# Web services for automated generation of focused Open Corpus Physical and Virtual Caches

Peter Hannon

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

2010

# Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Peter Hannon

September 2010

# Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____
Peter Hannon
September 2010

# Summary

There is a huge need for the ability to generate focused content caches, i.e. repositories of topic specific information. Focused content caches that can be generated and are accessible to the world wide web have the potential for use by a wide audience and have applications in many industries including eLearning and Tourism.

This research investigates the current state of the art in techniques and technologies of web Information Retrieval (IR). There are several open source tools and tool chains which could be utilised for the generation of focused content caches, however none offer the full "end to end" suite of services that are necessary and are also web accessible. The most promising candidate for use is the Open Corpus Content Service (OCCS) tool chain. However use of the OCCS is restricted to those with access to a particular server. The tool chain is invoked by executing Perl scripts after manually modifying configuration files. The system components are highly coupled and inaccessible to the web.

The Focused Content Generation System (FCGS) was developed to extend the OCCS and expose it to the world wide web. The FCGS web services are self configuring and allow intelligent management. The FCGS extends the OCCS to allow the generation of a repository of virtual links instead of physical content. This reduces the amount of storage space necessary as well as sidestepping any potential legal issues that arise as a result of hosting or refactoring proprietary content.

An evaluation of the FCGS is presented, which took the form of the development of a demonstrator client application, and the running of user and performance tests. The demonstrator client application verified that the FCGS web services were fit for use to generate focused content caches. The success of the demonstrator and performance tests clearly demonstrates the feasibility of providing a system for the generation of focused content caches that is web accessible, that is tunable and customisable and that provides interfaces for monitoring and management. The feedback from the user test (in the form of a questionnaire) indicates that the data and invocation requirements of FCGS are easily understood, the error handling and messages produced by the FCGS are informative, the FCGS web services perform in a timely manner and are fit for purpose and finally the management functionality provided by the FCGS is sufficient.

# Acknowledgements

First of all, I would like to thank my supervisor, Prof. Vincent Wade, whose vast knowledge and experience, patience and encouragement has made this work possible. I would also like to thank Séamus Lawless whose input to this thesis was invaluable.

I would like to thank my family. My parents, Gay and J.B. for their love and guidance and who encouraged me to embark on the journey of the Masters that this thesis is the culmination of. My girlfriend Alessia, whose patience and encouragement got me through a difficult year when free time was very hard to come by. My sisters Claire and Laura for being there and providing excellent examples of academic excellence for me to aspire to.

I would also like to extend my gratitude to the members of the Knowledge and Data Engineering Group (KDEG) in Trinity College Dublin. The contributions Stephen Curran, Ian O'Keeffe, Dominic Jones and Alex O'Connor made to this thesis are very much appreciated. Finally, thanks are also due to my Networks and Distributed Systems classmates.

You all have my sincerest gratitude.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

