweiligen Backend-Suchmaschine benötigt werden. (siehe Abschnitt 2.1.2).

4.3.2 Ergebnisbeschaffung

Dieser Abschnitt beschreibt die Beschaffung der Ergebnisse mittels der Backend-Suchmaschine. In der Implementierung wird nur eine Backend-Suchmaschine genutzt. Um Anforderung A1.3 zu erfüllen, die ein mögliches Austauschen von Backend-Suchmaschinen fordert, soll Polsearchine einen Algorithmus ähnlich des Dispatch Mechanisms aus Abschnitt 2.1.2 besitzen. Bei jeder Suchanfrage soll der Polsearchine-eigene Dispatch Mechanism die Eingabe des Endbenutzers an die zu nutzende Backend-Suchmaschine weiterleiten. Im Gegensatz zum im Abschnitt 2.1.2 beschriebenen Dispatch Mechanism soll diese Wahl unabhängig von der Nutzereingabe getroffen werden.

In Abschnitt 2.1.2 wurden Interface Agents vorgestellt. Diese kapseln die für die Backend-Suchmaschinen spezifische Logik von der Meta-Suchmaschine ab. Der Dispatch Mechanism wählt den zu nutzenden Interface Agent und leitet die Nutzereingabe an ihn weiter. Die Aufgabe des Interface Agents ist es, die Eingabe gegebenenfalls vorzuverarbeiten, die Suche mittels der Backend-Suchmaschine durchzuführen und anschließend die Ergebnisse zurückzugeben. Dadurch

erhalten alle Interface Agents dieselbe Eingabe und liefern die Backend-Suchmaschinenergebnisse in einem zueinander kompatiblen Format zurück. Dazu müssen alle Interface Agents dieselbe Schnittstelle implementieren. Die bei Polsearchine verwendeten Interface Agents sollen sich aus Gründen der Übersicht in zwei Klassen aufteilen. Die erste Klasse wird als *Retriever* bezeichnet und nutzt die zweite Klasse, den *Ergebnisparser*. Der Retriever soll die Anfrage an die Backend-Suchmaschine leiten. Seine Aufgabe ist es dabei, die Parametrisierung für die API-Anfrage durchzuführen. Zur Parametrisierung muss der Retriever etwa das „`filetype:`"-Präfix mittels der Heuristik aus Abschnitt 4.3.1 erkennen und verarbeiten. Es wird an dieser Stelle nicht genauer auf den Retriever eingegangen, da die durchgeführte Parametrisierung allein abhängig von der gewählten Backend-Suchmaschine ist. Eine konkrete Parametrisierung wird in Abschnitt 5.3.2 beschrieben. Der Retriever leitet die API-Rückgabe der Backend-Suchmaschine an den Ergebnisparser weiter. Dieser wird im nachfolgenden Abschnitt 4.3.3 behandelt.

4.3.3 Aufbau und Arbeitsweise des Ergebnisparsers

In diesem Abschnitt wird zuerst die Polsearchine-interne Modellierung von Ergebnissen vorgestellt. Es handelt sich dabei um Klassen und Schnittstellen, die von allen Interface Agents zur einheitlichen Rückgabe der Backend-Suchmaschinenergebnisse genutzt werden. Die Konvertierung der API-Rückgabe der Backend-Suchmaschinen in dieses einheitliche Format erfolgt durch den Ergebnisparser. Seine Arbeitsweise wird anschließend zur Beschreibung der Modellierung behandelt.

Die Modellierung der ungefilterten Ergebnisse stellt eine Abstraktion zwischen der Präsentation in Abschnitt 4.3.4 und der Interface Agents dar. Sie wird in Abbildung 4.4 zusammen mit der konkreten Modellierung der gefilterten Ergebnisse illustriert. Die für die Modellierung genutzten Klassen und Schnittstellen sollen nur die zur Präsentation benötigten Informationen beinhalten. Diese ergeben sich einerseits aus den in Anforderung A5 beschriebenen Hintergrundinformationen zur Darstellung gefilterter Ergebnisse. Andererseits handelt es sich um Informationen, die wahrscheinlich für eine API-Rückgabe einer Backend-Suchmaschine sind oder sonst anhand der API-Rückgabe bestimmt werden können. Ausgenommen der URL können Attribute auch leer sein. Die konkreten Implementierungen der Schnittstellen können diese auch erweitern, wie es in Abschnitt 5.4.1 beschrieben

ist. Dadurch ist es möglich, dem Nutzer zusätzliche, von der Backend-Suchmaschine bereitgestellte Informationen zu präsentieren.

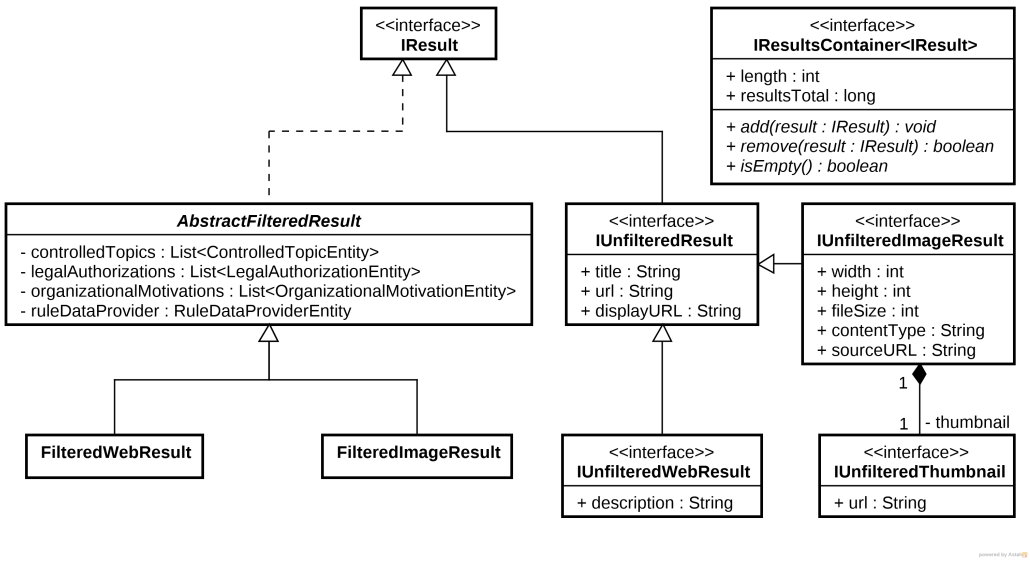

Abbildung 4.4: Diese Abbildung zeigt die Klassen und Schnittstellen, die zur abstrakten Repräsentation der Ergebnisse genutzt werden sollen.

Die allgemeinste Schnittstelle ist `IResult`. Sie stellt eine Obermenge für gefilterte und ungefilterte Ergebnisse dar. `IUnfilteredResult` ist die Schnittstelle für ungefilterte Suchmaschinenergebnisse. Als Attribute besitzt sie einen beschreibenden Titel (`title`), die URL des Ergebnisses (`url`) und die URL, die dargestellt wird (`displayURL`). Die `displayURL` soll den Grundaufbau der `url` erkennbar machen, ohne dabei eine Maximallänge zu überschreiten. Hierzu können URLs beispielsweise durch Auslassung einiger URL-Pfade optisch kürzer präsentiert werden. Es soll dem Ergebnisparser obliegen, wie diese gekürzte URL zur Darstellung erzeugt werden soll. Die in der Implementierung genutzte Suchmaschinen-API gibt bereits eine nutzbare, gekürzte URL zurück. Sie wurde übernommen, weshalb kein Algorithmus zur optischen Kürzung von URLs entwickelt wurde.

Webseitenergebnisse sollen durch `IUnfilteredWebResult` beschrieben werden. Diese Erweiterung von `IUnfilteredResult` soll sich einzig auf ein Attribut zur Speicherung von textuellen Beschreibungen des Ergebnisses beschränken (`description`). Suchmaschinen wie Microsoft Bing[4] geben hierfür Textzei-

[4] `http://www.bing.com/` letzter Zugriff 05.05.2013 15:45 Uhr

len aus der dazugehörigen Webseite zurück, die einige oder alle Suchbegriffe enthalten. Dadurch kann der Nutzer den Kontext seiner Suchbegriffe auf der Webseite besser einschätzen. `IUnfilteredImageResult` soll Bildergebnisse abbilden können. Zu den Attributen aus `IUnfilteredResult` kommen unter anderem die Bildbreite (`width`), Bildhöhe (`height`) und Dateigröße des Bildes (`fileSize`). Zusätzlich soll ein Attribut den Dateityp des Bildergebnisses beschreiben (`contentType`). Das vererbte `url`-Attribut soll auf die Bilddatei verweisen. `IUnfilteredImageResult` soll zusätzlich mittels des `sourceURL`-Attributs auf die Webseite verweisen in der das Bild eingebettet wurde. Jedes `IUnfilteredImageResult` soll eine Vorschau des modellierten Bildes durch eine Miniaturansicht (Englisch: „Thumbnail“) besitzen. Diese soll mit der Schnittstelle `IUnfilteredThumbnail` beschrieben werden. Durch die Nutzung eines separaten Bildes zur Miniaturansicht ist es möglich, die Vorschaubilder auf einer einheitlichen Größe zu präsentieren. Zusätzlich sinkt im Schnitt die auszuliefernde Datenmenge, da viele Bilder deutlich größer als ihre Miniaturansicht sind. Diese Miniaturansicht soll von einer URL referenziert werden (`url`).

Die Darstellung gefilterter Ergebnisse soll unabhängig von der genutzten Backend-Suchmaschine sein. Es wird auf die Nutzung von Schnittstellen für gefilterte Ergebnisse verzichtet, da die Backend-Suchmaschinen diese nicht erweitern sollen. Stattdessen soll die Abbildung der gefilterten Ergebnisse einmalig implementiert und von allen Interface Agents genutzt werden. Die abstrakte `AbstractFilteredResult`-Klasse modelliert gefilterte Ergebnisse. Sie wird durch die Klassen `FilteredWebResult` für gefilterte Webseitenergebnisse und `FilteredImageResult` für gefilterte Bildergebnisse konkretisiert. Die beiden Klassen enthalten keine zusätzlichen Attribute oder Methoden. Ihre Zweiteilung liegt in einer möglichen, späteren Erweiterung von Polsearchine begründet. Dabei könnten etwa gefilterte Bildergebnisse zusätzlich andere Arten von Hintergrundinformationen präsentieren als gefilterte Webseitenergebnisse. Die Attribute von `AbstractFilteredResult` modellieren die in Anforderung A6 geforderten Hintergrundinformationen.

Da der Interface Agent in den meisten Fällen nicht bloß ein einziges Ergebnis, sondern mehrere von der Backend-Suchmaschine erhält, wird ein *Behälter* (Englisch: „Container“) für `IResult`s eingeführt. Bei Behältern handelt es sich um Datentypen, die Elemente eines gemeinsamen Datentyps enthalten. Durch die Definition einer eigenen Schnittstelle können Informationen über die Ergebnisse

gespeichert werden. Die Schnittstelle für den Ergebnis-Behälter heißt `IResultsContainer`. Sie soll mit einem Attribut `length` die Anzahl der im Behälter enthaltenen Ergebnisse speichern. Diese Ergebnisse sind stets nur eine Teilmenge aller Ergebnisse, die auf die Nutzereingabe zurückgegeben werden können. Um die Anzahl aller auf die Nutzereingabe passenden Ergebnisse ebenfalls zu speichern, soll das Attribut `resultsTotal` verwendet werden. `IResultsContainer` soll Methoden zum Hinzufügen (`add()`) und Entfernen (`remove()`) von Ergebnissen bereitstellen. Nicht jede Suchmaschinen-API gibt eine leere Menge zurück, wenn keine weiteren Ergebnisse zur Suchanfrage existieren (siehe Abschnitt 5.4.1). Um trotzdem feststellen zu können, ob noch weitere Ergebnisse geladen werden können, ist eine `isEmpty()`-Methode in der Schnittstelle enthalten.

Der Ergebnisparser soll zuerst die URLs der Ergebnisse aus der Suchmaschinen-API-Rückgabe extrahieren. Für jedes Ergebnis muss jeweils die URL auf eine mögliche Regulierung geprüft werden. Durch die Entfernung ungenutzter Regeln in Abschnitt 4.2.5 sind dazu maximal zwei Abfragen auf den persistenten Speicher nötig. Zuerst soll der persistente Speicher nach einem `URLRuleEntity` durchsucht werden, dessen `Content WebPage` ist. Es muss zudem exakt dieselbe URL wie das zu prüfende Ergebnis besitzen. Wenn eine Regel im persistenten Speicher gefunden wurde, handelt es sich um die konkreteste Regel für dieses Ergebnis. Anderenfalls soll nach einer `URLRuleEntity` gesucht werden, die eine `WebSite` beschreibt. Die Suche soll dabei zusätzlich auf die Regeln eingeschränkt werden, deren URL den vollständigen Anfang der Ergebnis-URL bildet. Formuliert im zuvor genutzten Pseudocode (siehe Abschnitt 4.2.4) bedeutet dies, dass `(ergebnis.url).beginntMit(urlRuleEntity.url) == True` erfüllen muss. Sortiert nach absteigender Priorität der `URLRuleEntitys` befindet sich die anzuwendende Regel an erster Stelle der Suchrückgabe. Sollte auch diese Rückgabe leer sein, kann keine Regel auf das aktuelle Ergebnis angewandt werden. Dementsprechend muss aus der Rückfallregel bestimmt werden, ob Ergebnisse präsentiert werden dürfen oder nicht. Der Ergebnisparser soll die Rückfallregel einmalig vor Verarbeitung der Suchmaschinen-API-Rückgabe aus dem persistenten Speicher auslesen. Dadurch sollen die Zugriffe auf den persistenten Speicher reduziert werden. Für den Fall, dass das Ergebnis präsentiert werden darf, sollen die Informationen des Ergebnisses in eine Klasse extrahiert werden, die `IUnfilteredResult` implementiert. Anderenfalls sollen die Informationen aus der blockierenden Regel in eine Klasse extrahiert werden, die `Ab-`

`stractFilteredResult` umsetzt. In der aktuellen InFO-Version 0.1 enthält die Rückfallregel keine organisatorischen und rechtlichen Hintergrundinformationen. Sollte der Zugriff auf ein Ergebnis durch die Rückfallregel unterbunden werden, sollen die Attribute aus `AbstractFilteredResult` jeweils auf Polsearchine selbst verweisen. An dieser Stelle ist die Prüfung auf eine mögliche Regulierung abgeschlossen. Unabhängig von ihrem Ausgang befinden sich die zu präsentierenden Informationen des Ergebnisses oder der Regulierung nun innerhalb einer Klasse, die `IResult` implementiert. Diese Ergebnis-Klasse soll anschließend zu einer Implementierung des Ergebnis-Behälters `IResultsContainer` hinzugefügt werden. Auf diese Weise sollen auch die restlichen Ergebnisse der API-Rückgabe der Backend-Suchmaschine verarbeitet werden.

4.3.4 Präsentation der Ergebnisse

Aus Aufwandsgründen soll die gleichzeitige Suche mittels mehrerer Backend-Suchmaschinen und die damit verbundene Anforderung A1.2 vernachlässigt werden. Somit entfällt die Aufgabe des Display Mechanisms (siehe Abschnitt 2.1.2). Dieser würde die Ergebnisse mehrerer Backends-Suchmaschinen zusammenführen und vereinheitlichen. Statt eines Display Mechanisms sollen die Ergebnis-Behälter, die der Ergebnisparser erzeugt und befüllt, direkt für die Präsentation verarbeitet werden. Wenn keine Suchergebnisse zu dem Suchbegriff existieren, soll dem Nutzer dies mitgeteilt werden. Dem Nutzer soll auch mitgeteilt werden, falls ein Fehler die Ergebnisrückgabe verhindert. In diesem Abschnitt wird zuerst das Design von Webseitenergebnissen beschrieben. Anschließend folgt eine Beschreibung des Designs der Bildergebnisse. Abschließend wird begründet, wieso Polsearchine Ergebnisse nicht auf Seiten aufgeteilt, sondern stattdessen dynamisch nachgeladen werden sollen.

Abbildung 4.5 zeigt die konzeptionelle Präsentation der Webseitenergebnisse aufgeteilt in eine Karten-artige Darstellung. Die Ergebniskarten sollen in einer Liste untereinander angezeigt werden. Der Aufbau für Ergebnisse, die angezeigt werden dürfen, wird in der ersten Karte dargestellt. Sie sollen ihren Titel als eine zusammenfassende Überschrift nutzen. Zusätzlich soll ein Klick auf den Titel den Endbenutzer auf die URL des Ergebnisses weiterleiten. Unterhalb des Titels soll sich die anzuzeigende, möglicherweise gekürzte URL befinden. Dadurch soll der Nutzer nachvollziehen können, zu welchem Internetauftritt das Ergebnis gehört.

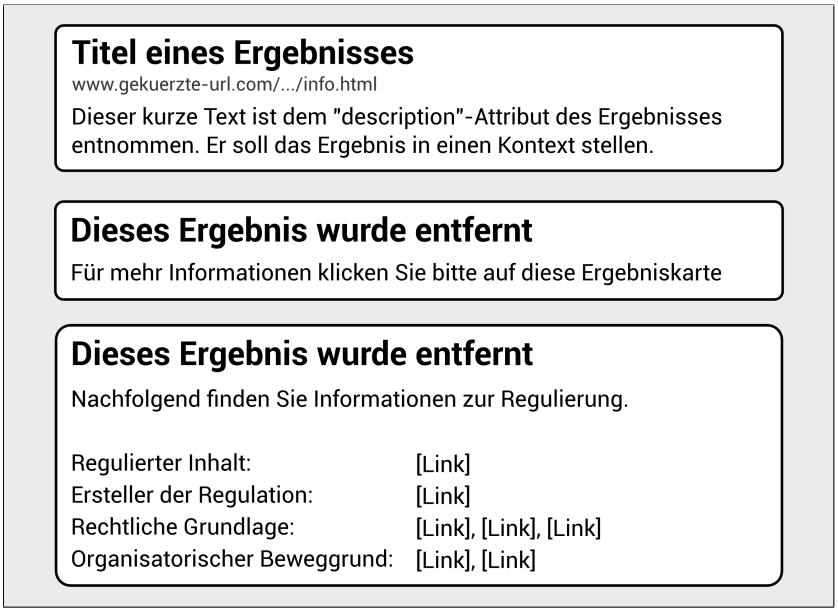

Abbildung 4.5: Diese Abbildung zeigt den konzeptionellen Aufbau zur Darstellung von Webseitenergebnissen. Es wird eine Karten-artige Darstellung für die Ergebnisse genutzt. Die erste Karte beinhaltet ein ungefiltertes Ergebnis. Das nachfolgende Ergebnis zeigt ein gefiltertes Ergebnis. Die dritte Karte stellt ein gefiltertes Ergebnis dar, nachdem darauf geklickt wurde.

Unter der anzuzeigenden URL soll die Beschreibung des Ergebnisses platziert werden. Die zweite Ergebniskarte zeigt ein gefiltertes Webseitenergebniss. Es besteht aus einem Text, der den Nutzer auf eine generelle Regulierung hinweist. Bei einem Klick auf diese Karte soll sie zu der dritten Karte erweitert werden. Dem Endbenutzer werden in der aufgeklappten Karte die konkreten Hintergrundinformationen zur Regulierung gemäß Anforderung A6 präsentiert. Dies ist sinnig, da der Umfang der Regulierungsinformationen in der InFO-Modellierung nach oben hin unbegrenzt ist (vergleiche Abbildung 2.3 und Abbildung 2.4). Würden die Hintergrundinformationen stattdessen direkt angezeigt, könnte das Anwendererlebnis (Englisch: „User experience“) durch zu viele, unmittelbar sichtbare Informationen negativ beeinträchtigt werden. Stattdessen soll dem Nutzer ein einheitliches Design präsentiert werden. In diesem sollen alle Webseitenergebnisse unabhängig von ihrer Regulierung eine Maximalhöhe nicht überschreiten. Dadurch wird auch die Menge an unmittelbar sichtbaren Informationen eingeschränkt.

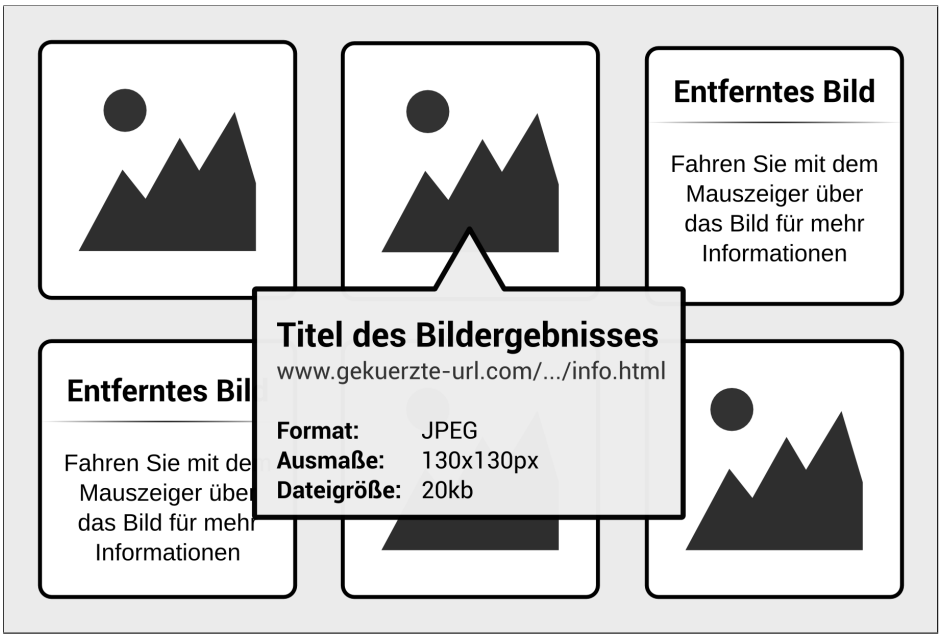

Abbildung 4.6: Diese Abbildung zeigt die konzeptionelle Darstellung von Bildergebnissen. Es wird eine eine Karten-artige Darstellung für die Ergebnisse genutzt. Damit die Informationsbox zum zweiten Bild von links gezeigt wird, muss der Mauszeiger über dem Bild platziert werden.