**AJAX**
Asynchronous JavaScript and XML.

**ANN**
Artificial Neural Network.

**API**
Application Programming Interface.

**BSD**
Berkeley Software Distribution.

**CNGL**
Centre for Next Generation Localisation.

**CSS**
Cascading Style Sheets.

**DNS**
Domain Name System.

**EE**
Enterprise Edition.

**FCGS**
Focused Content Generation System.

**GiB**
Gibibyte.

**GPL**
General Public License.

**HITS**
Hyperlink-Induced Topic Search.

**HTML**
HyperText Markup Language.

**HTTP**
: HyperText Transfer Protocol.

**idf**
: inverse document frequency.

**IR**
: Information Retrieval.

**ISC**
: Internet Systems Consortium.

**JAX-WS**
: Java API for XML Web Services.

**JMX**
: Java Management Extensions.

**JSON**
: JavaScript Object Notation.

**JTCL**
: Java Text Categorizing Library.

**JVM**
: Java Virtual Machine.

**KDEG**
: Knowledge and Data Engineering Group.

**LGPL**
: Lesser General Public License.

**OCCS**
: Open Corpus Content Service.

**ODP**
: Open Directory Project.

**P2P**
: Peer-to-peer.

**PDF**
: Portable Document Format.

**RDF**
: Resource Description Framework.

**RDV**
: Representative Document Vector.

**REST**
: Representational State Transfer.

**SOA**
: Service Oriented Architecture.

**SOAP**
: Simple Object Access Protocol.

**SQL**
: Structured Query Language.

**SVM**
: Support Vector Machine.

**SWD**
: Semantic Web Document.

**tf**
: term frequency.

**tf-idf**
: term frequency–inverse document frequency.

**URI**
: Uniform Resource Identifier.

**URL**
: Uniform Resource Locator.

**WAR**
: Web application ARchive.

**WSDL**
: Web Services Description Language.

**WTMS**
: Web Topic Management System.

**XML**
: eXtensible Markup Language.

# Chapter 1

# Introduction

## 1.1 Motivation

Current IR tool chains are tightly coupled and for the most part inaccessible for widespread use. They are typically made up of standalone applications, which require significant amounts of manual configuration and intervention at multiple points in the execution of the tool chain [42]. These tool chains are typically initiated by an HyperText Markup Language (HTML) or command-line interface after several crawling, indexing and classification parameters have been set and appropriate training data has been provided. These tool chains by their very nature as collections of standalone applications are inflexible. The configuration and tuning and as well initiation and management of an IR tool chain for generation of a focused corpus is for the most part only available in a very closed environment, to a select few with the required expertise and privileges. Access to the resultant information cache is also somewhat restricted and not easily consumed or refactored for use on the web. There is a clear limitation in the existing state of the art of IR tool chains. To be of widespread use these tool chains need to be loosely coupled, and to be easily accessible from anywhere on the web. With this accessibility should come the ability to configure, tune, and manage each separate component of the tool chain as necessary. With decoupling of the tool chain would come the ability to plug in and out different tools and so customise the tool chain for different IR tasks.

## 1.2 Research and Objectives

The research question posed in this thesis is: *(i) How can an FCGS be developed with loosely coupled components, that is exposed as web services, to allow intelligent management for auto-configuration, allowing each component to be invocable, tunable and customisable and (ii) how can such a system be engineered to generate a virtual cache of linked content, rather than actual harvested content.*

### 1.2.1 Research Objectives

Answering the research question can be broken down into the following objectives:

1. Investigate the current state of the art in techniques and technologies of web IR.
2. Analyse and determine a full set of requirements and inter-dependencies for exposing each component of an FCGS as web services.
3. Research and develop prototype web services to demonstrate the feasibility of developing an FCGS using Open Source software.

  - The web services should be self configuring and allow intelligent management.
4. Investigate performance and integration issues that arise in the building of these web services.
5. Determine the feasibility of generating a repository of virtual links instead of physical content.

### 1.2.2 Research Contribution

This thesis will make a significant contribution to the state of the art with the research and development of the FCGS. The FCGS web services have the potential to be used by wide audience of content consumers. The accessibility of these web services enables the general public, businesses and educational institutions to generate and access caches of focused topic content as they see fit. The potential to refactor and redistribute the information is also increased.

The FCGS provides web service interfaces that allow auto-configuration and intelligent management of the component services. The FCGS takes care of the underlying configuration of the service components, allowing the user to provide the minimum of parameters necessary to generate a focused content cache. The FCGS maintains and verifies the state of the individual service components, ensuring the services are invoked in the appropriate order. The FCGS has the ability to generate virtual caches instead of storing the actual harvested content. This reduces storage costs and also sidesteps the potential legal issues of storing proprietary content.

This thesis also contributes a significant amount of research into the state of the art of Open Source Web IR tool chains that allow the generation of focused content.

## 1.3 Technical Approach and Thesis Outline

The technical approach taken was as follows:

- The state of the art in crawling, classification, indexing, and IR tool chains was researched.
- Web services appropriate for requirements and suitable for evaluation were designed and built.
- A demonstrator client application which invokes the web services to generate a focused content cache was developed.
- A user test was conducted in order to evaluate the:
  - Usability of the web services.
  - Manageability of the web services.
  - Efficiency and efficacy of the web services.
- Performance tests of the web services were run and analysed.