Abbildung 4.6 zeigt die konzeptionelle Darstellung von Bildergebnissen. Sie sind von links nach rechts und von oben nach unten angeordnet. Dabei werden dem Nutzer die Miniaturansichten der ungefilterten Bilder angezeigt. Diese sollen eine maximale Breite und eine maximale Höhe besitzen, um die Bilder einheitlich präsentieren zu können. Bei einem Klick auf die Miniaturansicht soll der Nutzer direkt auf die URL der Bilddatei weitergeleitet werden. Fährt er mit dem Mauszeiger über eine Miniaturansicht, sollen ihm zusätzliche Informationen zu dem Bild angezeigt werden. Dies ist der Fall beim zweiten Bildergebnis der Abbildung 4.6. Der dort gezeigte Titel soll die URL verlinken, die auf die Seite verweist, in der das Bild eingebettet wurde. Unter dem Titel befindet sich erneut die anzuzeigende, möglicherweise gekürzte URL. Darunter sollen sich die übrigen Attribute der `IUnfilteredImageResult`-Schnittstelle befinden. Gefilterte Bildergebnisse werden in Abbildung 4.6 ebenfalls nur durch eine Miniaturansicht dargestellt. Dadurch soll die einheitliche Präsentation der anderen Ergebnisse unbeeinträchtigt bleiben. Es wird eine Grafik als Miniaturansicht für alle gefilterten Ergebnisse genutzt. Diese Grafik weist auf eine generelle Regulierung hin. Die zusätzlichen Informa-

tionen soll der Nutzer einsehen können, wenn er mit dem Mauszeiger über eine entsprechende Miniaturansicht fährt.

Aufgrund der hohen Menge an Suchergebnissen, die eine Suchanfrage erzeugen kann, können nicht alle Ergebnisse zur gleichen Zeit geladen werden. Dies ist aber auch nicht nötig, da Benutzer von Suchmaschinen im Schnitt pro Suchanfrage nur etwa acht Ergebnisse betrachten [CG07, GC07, LHB+08]. Zudem ergab eine Eye-Tracking-Studie von Granka et al. [GJG04] unter anderem, dass die meiste Aufmerksamkeit eines Nutzers auf Ergebnisse fällt, die direkt sichtbar sind. Nach dieser Studie sinkt die Aufmerksamkeit, wenn ein Nutzer ein Ergebnis erst durch Scrollen sehen kann. Sie sinkt allerdings noch stärker, wenn der Nutzer dafür eine weitere Seite aufrufen muss. Deshalb sollen Ergebnisse in Polsearchine nicht auf Seiten aufgeteilt werden. Stattdessen sollen sie dynamisch nachgeladen werden, sobald der Nutzer zum Ende der Ergebnisse scrollt. Jedes Nachladen soll dabei durch eine neue Suchanfrage realisiert werden. Da die Nutzereingabe weiterhin dieselbe ist, muss die Heuristik der Eingabeverarbeitung nicht erneut angewandt werden.

Kapitel 5

Implementierung

Dieses Kapitel beschreibt die Implementierung der Meta-Suchmaschine Polsearchine. Zuerst wird die Wahl der Programmiersprache und der Software-Plattform in Abschnitt 5.1 behandelt. Anschließend wird in Abschnitt 5.2 die Wahl einer Backend-Suchmaschine beschrieben. Dazu werden Anforderungen an eine Backend-Suchmaschine erarbeitet. Anhand dieser werden fünf potentielle Backend-Suchmaschinen untersucht und die Bing Search API ausgewählt. Der darauffolgende Abschnitt 5.3 behandelt die Konfiguration und Verwendung der Bing Search API für Polsearchine. Abschließend wird in Abschnitt 5.4 auf Umsetzungsdetails eingegangen.

5.1 Programmiersprache und Software-Plattform

Andere InFO-Entwicklungen wie die Nameserver-Erweiterung [Mos12] nutzen Java in ihrer Implementierung. Auch der in dieser Bachelorarbeit verwendete InFO-Parser [BGSS13] ist in Java geschrieben worden. Deshalb ist Polsearchine zum Großteil in Java geschrieben. Dies führt zu einheitlicheren InFO-Entwicklungen. Zusätzlich ermöglicht es, entwickelte Technologien wie den Parser leicht wiederverwenden zu können. Die Meta-Suchmaschine Polsearchine nutzt Java Platform, Enterprise Edition[1] 6 [CS+09] (kurz: *Java EE*) als Software-Plattform. Java EE ist auf die Entwicklung von Webanwendungen ausgelegt [Ora12]. Es wurde Version 6 gewählt, da sie zu Beginn der Implementierung die aktuellste stabile Version

[1] `http://www.oracle.com/technetwork/java/javaee/index.html` letzter Zugriff 16.04.2013, 09:45 Uhr

war. In der Industrie stellen Java-EE-Server einige der häufigst genutzten Anwendungsserver dar [DDL06]. Deshalb und aufgrund persönlicher Erfahrung fiel die Wahl auf Java EE. Die Erfahrung erlaubte einzuschätzen, dass eine Meta-Suchmaschine wie Polsearchine mit Java EE realisierbar ist. In den nachfolgenden drei Abschnitten wird der Aufbau einer Java-EE-Webanwendung beschrieben. Abschnitt 5.1.1 behandelt eine Schichten-basierte Betrachtung. Anschließend beschreibt Abschnitt 5.1.2 die entscheidenden Komponenten für Polsearchine, deren Aufgaben und ihr Zusammenspiel. Abschnitt 5.1.3 beschreibt knapp den Aufbau des gewählten Java-EE-Servers. Sofern nicht konkret auf Polsearchine bezogen, entstammen die Informationen dieser drei Abschnitte Oracle Corporation [Ora13].

5.1.1 Schichten-basierter Aufbau von Java-EE-Webanwendungen

Dieser Abschnitt beschreibt den Aufbau von Java-EE-Webanwendungen. Dazu existieren zwei unterschiedliche Sichtweisen, die abhängig vom Einsatz der Webanwendung sind. Zuerst wird ein dreischichtiger Aufbau vorgestellt. Dieser wird anschließend durch einen vierschichtigen Aufbau konkretisiert, welcher auch bei Polsearchine angewandt wird.

In einer ersten Sichtweise werden Java-EE-Webanwendungen üblicherweise als dreigeteilt beschrieben. Hierbei wird zwischen den ausführenden Systemen unterschieden. Diese sind Client-Rechner, Java-EE-Server und Datenbank-Server. Der Client-Rechner ist das System des Endbenutzers. Der Endbenutzer ruft die Webanwendung vom Java-EE-Server ab und interagiert mit ihr. Dies geschieht über seinen Webbrowser oder einer eigens dafür entwickelten, lokalen Anwendung. Der Java-EE-Server beinhaltet die *Geschäftslogik* der Webanwendung. Die Geschäftslogik steuert das Verhalten und beinhaltet die Funktionen der Webanwendung. Sie besitzt zudem die Möglichkeit mit dem Datenbank-Server zu kommunizieren. Dadurch kann der Java-EE-Server Daten dauerhaft speichern, sowie diese verändern oder löschen. Zusätzlich stellt der Java-EE-Server die *Präsentationslogik* bereit, wenn der Endbenutzer auf die Webanwendung mit einem Webbrowser zugreift. Im Java-EE-Kontext werden die Komponenten, die unmittelbar zur Erstellung und Interaktion mit der graphischen Benutzeroberfläche benötigt werden, als Präsentationslogik bezeichnet. Bei einer eigenständigen, lo-

kalen Anwendung befindet sich die Präsentationslogik auf dem Client-Rechner.

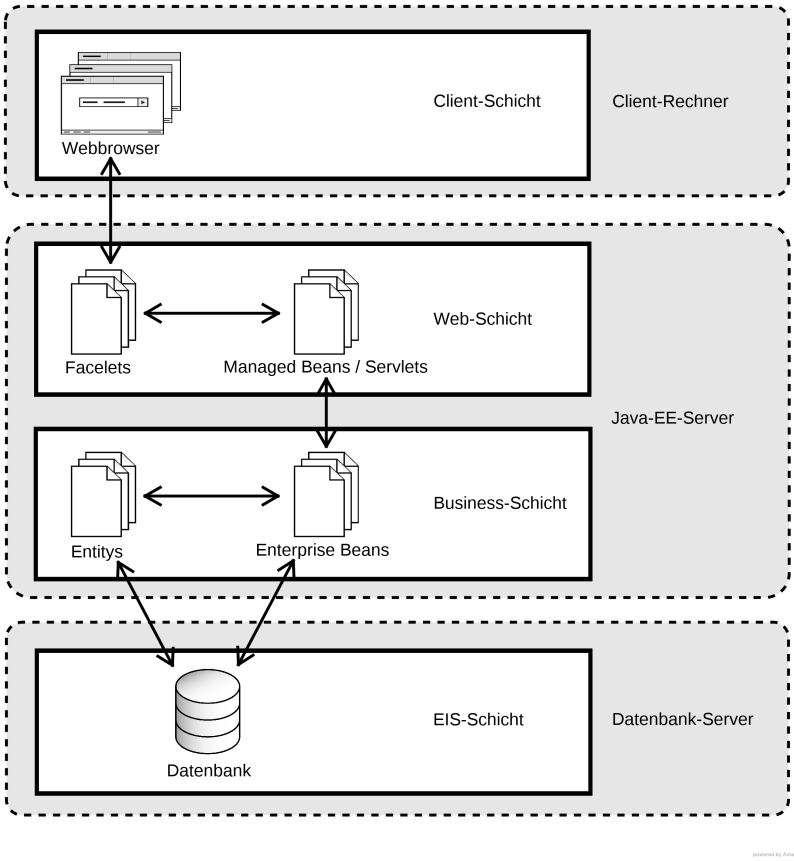

Abbildung 5.1: Diese Abbildung zeigt den Aufbau einer Java-EE-Webanwendung, die mithilfe eines Webbrowsers benutzbar ist. Die einzelnen Schichten einer Webanwendung werden durch die schwarzen Rechtecke dargestellt. In ihnen befinden sich die Polsearchine-Kernkomponenten. Pfeile visualisieren mögliche Interaktionen zwischen Komponenten. Die grauen, abgerundeten Rechtecke zeigen die Systeme, auf denen die Schichten liegen. (Abbildung angelehnt an [Ora13] Abbildungen 1-1, 1-4 und 1-5)

Es handelt sich bei Polsearchine um eine Meta-Suchmaschine, die in einem Webbrowser aufrufbar sein soll. Es bietet sich deshalb an, die Aufteilung einer Java-EE-Webanwendung auf diesen Fall zu konkretisieren. So ergibt sich eine zweite Sichtweise auf Java-EE-Webanwendungen, die eine präzisere Aufteilung anhand der Aufgabenbereiche statt der Systeme möglich. Die Aufteilung ist in 5.1 abgebildet und erfolgt in vier Schichten. Diese Schichten befinden sich innerhalb der Systeme. Der Client-Rechner beinhaltet die *Client-Schicht*. Die Präsentations-

logik befindet sich innerhalb der *Web-Schicht*. In der *Business-Schicht* liegt die Geschäftslogik. Sowohl die Web- als auch die Business-Schicht liegen innerhalb des Java-EE-Servers. Die letzte Schicht liegt im Datenbank-Server und beinhaltet die Daten. Sie wird *EIS-Schicht* genannt. *EIS* steht für betriebliches Informationssystem (Englisch: „Enterprise Information System). Die Nutzung eines EIS erlaubt es, dass Daten gemeinsam von mehreren unterschiedlichen Anwendungen genutzt werden können. Das betriebliche Informationssystem ist dabei unabhängig von der Java-EE-Webanwendung. Wenn die Webanwendung die einzige Anwendung mit Zugriff auf die Daten ist, kann auch eine einzelne Datenbank statt eines EIS genutzt werden.

5.1.2 Funktionen und Interaktionen von Java-EE-Komponenten

In diesem Abschnitt wird auf einige Komponenten eingegangen, die in einer Java-EE-Webanwendung genutzt werden. Die Auswahl wurde dabei auf die in Polsearchine verwendeten Komponenten beschränkt. Deshalb wird in diesem Abschnitt von einer Webanwendung ausgegangen, die vom Endbenutzer in einem Webbrowser genutzt wird. Abbildung 5.1 zeigt die anschließend behandelten Komponenten, ihr Zusammenspiel und ihre Zuordnung zu Schichten und Systemen. Die unterste Schicht ist die EIS-Schicht. Sie beinhaltet als wichtigste Komponente Datenspeicher. In dieser Implementierung wird eine Datenbank als persistenter Datenspeicher genutzt. In ihr werden von Polsearchine zur Erfüllung von Anforderung A8.6 die Benutzerkonten gespeichert (siehe Abschnitt 5.4.2). Zusätzlich werden in der Datenbank Informationen der Policy-Dateien gespeichert (siehe Abschnitt 5.4.2). Zur Erfüllung von Anforderung A7.2 werden sie als Entitys gespeichert.

Die Entitys werden als Java-Objekte auf der Business-Schicht erzeugt und verarbeitet. Ihre Methoden beschränken sich auf Zugriffsmethoden und ähnlich triviale Funktionen. Zugriffsmethoden sind Methoden, welche ein Attribut eines Objekts zurückgeben oder ändern. Die Speicherung in und das Lesen aus der Datenbank erfolgt über die von Java EE bereitgestellte Java Persistence API[2] (kurz: *JPA*). JPA ermöglicht ein objektrelationales Abbilden der Entitys. Das bedeutet,

[2] `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html` letzter Zugriff 27.07.2013, 15:00 Uhr

dass Java-Objekte verlustfrei in der Datenbank gespeichert und aus ihr gelesen werden können. In der Business-Schicht befinden sich auch *Enterprise Beans*. Sie beinhalten die Geschäftslogik und somit komplexere Methoden. Enterprise Beans besitzen keine Daten, sondern arbeiten auf den Entitys. So werden die Daten von den eigentlichen Funktionen der Geschäftslogik getrennt.

In der Web-Schicht befinden sich *Facelets*. Sie stellen die Oberfläche bereit, die später an den Client ausgeliefert wird. Es handelt sich dabei um `XHTML`-Dateien mit erweiterter Syntax. Diese spezielle Syntax erlaubt eine dynamische Erstellung der später an den Client übertragenen, reinen `XHTML`-Dateien. Zum Beispiel können so Templates oder Kontrollstrukturen wie Schleifen zur Ausgabe genutzt werden. Bei Templates handelt es sich um Facelets, die als Vorlage für andere genutzt werden können. So ist es möglich, mehrfach genutzte Teile wie beispielsweise den Kopf einer Webseite wiederzuverwenden. Facelets kommunizieren nicht direkt mit der Business-Schicht. Stattdessen interagieren sie mit *Servlets* und *Managed Beans*. Diese sind ebenfalls Komponenten der Web-Schicht. Sie interagieren mit den Enterprise Beans. Es handelt sich bei Servlets und Managed Beans um Java-Objekte. Durch sie erhält die Präsentationslogik Zugriff auf die Methoden und über diese auch Zugriff auf die Daten der Geschäftslogik. Servlets und Managed Beans können zwar auf gleiche Weise mit der Business-Schicht interagieren, sie werden aber für unterschiedliche Zwecke verwendet. Servlets wurden für zustandslose, asynchrone Aufrufe innerhalb der Facelets entwickelt. Durch Servlets kann in den dargestellten Webseiten Inhalt dynamisch nachgeladen werden. Managed Beans hingegen wurden zur Datenmanipulation im Hintergrund entworfen. Im Gegensatz zu Servlets werden ihre Rückgabewerte meist nicht an die Oberfläche weitergegeben. Managed Beans können zudem einen Zustand besitzen. So können sie beispielsweise genutzt werden, um Eingabedaten über mehrere Formulare hinweg vorzuhalten, ohne diese Daten persistent speichern zu müssen. Die Interaktion von Facelets mit Managed Beans geschieht über die erweiterte Syntax der Facelets. Eine Kommunikation der Facelets mit Servlets findet hingegen mittels HTTP-Anfragen statt.

Als letzte wichtige Komponente kann der Webbrowser des Endbenutzers gesehen werden. Der Webbrowser befindet sich auf dem Client-Rechner und somit in der Client-Schicht. Über den Webbrowser stellt der Endbenutzer Anfragen an den Java-EE-Server. Hinter jeder Anfrage steht der Aufruf eines Facelets. Dieses liefert anschließend die graphische Benutzeroberfläche aus.

5.1.3 Aufbau des Java-EE-Servers

Dieser Abschnitt stellt den verwendeten Java-EE-Server und seinen grundlegenden Aufbau knapp vor. Dabei wird auch darauf eingegangen, welche Komponenten die Webanwendung für einen Datenbankzugriff benötigt. Zudem beschreibt dieser Abschnitt den grundlegenden Aufbau einer Java-EE-Webanwendung als Datei zur Installation durch den Java-EE-Server.

Als Java-EE-Server wurde GlassFish[3] Server Open Source Edition 3.1.2 gewählt. Dieser gilt als die Referenzimplementation für Java EE 6. Zusätzlich wird er standardmäßig mit Entwicklungsumgebungen wie NetBeans und Eclipse ausgeliefert [Ora10]. Das ermöglicht eine Fehlersuche und -beseitigung (Englisch: „Debugging“) während die Anwendung auf dem Server ausgeführt wird. Die Entwicklung wird dadurch erleichtert. GlassFish kann mehrere *Domänen* (Englisch: „Domains“) besitzen. Bei jeder Domäne handelt es sich um eine Server-Instanz, auf der eine oder mehrere Webanwendungen ausgeführt werden. Somit kann GlassFish mehrere Java-EE-Server betreiben. Bei Polsearchine handelt es sich nach Anforderung A1.7 um genau eine Webanwendung. Sie wird deshalb innerhalb einer eigenen Domäne ausgeführt.

Damit eine Webanwendung Datenbankzugriff erhält, bedarf es eines *Connection Pool*s. In diesem werden Informationen wie Datenbankname und Adresse gespeichert werden. Mittels des Connection Pools baut GlassFish eine durchgehende Verbindung zur Datenbank auf. Für eine Datenbankabfrage muss dadurch keine neue Verbindung aufgebaut werden. Zusätzlich können Connection Pools von mehreren Webanwendungen auf demselben GlassFish-Server genutzt werden. Damit dies möglich ist, besitzen Webanwendungen keinen direkten Zugriff auf den Connection Pool. Stattdessen greifen sie auf ein *JDBC-Resource*-Objekt als Datenquelle zu. Dieses wird bei Polsearchine von JPA genutzt (siehe Abschnitt 5.1.2). JPA ist eine API, die von Polsearchine zum Speichern und Laden von Entitys genutzt wird.

Das Installieren einer Java-EE-Webanwendung auf einem Java-EE-Server wird als *Deploying* bezeichnet. Dazu wird die Webanwendung als Datei vom Server eingelesen und verarbeitet. Eine Java-EE-Webanwendung wird als Datei mit der Endung `EAR` vertrieben. Es handelt sich dabei um eine `JAR`-Datei. Diese ist ein Archiv, das neben der Webanwendung und Abhängigkeiten auch Meta-Informationen ent-

[3] `https://glassfish.java.net/` letzter Zugriff 26.04.2013, 20:30 Uhr

hält. Die Meta-Informationen sind beispielsweise globale Anwendungsvariablen (siehe Abschnitt 5.4.4) oder der Pfad, unter dem die Anwendung eingebunden werden soll. Über letzteren kann etwa spezifiziert werden, dass eine Anwendung über die URL des Servers beispielsweise mit dem Pfad `/meineAnwendung` erreichbar sein soll. Bei den Abhängigkeiten handelt es sich um Java-Bibliotheken, die sich in einem `lib`-Ordner des `EAR`-Archivs befinden. Die Webanwendung selbst teilt sich innerhalb des `EAR`-Archivs in zwei weitere `JAR`-Dateien auf. Dabei besitzt das Archiv, welches die Web-Schicht beinhaltet, die Endung `WAR`. Dort befinden sich unter anderem die Facelets, Servlets und Managed Beans. Das Archiv, das die Business-Schicht enthält, besitzt eine `JAR`-Endung. Sie beinhaltet unter anderem die Entitys und Enterprise Beans. Die Installation von Polsearchine wird in Anhang A und das Deploying konkret in Abschnitt A.7.3 beschrieben.