The generation of a focused content cache requires three key tasks to be performed: crawling, document classification, and indexing. A review of the state of the art in these three areas was undertaken, and is presented in Chapter 2. This review discusses the main concepts, algorithms and techniques associated with these tasks. It is followed by a survey of the available open source tools and tool chains that can be used to accomplish these tasks in Chapter 3. The tools and tool chains are analysed and compared to determine their applicability for generating a focused content cache.

The FCGS web services are designed and built to meet the research objectives. The design of the FCGS is described in Chapter 4 and the implementation and specification can be found in Chapter 5. The FCGS was evaluated by developing a demonstrator application and conducting user and performance tests. The user test required the participants to build a client application having been

given access to the web services, the demonstrator application and service Application Programming Interface (API) documentation. The participants were then required to complete a questionnaire to determine their opinion of the usability, efficacy, efficiency and manageability of the services. They also had the opportunity to comment on features they valued as well as suggest improvements. Performance tests were run on each of the web services in order to get a baseline of the performance of the FCGS. Details of the demonstrator application as well as the results and analysis of the user and performance tests are presented in Chapter 6.

This thesis concludes with Chapter 7, which discusses the objectives and achievements of this research, the key contributions to the state of the art of focused content cache generation as well as future work that could be undertaken.

# Chapter 2

# Crawling, Classification and Indexing

## 2.1 Introduction

Examining the techniques and technologies for the generation of a focused content caches cannot be addressed without first exploring IR on the world wide web. The purpose of this chapter is to inform the reader of the techniques for discovering, classifying and harvesting content for topic specific repositories. The chapter will first describe some of the prominent Web IR algorithms. That will be followed by Section 2.3 which describes crawling, and in particular focused crawling which allows the discovery of topic specific content. Paramount to the generation of focused content is a means of determining the relevancy of a downloaded document, document classification will be discussed in Section 2.4. Finally the content of downloaded documents needs to be represented in a way to allow in conjunction with a user query the location and ranking of relevant content. The process of generating a document representative to facilitate search and retrieval (also known as indexing) and models for retrieval will be discussed in Section 2.5.

## 2.2 Web IR Algorithms

The effectiveness of IR is often measured using two terms: *precision* and *recall*. Precision is the number of relevant documents retrieved divided by the total number of documents retrieved. Recall is the number of relevant documents retrieved divided the total number of relevant documents (whether they were retrieved or not). Both of these measures can be said to be subjective, as determining if a document is relevant requires knowledge of the user's intent when they made the query in the first place. Recall as a measure is not useful in the context of the web, as it requires knowledge of all of the relevant documents in advance of the query. As the web is constantly growing with new content being generated all the time, this cannot be known. Web-based IR algorithms typically employ some sort of link analysis algorithm to effectively rank web pages. Garfield's impact factor [34], which defines a measure of how impactful an academic or scientific journal is, can be determined by computing the average amount of citations papers published in the journal have received in the past two years. This concept is a key foundation of the web page ranking algorithms in use today. Pinski and Narin [48] suggested that a citation from a more influential paper should be weighted more, this concept can also be extended to web page ranking algorithms.

### 2.2.1 HITS

Hyperlink-Induced Topic Search (HITS) is a link analysis algorithm for rating web pages, and uses the Hubs and Authorities [40] theory. Hubs are web pages that serve as directories and link directly to authoritative pages. A good Hub will link to many Authorities, and a good Authority will be linked to by many different hubs. The HITS algorithm operates as follows. A focused subgraph of the web is generated from the results of a text based search engine for a given query. The subgraph is then enlarged by adding the links in each page (up to certain threshold) of the subgraph. The subgraph is then pruned by removal of *intrinsic* links, that is links within the same domain name (as they are often purely navigational). Thus leaving the *transverse* links, that is the links across different domain names. Next the Hub and Authority values are computed for each page in the subgraph. To begin each page is given a Hub value and an Authority value of 1. Then the following 4 steps are followed:

1. Each page's Authority value is refreshed to be equal to the sum of the Hub values of all the pages linking to it.
2. Each page's Hub value is refreshed to be equal to the sum of the Authority values of each page it links to.
3. The Hub and Authority values are normalised by dividing each Hub value by the sum of the squares of all the Hub values and dividing each Authority value by the sum of the squares of all the Authority values.
4. The 3 previous steps are repeated as necessary.

### 2.2.2 PageRank

Unlike HITS, the PageRank [47] algorithm is run at index time, and extends the Hubs and Authorities concept. A link from page A to page B is counted as a vote for page B by page A. The importance of the page casting the vote is also taken into account; a vote by a more important page is given more weight. PageRank forms the basis of the Google search engine, and as time has gone on various other factors have been incorporated into the algorithm, but Google does not reveal specific details in order to preserve their market lead and also to prevent manipulation of the search results.

The PageRank algorithm is a probability distribution which represents how likely it is that a person surfing the web randomly traversing links will arrive at any particular page. The PageRank for a given page is a probability value between 0 and 1 indicating how likely it is to arrive at the page via a random link. Any page which links to a given page contributes their PageRank value divided by the normalised[1] number of links on each linking page. The contributions from all linking pages are summed to determine a PageRank for a given page. The general form can be specified as:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

The PageRank for page $u$ is related to the PageRanks of each page $v$ in the set $B_u$ divided by the number of $L(v)$ from links from page $v$. $B_u$ is the set of all pages that link to page $u$.

The simplified form of the algorithm detailed above was enhanced with the addition of a damping factor. This models the fact that the random surfer will eventually get bored and stop traversing links. An acceptable value for the damping factor is a probability of 0.85.

[1] Each specific Uniform Resource Locator (URL) is counted only once.

The revised formula can be seen below:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

$p_i, \ldots, p_N$ are the pages being considered. The set $M(p_i)$ contains all pages linking to $p_i$. $L(p_j)$ is the number of out-links (outward hyperlinks) on page $p_j$ and $N$ is the total number of pages. From the formula above you can see that the damping factor $d$ has the effect of decreasing a given page's PageRank. It should be noted that pages with no out-links are assumed to link out to all other pages in the collection, their PageRank values shared equally amongst all other pages.

## 2.3 Crawling

Web search engines such as Google, attempt to do a breadth-first crawl, i.e. traverse the entire web, download each page and index it for use in later queries. However there are parts of the *deep web*[2] that are not accessible, and cannot be indexed by search engines. In 2004 it was estimated that the deep web is on the scale of 307,000 sites, 450,000 databases, and 1,258,000 interfaces [37]. About a third of this was indexed by the major search engines.

A search engine typically uses a web crawler to traverse the web in order to discover new and updated web pages. Web pages that are retrieved by the crawler have their out-links added to the crawl frontier — the list of unvisited pages. A crawler may be configured to download a fixed number of pages, or it can continue crawling until storage or CPU resources have been consumed. Determining which page to retrieve next in the crawl frontier has been the focus of a huge amount of research in this area. Search engines typically employ a breadth-first crawl as they can potentially be invoked with any query. The aim of focused crawling is to crawl using a best-first ordering of the pages in the crawl frontier. The following subsections will discuss focused, intelligent, learning and semantic crawlers.

### 2.3.1 Focused Crawlers

Focused crawlers are programs which are used to selectively retrieve web pages based on predefined criteria or topics. Because focused crawlers target a narrower slice of the web, and can crawl this slice to a greater depth, they can potentially retrieve information that was missed by a traditional breadth-first crawler. Focused crawlers make use of the phenomenon of Topical Locality [31] i.e. the content of a linked page is likely to be related to the content of the page on which it was linked. Links that appear closer together on a web page tend to be more related to each other and the anchor text can provide useful information about the relevancy of the linked content. Before a focused crawl is conducted a set of locations from which to begin the crawl are required, these are known as seed pages. The relevancy of the returned result is highly dependent on the selection of good seed pages. Seed pages should either contain highly relevant content, or else link to highly relevant content in a minimum amount of link traversals.