5.2 Wahl der Backend-Suchmaschine

Polsearchine ist durch die Konzeption als Meta-Suchmaschine auf die Suchergebnisse einer anderen Suchmaschine angewiesen. Somit muss mindestens eine geeignete Backend-Suchmaschine als Grundlage für Polsearchine gefunden werden. Da der gleichzeitige Einsatz mehrerer Backend-Suchmaschinen die Komplexität deutlich steigern würde, soll bei Polsearchine als Machbarkeitsnachweis darauf verzichtet werden. Die dazugehörige Kann-Anforderung A1.2 wird somit nicht erfüllt. Stattdessen wird nachfolgend nur die Wahl einer einzigen Backend-Suchmaschine beschrieben. Zuerst werden die Anforderungen für diese Wahl in Abschnitt 5.2.1 vorgestellt und erläutert. Im Anschluss werden die fünf wichtigsten Suchmaschinen in Abschnitt 5.2.2 anhand der Anforderungen evaluiert. Die Wichtigkeit einer Suchmaschine wird mittels ihres weltweiten Marktanteils bestimmt. Der Abschnitt endet mit der begründeten Entscheidung für die Bing Search API als Backend-Suchmaschine.

5.2.1 Anforderungen zur Wahl der Backend-Suchmaschine

Dieser Abschnitt stellt die Anforderungen an die Wahl der Backend-Suchmaschine vor. Jede Anforderung besitzt eine Nummerierung. Die Nummerierung wird im darauffolgenden Abschnitt 5.2.2 genutzt. Nach der Nummerierung folgt ein

Schlagwort, welches das Thema der Anforderung umschreibt. Der erste Satz ist jeweils die eigentliche Anforderung. Anschließend folgt für jede Anforderung eine Begründung für ihre Auswahl. Sofern benötigt wird dieser Teil mit Erklärungen zum Ausmaß der Anforderung ergänzt.

R1 Indexgröße Der Suchmaschinenindex soll möglichst umfangreich sein. Ein Suchmaschinenindex ist ein Datenspeicher, der gesammelte Informationen über beispielsweise Webseiten beinhaltet (siehe Abschnitt 2.1.1). Durch einen großen Index soll verhindert werden, dass der Nutzer den Eindruck gewinnt, es würden Ergebnisse zurückgehalten werden. Stattdessen wären die einzigen ausbleibenden Ergebnisse die von Polsearchine gefilterten.

R2 Vorfilterung Die Suchmaschine soll Ergebnisse nicht selbst filtern. Manche Filterungen, wie die von Google SafeSearch vorgenommenen, können deaktiviert werden [SWP04]. Andere liegen allerdings außerhalb der Kontrollmöglichkeiten. Deshalb sind sie problematisch für eine regulierende Meta-Suchmaschine wie Polsearchine. Beispiele für eine solche Filterung wurden in dem einleitenden Kapitel 1 aufgezeigt.

R3 Berechtigung Die Benutzung der Suchmaschine in Polsearchine muss vom Anbieter erlaubt worden sein. Das Parsen von Ergebnisseiten, wird von Suchmaschinenanbietern nicht explizit gestattet. Somit handelt es sich um eine rechtliche Grauzone. Deshalb ist es sinnig, sich direkt auf bereitgestellte APIs zu fokussieren. Damit eine API als Basis für Polsearchine genutzt werden kann, muss eine derartige Nutzung rechtlich konform zu den Nutzungsbedingungen sein. Diese Nutzungsbedingungen können Anforderungen an die Anwendung stellen. Sie können auch die Benutzung nur unter Einschränkungen erlauben. Mehrfach wird beispielsweise eine Veränderung der Ergebnismenge verboten, sofern sie nicht aufgrund nationalen Rechts erforderlich ist (vgl. [Mic12b, Yan12d]). Dies schließt unter anderem eine Umsortierung mit ein. Weiter schränken manche API-Nutzungsbedingungen zum Beispiel teilweise die Anzeige von Werbung auf der Seite der Ergebnisse ein (vgl. [Mic12b]). Viele solcher Einschränkungen sind allerdings irrelevant für Polsearchine. Dies liegt daran, dass Polsearchine kein eigenes Rankingmodul besitzt und nur eine Backend-Suchmaschine nutzen soll. Die Ergebnismenge wird deshalb nicht umsortiert. Ebenso ist eine Platzierung von Werbung für Polsearchine irrelevant, da es sich nur um einen Mach-

barkeitsnachweis handelt. Bezüglich der Nutzungsbedingungen bleibt der Fokus vor allem darauf, dass die Benutzung der API in einer Meta-Suchmaschine erlaubt sein muss. Zusätzlich muss es erlaubt sein, Ergebnisse legitimiert zurückzuhalten. Nur so können Ergebnisse von Polsearchine gefiltert werden.

R4 Kosten Die Verwendung der API soll möglichst geringe Kosten erzeugen. Da es sich bei Polsearchine nur um einen Machbarkeitsnachweis handelt, ist nicht mit sehr vielen Suchanfragen zu rechnen. Aus dem Grund sollen etwaige Kosten für 5000 Suchanfragen pro Monat als Vergleichswert berechnet werden. Die Anforderung ist somit optimal erfüllt, wenn eine API kostenlos mindestens diese Anzahl an Suchanfragen erlaubt.

R5 Funktionsumfang Die API soll sich nicht auf die Suche nach Webseiten beschränken. Es sollten somit nicht ausschließlich Webseiten als Ergebnisse zurückgeben werden können. Das ist wichtig zur potentiellen Erfüllung von zwei an die Entwicklung von Polsearchine gestellten Kann-Anforderungen. Diese fordern eine Suche nach Videos (Anforderung A1.5) und Bildern (Anforderung A1.6).

R6 Übertragbarkeit Die API muss auf mehreren Rechnern gleichzeitig nutzbar sein. So kann die Entwicklung unabhängig von dem Server geschehen, welcher später Polsearchine bereitstellen soll. Zusätzlich erhöht dies die Portabilität der Anwendung. Dadurch kann Polsearchine problemlos auf einem anderen Server eingesetzt werden.

5.2.2 Suchmaschinen-APIs in Hinblick auf die Anforderungen und Wahl einer API

Meta-Suchmaschinen wie Dogpile[4], Metacrawler[5] oder DuckDuckGo[6] bieten nach ausgiebiger Recherche keine APIs zur Suche nach Webseiten, Bildern oder Videos an. Da diese Meta-Suchmaschinen selbst auf andere Suchmaschinen-APIs zurückgreifen, sind sie an deren rechtliche Bedingungen gebunden. Die Nutzungsbedingung der verwendeten Suchmaschinen-APIs verbieten dabei häufig die

[4] `http://www.dogpile.com/` letzter Zugriff 10.04.2013, 18:00 Uhr
[5] `http://www.metacrawler.com/` letzter Zugriff 10.04.2013, 18:00 Uhr
[6] `https://duckduckgo.com/` letzter Zugriff 10.04.2013, 18:00 Uhr

Weitergabe der Ergebnisse (vgl. [Mic12b, Yah12a, Yan12d]). Deshalb werden nachfolgend nur APIs reiner Suchmaschinen betrachtet. Soweit möglich werden sie anhand der Anforderungen R1 bis R6 auf eine mögliche Nutzung für Polsearchine geprüft. In Hinblick auf Anforderung R1 ist es allerdings schwierig, vergleichbare Indexgrößen einzelner Anbieter zu ermitteln. Stattdessen werden die weltweit fünf meistgenutzten Suchmaschinenanbieter betrachtet. Diese sind in absteigender Reihenfolge Google Inc., Baidu, Inc., Yahoo! Inc., Yandex und Microsoft Corporation [Sul13]. Auch in Hinblick auf Anforderung R2 ist es schwierig, konkrete Angaben zu vorgenommener Filterung zu erfahren. Deshalb muss dieser Punkt anschließend vernachlässigt werden.

Die bekannteste Suchmaschine ist Google Search (vgl. [BB05]). Für sie existieren zwei offizielle APIs [Goo12a, Goo12b]. Eine der beiden wird nicht mehr weiterentwickelt und kann jederzeit gänzlich entfernt werden [Goo12a]. Viele API-Nutzungsbedingungen behalten sich dieses Recht vor (vgl. bspw. [Yah12a]). Das Eintreten eines solchen Szenarios ist aber bei Benutzung dieser ersten API nach den Nutzungsbedingungen [Goo10] sehr wahrscheinlich. Somit ist aufgrund dieser Ungewissheit die Nutzung der beschriebenen API keine Option. Die andere API operiert nur auf Google-Custom-Search-Ergebnissen [Goo12b]. Google Custom Search beschränkt sich auf eine voreingestellte Menge an Webseiten [Goo12b]. Dadurch ist es nur möglich, auf einer kleinen Teilmenge der von Google Search indizierten Webseiten zu suchen. Dies widerspricht der Anforderung R1. Somit stellt Google Inc. keine API bereit, die für Polsearchine nutzbar wäre. Dadurch sind die Anforderungen R3, R4, R5 und R6 nicht bewertbar. Bezüglich möglicher Regulierungen ist bekannt, dass in manchen Ländern wie Frankreich oder Deutschland eine nicht deaktivierbare Vorfilterung stattfindet (siehe Kapitel 1). Da Google Inc. keine für Polsearchine taugliche Suchmaschinen-API anbietet, wurde nicht weiter geprüft, ob sich die Vorfilterung auch auf die APIs auswirkt. Anforderung R2 kann somit ebenso wenig bewertet werden.

Die Suchmaschine der Baidu, Inc. hält den größten Suchmaschinen-Marktanteil in China [Ye07]. Es ist bekannt, dass Baidu, Inc. Ergebnisse im Sinne der chinesischen Regierung filtert [EHLM09]. Eine manuell durchgeführte Suche[7] nach der offiziellen Homepage der Organisation Human Rights Watch „`hrw.org`“ bestätigte dies. Es fehlten konkrete Ergebnisse zur Organisation und ein Text wies darauf hin, dass die URL „`hrw.org`“ nicht gefunden werden konnte. Bei ei-

[7]Stand des Tests 18.09.2013, 18:00 Uhr

ner anschließenden Suche nach der vollständigen URL „http://www.hrw.org“ wurde keine Webseite ausgeliefert. Somit wird Anforderung R2 nicht erfüllt. Die Recherche nach einer API wurde durch sprachliche Barrieren erschwert. Sowohl die Suchoberfläche[8] als auch der offizielle Internetauftritt bezüglich APIs[9] wurden auf Chinesisch verfasst. Webseiten der Baidu, Inc., die sich auf die API beziehen, bieten Übersetzungen auf Englisch an. Aus diesen geht hervor, dass Baidu, Inc. die APIs Baidu OMS API[10], Baidu SMS API[11] und Baidu NMS API[12] bereitstellt [Bai11, Bai10]. Die Baidu OMS API beinhaltet die anderen beiden und bietet ausschließlich unterschiedliche Marketingdienste an [Bai13]. Sie stellt keine Methoden bereit, die eine Suche nach Webseitenergebnissen oder anderen ermöglicht. Aus diesem Grund sind die unmittelbar daran geknüpften Anforderungen R3 bis R6 nicht bewertbar.

Yahoo! Search ist die weltweit dritt-häufigst genutzte Suchmaschine [Sul13]. Zusätzlich besaß sie im Februar 2013 den dritt-größten Suchmaschinen-Marktanteil in den Vereinigten Staaten [com13]. Yahoo! Boss ist die API für Yahoo! Search [Yah13]. Anforderung R2 kann nicht bewertet werden, da keine Details zu einer möglichen Filterung bekannt sind. Eine manuell durchgeführte Suche[13] nach „http://stormfront.org“ mit der deutschen[14], französischen[15] und britischen[16] Variante von Yahoo! Search zeigte allerdings gefilterte Ergebnisseiten für die deutsche und französische Lokalisierung. Der Endbenutzer wird nicht über eine Regulierung informiert. Es ist somit zumindest denkbar, dass die API eine ähnliche Form der Regulierung erfährt. Die Nutzungsbedingungen [Yah12a] erfüllen Anforderung R3. Jede Suchanfrage ist mit Kosten verbunden. So kosten 5000 Suchanfragen mit bis zu 50 Ergebnissen bis zu $4 [Yah12b]. Damit ist Anforderung R4 zwar nicht verletzt, aber auch nicht optimal erfüllt. Mit der API kann neben Webseiten auch nach Bildern und Videos gesucht werden [Yah13].

[8] http://www.baidu.com/ letzter Zugriff 22.07.2013, 21:30 Uhr
[9] http://apihome.baidu.com/ letzter Zugriff 22.07.2013, 21:30 Uhr
[10] http://yingxiao.baidu.com/support/apien/node_4727.html letzter Zugriff 22.07.2013, 22:00 Uhr
[11] http://yingxiao.baidu.com/support/apien/node_4759.html letzter Zugriff 22.07.2013, 22:00 Uhr
[12] http://yingxiao.baidu.com/support/apien/node_5530.html letzter Zugriff 22.07.2013, 22:00 Uhr
[13] Stand: 09.09.2013, 20:30 Uhr
[14] http://de.search.yahoo.com/ letzter Zugriff 09.09.2013, 20:30 Uhr
[15] http://fr.search.yahoo.com/ letzter Zugriff 09.09.2013, 20:30 Uhr
[16] http://uk.search.yahoo.com/ letzter Zugriff 09.09.2013, 20:30 Uhr

Dadurch ist Anforderung R5 erfüllt. Eine Einschränkung der Nutzung der API auf bestimmte Rechner existiert nicht. Somit wird Anforderung R6 erfüllt.

Die Suchmaschine von Yandex war 2011 die in Russland am häufigsten genutzte [Cri11]. Sofern nicht anders angemerkt, entstammen die nachfolgenden Informationen der Yandex.XML Anleitung für Entwickler [Yan13]. Im Jahr 2011 weitete Yandex sein Angebot auf den türkischen Raum aus [Yan12a]. Aus diesem Grund ermöglicht die Yandex Suchmaschinen-API Yandex.XML lokalisierte russische, türkische und weltweite Ergebnisse. Die Lokalisierung drückt sich durch unterschiedliche Sortierung der Ergebnisse und einer unterschiedlichen Indexmenge aus. Eine Lokalisierung wird während der Registrierung zur Nutzung der API gewählt. Sie ist anschließend unveränderlich. Ob dies zu Filterungen führt oder ob die API allgemein filtert, ist nicht bekannt. Eine manuelle Suche mit Yandex[17] nach „`http://stormfront.org`“ zeigte keine Filterung. Da Yandex allerdings keine französische oder deutsche Lokalisierung anbietet, konnte dieser Test nicht für Länder wiederholt werden, in denen eine Filterung wahrscheinlich wäre. Somit kann Anforderung R2 nicht bewertet werden Die Nutzungsbedingungen sind ausreichend für eine Entwicklung mit Polsearchine (vgl. [Yan12c, Yan12b, Yan12d, Yan12e]). Damit ist Anforderung R3 erfüllt. Bei der Registrierung zur Nutzung der API kann eine Telefonnummer angegeben werden. Wird diese von Yandex bestätigt, können täglich bis zu 1000 Suchanfragen kostenfrei durchgeführt werden. Anforderung R4 ist damit optimal erfüllt. Da mit Yandex.XML allerdings nur nach Webseiten gesucht werden kann, ist Anforderung R5 verletzt. Zusätzlich wird die Nutzung der API bei der Registrierung an eine IP-Adresse gebunden. Dadurch wird Anforderung R6 nicht erfüllt.

Die Suchmaschine der Microsoft Corporation Bing ist die weltweit fünfthäufigst genutzte Suchmaschine [Sul13]. In den Vereinigten Staaten hält sie allerdings den zweitgrößten Anteil am Suchmaschinen-Markt [com13]. Die Bing Search API beinhaltet eine Vorfilterung ähnlich Google SafeSearch. Sie kann deaktiviert werden. Eine manuell durchgeführte Suche[18] nach „`http://stormfront.org`“ mit der deutschen[19], französischen[20] und britischen[21] Lokalisierung von Bing zeigte eine Filterung für die deutsche und französische Variante. Statt konkreter

[17] `http://www.yandex.ru/` letzter Zugriff 18.09.2013, 19:00 Uhr
[18] Stand 10.09.2013, 01:30 Uhr
[19] `http://www.bing.com/?cc=de` letzter Zugriff 10.09.2013, 01:30 Uhr
[20] `http://www.bing.com/?cc=fr` letzter Zugriff 10.09.2013, 01:30 Uhr
[21] `http://www.bing.com/?cc=gb` letzter Zugriff 10.09.2013, 01:30 Uhr

Informationen zur Filterung verweist Bing bei einer durchgeführten Filterung nur auf allgemeine Bedingungen zur Filterung[22]. Ein genaues Ausmaß der Vorfilterung und die damit verbundene Einschränkung der Anforderung R2, ist nicht bekannt. In Abschnitt 5.3 finden sich zu diesem Punkt Ergebnisse oberflächlicher Tests mit der API. Die Nutzungsbedinungen der Bing Search API [Mic12b] erfüllen Anforderung R3. Da Kosten erst ab über 5000 Suchanfragen pro Monat mit jeweils bis zu 50 Ergebnissen anfallen [Mic12a], wird Anforderung R4 als optimal erfüllt gewertet. Weiter ist die API unter anderem auch für eine Suche nach Bildern oder Videos einsetzbar [Mic12a]. Damit wird Anforderung R5 erfüllt. Zudem ist der Einsatz auf mehreren Rechnern möglich, wodurch Anforderung R6 erfüllt ist.

Aus dieser Untersuchung geht hervor, dass für die Entwicklung von Polsearchine die APIs von Yahoo! Inc. und Microsoft Corporation infrage kommen. Da die Bing Search API ein kostenfreies Kontingent an Suchanfragen bereitstellt, wurde sie der Yahoo! Boss API vorgezogen. Zudem wurde bei Bing im Gegensatz zu Yahoo! Search auf eine durchgeführte Filterung hingewiesen.

5.3 Verwendung der Bing Search API

Die anschließenden Informationen zur Parametrisierung entstammen der Schema-Dokumentation der Bing Search API [Mic13]. Eine Suchanfrage wird über eine URL durchgeführt. Tabelle 5.1 zeigt beispielhaft eine solche URL. In dieser Tabelle werden alle anschließend vorgestellten Parameter genutzt und deren Wirkung zusammengefasst. Damit es sich um eine gültige URL handelt, müssen die Parameterwerte gegebenenfalls kodiert werden. Dadurch wird in Tabelle 5.1 beispielsweise anstatt eines Apostrophs „`%27`“ genutzt. Die Bing Search API orientiert sich bei dieser Kodierung an RFC 3986 [BLFM+05]. Für die Nutzung mit Polsearchine lässt sich die URL in zwei Teile aufteilen. Der erste ist statisch und der zweite dynamisch. Bei jeder Suchanfrage kann sich der dynamische Teil verändern. Er ist abhängig von der Nutzereingabe und -interaktion. Der statische Teil wird hingegen durch Polsearchine festgelegt und bleibt konstant bei jeder Suchanfrage. Die Werte für den statischen Teil müssen somit im vornherein ausgewählt werden. Ohne diese Festsetzung der Werte nutzt die Bing Search API eigene Rückfallwerte.

[22] `https://onlinehelp.microsoft.com/de-DE/bing/ff808447.aspx` und `https://onlinehelp.microsoft.com/fr-FR/bing/ff808447.aspx` letzter Zugriff 18.09.2013, 18:30 Uhr

Nachfolgend werden die genutzten Parameter in Abschnitt 5.3.1 erklärt. Anschließend wird die Wahl ihrer Werte, soweit nötig, in Abschnitt 5.3.2 begründet.

Beispielfragment	**Erläuterung**	
`https://api.datamarket. ↩ azure.com/Bing/Search/ ↩ Composite?`	Stamm-URL der Bing Search API	**Statisch**
`$format=JSON`	Rückgabe als JSON	
`&Adult=%27Off%27`	Deaktivieren des Adult-Filters	
`&WebOptions=%27DisableQuery ↩ Alterations%2BDisableHost ↩ Collapsing%27`	Tippfehler nicht korrigieren und mögliche Duplikate anzeigen	
`&$top=50`	Rückgabe von 50 Ergebnissen	
`&Market=%27en-US%27`	US-Lokalisierung	
`&Sources=%27Web%27`	Suche nach Webseiten	**Dynamisch**
`&Query=%27Llamas%27`	Suche nach dem Wort „`Llamas`“	
`&$skip=23`	Überspringen der ersten 23 Ergebnisse	
`&WebFileType=%27PDF%27`	Einschränkung auf PDF-Dokumente	

Tabelle 5.1: Die Tabelle zeigt den Beispielaufbau einer Bing-Search-API-URL. „Statisch“ bedeutet, dass diese Werte bei Polsearchine über alle Suchanfragen hinweg gleich bleiben. Dementsprechend können sich die Fragmente, die als „Dynamisch“ gekennzeichnet sind, bei jeder Anfrage verändern.