[2]The deep web is content which is invisible to search engines. There are various reasons this can be the case: pages are dynamically generated, pages contain no in-links, page access requires registration and login, page content varies with context, page access requires links generated by a scripting language or pages contain textual content in specific file formats not handled by search engines.

**Methods of Focused Crawling**

**Link prioritisation** is used by both Fish Search [32] and Shark Search [38] approaches. Fish Search employs key-word matching and assigns a binary priority value to potential download pages, 1 if the page is relevant to the search, 0 if not. The Shark Search approach, expands on this by employing a Vector Space Model (see 2.5.2.2 for details) to assign priority values more precisely than the binary values assigned by Fish Search. When assigning a priority value, Shark Search considers the content of the page, the priority value of the parent page, the anchor text, and the text in close proximity to the links.

In the Web Topic Management System (WTMS) [46] a user supplies seed pages or key-words from which seed pages can be generated. The crawler creates an Representative Document Vector (RDV) derived from key-words which occur frequently in the downloaded seed pages. The links in the seed pages are downloaded and a Vector Space Model is used to determine their similarity to the RDV. Pages with a similarity in excess of a certain set value are indexed and their links added to the crawl frontier. These out-links are then retrieved until the crawl frontier is exhausted. The in-links, that is the pages linking to a particular page, can be obtained from the Google search engine. The crawler downloads the in-links of the seed pages and indexes them if they are sufficiently similar to the RDV. Then their in-links are added to the crawl frontier and so on. In order to reduce the number of irrelevant downloaded pages the crawler uses 2 heuristics. The first specifies that a linked page will only be downloaded if it is sufficiently *near* in the directory hierarchy. Pages in parent or child directories will be downloaded, as well as sections of sibling directories. The second heuristic specifies that if more than 25 pages have been downloaded from a directory and 90% discarded (because of insufficient similarity to the RDV) then no further pages will be downloaded from that directory.

**Taxonomic Crawling** was suggested by Chakrabarti1 et al [30] as an improvement to focused crawling. This involves the addition of the *classifier* and the *distiller* programs to provide guidance to the crawler. The classifier's responsibility is evaluating the relevance of a retrieved document, while the distiller seeks to identify hypertext nodes which would lead to many relevant pages within a minimum of link traversals. The system is set up with an initial coarse grained canonical classification tree, such as the Open Directory Project (ODP) [20]. The user imports some example URLs, and the system proposes classes that fit the examples best. The user can accept these proposals or if the taxonomy is too coarse, refine categories and reassign examples to the new categories. The system will also propose some additional URLs that appear similar to the given examples. The classifier then integrates the selections of the user into the statistical class models. A crawl can now be initiated. During the crawl the distiller identifies hubs and the (re)visit priorities of these pages and their immediate neighbours are increased. From time to time the user can give feedback on resource lists which is fed back to the classifier and distiller.

**Genetic Algorithms** can be applied to IR, whereby the process of natural selection to applied to solutions, effective solutions thrive, and ineffective solutions go extinct. InfoSpiders [45] is an example of this, a collection of crawler agents traverse the web searching for relevant content, starting at a randomly chosen seed page. They are given a random behaviour and a supply of energy. Agents which are successful have their energy replenished and survive, those that do not return relevant content die off. Agents can change their behaviour by determining the best strategy for having their energy replenished, and agents can produce offspring where their behaviours are combined if two agents reach a page at the same time.

**Context Graph.** One method to improve the priority value given to a page is by using a context model. This models the context within which the relevant content is usually found on the web. The model or context graph is generated by using an existing search engine to find pages which link to a set of seed pages. The found pages are assigned a relational value based on the least amount of link traversals it takes to reach one of the seed pages. Then during a crawl the link distance between an unknown page and a relevant page can be estimated. Thus the crawler can determine if it is worthwhile to crawl deeper, given the page's context. Links which are closer the relevant pages are prioritised.