5.3.1 Beschreibung der verwendeten Parameter

Grundlegend für die Suche ist die Stamm-URL der Bing Search API. Diese wird durch URL-Query-Parameter ergänzt. Die Reihenfolge der nachfolgenden Erläuterungen richtet sich dabei an Tabelle 5.1. Es ist allerdings auch möglich, die Parameter in einer anderen Reihenfolge zu nutzen. Dies hat keine Auswirkung auf die Rückgabe. Der `$format`-Parameter bestimmt das Format der Rückgabe. Die Ergebnisse der Anfrage können im XML- oder JSON-Format zurückgegeben werden. Der `Adult`-Parameter bietet eine Regulierung für sexuell anstößige Inhalte an. Mit ihm kann die Striktheit der Regulierung angepasst oder deaktiviert werden. Der `WebOptions`-Parameter ermöglicht zwei Anpassungen. Es kann verhindert werden, dass die Bing Search API den Suchbegriff verändert. Ist diese Option

nicht gesetzt, kann beispielsweise eine Suche nach „Shcule“ als eine Suche nach dem Begriff „Schule“ interpretiert werden. Zusätzlich können mit dem Web-Options-Parameter Ergebnisse ignoriert werden, die mehrfach auf demselben Internetauftritt auftauchen. Diese Option soll Duplikate in den Ergebnissen verhindern [Riv10, Mic10]. Der $top-Parameter gibt an, wie viele Ergebnisse maximal zurückgegeben werden sollen. Die Bing Search API gibt standardmäßig maximal 50 zurück. Der Market-Parameter spezifiziert die Lokalisierung. Diese betrifft jede Suchanfrage. Wird kein Wert mitgegeben, ermittelt die API automatisiert eine Lokalisierung. Dies geschieht unter anderem anhand der IP-Adresse des Suchanfragesenders. Für die API ist der Sender der Server, auf dem Polsearchine läuft.

Der Sources-Parameter bestimmt die Art der Ergebnistypen. Durch die Auswahl einer oder mehrerer Arten mittels entsprechenden Werts kann nach Webseiten, Bildern, Videos oder Nachrichten gesucht werden. Als weitere Arten definiert die Bing Search API die Möglichkeit der Suche nach ähnlichen Begriffen oder nach Vorschlägen für mögliche Tippfehler. All diese sechs Arten können auch kombiniert verwendet werden. Der Query-Parameter beinhaltet den Suchbegriff und ist somit essentiell. Die Nutzereingabe entspricht dem Suchbegriff. Sie muss deshalb unter Umständen vorverarbeitet werden, damit es sich unabhängig von der Eingabe um eine gültige URL handelt. Der $skip-Parameter dient dem Überspringen von Ergebnissen. Um n Ergebnisse zu überspringen, wird „$skip $= n$“ verwendet. In dem Fall ist das erste zurückgegebene Ergebnis das $n + 1$-te aller Ergebnisse. Dieser Parameter kann beispielsweise genutzt werden, um Suchergebnisse auf mehrere Ergebnisseiten zu verteilen. Polsearchine nutzt ihn, um dynamisch Ergebnisse nachzuladen (siehe Abschnitt 4.3.4). Der WebFileType-Parameter ermöglicht die Suche nach Dateien anhand ihres Dateityps. Die möglichen Dateitypen sind dabei API-seitig beschränkt und beinhalten unter anderem DOC, PDF, HTML und TXT.

5.3.2 Begründung genutzter Werte zur Parametrisierung

Dieser Abschnitt behandelt, welche statischen Werte für Polsearchine gewählt wurden. Der $format-Parameter wurde auf „JSON“ gesetzt, weil die JSON-Rückgabe wesentlich kompakter ist. Sie besitzt weiter keine Nachteile gegenüber XML. Der Adult-Parameter kontrolliert das Ausmaß der Vorfilterung bezüglich sexueller

Inhalte. Diese muss gänzlich deaktiviert werden, indem „Off“ als Wert gesetzt wird. So wird eine Vorfilterung unterbunden und damit Anforderung R2 nicht verletzt. Der WebOptions-Parameter wird mit dem Wert „DisableQueryAlterations+DisableHostCollapsing“ genutzt. Das Pluszeichen muss in der URL durch ein „%2B“ ersetzt werden, damit sie gültig ist. Durch „DisableQueryAlterations“ wird sichergestellt, dass die Bing Search API die Nutzereingabe nicht verfälscht. Zusätzlich wird durch „DisableHostCollapsing“ an dieser Stelle eine Vorfilterung im Sinne von Anforderung R2 unterbunden. Ohne diese Option würden vermeintliche Duplikate auf demselben Internetauftritt aus der Ergebnismenge entfernt werden. Der $top-Parameter wird mit dem Wert „50“ genutzt, so dass maximal 50 Ergebnisse zurückgegeben werden. Dieser Wert wurde gewählt, um möglichst viele Ergebnisse bei einer kostenlosen Nutzung der Bing Search API anzeigen zu können (siehe Abschnitt 5.2.2). Den Wert für den Market-Parameter zu wählen, ist nicht trivial. Die Schema-Dokumentation [Mic13] gibt keine Auskunft darüber, ob der Market-Parameter nur die Ergebnissortierung beeinflusst oder auch zu Vorfilterungen führt. Um einen Markt wählen zu können, wurden einige Ausmaße der Regulierung durch die API grob untersucht. Diese Untersuchung erhebt keinen Anspruch auf Vollständigkeit oder absolute Korrektheit. Das Ziel ist schlichtweg die Bestimmung eines brauchbaren Market-Parameters.

Für den nachfolgenden Versuch wurden die Ergebnisse der Bing Search API für sieben Lokalisierungen bei drei Suchbegriffen betrachtet. Die gewählten Market-Parameterwerte sind „ar-XA“ (Saudi Arabien), „de-DE“ (Deutschland), „de-AT“ (Österreich), „en-GB“ (Vereinigtes Königreich), „en-US“ (Vereinigte Staaten), „fr-FR“ (Frankreich) und „zh-CN“ (China). Dabei fiel die Wahl auf Länder, bei denen eine Regulierung der Inhalte bekannt ist oder die für diese Bachelorarbeit von Bedeutung sind. So wurde in der Einleitung Regulierungen bei der französisch- und deutschlokaliserten Version von Google Search beschrieben. Saudi Arabien und China filtern nach Faris [FV08] stark. Polsearchine besitzt eine englische graphische Benutzeroberfläche. Deshalb ist eine etwaige Filterung bei der Lokalisierung für die Vereinigten Staaten oder das Vereinigte Königreich als Hauptvertreter der englischen Sprache relevant. Österreich befindet sich unter den getesteten Lokalisierungen, um eine mögliche Regulierung mit der deutschen Lokalisierung zu vergleichen. Somit kann festgestellt werden, ob es sich um eine Filterung bei deutscher Sprache oder deutscher Region handelt.

Für die Tests wurden Suchbegriffe für Webseiten gewählt, die repräsentativ in die Kategorien Pornographie, Rassismus und Menschenrechte fallen. Die Bing-Inhaltsfilterung wurde mittels des `Adult`-Parameters deaktiviert. Es wurde dabei nur nach Webseiten und nicht beispielsweise Bildern oder Videos gesucht. Zusätzlich wurde die Option „`DisableQueryAlterations`“ aktiviert. Die Tests

		Bing-Search-API-Lokalisierungsparameter						
		ar-XA	de-AT	de-DE	en-GB*	en-US	fr-FR	zh-CN
Begriff	youporn	-	U	-	U	U	U	F
	stormfront.org	U	U	F	U	U	F	-
	human rights watch	U	U	U	U	U	U	U

Tabelle 5.2: Suchergebnisse bei unterschiedlichen Suchbegriffen mit verschiedenen Lokalisierungsparametern. Alle Testergebnisse, außer gekennzeichnete, sind vom 29.11.2012 und 12.04.2013. Mit „*“ gekennzeichnete Ergebnisse sind nur vom 12.04.2013. „U“ beschreibt ungefilterte und „F“ gefilterte Ergebnisse. Auf eine fehlende Ergebnisrückgabe weist „-“ hin. Zur besseren Übersicht sind die Zellen der teilweise oder gänzlich gefilterten Ergebnisse grau hinterlegt.

wurden am 29.11.2012 und erneut am 12.04.2013 mit zusätzlicher Untersuchung der Lokalisierung für das Vereinigte Königreich durchgeführt. Eine Veränderung des Filterverhaltens war über diesen Zeitraum nicht feststellbar. Die Ergebnisse sind in Tabelle 5.2 zusammengefasst. Aus Gründen des Umfangs wurden jeweils nur die ersten 15 Ergebnisse betrachtet. Von diesen wurde anschließend auf die gesamte Ergebnismenge geschlossen. Dabei wurde die Ergebnismenge in drei Kategorien unterteilt. Wenn augenscheinlich keine Filterung feststellbar war, so wurden die Ergebnisse als ungefiltert („U“) festgehalten. Fehlten die offensichtlich gesuchten Webseiten und wurden auch keine thematisch ähnlichen Seiten zurückgegeben, wurde die Rückgabe als gefiltert („F“) bewertet. Gab es keine Ergebnisse, wurde dies dementsprechend als keine Ergebnisrückgabe („-“) notiert.

Saudi Arabien, Deutschland und China geben jeweils einmal für einen bestimmten Suchbegriff keine Ergebnisse zurück. Frankreich und Deutschland filtern die Suche nach „`stormfront.org`“ ähnlich wie bei Google Search. Interessant für die spätere Wahl des `Market`-Parameters sind die Lokalisierungen, bei denen keine Filterung auftrat. Im Test waren dies Österreich, Vereinigtes Königreich und

Vereinigte Staaten. Damit bleibt es nach diesen Tests möglich, eine deutschsprachige Lokalisierung zu verwenden. Da Polsearchine ausschließlich eine englische Benutzeroberfläche besitzt, wurde die österreichische Lokalisierung nicht übernommen. Stattdessen wird als Ergebnis dieser Tests die Lokalisierung für die Vereinigten Staaten verwendet. Der Vorzug gegenüber des Vereinigten Königreichs liegt einzig in der größeren Einwohnerzahl der Vereinigten Staaten begründet.

5.4 Umsetzungsdetails

Die Implementierung erfolgte analog zur Konzeption, welche in Kapitel 4 vorgestellt wurde. Dieser Abschnitt beinhaltet Entwicklungs- und Bing-spezifische Details. In Abschnitt 5.4.1 wird zuerst die Implementierung der Ergebnisschnittstellen vorgestellt. Abschnitt 5.4.2 beschreibt die umgesetzte, persistente Speicherung von Benutzerkonten und Entitys. Der darauffolgende Abschnitt 5.4.3 behandelt Details zur Umsetzung der Anmeldung mit Benutzerkonten. Abschließend wird in Abschnitt 5.4.4 die implementierte Konfigurationsmöglichkeit vorgestellt. Die Webanwendung und ihre Webseiten sind gänzlich in Englisch gehalten. Bildschirmfotos der implementierten Webseiten befinden sich in Anhang B.

5.4.1 Implementierung der Ergebnisschnittstellen

Dieser Abschnitt beschreibt die Implementierung der Ergebnisschnittstellen für die Nutzung mit der Bing Search API. Dabei wird auf Besonderheiten bei der Bing-Search-API-Rückgabe eingegangen und die Umsetzung der `isEmpty()`-Methode hergeleitet.

Abbildung 5.2 zeigt die Realisierung der in Abschnitt 4.3.3 entworfenen Schnittstellen. Die Attribute des `BingThumbnail` sowie die Bing-interne ID der Ergebnisse `bingID` finden keine Verwendung in der Verarbeitung und Präsentation der Ergebnisse. Sie befinden sich aus Gründen der Vollständigkeit in diesem Klassendiagramm, da sie von der Bing Search API zurückgegeben werden. `AbstractBingResult` implementiert die Zugriffsmethoden für die Attribute aus `IUnfilteredResult`. Das `offset`-Attribut im `BingResultsContainer` wird von der Bing Search API zurückgegeben und soll den Startwert der Ergebnisrückgabe modellieren. Wenn das `offset`-Attribut beispielsweise den Wert „`5`“ annimmt, soll das erste im `BingResultsContainer` enthaltene Ergebnis das

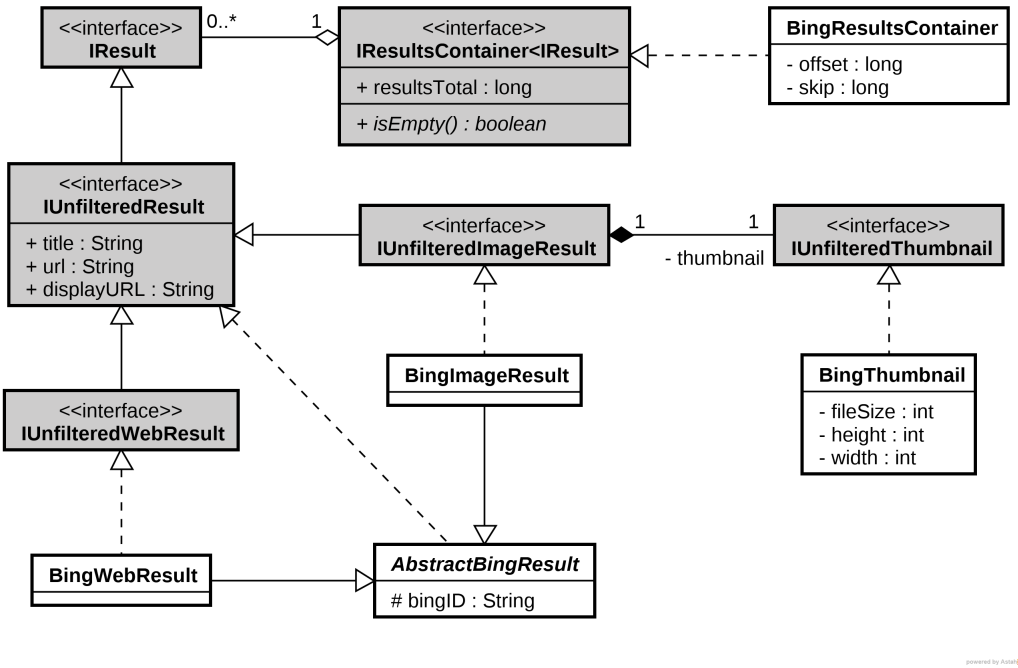

Abbildung 5.2: Dargestellt sind die Realisierungen der Schnittstellen aus Abbildung 4.4. Die Schnittstellen sind zur besseren Unterscheidung grau hinterlegt. Es wurden nur die Attribute und Methoden der realisierten Schnittstellen abgebildet, die in der Beschreibung in Abschnitt 5.4.1 erwähnt werden.

sechste aller Ergebnisse sein. Der `offset`-Wert sollte damit mit dem `skip`-Wert der Parametrisierung der Suchanfrage übereinstimmen (siehe Abschnitt 5.3). Nach mehrfachen Tests wurde allerdings ein undefiniertes Verhalten festgestellt, sobald der `offset`-Wert größer als die gesamte Ergebnismenge (`resultsTotal`) ist. Zudem ist in diesem Fall die Bing-Search-API-Rückgabe nicht leer, sondern gibt Ergebnisse beginnend von `offset` „`0`“ zurück. Es ist dadurch nicht möglich, die `isEmpty()`-Methode anhand der zurückgegebenen Ergebnismenge oder des `offset`-Werts zu bestimmen. Deshalb enthält der `BingResultsContainer` zusätzlich ein `skip`-Attribut, welches denselben Wert wie der `skip`-Parameter der zuvor durchgeführten Suchanfrage besitzt. Damit gilt, dass der `BingResultsContainer` nach `isEmpty()` leer ist, wenn der Startwert `skip` größer oder gleich der gesamten Ergebnismenge `resultsTotal` ist.

5.4.2 Persistente Speicherung

Dieser Abschnitt behandelt die Umsetzung der persistenten Speicherung. Hierbei wird die in der GlassFish-Installation enthaltene Datenbank Java DB[23] als persistenter Datenspeicher genutzt. Zuerst wird die Modellierung der Benutzerkonten behandelt. Dabei wird auch auf die Verwaltung der Benutzerkonten eingegangen. Abschließend wird die Speicherung der Entitys beschrieben.

Benutzername und Passwort werden in einer gemeinsamen Tabelle gespeichert. Um die Eindeutigkeit des Benutzernamens nach Anforderung A8.3 zu gewährleisten, wird der Benutzername als Primärschlüssel der Tabelle genutzt. Das Passwort wird zur Erfüllung von Anforderung A8.4 als SHA-256-Hashsumme gespeichert. Eine weitere Tabelle modelliert Rechtegruppen. Auch Rechtegruppen werden eindeutig als Primärschlüssel modelliert. Diese Rechtegruppen werden über eine dritte Tabelle mit der Benutzername-Passwort-Tabelle verknüpft. Hierdurch wird Anforderung A8.5 erfüllt, die eine Assoziation von Benutzerkonten mit Rechtegruppen fordert. Einem Benutzer können somit mehrere Rechtegruppen zugewiesen werden. In der dritten Tabelle wird die Kombination der Fremdschlüssel Benutzername und Rechtegruppe als zusammengesetzter Primärschlüssel genutzt. Durch die Aufteilung in drei Tabellen werden Attribut-Redundanzen vermieden. Zeitgleich bleiben die Benutzername-Passwort-Tabelle und die Rechtegruppe-Tabelle erweiterbar. Änderungen an den Benutzerkonten wie das Hinzufügen und Löschen können nur direkt auf der Datenbank durchgeführt werden. Diese Umsetzung wurde als ausreichend erachtet, da die Anzahl der benötigten Benutzerkonten als gering eingeschätzt wurde. Das liegt daran, dass nur eine Rechtegruppe umgesetzt wurde. Diese modelliert Administratornutzer, deren erweiterte Rechte wenigen Nutzern vorbehalten bleiben sollten. Die Kann-Anforderung A8.8, welche eine graphische Benutzeroberfläche zur Verwaltung der Benutzerkonten fordert, wird somit verletzt.

Bei der Speicherung in der Datenbank wird jeder Entity-Typ in einer eigenen Tabelle gespeichert. Verknüpfungen zwischen Entitys untereinander erfolgen jeweils über eine zusätzliche Tabelle. Diese besteht ähnlich wie die Benutzername-Rechtegruppe-Tabelle nur aus den Primärschlüsseln der jeweiligen Entitys. Als Primärschlüssel für die einzelnen Entitys sollen künstliche IDs erzeugt werden.

[23] `http://www.oracle.com/technetwork/java/javadb/overview/index.html` letzter Zugriff 19.09.2013, 02:00 Uhr

Die URIs der InFO-Elemente als ID zu nutzen, ist nicht praktikabel, da diese sehr lang sind. Dadurch würden die Entity-Entity-Tabellen unnötig groß werden.

5.4.3 Anmeldung

Dieser Abschnitt beschreibt zuerst den Funktionsumfang der umgesetzten Anmeldung für Endbenutzer mit Benutzerkonten. Danach wird knapp erläutert, wie vom Nutzer eingegebene Anmeldedaten mit der Datenbank abgeglichen werden.

Die Anforderung A8.1 verlangt die Möglichkeit einer Anmeldung von Endbenutzern mit einem Benutzerkonto. Diese Anforderung wird durch eine mögliche Anmeldung mittels HTTP-Authentifizierung erfüllt. Polsearchine verwendet dazu eine Basic Authentication [FHBH+99]. Diese hat den Nachteil, dass eine Abmeldung nicht definiert ist. Die durch die Kann-Anforderung A8.2 geforderte Abmeldung wird somit nicht unterstützt. Anforderung A8.1.1 fordert eine verschlüsselte Verbindung während der Übertragung der Anmeldedaten. Sie wird dadurch erfüllt, dass alle Interaktionen mit Polsearchine über das Protokoll TLS 1.0 [DAF+99] erfolgen. Es handelt sich dabei um ein Netzwerkprotokoll zur verschlüsselten Übertragung von Daten.

Damit eine Anmeldung möglich ist, muss ein *JDBC Realm* existieren. Die Informationen dazu entstammen Oracle Corporation [Ora13]. Es handelt sich beim JDBC Realm um eine Abkapselung der Datenbank, die nur die zur Nutzerauthentifizierung benötigten Daten enthält. Der Realm nutzt für den Datenbankzugriff das in Abschnitt 5.1.3 vorgestellte JDBC-Resource-Objekt. Über dieses hat der JDBC Realm Zugriff auf die Tabellen, die Nutzernamen, Passwörter und Rechtegruppen beinhalten. Die HTTP-Authentifizierung nutzt diesen Realm, um die Eingabedaten mit den Benutzerkonten der Datenbank abzugleichen. Dabei liest sie über den JDBC Realm auch die Rechtegruppe des Benutzerkontos aus. Für den Zugriff auf das Backend muss ein Benutzerkonto mit der Rechtegruppe „`POLSEARCHINE_ADMIN`“ verknüpft sein. Ein Benutzerkonto mit einer solchen Verknüpfung entspricht dem in Abschnitt 3.5 eingeführten Administratornutzer. Dadurch wird der Zugriff auf das Backend auf Administratornutzer eingeschränkt und Anforderung A9.1 erfüllt. Die Einrichtung des Realms wird in Anhang A.7.2 behandelt.