### 2.3.2 Intelligent Crawlers

An intelligent crawler does not begin with seed pages, instead predicates are defined. The predicates can be in-linking web page content, candidate URL structure, or other behaviours of the in-linking web pages or siblings. They are then used to estimate the probability a given page will satisfy a query on a particular topic. The web's link structure is learned and so the classifier is trained as the crawl progresses. The crawl begins at general points on the web, and then focuses in on relevant content as it comes across pages which match the predicates.

### 2.3.3 Learning Crawlers

Instead of seed pages, a learning crawler is started with a training set. A training set will consist of example pages indicating which pages are deemed relevant to the topic of the crawl. It may optionally include the links traversed leading to the relevant pages and also may include pages that are not relevant to crawl. Once the crawl has been initiated, a classifier analyses retrieved content and determines content relevancy, and assigns a priority. Naive Bayes classifiers, decision trees, neural nets and Support Vector Machines (SVMs) have been used as classifiers for Learning crawlers (see Section 2.4). The Context Graph method is extended using the Hidden Markov Model crawler. A user surfs the web searching for pages stating if a visited web page is relevant or not. The sequence of page visits is recorded and can then be used to train the crawler, helping it to recognise paths which lead to relevant content.

### 2.3.4 Semantic Crawlers

As far as best-first crawlers are concerned, content relevance equates to content similarity. Lexical term matching is used to determine if two documents are similar. This does not allow for terms that are semantically similar but lexically different. As a result of this traditional crawlers will ignore content that is lexically dissimilar but semantically comparable to relevant content. Semantic crawlers attempt to address this issue using term ontologies. These term ontologies allow similar terms to be related to each other using the various type of semantic links — Is-A, Is-Part-Of etc.

#### 2.3.4.1 Swoogle

Semantic web crawlers are very limited as Semantic Web Documents (SWDs) are sparsely distributed throughout the web, and seeding pages are hard to obtain. It is quite expensive to have to confirm a document contains Resource Description Framework (RDF) content when scaling for millions of documents. Swoogle [33] addresses this by using a hybrid Semantic Web harvesting framework. This utilises user submissions of SWDs, it harvests URLs from Google queries for pages with file

type RDF, it does conventional web crawling, and RDF crawling. The RDF crawling will confirm the document is an SWD, metadata (data at the syntax and semantic level) can be retrieved and added to the Swoogle sample dataset. An inductive learner can then use this dataset to generate new seeds for further Google queries and conventional web crawling.

### 2.3.5 Comparative Analysis

Generic web crawlers tend to crawl in a breadth first manner, they are designed to fulfil the information needs of vast majority of the public. Hence they cannot address the need for bodies of relevant information in specific subject areas. Focused crawling can fill this gap. Focused crawlers can be intelligent and adaptive and so offer a much richer experience when looking to explore the web in a subject specific fashion. When prioritising links, breadth-first has been shown to be the link-ordering algorithm of choice. Taxonomic crawling can improve matters, but it can often require a user to possess certain level of technical expertise in order to train the crawler. Intelligent crawls which begin as general purpose crawls will be take longer and subsequently require more resources. Context graphs rely on existing search engines for back-crawling and Genetic algorithms have been found to be inferior to standard best-first algorithms.

## 2.4 Classification

Classification is used to assign a document to a particular category (also referred to as a class) or categories, based on its contents. Classifiers will typically require *training* in order to generate a classification model, which can then be used to categorise a document. The following subsections will discuss the *Naive Bayes*, *Decision Tree*, *Artificial Neural Network (ANN)* and *SVM* classifiers, their training requirements and finally analyse their strengths and weaknesses.

### 2.4.1 Naive Bayes

A Naive Bayes classifier is a probabilistic classifier which applies Bayes theorem [28] with naive independence assumptions and has been in use in IR for over 40 years. When determining if an document should belong to a certain class, each property necessary for conferring membership of a class is treated as being independent of all the other properties, even if one property is related or depends on another. Each property independently contributes to the probability that the document is a member of the class. Because of the ease of implementation and speed of use, Naive Bayes is often used as a baseline for classification. It requires minimal training data for estimating the parameters required for classification. However it currently has a reputation for poor performance, but with some modifications it has been shown [50] to achieve better performance, close to SVM techniques.