5.4.4 Globale Anwendungsvariablen

Dieser Abschnitt beschreibt die globalen Anwendungsvariablen. Dabei handelt es sich um anwendungsweite Parameter, die sich an einem zentralen Speicherort befinden. Sie bieten Konfigurationsmöglichkeiten für das Verhalten von Polsearchine. Als Speicherort wurde die `application.xml` der Webanwendung gewählt. Die Datei ist standardmäßig in Java-EE-Webanwendungen enthalten und speichert Meta-Informationen über die Anwendung [Ora13]. Durch die Nutzung der `application.xml` befinden sich die globalen Anwendungsvariablen außerhalb des eigentlichen Programmcodes. Das erleichtert die Verwaltung, da sie kein Verständnis des Programmcodes voraussetzt. Zusätzlich wird die Flexibilität der Anwendung erhöht, da eine Änderung einer globalen Anwendungsvariable keine neue Kompilierung des Programms erfordert. Allerdings muss ein Deploying durchgeführt werden, damit die Änderungen von der Webanwendung umgesetzt werden. Mit Deploying wird das Installieren der Webanwendung auf dem Server bezeichnet (siehe Abschnitt 5.1.3). Das Deploying ist nötig, da sich die XML-Datei innerhalb der Webanwendung befindet. Die Webanwendung wird als eine einzelne Datei vertrieben und einmalig beim Installieren der Anwendung verarbeitet (siehe Abschnitt 5.1.3). Dadurch, dass sich die `application.xml` innerhalb der vertriebenen Datei befindet, kann die Webanwendung sie nicht verändern. Infolgedessen wird die Kann-Anforderung A11.4 verletzt, die eine Änderung der Konfiguration aus dem Backend heraus fordert. Anforderung A11.3 fordert die Anzeige der aktuellen Konfiguration im Backend. Diese Anforderung wird erfüllt, indem die globalen Anwendungsvariablen im Backend angezeigt werden. Die nachfolgende Aufzählung zeigt die umgesetzten globalen Anwendungsvariablen. Falls eine Anforderung eine Konfigurierbarkeit forderte, wird diese dadurch erfüllt. Dies wird in der Aufzählung durch Aufführen der Anforderung in Klammern deutlich gemacht.

- Backend-Suchmaschine (Anforderung A1.3)
- IP-Adressen-Muster (Anforderung A2)
- Dateipfad der Erweiterungen der Datenschutzerklärung (Anforderung A4.1.1)
- Dateipfad zur Speicherung der Policy-Dateien (Anforderung A10.1)
- Dateiendung für Policy-Dateien (Anforderung A10.2)

- Bing-Search-API-Schlüssel
- URI zum Verweis auf Polsearchine

Der Bing-Search-API-Schlüssel wird benötigt, um die Bing Search API zu nutzen. Er wird bei der Registrierung zur Nutzung der API[24] erstellt und ist an einen Microsoft Account[25] gebunden. Die Auslagerung dieses Schlüssels in eine globale Anwendungsvariable erleichtert die Nutzung von Polsearchine durch unterschiedliche Betreiber mit jeweils eigenen Schlüsseln. Eine weitere globale Anwendungsvariable wird verwendet, um einen Verweis auf Polsearchine zu speichern. Es handelt sich dabei um eine URI. Sie wird genutzt, wenn Ergebnisse aufgrund der Rückfallregel blockiert werden (siehe Abschnitt 4.3.3). Die URI kann alternativ auch auf eine Webseite verweisen, die allgemeine Informationen über Regulierungen enthält. Damit kann kompensiert werden, dass die Rückfallregel keine rechtlichen oder organisatorischen Hintergrundinformationen bereitstellt.

[24] `https://datamarket.azure.com/dataset/bing/search` letzter Zugriff 19.09.2013, 18:30 Uhr

[25] `http://windows.microsoft.com/en-US/windows-live/sign-in-what-is-microsoft-account` letzter Zugriff 19.09.2013, 18:30 Uhr

Kapitel 6

Ergebnisse der Implementierung

In diesem Kapitel werden die Ergebnisse der Implementierung vorgestellt. Abschnitt 6.1 zeigt tabellarisch, inwieweit die Anforderungen aus Kapitel 3 durch die Implementierung umgesetzt werden konnten. In Abschnitt 6.2 wird der realisierte Funktionsumfang von Polsearchine zusammengefasst. Abschließend werden in Abschnitt 6.3 mögliche Weiterentwicklungen aufgezeigt, die sich aus der Konzeption und Umsetzung von Polsearchine ergeben. Soweit dies möglich ist, werden dabei auch Lösungsansätze vorgestellt.

6.1 Tabellarischer Vergleich der Anforderungen und Implementierung

Dieser Abschnitt fasst die Umsetzung der Anforderungen durch die Implementierung in Tabelle 6.1 zusammen. Dabei ist jede Anforderung aus Kapitel 3 mit ihrem Bezeichner aufgelistet. Es ist vermerkt, ob es sich um eine Muss- („M“), Soll- („S“) oder Kann-Anforderung („K“) handelt. Eine Kurzbschreibung der Anforderung fasst ihren Inhalt zusammen. Rechts davon befindet sich jeweils der Status der Umsetzung. Der Status wird unterschieden in umgesetzt („J“), nicht umgesetzt („N“) und teilweise umgesetzt („T“). In einer letzten Spalte befinden sich Anmerkungen zur Implementierung.

Anf.	Typ	Inhalt der Anforderung	St.	Anmerkung
A1	M	Meta-Suchmaschine	T	Siehe Unteranforderungen

Anf.	Typ	Inhalt der Anforderung	St.	Anmerkung
A1.1	M	Mindestens eine Backend-Suchmaschine	J	Bing Search API
A1.2	K	Mehrere Backend-Suchmaschinen parallel	N	Hoher Aufwand und häufig erschwert durch Nutzungsbedingungen von Suchmaschinen-APIs
A1.3	M	Austauschbare Backend-Suchmaschine	J	Abstraktion durch Schnittstellen
A1.4	M	Suche nach Webseiten	J	-
A1.5	K	Suche nach Videos	N	-
A1.6	K	Suche nach Bildern	J	-
A1.7	M	Polsearchine muss eine Webanwendung sein.	J	Java-EE-Webanwendung
A1.8	S	Suche nach Dateitypen	J	Schnittstelle für unterstützte Dateitypen
A2	S	IP-Adressen-Muster	J	Globale Anwendungsvariable
A3	M	Impressum	J	-
A4	M	Datenschutzerklärung	J	-
A4.1	M	Dynamische Erweiterung der Datenschutzerklärung	J	Mittels globaler Anwendungsvariable
A4.1.1	S	Dateipfad für Erweiterungen von Datenschutzerklärungen	J	Globale Anwendungsvariable
A5	M	Regulierung anhand von InFO-Policy-Dateien	T	`Sender`, `SenderSpecifier`, einige Policy-Priorisierungsalgorithmen und ausführliche Prüfung auf `EnforcingSystem` nicht umgesetzt
A5.1	K	Regulierung anhand von Hashwerten	N	Nicht vollständig ausgearbeitet in InFO-Version 0.1
A5.2	M	Regulierung anhand von URLs	J	-
A5.3	K	Umsetzung von Veränderungen an Policys zur Laufzeit	N	Umsetzung erfolgt nur beim Anwendungsstart
A6	M	Hintergrundinformationen zu jeder Regulierung	J	-
A6.1	M	Anzeige des Erstellers der Regel	J	-
A6.2	M	Anzeige des Themas der Regel	J	-

Anf.	Typ	Inhalt der Anforderung	St.	Anmerkung
A6.3	M	Anzeige der rechtlichen Grundlage/des organisatorischen Beweggrunds	J	-
A7	M	Verarbeitung von Policys	T	Bei jedem Anwendungsstart, siehe Unteranforderungen
A7.1	M	Umwandlung der Policy-Dateien in Entitys	J	Bei jedem Anwendungsstart
A7.2	M	Persistente Speicherung von Entitys	J	In einer Datenbank
A7.3	M	Eindeutige Feststellbarkeit einer Regulierung anhand der URL	J	Durch Policy-Verarbeitung
A7.4	M	Regulierung zur Laufzeit ausschließlich durch Entitys	T	Mögliche Vorfilterung der Backend-Suchmaschine unklar
A8	M	Unterstützung von Benutzerkonten	T	Siehe Unteranforderungen
A8.1	M	Anmeldung	J	HTTP Basic Authentication
A8.1.1	S	Verschlüsselte Übertragung der Anmeldedaten	J	Alle Zugriffe auf Polsearchine erfolgen über TLS 1.0
A8.2	K	Abmeldung	N	Nicht unterstützt in HTTP Basic Authentication
A8.3	M	Einmaliger Nutzername	J	Nutzername ist Primärschlüssel in der Datenbank
A8.4	M	Benutzerkonto muss Nutzername und Passwort besitzen	J	-
A8.5	M	Assozation von Benutzerkonten mit Rechtegruppen	J	-
A8.6	M	Persistente Speicherung von Benutzerkonten	J	Erfolgt in Datenbank
A8.7	M	Speicherung des Passworts als Hashwert	J	SHA-256-Hashsumme
A8.8	K	Graphische Oberfläche zur Benutzerkontenverwaltung	N	Verwaltung nur via Datenbank
A9	S	Verwaltung der Policy-Dateien über eine Webseite	T	Befindet sich im Backend, siehe Unteranforderungen
A9.1	M	Einschränkung der Verwaltung auf eine Rechtegruppe	J	Backendzugriff ist auf Rechtegruppe eingeschränkt

Anf.	Typ	Inhalt der Anforderung	St.	Anmerkung
A9.2	M	Anzeige der Namen der Policy-Dateien in der Verwaltung	J	-
A9.3	K	Menschenlesbare Kurzbeschreibung für Policys	N	Kann etwa durch OWL-Annotation umgesetzt werden
A9.4	M	Hinzufügen von Policy-Dateien durch Hochladen	J	-
A9.5	M	Möglichkeit des Löschens von Policy-Dateien	J	-
A10	S	Speicherung der Policy-Dateien auf dem Server-Dateisystem	J	-
A10.1	S	Dateipfad zur Speicherung der Policy-Dateien	J	Globale Anwendungsvariable
A10.2	S	Dateiendung zur Speicherung der Policy-Dateien	J	Globale Anwendungsvariable
A11	M	Backend	T	Siehe Unteranforderungen
A11.1	S	Backend als Webseite	J	-
A11.2	M	Backendzugriff nur als Administratornutzer	J	-
A11.3	K	Anzeige aktueller Konfiguration im Backend	J	Anzeige von globalen Anwendungsvariablen
A11.4	K	Veränderung der Konfiguration aus Backend heraus	N	`application.xml` kann aus Webanwendung heraus nicht geschrieben werden
A11.5	K	Direkte Umsetzung von Konfigurationsänderungen	N	GlassFish liest globale Anwendungsvariablen einmalig beim Deploying ein
A11.6	S	Verwaltung der Policy-Dateien über das Backend	J	-

Tabelle 6.1: Diese Tabelle listet alle Anforderungen in der Spalte „Anf.“ auf. Die „Typ“-Spalte gibt an, ob es sich bei der Anforderung um eine Muss- („M“), Soll- („S“) oder Kann-Anforderung („K“) handelt. Der Inhalt der Anforderung wird knapp in der gleichnamigen Spalte zusammengefasst. Die „St.“-Spalte beschreibt den Status der Implementierung. Umgesetzt wird mit einem „J“ gekennzeichnet, nicht umgesetzt mit einem „N“ und teilweise umgesetzt mit einem „T“. In der letzten Spalte befinden sich Anmerkungen zur (Nicht-) Umsetzung der Anforderung.

6.2 Realisierter Funktionsumfang

Der realisierte Funktionsumfang von Polsearchine ergibt sich aus den Anforderungen (Kapitel 3), der Konzeption (Kapitel 4) und der Implementierung (Kapitel 5). In diesem Abschnitt wird er zusammenfassend vorgestellt.

Mit Polsearchine konnte eine regulierende Meta-Suchmaschine entwickelt werden. Sie wurde als Java-EE-Webanwendung umgesetzt, die ihre Ergebnisse über die Bing Search API bezieht. Durch abstrahierende Programmierschnittstellen ist es möglich, die Backend-Suchmaschine auszutauschen. Polsearchine unterstützt sowohl eine Suche nach Webseiten als auch nach Bildern. Zusätzlich kann die Suche auf Dateitypen eingeschränkt werden, die von der aktiven Backend-Suchmaschine unterstützt werden. Die Ergebnisse werden nicht auf einzelne Seiten aufgeteilt, sondern dynamisch nachgeladen. Mittels eines IP-Adressen-Musters kann die Suchfunktion anhand der IP-Adresse des Endbenutzers eingeschränkt werden. Die entwickelte Webanwendung besitzt jeweils eine Webseite für ein Impressum und eine Datenschutzerklärung. Abhängig von der gewählten Backend-Suchmaschine wird die Datenschutzerklärung dynamisch erweitert.

Polsearchine unterstützt die Anmeldung mittels Benutzerkonten. Alle Seitenzugriffe und somit auch die Übertragung der Anmeldedaten erfolgen verschlüsselt. Passwörter werden als Hashwert gespeichert. Benutzerkonten können mit Rechtegruppen verknüpft werden. Eine konkrete Rechtegruppe ermöglicht den Zugriff auf das Backend. Dieses zeigt die Policy-Dateien, die sich auf dem Dateisystem des Servers befinden. Vom Backend aus können vorhandene Policy-Dateien gelöscht und neue hinzugefügt werden. Ein Algorithmus speichert die Dateien dabei nach einem einheitlichen Benennungsschema auf dem Dateisystem. Das Backend zeigt zudem die aktuelle Konfiguration von Polsearchine. Konfigurationen erfolgen in einer Datei außerhalb des Quellcodes.

Die Regulierung erfolgt durch InFO-Policy-Dateien, die zu jedem Anwendungsstart verarbeitet werden. Dabei werden Regeln, Policys und Meta Policys validiert. Anschließend werden Policys und Regeln priorisiert und Konflikte zwischen Regeln gelöst. Zusätzlich führt Polsearchine eine Entfernung redundanter Regeln durch. Dadurch ist für jedes Ergebnis anhand seiner URL eindeutig feststellbar, ob und wie es reguliert wird. Die essentiellen Informationen der Policy-Dateien zur Regulierung werden persistent in einer Datenbank gespeichert. Regulierungen können somit deutlich schneller durchgeführt werden, als wenn

für jedes darzustellende Ergebnis eine Policy-Verarbeitung erfolgen müsste. Gefilterte Ergebnisse erlauben dem Endbenutzer die rechtlichen und organisatorischen Hintergrundinformationen ihrer Regulierung einzusehen.

6.3 Mögliche Weiterentwicklungen

Abschnitt 6.1 zeigt, dass Polsearchine die Anforderungen zu einem Großteil umsetzt. Aus den nicht oder nur teilweise umgesetzten Anforderungen ergeben sich mögliche Erweiterungen für Polsearchine. Es handelt sich dabei vor allem um Erweiterungen, die die Arbeit mit Polsearchine als Endbenutzer oder Administratornutzer einfacher gestalten würden. Weiter zeigten sich während der Entwicklung Bereiche auf, denen zusätzliche Arbeit gewidmet werden sollte. Diese potentiellen Erweiterungen werden nachfolgend vorgestellt. Wenn möglich, wird zudem ein knapper Lösungsansatz beschrieben.

Anforderung A1.2 fordert eine parallele Nutzung mehrerer Backend-Suchmaschinen. Diese Anforderung wurde aufgrund des mit ihr verbundenen großen Aufwands nicht implementiert. Zusätzlich verhindern oder erschweren die Nutzungsbedingungen einiger Suchmaschinen-APIs eine gleichzeitige Nutzung mit anderen Suchmaschinen (vgl. [Mic12b, Yan12d]). Eine Weiterentwicklung kann Polsearchine zur parallelen Nutzung mit mehreren Backend-Suchmaschinen anpassen. Zuvor müssen aber Suchmaschinen-APIs gefunden werden, deren Nutzungsbedingungen einen solchen Einsatz erlauben.

Polsearchine wurde als Meta-Suchmaschine konzipiert, die ihre Suchergebnisse von einer Backend-Suchmaschine erhält. Dabei erfolgt die eigens durchgeführte Regulierung gemäß Anforderung A7.4 ausschließlich anhand der Entitys. Bei der Wahl der Backend-Suchmaschine in Abschnitt 5.2 wurde dies mit Anforderung R2 berücksichtigt. Mit dieser Anforderung R2 wurde gefordert, dass die Backend-Suchmaschine nicht vorfiltert. Die im Rahmen dieser Bachelorarbeit dazu durchgeführten Untersuchungen stellen nur einen ersten Schritt dar. Weiterführende Untersuchungen zur Regulierung von Suchmaschinen sind nötig.

Anforderung A5 fordert die Regulierung anhand von InFO-Policy-Dateien. Diese wurde nur teilweise umgesetzt. Die Verarbeitung der Sprachelemente `Sender` und `SenderSpecifier` wurde aus Zeitgründen nicht implementiert. Es wurden einige Priorisierungsalgorithmen nicht umgesetzt, da der verwendete InFO-Parser dazu nötige Sprachelemente nicht unterstützt (siehe Abschnitt 4.2.3).

Die Prüfung des Umsetzungssystems der Policys und Meta Policys wurde aufgrund eingeschränkter Unterstützung des `TechnicalSystem`s durch den InFO-Parser vernachlässigt (siehe Abschnitt 4.2.2). Eine Erweiterung kann die Unterstützung der Sprachelemente in Polsearchine und dem InFO-Parser implementieren. Anschließend können die fehlenden Algorithmen in Polsearchine umgesetzt werden. Zusätzlich kann eine Weiterentwicklung die Regulierung anhand von Hashwerten gemäß der Anforderung A5.1 umsetzen. Sie wurde in dieser Bachelorarbeit vernachlässigt, da SEFCO in Version 0.1 nicht definiert, über welche Teile einer Webseite Hashwerte berechnet werden sollen.

Aktuell zeigt die Übersicht der Policy-Dateien nach Anforderung A9.2 die Dateinamen an. Um den Inhalt der Dateien einfacher verständlich zu machen, kann eine Erweiterung die nicht erfüllte Anforderung A9.3 umsetzen. Sie fordert die Darstellung einer Kurzbeschreibung für Policys. Zwar bietet InFO dazu kein eigenes Sprachelement an, der Aufbau der Ontologie auf OWL erlaubt aber die Nutzung des Sprachelements `owl:AnnotationProperty` [BHH+04]. Dieses ermöglicht Teile von Ontologien wie etwa einzelne Policys mit Kommentaren zu versehen. Das darin benutzbare `rdfs:label`-Sprachelement dient als natürlichsprachlicher Bezeichner [BHH+04]. Es kann somit zur Erfüllung von Anforderung A9.3 verwendet werden.

Die Benutzerkontenverwaltung setzt aktuell Datenbankzugriff und SQL-Vorwissen heraus (siehe Abschnitt 5.4.2). Um die Benutzerkontenverwaltung einfacher zu gestalten, kann sie nach Anforderung A8.8 als graphische Benutzeroberfläche umgesetzt werden. In dieser könnten über Eingabeformulare neue Benutzerkonten erzeugt werden. Weiter könnten vorhandene in einer Liste dargestellt und von dieser aus bearbeitet oder gelöscht werden.

Polsearchine nutzt eine HTTP-Authentifizierung zur Anmeldung von Benutzerkonten, die keine Abmeldung unterstützt (siehe Abschnitt 5.4.3). Eine Erweiterung würde diese in Anforderung A8.2 geforderte Abmeldung umsetzen. Dazu könnte statt einer HTTP-Authentifizierung eine Formular-basierte Anmeldung implementiert werden, die eine Abmeldung unterstützt (siehe [Ora13]).

Anforderung A7 verlangt, dass Polsearchine Policys verarbeiten können muss. Dies geschieht aktuell jeweils einmalig beim Anwendungsstart (siehe Abschnitt 4.2). Eine Erweiterung kann die Verarbeitung während des laufenden Betriebs ermöglichen. Dadurch werden Änderungen an Policys nicht erst beim erneuten Anwendungsstart angewandt. Für eine Ausführung im laufenden Betrieb sind jedoch

zwei Vorkehrungen nötig. Erstens muss garantiert werden, dass alle Datenmanipulation zusammen atomar sind (siehe [HR83]). Das bedeutet, dass die Datenbank vollständig auf ihren vorherigen Zustand zurückgesetzt werden muss, sollte die Policy-Verarbeitung fehlschlagen. Zweitens muss die gesamte Datenmanipulation isoliert erfolgen (siehe [HR83]). Der Lesezugriff auf die Datenbank muss dazu während der Manipulation exklusiv für die Policy-Verarbeitung gesperrt werden. Anderenfalls könnten Endbenutzern falsch regulierte Ergebnisse angezeigt werden, falls sie eine Suchanfrage während einer Policy-Verarbeitung senden. Schreibzugriffe müssen hingegen nicht unterbunden werden, da nur die Policy-Verarbeitung auf die Datenbank schreibt.

Eine Weiterentwicklung von Polsearchine kann die Anforderung A11.4 implementieren, die eine Veränderung der Konfiguration über das Backend ermöglicht. Aktuell werden die Konfigurationen anhand der `application.xml` vorgenommen (siehe Abschnitt 5.4.4). Das führt zudem dazu, dass Änderungen an der Konfiguration erst nach einem erneuten Deploying in Kraft treten (siehe Abschnitt 5.4.4). Eine Umsetzung von Änderungen an der Konfiguration zur Laufzeit (Anforderung A11.5) kann implementiert werden. Eine Lösung muss dazu die globalen Anwendungsvariablen außerhalb der Java-EE-Webanwendung speichern.

Es ist eine Überarbeitung der Benutzeroberfläche zur Nutzung auf touchbasierten Endgeräten denkbar. Diese Geräte besitzen meist keinen Mauszeiger, sondern werden über eine Eingabe mittels der Finger gesteuert. Die Bildersuche zeigt allerdings zusätzliche Informationen zu Bildergebnissen nur, wenn ein Mauszeiger auf der Miniaturansicht platziert ist (siehe Abschnitt 4.3.4). Eine alternative Umsetzung kann das Anzeigen der zusätzlichen Informationen durch einen Klick auf die Ergebniskarte ermöglichen.

Kapitel 7

Fazit

In dieser Bachelorarbeit wurde die prototypische Meta-Suchmaschine Polsearchine[1] entwickelt. Sie bezieht ihre Ergebnisse von einer austauschbaren Backend-Suchmaschine und ermöglicht eine regulierende Suche nach Webseiten und Bildern. Die Regulierung erfolgt mittels Policy-Dateien, die mit SEFCO erstellt werden können. SEFCO ist eine Erweiterung der Ontologie InFO für Suchmaschinen. InFO stellt auch Erweiterungen zur Internetregulierung auf DNS- und IP-Ebene bereit. Hierdurch ist eine Regulierung von Suchmaschinenergebnissen möglich, die mit der Zugriffsregulierung auf Webseiten oder Internetauftritte übereinstimmt. Dies erschwert das Umgehen von Regulierungen auf Suchmaschinenebene.

Polsearchine kennzeichnet gefilterte Ergebnisse, so dass sie für den Endbenutzer als solche erkennbar sind. Sie behalten ihre Position in der Liste der Ergebnisse bei. Dadurch kann ein Endbenutzer nachvollziehen, an welcher Stelle ein Ergebnis gefiltert wurde. Es ist ihm außerdem möglich, zu jedem gefilterten Ergebnis die rechtlichen und organisatorischen Hintergrundinformationen der Regulierung einzusehen. Die durchgeführte Internetregulierung ist damit transparenter als die beobachtete Filterung bei Google Search, Bing, Baidu oder Yahoo! Search. Das gilt allerdings nur, falls die Ergebnisse der genutzten Backend-Suchmaschine nicht vorgefiltert sind. Für die in dieser Bachelorarbeit verwendete Bing Search API konnte eine Vorfilterung weder belegt noch widerlegt werden. Neben weiteren aufgezeigten Möglichkeiten der Weiterentwicklung ergibt sich daraus eine potentielle Folgearbeit. Durch sie sollte analysiert werden, wie stark Vorfilterung bei unterschiedlichen Suchmaschinen ausgeprägt ist.

[1] `https://search.icp.it-risk.iwvi.uni-koblenz.de/`

Anhang A

Installation und Konfiguration

Dieser Anhang beschreibt die Installationen und Konfiguration, die nötig sind, um die entwickelte Suchmaschine nutzen zu können. Im ersten Abschnitt A.1 werden die Vorbereitungen für das Host-System behandelt. Abschnitt A.2 beschreibt die Installation von GlassFish. Für die darauffolgenden Schritte wird ein Vorwissen bei drei Programmen benötigt. Diese Programme sind in der GlassFish-Installation enthalten. Ihre Anwendungsbereiche und Nutzung werden im Abschnitt A.3 beschrieben. Die Installation kann daraufhin mit der Erstellung der Datenbank fortgesetzt werden. Dies wird in Abschnitt A.4 behandelt. Abschnitt A.5 beschreibt die Arbeitsschritte, die zur Einrichtung eines Polsearchine-Administratornutzers nötig sind. Anschließend wird die Erstellung und Einrichtung der Domäne beschrieben. Hierbei gibt es zwei Herangehensweisen. Die erste Möglichkeit ist, eine bereits eingerichtete Domäne zu kopieren. Dies erspart jegliche Konfiguration. Die alternative Vorgehensweise beschreibt die Installation und Konfiguration von GlassFish Server und Polsearchine von Grund auf. Diese Herangehensweise ist deutlich aufwändiger. Sie kann sich aber für Nutzer ohne oder mit wenig Erfahrung mit GlassFish als hilfreich erweisen. Die schnelle Installation durch Kopieren wird in Abschnitt A.6 beschrieben. Demgegenüber steht die ausführliche Neueinrichtung, welche in Abschnitt A.7 behandelt wird. Unabhängig von der gewählten Herangehensweise entspricht die Nummerierung der Abschnitte der Reihenfolge der Durchführung.

A.1 Host-System vorbereiten

Dieser Abschnitt beschreibt die Vorbereitung eines Host-Systems zur anschließenden Installation der entwickelten Suchmaschine. Genutzt wird ein Computer mit Debian Squeeze[1] für 32-Bit Architekturen. Die Suchmaschine nutzt einige Funktionen, die erst ab OpenJDK 7[2] unterstützt werden. Da dieses nicht in den Repositories von Debian Squeeze liegt, müssen die Pakete aus Debian Wheezy[3] genutzt werden. Die Installation läuft wie folgt ab:

```
1 echo "deb http://ftp.de.debian.org/debian wheezy main" | \
2     tee -a /etc/apt/sources.list
3 apt-get update
4 apt-get install openjdk-7-jdk
5 sed -i '$d' /etc/apt/sources.list
6 apt-get update
```

A.2 GlassFish installieren

Die in dieser Bachelorarbeit entwickelte Suchmaschine wird in einer Domäne des GlassFish Server Open Source Edition 3.1.2.2 ausgeführt (siehe Abschnitt 5.1.3). Er kann über die Webseite archivierter GlassFish-Versionen[4] oder der Webseite der verwendeten GlassFish-Version[5] heruntergeladen werden. Zur Installation wurde das englische ZIP-Archiv[6] verwendet. Dazu muss das Archiv nach dem Herunterladen in `/opt/` entpackt werden:

```
1 unzip glassfish-3.1.2.2.zip -d /opt/
```

An dieser Stelle ist die Installation des Servers in den Ordner `/opt/glassfish3/` abgeschlossen.

[1] `http://www.debian.org/releases/stable/` letzter Zugriff 27.03.2013, 16:45 Uhr

[2] `http://openjdk.java.net/projects/jdk7/` letzter Zugriff 03.04.2013, 15:45 Uhr

[3] `http://www.debian.org/releases/wheezy/` letzter Zugriff 28.03.2013, 01:00 Uhr

[4] `http://glassfish.java.net/download-archive.html` letzter Zugriff 08.09.2013, 15:00 Uhr

[5] `http://glassfish.java.net/downloads/3.1.2.2-final.html` letzter Zugriff 08.09.2013, 15:00 Uhr

[6] `http://download.java.net/glassfish/3.1.2.2/release/glassfish-3.1.2.2.zip` letzter Zugriff 18.04.2013, 20:51 Uhr

A.3 Wichtige Programme

Dieser Teil stellt drei wichtige Programme vor, die GlassFish bereitstellt. Die Programme heißen `asadmin`, `DAS` und `ij`. Sie werden in den Abschnitten A.4, A.5, A.7.1, A.7.2 und A.7.3 während der Installation von Polsearchine benötigt. Deshalb werden die Programme hier knapp beschrieben. Dabei wird deren Einsatzzweck erläutert. Zusätzlich wird darauf eingegangen, wie sie gestartet und gegebenenfalls beendet werden. Hierbei wird stets davon ausgegangen, dass GlassFish in den Ordner `/opt/glassfish3/` installiert wurde. Eine derartige Installation ist unter A.1 beschrieben. `DAS` besitzt eine graphische Benutzeroberfläche. Eine solche Oberfläche heißt im Englischen „graphical user interface“ und wird mit *GUI* abgekürzt. Das textuelle Pendant dazu ist die Kommandozeile. Diese wird mit *CLI* für die Englische Bezeichnung „command-line interface“ abgekürzt. Sowohl `asadmin` als auch `ij` werden über ein CLI genutzt.

A.3.1 CLI-GlassFish-Administration mit `asadmin`

Um GlassFish zu verwalten, wird das Programm `/opt/glassfish3/bin/asadmin` verwendet. Nach dem Starten von `asadmin` kann sich ein GlassFish-interner Administrator mittels „`login`“ anmelden. Der GlassFish-interne Administrator ist nicht zu verwechseln mit dem Systemadministrator. Ersterer ist ein spezieller Nutzer, der nur der GlassFish-Installation bekannt ist. Wichtig ist bei einer Anmeldung, dass die entsprechende *Domäne* zuvor gestartet worden sein muss. Die Erstellung einer solchen wird in Abschnitt A.7 beschrieben. Abhängig von den Dateisystemberechtigungen ist es nötig, `asadmin` als Systemadministrator auszuführen. Wenn die Arbeit mit `asadmin` abgeschlossen wurde, kann „`exit`“ zum Beenden des Programms genutzt werden. Etwaig gestartete Domänen werden dadurch nicht beendet.

A.3.2 GUI-Domänenadministration mit `DAS`

Zur Konfiguration einer Domäne wird eine graphische Benutzeroberfläche angeboten. Diese ist in einem Webbrowser erreichbar und heißt *Domain Admin Server*. Sie wird mit *DAS* abgekürzt. Die Oberfläche kann über die IP-Adresse des Systems aufgerufen werden. Dazu muss als Port der für `DAS` konfigurierte Port mit ange-

Abbildung A.1: Dieses bearbeitete Bildschirmfoto von `DAS` zeigt in der linken Hälfte alle Punkte, die in diesem Anhang A benötigt werden.

geben werden. In der Standardeinstellung ist dies 4848. Damit `DAS` erreichbar ist, muss die Domäne mittels `asadmin` (siehe Abschnitt A.3.1) gestartet worden sein:

```
1 start-domain polsearchine
```

Die Grafik A.1 zeigt ein bearbeitetes Bildschirmfoto von `DAS`. Im linken Bereich der Oberfläche befindet sich eine Navigation. Über diese sind diverse Konfigurations- und Überwachungsmöglichkeiten erreichbar. Dabei zeigt Grafik A.1 ausschließlich die Auswahlmöglichkeiten, die in den anschließenden Abschnitten dieses Anhangs A genutzt werden.

A.3.3 CLI-Datenbankinteraktion mit `ij`

Die mit GlassFish eingesetzte Datenbank ist Apache Derby[7], auch bekannt unter dem Namen JavaDB. Durch SQL-Befehle kann mit `ij` direkt mit der Datenbank interagiert werden [Apa13]. Das Programm `ij` wird wie folgt gestartet:

```
1 /opt/glassfish3/javadb/bin/ij
```

Anschließend kann die Verbindung zu einer Datenbank erstellt werden. Ab dem Zeitpunkt können SQL-Befehle auf der Datenbank ausgeführt werden. Darauf

[7] `http://db.apache.org/derby/` letzter Zugriff 05.04.2013, 15:45 Uhr

wird hier nicht weiter eingegangen. Wenn die Arbeit abgeschlossen wurde, kann `ij` wie folgt beendet werden:

```
1 exit;
```

A.4 Erstellen der Datenbank

Polsearchine speichert die eigenen Administratornutzer und die Entitys in einer Datenbank. Damit dies möglich ist, muss die Datenbank erstellt werden. Zuerst wird unter `asadmin` (siehe Abschnitt A.3.1) der Datenbankserver gestartet:

```
1 start-database
```

Nachfolgend wird vom Datenbankserver auf das Verzeichnis `/opt/glassfish-3/glassfish/databases` geschrieben. Es ist also wichtig, dass `start-database` mit ausreichenden Dateisystemberechtigungen gestartet wurde. Es muss das Derby `ij`-Programm (siehe A.3.3) aufgerufen werden. Für die zu erstellende Datenbank mit dem Namen „`polsearchine`“ wird der Nutzer „`polsearchineDB`“, das Passwort „`useADifferentPassword`“ und der Port 1527 verwendet. Es ist empfehlenswert, ein anderes, stärkeres Passwort zu wählen. Mit dem nachfolgenden Befehl wird die Datenbank erzeugt:

```
1 CONNECT 'jdbc:derby://localhost:1527/polsearchine; ↩
    user=polsearchineDB; password=useADifferentPassword; ↩
    create=true';
```

Das Datenbankschema wird bei der Installation von Polsearchine in Abschnitt A.7.3 automatisch erstellt. Somit ist die Erstellung der Datenbank bereits abgeschlossen. In Abschnitt A.7.2 ist beschrieben, wie die Domäne konfiguriert werden muss, um die Datenbank nutzen zu können.

A.5 Einrichtung eines Administratornutzers

Dieser Abschnitt setzt voraus, dass eine Datenbank, wie in Abschnitt A.4 beschrieben, erstellt wurde. Es wird nun ein Administratornutzer erzeugt. Der Administratornutzer kann sich über die graphische Oberfläche der Suchmaschine anmelden. Im Backendbereich kann er Policys hochladen, löschen und die aktuelle Konfiguration einsehen. Dieser soll die Daten aus der folgenden Tabelle besitzen:

Name	Rechtegruppe	Passwort
admin	POLSEARCHINE_ADMIN	dontUseThisPassword

Das Nutzerpasswort darf nach Anforderung A8.7 nicht im Klartext in der Datenbank gespeichert werden. Deshalb muss ein SHA-256-Hashsumme von dem zu nutzenden Passwort erstellt werden. Dazu kann unter Debian wie folgt vorgegangen werden:

```
1 echo -n "dontUseThisPassword" | sha256sum
```

Die Ausgabe dieses Befehls wird später verwendet. Zuerst muss aber die Rechtegruppe für Zugriff auf das Backend in der Datenbank erstellt werden. Hierzu wird das `ij`-Programm (siehe Abschnitt A.3.3) gestartet. Anschließend wird die Rechtegruppe mittels folgendem SQL-Befehl erzeugt:

```
1 INSERT INTO POLSEARCHINEGROUP
2 VALUES ('POLSEARCHINE_ADMIN');
```

Nun wird ein Nutzer für das Backend erzeugt. Das Passwort ist die zuvor erzeugte Hash-Summe. Dadurch hat es eine Länge, die über die Seitenbreite hinausragt und musste ungünstiger Weise manuell umgebrochen werden. Es gilt aber auch hier, dass dieses Passwort nicht übernommen, sondern ein eigenes gesetzt werden sollte.

```
1 INSERT INTO POLSEARCHINEUSER
2 VALUES ('admin', '8fb939de0468016814d887071b5beee1bc2921 ↩
      99ca3664d981de95e18260d8bd');
```

Zum Schluss werden Nutzer und Rechtegruppe miteinander verknüpft:

```
1 INSERT INTO USER_GROUP
2 VALUES ('POLSEARCHINE_ADMIN', 'admin');
```

Ein erster Administratornutzer wurde nun erstellt.

A.6 Installation durch Kopieren

Die einfachste und schnellste Form der Installation von Polsearchine ist es, ein fertiges Archiv zu entpacken. Alternativ kann die Domäne auch manuell erstellt und eingerichtet werden (siehe Abschnitt A.7). Für eine Installation durch Kopie-

ren muss die Datei `polsearchine.tar.bz2` aus dem `dist`-Ordner im SVN[8] heruntergeladen werden. Anschließend wird sie wie folgt extrahiert:

```
1 tar xfjv polsearchine.tar.bz2 -C /opt/glassfish3/
```

Der extrahierende Nutzer muss dabei die Berechtigungen besitzen, einen Ordner in `/opt/glassfish3/glassfish/domains/` zu erstellen. Damit ist die Installation abgeschlossen und Polsearchine kann mittels `asadmin` (siehe Abschnitt A.3.1) gestartet werden:

```
1 start-domain polsearchine
```

A.7 Manuelle Einrichtung der Domäne

Nachfolgend wird eine Domäne für Polsearchine von Grund auf neu eingerichtet. Abschnitt A.7.1 beschreibt die Erstellung der Domäne. Hierbei wird auch auf die Schritte eingegangen, die für eine vollständige TLS-Verschlüsselung aller Polsearchine-Zugriffe benötigt werden. Abschnitt A.7.2 behandelt die Installation der Datenbank. Dadurch erhält die Webanwendung Zugriff auf die Datenbank. Der abschließende Abschnitt A.7.3 thematisiert die Erstellung der Pfade, die von einigen globalen Anwendungsvariablen (siehe Abschnitt 5.4.4) beschrieben werden.

A.7.1 Erstellung der Domäne

Die nachfolgenden Befehle werden alle innerhalb des `asadmin`-Programms ausgeführt (siehe auch Abschnitt A.3.1). Zuerst wird unter der Domäne „polsearchine“ die Suchmaschine wie folgt eingerichtet:

```
1 create-domain --adminport 4848 polsearchine
```

Der Adminport wurde hierbei auf den für GlassFish üblichen Port 4848 gesetzt. Während der Ausführung von `create-domain` wird nach einem internen Administratornamen der GlassFish-Installation und seinem Kennwort gefragt. Dieser Schritt sollte keineswegs übersprungen werden. Stattdessen empfiehlt es sich, einen eigenen Administrator mit einem sicheren Kennwort zu erstellen.

[8] `https://svn.uni-koblenz.de/aggrimm/BA-Ruster/dist/`

Der von Polsearchine genutzte Policy-Parser (siehe [BGSS13]) nutzt indirekt Xerces2 [9]. Es handelt sich dabei um einen Java-Parser. Dieser führt mit GlassFish zu Problemen, wenn die Binärdateien `xercesImpl.jar` und `xml-apis.jar` nicht in dem `lib/ext/`-Verzeichnis der soeben erstellten Domäne vorliegen. Für Polsearchine wurde die Binärdateien von Xerces2 Java 2.11.0[10] eingesetzt. Um diese in das gewünschte Verzeichnis zu extrahieren, muss zunächst `asadmin` beendet werden. Anschließend wird folgender Befehl genutzt:

```
1 tar xfzv Xerces-J-bin.2.11.0.tar.gz \
2     xerces-2_11_0/xercesImpl.jar \
3     xerces-2_11_0/xml-apis.jar
4 mv xerces-2_11_0/* \
5     /opt/glassfish3/glassfish/domains/polsearchine/lib/ext
6 rm -r xerces-2_11_0
```

Nach einem erneuten Aufrufen von `asadmin` ist es nun möglich, sich als Administrator anzumelden, nachdem der Server gestartet wurde:

```
1 start-domain polsearchine
2 login
```

Bevor der Adminstratornutzer auch auf `DAS` (siehe Abschnitt A.3.2) zugreifen kann, muss er dafür freigeschaltet werden. Dazu wird der nachfolgende Befehl ausgeführt, während der Server läuft.

```
1 enable-secure-admin --host localhost --port 4848
```

Wie die Ausgabe beschreibt, muss der Server neugestartet werden:

```
1 restart-domain polsearchine
```

Damit ein Suchmaschinennutzer später keine Ports beim Aufruf der Suchmaschine eingeben muss, sollen HTTP-Zugriffe über Port 80 und HTTPS-Zugriffe über Port 443 erfolgen. Die Ports unterhalb von 1024 sind als Systemports reserviert [Int13]. Deshalb sollen Zugriffe auf diese Ports auf die GlassFish-Ports 8080 für HTTP und 8181 für HTTPS weitergeleitet werden. Hierzu muss der Systemadministrator unter Debian folgende Befehle ausführen:

```
1 iptables -t nat -A PREROUTING -p tcp --dport 80 \
```

[9] `https://xerces.apache.org/xerces2-j/` letzter Zugriff 04.06.2013, 17:30 Uhr

[10] `http://mirror.arcor-online.net/www.apache.org//xerces/j/binaries/Xerces-J-bin.2.11.0.tar.gz` letzter Zugriff 04.06.2013, 17:30 Uhr

```
    -j REDIRECT --to-port 8080
iptables -t nat -A PREROUTING -p tcp --dport 443 \
    -j REDIRECT --to-port 8181
```

Nach Anforderung A8.1.1 sollen zusätzlich alle Zugriffe mit TLS verschlüsselt werden. Zu diesem Zeitpunkt sind unverschlüsselte Verbindungen über manuelle Zugriffe auf Port 8080 weiterhin möglich. Dies soll unterbunden werden, ohne eine systemweite Weiterleitung durchzuführen. Anderenfalls wäre der Port 8080 für andere Programme auf diesem System nicht mehr einsetzbar. Für die Weiterleitung muss `asadmin` gestartet werden. Das Vorgehen folgt dem von Lee [Lee10] beschriebenen Verfahren:

```
create-protocol --securityenabled=false http-redirect
create-protocol-filter --protocol http-redirect --classname ↩
   com.sun.grizzly.config.HttpRedirectFilter redirect-filter
create-protocol --securityenabled=false pu-protocol
create-protocol-finder --protocol pu-protocol ↩
   --targetprotocol http-listener-2 --classname ↩
   com.sun.grizzly.config.HttpProtocolFinder http-finder
create-protocol-finder --protocol pu-protocol ↩
   --targetprotocol http-redirect --classname ↩
   com.sun.grizzly.config.HttpProtocolFinder http-redirect
set configs.config.server-config.network-config. ↩
    network-listeners.network-listener. ↩
    http-listener-1.protocol=pu-protocol
```

Es muss nun erneut `DAS`, wie in Abschnitt A.3.2 beschrieben, aufgerufen werden. Nach dem Einloggen als GlassFish-interner Administrator muss ein Parameter zu den JVM-Einstellungen hinzugefügt werden. Dazu wird in der linken Seitenleiste zu „Configurations“, dem Unterpunkt „server-config“ und dort „JVM-settings“ navigiert. Anschließend muss im Reiter „JVM Options“ auf „Add JVM Option“ geklickt und der Parameter „`-Dfile.encoding=UTF8`“ eingetragen werden. Dieser Vorgang wird mit einem Klick auf „Save“ gespeichert.

A.7.2 Installation der Datenbank

Damit die Datenbank später von der Suchmaschine genutzt werden kann, müssen eine JDBC Resource, ein Connection Pool sowie ein JDBC Realm erstellt werden (siehe Abschnitt 5.1.3 und Abschnitt 5.4.3). Dazu wird, wie in in Abschnitt A.3.2 be-

schrieben, `DAS` gestartet. Unter „Resources“ in der Navigationsleiste links befindet sich der Eintrag „JDBC Connection Pools“. Darunter muss auf „New…“ geklickt werden. Die Werte sind dabei aus der nachfolgenden Tabelle zu übernehmen:

Pool Name	`PolsearchinePool`
Resource Type	`javax.sql.DataSource`
Database Driver Vendor	`JavaDB-30`

Nach dem Klick auf „Next“ müssen einige „Additional Properties“ anhand der anschließenden Tabelle angepasst werden:

driverClass	`org.apache.derby.jdbc.ClientDriver`
password	`useADifferentPassword`
portNumber	`1527`
databaseName	`polsearchine`
user	`polsearchineDB`
serverName	`localhost`
securityMechanism	`3`
ssl	`off`

Wurden bei der Erstellung der Datenbank (siehe Abschnitt A.4) nicht die Beispielwerte gewählt, müssen die Änderungen analog dazu übernommen werden.

In der linken Navigationsleiste muss nun auf „JDBC Resources“ geklickt werden. Dort wird „New…“ ausgewählt. Die Werte sollten mit denen aus der nachfolgenden Tabelle übereinstimmen:

JNDI Name	`jdbc/Polsearchine`
Pool Name	`PolsearchinePool`

Abschließend wird mit einem Klick auf „OK“ gespeichert. Um einen neuen JDBC Realm zu erstellen, muss in der Navigationsleiste von „Configurations“ nach „server-config“, „Security“ bis hin zu „Realms“ navigiert werden. Dort muss erneut auf „New…“ geklickt und anhand der folgenden Tabelle der Realm erstellt werden:

Name	`polsearchineRealm`
Class Name	`com.sun.enterprise.security. ↩ auth.realm.jdbc.JDBCRealm`
JAAS Context	`jdbcRealm`
JNDI	`jdbc/Polsearchine`
User Table	`POLSEARCHINEUSER`

User Name Column	NAME
Password Column	PASSWORDDIGEST
Group Table	USER_GROUP
Group Table User Name Column	USERS_NAME
Group Name Column	GROUPS_NAME
Database User	polsearchineDB
Database Password	useADifferentPassword
Digest Algorithm	SHA-256
Password Encryption Algorithm	SHA-256
Encoding	Hex

Auch hier ist es wichtig zu beachten, dass die Werte nur korrekt sind, wenn exakt nach dem Beispiel in Abschnitt A.4 vorgegangen wurde. Abschließend wird dieser Vorgang durch Klicken auf „Save“ beendet. Die Datenbank ist nun für diese Domäne nutzbar.

A.7.3 Installation von Polsearchine

Polsearchine benötigt zwei Ordner. Deren Dateipfade werden durch die globalen Anwendungsvariablen nach Anforderung A10.1 und Anforderung A4.1.1 festgelegt. In diesem Unterabschnitt wird davon ausgegangen, dass die Anwendungsvariablen jeweils einen Unterordner innerhalb des Verzeichnisses der Domäne beschreiben. Diese Ordner müssen manuell erstellt werden:

```
1 cd /opt/glassfish3/glassfish/domains/polsearchine/
2 mkdir legalText owl
```

Es ist wichtig, dass die Domäne später so gestartet wird, dass sie Schreib- und Lesezugriff auf den Ordner `owl/` und Lesezugriff auf den Ordner `legalText/` besitzt (siehe Abschnitt 4.2.1). Auf diesen Ordnern arbeitet später Polsearchine. Im Ordner `legalText/` muss mindestens eine Datei existieren, deren Name identisch mit dem genutzten Suchmaschinen-Parameter ist. Standardmäßig ist dies „`bing`“. Initial kann sie leer erstellt werden:

```
1 touch legalText/bing
```

Bevor Polsearchine öffentlich eingesetzt wird, sollte die Datei einen Rechtstext enthalten. Dieser muss die Rechtslage bezüglich der Nutzung der Suchmaschinen-API erläutern.

Als Nächstes muss das Polsearchine Enterprise Archive aus dem SVN[11] heruntergeladen und zur Domäne hinzugefügt werden. Der nachfolgende Befehl geht davon aus, dass der Nutzer sich im selben Pfad befindet, in dem das Enterprise Archive liegt:

```
1 mv Polsearchine.ear \
2     /opt/glassfish3/glassfish/domains/polsearchine/ ↩
      autodeploy/.
```

Bevor Polsearchine einsatzbereit ist, muss eine Datenbank erstellt (siehe Abschnitt A.4) und zu der Domäne hinzugefügt (siehe Abschnitt A.7.2) worden sein. Damit sich Nutzer später anmelden können, müssen die Rechtegruppen der Datenbank der Suchmaschine zugänglich gemacht werden. Hierzu wird `asadmin` (siehe Abschnitt A.3.1) gestartet und folgender Befehl ausgeführt:

```
1 set server-config.security-service.activate-default- ↩
      principal-to-role-mapping=true
```

Abschließend sollte Polsearchine, nach dem Neustarten der Domäne in `asadmin`, eingerichtet sein:

```
1 restart-domain polsearchine
```

Durch Aufrufen der externen IP-Adresse des Host-Systems in einem Webbrowser ohne Angabe eines Ports sollte Polsearchine erreichbar sein. Ist dies nicht der Fall, muss das Enterprise Archive manuell eingebunden werden. Dafür wird `DAS` (siehe Abschnitt A.3.2) gestartet und „Applications“ in der Navigation links ausgewählt. Ist die Liste leer, muss auf „Deploy…“ geklickt und das Enterprise Archive manuell ausgewählt werden. Mit „OK“ wird diese Auswahl bestätigt. Sollte das manuelle Einbinden nicht funktionieren, zeigt `DAS` Fehlermeldungen. Diese helfen bei der Lösung von etwaigen Problemen. Sie werden auch in den Serverlogs gespeichert. Spätestens jetzt sollte Polsearchine mit einem Webbrowser erreichbar sein.

[11] `https://svn.uni-koblenz.de/aggrimm/BA-Ruster/dist/`

Anhang B

Bildschirmfotos der Implementierung

In diesem Anhang befinden sich Bildschirmfotos der Implementierung. Abbildung B.1 zeigt die Startseite von Polsearchine. Wenn von dieser Seite aus eine Suche gestartet wird, wird nach Webseitenergebnissen gesucht. Dies wird über den dunkelblauen Reiter in der Kopfleiste angezeigt.

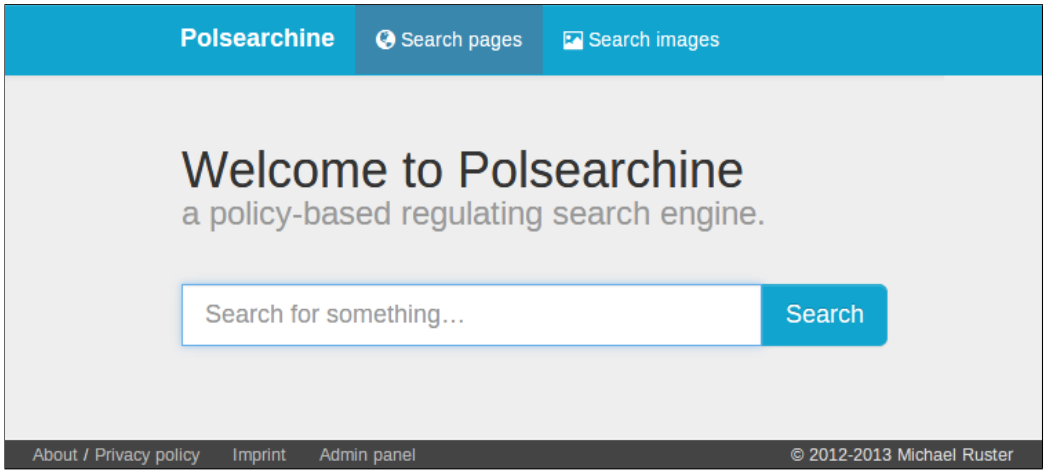

Abbildung B.1: Dieses Bildschirmfoto zeigt die Startseite von Polsearchine.

Eine Ergebnisseite für Webseitenergebnisse findet sich als Bildschirmfoto in Abbildung B.2. Die Grafik in der Kopfleiste „results by bing“ wird dynamisch anhand der globalen Anwendungsvariable für die aktive Suchmaschine gewählt.

Das Bildschirmfoto zeigt eine Suche nach „`stormfront.org`“, bei der die beiden unteren Ergebnisse gefiltert wurden. Die zweite Ergebniskarte stellt ein gefiltertes Ergebnis dar. Darunter befindet sich eine Ergebniskarte für ein gefiltertes Ergebnis, die zuvor angeklickt wurde. Dadurch werden die Hintergrundinformationen präsentiert, wie es in Abschnitt 4.3.4 konzipiert wurde.

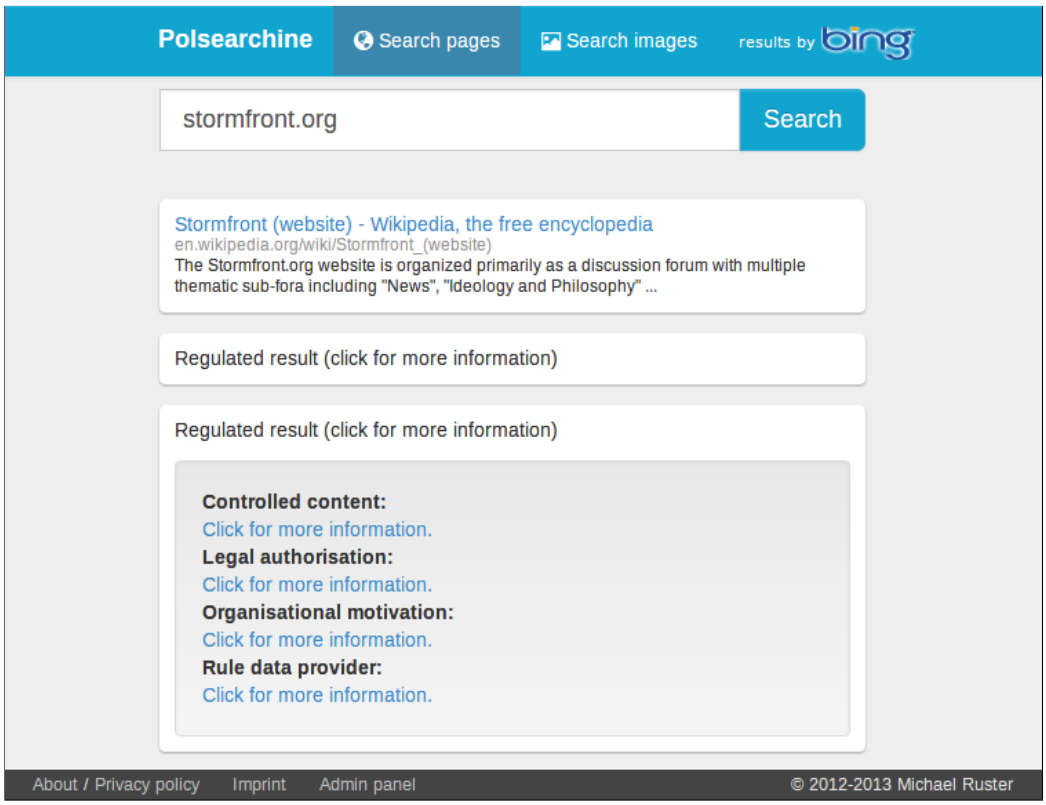

Abbildung B.2: Dieses Bildschirmfoto zeigt die Umsetzung der Darstellung von Webseitenergebnissen. Das Bildschirmfoto zeigt ein künstliches Szenario, dessen Ergebnisrückgabe Abbildung 4.5 stark ähnelt. Die erste Karte enthält ein dargestelltes Ergebnis und die anderen beiden jeweils ein gefiltertes. Dabei zeigt die dritte Karte wie ein gefiltertes Ergebnis dargestellt wird, wenn auf die Karte geklickt wurde.

Abbildung B.3 zeigt ein Bildschirmfoto einer Ergebnisseite für Bildergebnisse. Das Bildschirmfoto zeigt eine Suche nach „`uni koblenz`“. Zu Beispielzwecken wurden Bildergebnisse des Internetauftritts `http://www.uni-koblenz-landau.de` gefiltert. Damit die zusätzlichen Informationen zum zweiten Bild von links angezeigt werden, wurde der Mauszeiger über der Ergebniskarte platziert. Er ist auf dem Bildschimfoto nicht abgebildet.

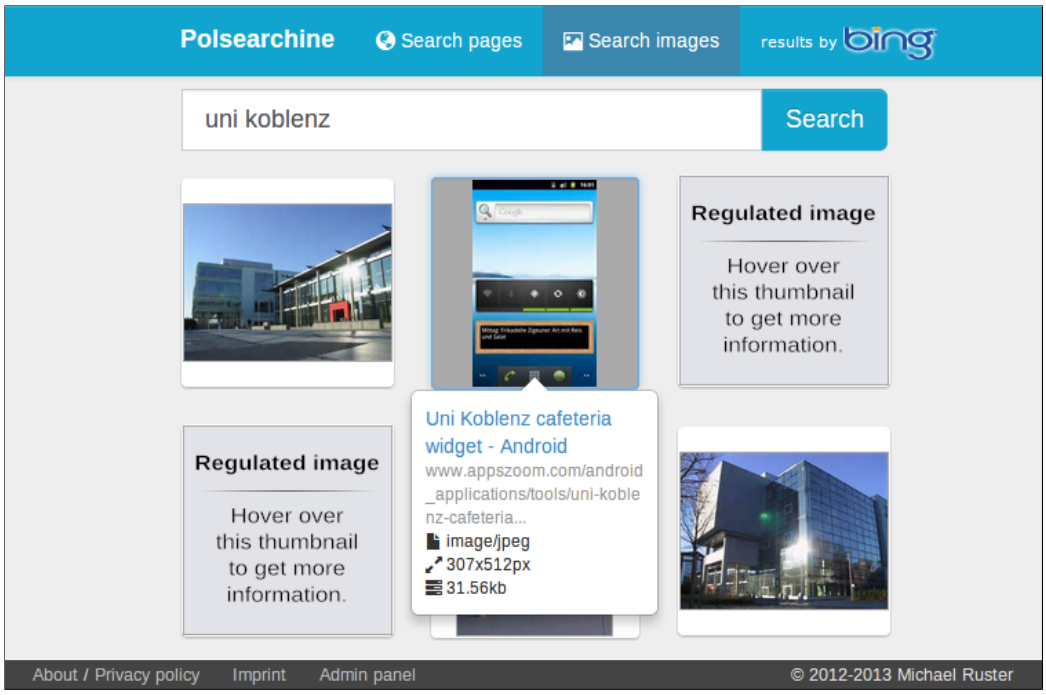

Abbildung B.3: Dieses Bildschirmfoto zeigt die Umsetzung der Darstellung von Bildergebnissen. Das Bildschirmfoto zeigt ein künstliches Szenario, dessen Ergebnisrückgabe Abbildung 4.6 stark ähnelt. Ein Mauszeiger ist dabei über der zweiten Ergebniskarte von links platziert. Dadurch werden die zusätzlichen Informationen zum dazugehörigen Bild angezeigt.

Bei Abbildung B.4 handelt es sich um ein bearbeitetes Bildschirmfoto. Es zeigt die vier in Abschnitt 4.3.1 und Abschnitt 4.3.4 beschriebenen Benachrichtigungen, die dem Endbenutzer in verschiedenen Situationen gezeigt werden. Die Grafik in der Meldung „Results are being loaded - please wait.“ ist animiert. Durch das Drehen der Pfeile soll dem Endbenutzer suggeriert werden, dass seine Suchanfrage verarbeitet wird.

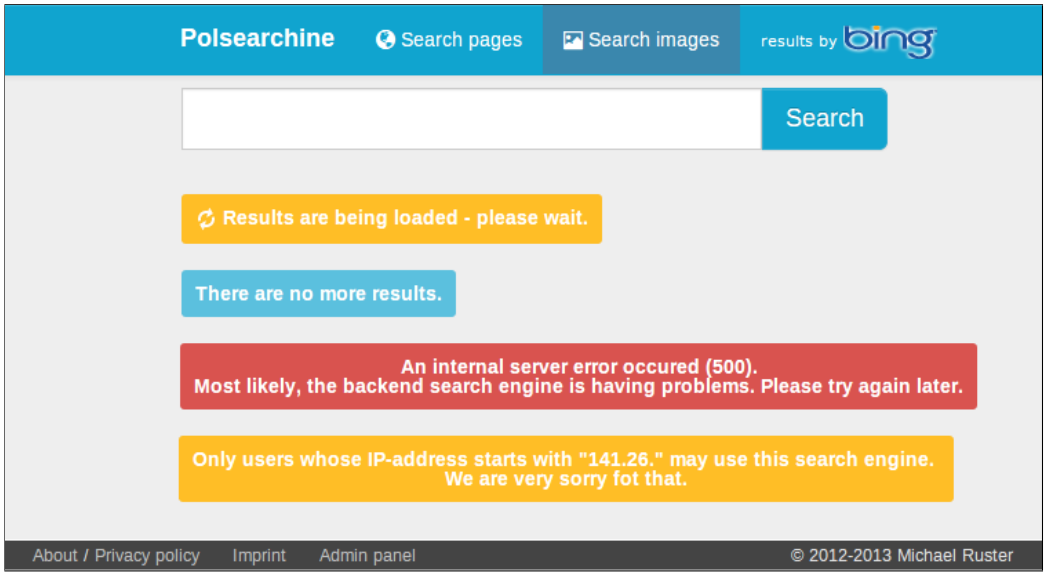

Abbildung B.4: Dieses bearbeitete Bildschirmfoto zeigt die Meldungen, die einem Endbenutzer zu unterschiedliche Situationen präsentiert werden. Die erste Meldung wird ihm angezeigt, wenn Ergebnisse geladen werden. Sollten keine weiteren Ergebnisse mehr existieren, wird dem Nutzer die zweite Meldung gezeigt. Die dritte Meldung wird gezeigt, wenn während der Ergebnisbeschaffung oder -verarbeitung ein Fehler aufgetreten ist. Wenn die IP-Adresse eines Nutzers durch das IP-Adressen-Muster ausgeschlossen wird, wird ihm die vierte Meldung angezeigt. Das IP-Adressen-Muster ist in diesem Bildschirmfoto „`141.26.`“.

Abbildung B.5 zeigt ein Bildschirmfoto des Backends. Dieses ist durch einen Klick auf „Admin panel“ in der Fußzeile erreichbar. Daraufhin wird der Endbenutzer nach Nutzername und Passwort gefragt. Wenn es sich um die Anmeldedaten eines Administratornutzers handelt, wird das Backend gezeigt. Ansonsten wird der Endbenutzer auf eine Webseite geleitet, die ihn auf eine fehlgeschlagene Anmeldung aufmerksam macht. Ein Administratornutzer kann im Backend die aktuelle Konfiguration einsehen. Darunter befindet sich eine Liste der gespeicherten Policy-Dateien. Beim Drücken des Knopfes mit der Aufschrift „Select policy…“

wird ein lokaler Dateibrowser geöffnet. Über diesen kann eine Policy-Datei ausgewählt und anschließend hochgeladen werden.

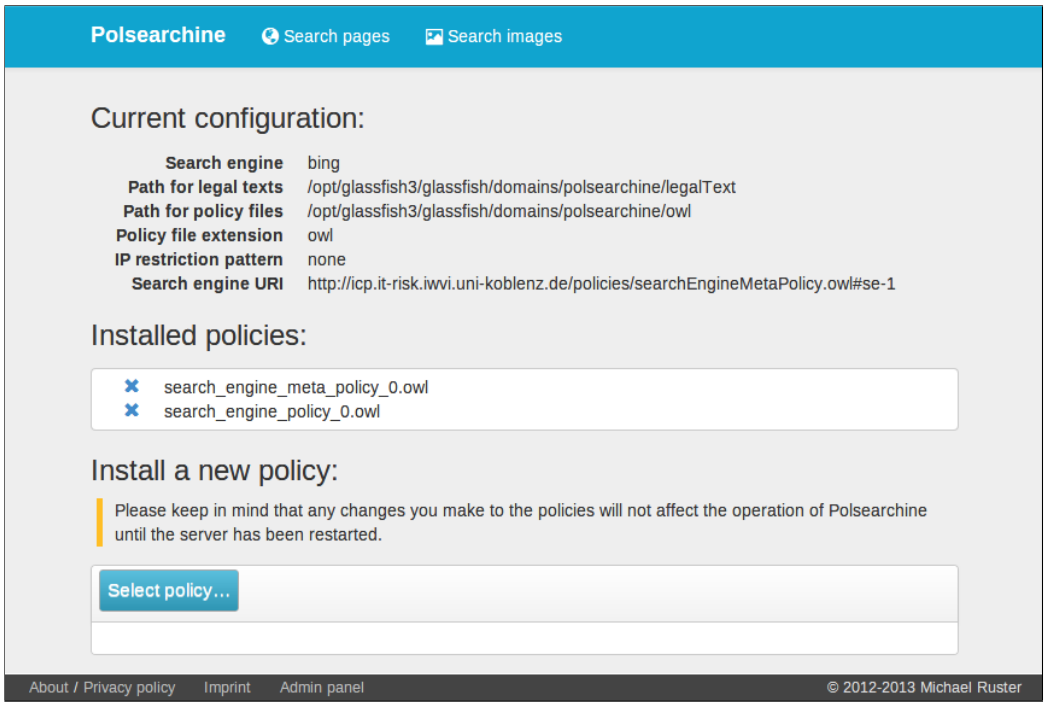

Abbildung B.5: Dieses Bildschirmfoto zeigt die Umsetzung des Backends. Unterhalb der Überschrift „Current configuration“ befinden sich die Werte der globalen Anwendungsvariablen. Darunter sind die Policy-Dateien im Policy-Dateipfad (siehe Abschnitt 4.1) dargestellt. Durch einen Klick auf das blaue „X“ können sie jeweils gelöscht werden. Unterhalb der Liste von gespeicherten Policy-Dateien befindet sich ein Knopf zum Hochladen weiterer Policy-Dateien.

Literaturverzeichnis

[ACGM+01] ARASU, Arvind ; CHO, Junghoo ; GARCIA-MOLINA, Hector ; PAEPCKE, Andreas ; RAGHAVAN, Sriram: Searching the web. In: *ACM Transactions on Internet Technology (TOIT)* 1 (2001), Nr. 1, S. 2–43

[Apa13] APACHE SOFTWARE FOUNDATION: *Step 2: ij Basics*. `http://db.apache.org/derby/papers/DerbyTut/ij_intro.html`, Februar 2013. – Online, letzter Zugriff 05.04.2013, 15:15 Uhr

[Bai10] BAIDU, INC.: *Baidu Online Marketing Service API*. `http://support.baidu.com/apien/?action=main`, 2010. – Online, letzter Zugriff 22.07.2013, 22:30 Uhr

[Bai11] BAIDU, INC.: *Baidu Online Marketing Services API Manaul*. `http://apihome.baidu.com/docs/index.php/En_sms_v2_sp_introduction_aboutThis/`, 2011. – Online, letzter Zugriff 22.07.2013, 22:30 Uhr

[Bai13] BAIDU, INC.: *What does the Baidu Online Marketing Services API offer*. `http://yingxiao.baidu.com/support/apien/detail_4755.html`, 2013. – Online, letzter Zugriff 22.07.2013, 23:00 Uhr

[BB05] BROPHY, Jan ; BAWDEN, David: Is Google enough? Comparison of an internet search engine with academic library resources. In: *Aslib Proceedings* Bd. 57 Emerald Group Publishing Limited, 2005, S. 498–512

[BGSS13] BALKE, Alexander ; GORBULSKI, Felix ; SCHENS, Artur ; SCHENS, Erwin: *Projektpraktikum Phi: Implementation eines InFO-konformen Application-Level Proxy-Servers*. 2013

[BHH+04] BECHHOFER, Sean ; HARMELEN, Frank v. ; HENDLER, Jim ; HORROCKS, Ian ; MCGUINNESS, Deborah L. ; PATEL-SCHNEIDER, Peter F. ; STEIN, Lynn A.: OWL web ontology language overview. In: *W3C Recommendation* 10 (2004)

[BLC95] BERNERS-LEE, Tim ; CONNOLLY, Dan: Hypertext Markup Language-2.0. 1995. – RFC 1866

[BLFM+05] BERNERS-LEE, Tim ; FIELDING, Roy T. ; MASINTER, Larry u. a.: Uniform Resource Identifier (URI): Generic Syntax. InternetEngineering Task Force, Januar 2005. – RFC 3986

[BP98] BRIN, Sergey ; PAGE, Lawrence: The anatomy of a large-scale hypertextual Web search engine. In: *Computer networks and ISDN systems* 30 (1998), Nr. 1, S. 107–117

[Bra97] BRADNER, Scott: Key words for use in RFCs to Indicate Requirement Levels. 1997. – RFC 2119

[CG07] CUTRELL, Edward ; GUAN, Zhiwei: What are you looking for?: an eye-tracking study of information usage in web search. In: *Proceedings of the SIGCHI conference on Human factors in computing systems* ACM, 2007, S. 407–416

[com13] COMSCORE, INC.: comScore Releases February 2013 U.S. Search Engine Rankings. (2013), März. `http://www.comscore.com/Insights/Press_Releases/2013/3/comScore_Releases_February_2013_U.S._Search_Engine_Rankings`. – Online, letzter Zugriff 11.04.2013, 15:45 Uhr

[Cri11] CRISP, Toby: Yandex: From Russia with Love. (2011), Juni. `http://www.comscore.com/Insights/Blog/Yandex_From_Russia_with_Love`. – Online, letzter Zugriff 10.04.2013, 19:45 Uhr

[CS+09] CHINNICI, Roberto ; SHANNON, Bill u. a.: JSR 316: Java™ Platform, Enterprise Edition 6 (Java EE 6) Specification. 2009. – Java Specification Request

[DAF+99] DIERKS, Tim ; ALLEN, Karlton Philip L. Christopher ; FREIER, Alan O. ; KOCHER, Paul C. u. a.: The TLS Protocol - Version 1.0. 1999. – RFC 2246

[DDL06] DESERTOT, Mikael ; DONSEZ, Didier ; LALANDA, Philippe: A dynamic service-oriented implementation for java ee servers. In: *Services Computing, 2006. SCC'06. IEEE International Conference on* IEEE, 2006, S. 159–166

[DH97] DREILINGER, Daniel ; HOWE, Adele E.: Experiences with selecting search engines using metasearch. In: *ACM Transactions on Information Systems (TOIS)* 15 (1997), Nr. 3, S. 195–222

[Dia08] DIAZ, Alejandro M.: Through the Google goggles: Sociopolitical bias in search engine design. In: *Web Search* (2008), S. 11–34

[DR08] DEIBERT, Ronald J. ; ROHOZINSKI, Rafal: Good for liberty, bad for security? Global civil society and the securitization of the Internet. In: *Access Denied: The Practice and Policy of Global Internet Filtering, ed. Ronald J. Deibert, John Palfrey, Rafal Rohozinski, and Jonathan Zittrain*. The MIT Press, 2008, S. 123–149

[DR10] DEIBERT, Ronald J. ; ROHOZINSKI, Rafal: Beyond Denial: introducing next-generation information access controls. In: DEIBERT, Ronald (Hrsg.) ; PALFREY, John (Hrsg.) ; ROHOZINSKI, Rafal (Hrsg.) ; ZITTRAIN, Jonathan (Hrsg.): *Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace*. The MIT Press, 2010, Kapitel 1, S. 3–13

[EHLM09] ERIXON, Fredrik ; HINDLEY, Brian ; LEE-MAKIYAMA, Hosuk: Protectionism Online: Internet Censorship and International Trade Law / ECIPE. 2009. – working paper

[FHBH+99] FRANKS, John ; HALLAM-BAKER, Phillip M. ; HOSTETLER, Jeffery L. ; LAWRENCE, Scott D. ; LEACH, Paul J. ; LUOTONEN, Ari ; STEWART, Lawrence C.: HTTP Authentication: Basic and Digest Access Authentication. 1999. – RFC 2617

[FV08] FARIS, Robert ; VILLENEUVE, Nart: Measuring global Internet filtering. In: *Access Denied: The practice and policy of global Internet filtering*. The MIT Press, 2008, S. 5–28

[G+09] GROUP, W3C Owl W. u. a.: OWL 2 web ontology language document overview. In: *W3C Recommendation* 27 (2009), S. 1205–1214

[GC07] GUAN, Zhiwei ; CUTRELL, Edward: An eye tracking study of the effect of target rank on web search. In: *Proceedings of the SIGCHI conference on Human factors in computing systems* ACM, 2007, S. 417–420

[Geo13] GEORGIA TECH RESEARCH CORPORATION: *GVU's 9th WWW Survey Results*. `http://www.cc.gatech.edu/gvu/user_surveys/survey-1998-04/`, 2013. – Online, letzter Zugriff 16.03.2013, 15:00 Uhr

[GJG04] GRANKA, Laura A. ; JOACHIMS, Thorsten ; GAY, Geri: Eye-tracking analysis of user behavior in WWW search. In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* ACM, 2004, S. 478–479

[Goo10] GOOGLE INC.: *Web Search API Terms of Service - Google Web Search API (Deprecated) —Google Developers*. `https://developers.google.com/web-search/terms`, 2010. – Online, letzter Zugriff 23.07.2013, 00:00 Uhr

[Goo12a] GOOGLE INC.: *Google Web Search API (Deprecated) — Google Developers*. `https://developers.google.com/web-search/`, 2012. – Online, letzter Zugriff 23.02.2013, 15:30 Uhr

[Goo12b] GOOGLE INC.: *Overview - Custom Search — Google Developers*. `https://developers.google.com/custom-search/v1/overview`, August 2012. – Online, letzter Zugriff 07.04.2013, 17:00 Uhr

[HR83] HAERDER, Theo ; REUTER, Andreas: Principles of transaction-oriented database recovery. In: *ACM Computing Surveys (CSUR)* 15 (1983), Nr. 4, S. 287–317

[HUHN01] HUANG, Lieming ; ULRICH, Thiel ; HEMMJE, Matthias ; NEUHOLD, Erich J.: Adaptively constructing the query interface for meta-search

engines. In: *Proceedings of the 6th international conference on Intelligent user interfaces* ACM, 2001, S. 97–100

[Int13] INTERNET ASSIGNED NUMBERS AUTHORITY: *Service Name and Transport Protocol Port Number Registry*. `http://www.iana.org/assignments/port-numbers`, April 2013. – Online, letzter Zugriff 03.04.2013, 14:30 Uhr

[JHW07] JUNG, Seikyung ; HERLOCKER, Jonathan L. ; WEBSTER, Janet: Click data as implicit relevance feedback in web search. In: *Information Processing & Management* 43 (2007), Nr. 3, S. 791–807

[JSK07] JANSEN, Bernard J. ; SPINK, Amanda ; KOSHMAN, Sherry: Web searcher interaction with the Dogpile. com metasearch engine. In: *Journal of the American Society for Information Science and Technology* 58 (2007), Nr. 5, S. 744–755

[Kas12] KASTEN, Andreas: *Flow Control Ontology —ICP Wiki*. `http://icp.it-risk.iwvi.uni-koblenz.de/wiki/Flow_Control_Ontology`, Oktober 2012. – Online, letzter Zugriff 17.07.2013, 17:00 Uhr

[Kas13] KASTEN, Andreas: *Search Engine-based Flow Control Ontology —ICP Wiki*. `http://icp.it-risk.iwvi.uni-koblenz.de/wiki/Search_Engine-based_Flow_Control_Ontology`, Januar 2013. – Online, letzter Zugriff 17.07.2013, 17:00 Uhr

[KR09] KASSIM, Junaidah M. ; RAHMANY, Mahathir: Introduction to Semantic Search Engine. In: *Electrical Engineering and Informatics, 2009. ICEEI'09. International Conference on* Bd. 2 IEEE, 2009, S. 380–386

[KS12] KASTEN, Andreas ; SCHERP, Ansgar: *A pattern system for information flow control on the Internet*. 2012. – unveröffentlicht

[KT00] KOBAYASHI, Mei ; TAKEDA, Koichi: Information retrieval on the web. In: *ACM Comput. Surv.* 32 (2000), Juni, Nr. 2, S. 144–173

[Lee10] LEE, Justin: *Port Unification in GlassFish 3 - Part 1 — antwerkz*. `http://antwerkz.com/port-unification-in-`

glassfish-3-part-1/, Juni 2010. – Online, letzter Zugriff 27.03.2013, 22:30 Uhr

[Lew05] LEWANDOWSKI, Dirk: *Web Information Retrieval: Technologien zur Informationssuche im Internet*, Heinrich Heine University Düsseldorf, Diss., 2005. – 1–248 S.

[LHB+08] LORIGO, Lori ; HARIDASAN, Maya ; BRYNJARSDÓTTIR, Hrönn ; XIA, Ling ; JOACHIMS, Thorsten ; GAY, Geri ; GRANKA, Laura ; PELLACINI, Fabio ; PAN, Bing: Eye tracking and online search: Lessons learned and challenges ahead. In: *Journal of the American Society for Information Science and Technology* 59 (2008), Nr. 7, S. 1041–1052

[LLG98] LAWRENCE, Steve ; LEE GILES, C: Inquirus, the NECI meta search engine. In: *Computer networks and ISDN systems* 30 (1998), Nr. 1, S. 95–105

[Mic10] MICROSOFT CORPORATION: *Bing Developer FAQs*. http://download.microsoft.com/download/1/A/2/1A24833E-E774-4D27-9BA0-94BA077BB84E/DeveloperFAQs.pdf, 2010. – Online, letzter Zugriff 24.05.2013, 15:50 Uhr

[Mic12a] MICROSOFT CORPORATION: *Bing Search API | Windows Azure Marketplace*. http://datamarket.azure.com/dataset/bing/search, 2012. – Online, letzter Zugriff 23.02.2013, 15:30 Uhr

[Mic12b] MICROSOFT CORPORATION: *Bing Search API Terms Of Use*. https://datamarket.azure.com/dataset/5BA839F1-12CE-4CCE-BF57-A49D98D29A44#terms, April 2012. – Online, letzter Zugriff 10.04.2013, 17:15 Uhr

[Mic13] MICROSOFT CORPORATION: *Schema Tabular Documentation for the Bing Search API*. http://go.microsoft.com/fwlink/?LinkID=272627, Januar 2013. – Online, letzter Zugriff 26.04.2013, 22:15 Uhr

[MJ08] MANOJ, M ; JACOB, Elizabeth: Information retrieval on Internet using meta-search engines: A review. In: *Journal of scientific & industrial research* 67 (2008), S. 739–746

[Mos12] MOSEN, Dominik: Erweiterung eines Nameservers zur policy-basierten Internetregulierung / Universität Koblenz-Landau. 2012. – Bachelorarbeit

[Nat98] NATHENSON, Ira S.: Internet infoglut and invisible ink: Spamdexing search engines with meta tags. In: *Harv. JL & Tech.* 12 (1998), S. 43

[Ora10] ORACLE CORPORATION: *Introduction to Java Platform, Enterprise Edition 6*, April 2010. `http://www.oracle.com/us/products/middleware/application-server/050871.pdf`. – Online, letzter Zugriff 26.04.2013 23:00 Uhr

[Ora12] ORACLE CORPORATION: *Your First Cup: An Introduction to the Java™ EE Platform*, April 2012. `http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf`. – Online, letzter Zugriff 15.04.2013, 23:45 Uhr

[Ora13] ORACLE CORPORATION: *The Java EE 6 Tutorial*, Januar 2013. `http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf`. – Online, letzter Zugriff 25.04.2013 14:30 Uhr

[PD05] PESHAVE, Monica ; DEZHGOSHA, Kamyar: *How search engines work and a web crawler application*. `http://www.micsymposium.org/mics_2005/papers/paper89.pdf`, 2005. – Online, letzter Zugriff 26.04.2013, 22:00 Uhr

[Riv10] RIVERA, Rafael: *Use Bing API and MSDN metadata to generate code automagically (Part 1)*. `http://withinwindows.com/within-windows/2010/10/16/use-bing-api-and-msdn-metadata-to-generate-code-automagically-part-1`, Oktober 2010. – Online, letzter Zugriff 24.05.2013, 14:30 Uhr

[RLHJ+99] RAGGETT, Dave ; LE HORS, Arnaud ; JACOBS, Ian u. a.: HTML 4.01 Specification. In: *W3C recommendation* 24 (1999)

[Sar03] SARR, Mansour: *Improving precision and recall using a spell checker in a search engine*, KTH Royal Institute of Technology, Diplomarbeit, 2003

[SBEW01] STEINER, Michael ; BUHLER, Peter ; EIRICH, Thomas ; WAIDNER, Michael: Secure password-based cipher suite for TLS. In: *ACM Transactions on Information and System Security* 4 (2001), Nr. 2, S. 134–157

[SE97] SELBERG, Erik ; ETZIONI, Oren: The MetaCrawler architecture for resource aggregation on the Web. In: *IEEE expert* 12 (1997), Nr. 1, S. 11–14

[Sul13] SULLIVAN, Danny: *Google Still World's Most Popular Search Engine By Far, But Share Of Unique Searchers Dips Slightly*. `http://searchengineland.com/google-worlds-most-popular-search-engine-148089`, Februar 2013. – Online, letzter Zugriff 10.04.2013, 21:15 Uhr

[SWP04] SUA, Kui C. ; WALDREN, Steve E. ; PATRICK, Timothy B.: Differences in the effects of filters on health information retrieval from the internet in three languages from three countries: A comparative study. In: *Proceedings of MEDINFO* 2004 (2004)

[TK06] TOYODA, Masashi ; KITSUREGAWA, Masaru: What's really new on the web?: identifying new pages from a series of unstable web snapshots. In: *Proceedings of the 15th international conference on World Wide Web* ACM, 2006, S. 233–241

[Ung95] UNGER, Stephen H.: Hazards, critical races, and metastability. In: *Computers, IEEE Transactions on* 44 (1995), Nr. 6, S. 754–768

[Vau04] VAUGHAN, Liwen: New measurements for search engine evaluation proposed and tested. In: *Information Processing & Management* 40 (2004), Nr. 4, S. 677–691

[Wil10] WILLIAMSON, Victor L.: Goal-oriented Web search / Massachusetts Institute of Technology. 2010. – Dissertation

[Yah12a] YAHOO! INC.: *Yahoo! BOSS Services Terms of Use*. `http://info.yahoo.com/legal/us/yahoo/boss/tou/`, September 2012. – Online, letzter Zugriff 11.04.2013, 16:30 Uhr

[Yah12b] YAHOO! INC.: *Yahoo! BOSS – Pricing*. http://info.yahoo.com/legal/us/yahoo/boss/pricing/, November 2012. – Online, letzter Zugriff 11.04.2013, 17:00 Uhr

[Yah13] YAHOO! INC.: *Yahoo! Search BOSS - YDN*. http://developer.yahoo.com/boss/search/, 2013. – Online, letzter Zugriff 23.02.2013, 15:30 Uhr

[Yan06] YANG, Tao: *Large scale internet search at ask.com*. http://www.stanford.edu/group/mmds/slides/yang-mmds.pdf, 2006. – Online, letzter Zugriff 26.04.2013, 22:00 Uhr – Präsentation

[Yan12a] YANDEX LLC: *Fact sheet*. http://download.yandex.ru/company/Fact_Sheet_Q4_2012.pdf, 2012. – Online, letzter Zugriff 09.04.2013, 16:00 Uhr

[Yan12b] YANDEX LLC: *User Agreement for Yandex Services*. http://legal.yandex.com/rules/, April 2012. – Online, letzter Zugriff 20.05.2013, 23:00 Uhr

[Yan12c] YANDEX LLC: *Yandex Terms of Service — Legal documents*. http://legal.yandex.com/termsofservice/, September 2012. – Online, letzter Zugriff 20.05.2013, 23:30 Uhr

[Yan12d] YANDEX LLC: *Yandex.XML (Web Search API) Terms of Service*. http://legal.yandex.com/xml/, Oktober 2012. – Online, letzter Zugriff 20.05.2013, 23:00 Uhr

[Yan12e] YANDEX LLC: *Yandex.XML (Web Search API) Terms of Service*. http://legal.yandex.com/privacy/, September 2012. – Online, letzter Zugriff 20.05.2013, 23:15 Uhr

[Yan13] YANDEX LLC: *Yandex.XML. Developer's guide. Version 1.1*. http://api.yandex.com/xml/doc/dg/yandex-xml-dg.pdf, April 2013. – Online, letzter Zugriff 09.04.2013, 17:00 Uhr

[Ye07] YE, Zhenghua: An exploratory study of search advertising in China. 2007. – Dissertation

[ZP08] ZITTRAIN, Jonathan ; PALFREY, John: Internet filtering: The politics and mechanisms of control. In: DEIBERT, Ronald (Hrsg.) ; PALFREY, John (Hrsg.) ; ROHOZINSKI, Rafal (Hrsg.) ; ZITTRAIN, Jonathan (Hrsg.): *Access Denied: The Practice and Policy of Global Internet Filtering*. The MIT Press, 2008, S. 29–57

[ZRQ+05] ZHOU, Yilu ; REID, Edna ; QIN, Jialun ; CHEN, Hsinchun ; LAI, Guanpi: US domestic extremist groups on the Web: link and content analysis. In: *Intelligent Systems, IEEE* 20 (2005), Nr. 5, S. 44–51