### 2.4.2 Decision Trees

A classifier will use a decision tree (also known as a classification tree) to generate a model that predicts the class of an inputted document. The classification decision tree is constructed as follows, the root node of the tree contains all documents. An internal node contains a subset of the documents of its parent node, derived according to one attribute. The arc between a parent an child node is labelled with a predicate to apply to the attribute at the parent node. A leaf node is labelled with a class. The tree is recursively partitioned from the root node. Documents are split into subsets

according to an attribute, the attribute with the highest *Information Gain* [41] or using the *Gini Index of Diversity* [35] is chosen first. Trees can grow to be excessively large and so must be pruned to compensate for overfitting.

### 2.4.3 Artificial Neural Networks

ANNs are electronic representations of the neural structure of the brain. The network is trained by processing records sequentially and learning by comparing its classification with the known classification. Errors from the first classification are fed back [36] into the system and used to modify the algorithm for classification of the second record and so on. A neuron takes a set of inputs, each of which has an associated connection weight (initially a random value), it sums the weights and maps the result to an output. The network is composed of layers which are themselves composed of connected neurons. There is an input layer connected to one or more hidden layers which are connected to an output layer. The input layer is simply takes the input values of a record and passes them to the connected hidden layer. The output layer has a separate output for each individual class. A record processed by an ANN results in a value at each output in the output layer, the record is then categorised as belonging to the class with the highest value. Because the classes are known for the records in the training phase, the outputs can be assigned 1 for the correct class and 0 for all the others. The calculated values are compared to the correct values to generate an error value which can be fed back into the hidden layer so the input connection weights in the neurons can be adjusted to produce better results the next time a record is processed. The same training data can be processed several times to further refine the weights. It is possible that a neural network does not learn, this is the case if the correct output cannot be derived from the input data or there is not enough training data to sufficiently refine the network.

### 2.4.4 Support Vector Machines

Proposed in 1992 by Boser, Guyon and Vapnik [29], SVMs are used for separating data into one of two categories. An N-dimensional hyperplane model is built from a training set. The training set consists of examples, each one labelled as belonging to one of two categories. The examples are mapped as points in a hyperplane, such that examples in separate categories are separated by the widest possible margin in the space. Once trained a new example will be mapped to the space and estimated to belong to the category on whichever side of the margin they have been mapped to. The ability of an SVM to learn is independent of the dimensionality of the hyperplane. In the context of text classification a document will be reduced to *feature vectors*. Each feature vector will typically be a word stem (as a result of a prior stemming process), after all the stop-words have been removed from the text. Document text that produces a high dimensional input space, few irrelevant features and document vectors are sparse and these properties are well suited use with SVMs.

### 2.4.5 Comparative Analysis

Naive Bayes classifiers are well suited for numeric and textual data, and are easy to implement when compared with the other classifiers as well as having low overhead. However the independence assumption does not fit with reality and it performs very poorly with correlated data.

The advantages of using a Decision Tree classifier are that is is easy to generate rules and reduce the problem complexity. However the time spent training can be expensive, and once a mistake is

made at a node, any nodes in the sub-tree are incorrect. It is possible the excessively large trees will be generated so a means of pruning must be devised.

ANNs are capable of producing good results in complex domains. They are suitable in the discrete and continuous domain. Classification is fast but training quite slow. Learned results can be difficult to interpret compared with decision trees and like decision trees they may suffer from overfitting.

SVMs are superior to ANNs in capturing inherent data characteristics. An SVM's ability to learn is independent of the dimensionality of the hyperplane. In tests SVMs tend to outperform the other classifiers.

## 2.5 Indexing

With the continuous production of content on the web, the importance of appropriate indexing of this content has never been greater. The importance of indexing in IR cannot be understated. Indexing refers to the process of representing the content to allow fast and accurate IR. In the context of IR indexing treats a document as an unordered set of words. Indexing attempts to do some term analysis of the document in order to decide on its subject matter. The following section will describe the means and types of indexing, the IR models used and their strengths and weaknesses.

### 2.5.1 Document Representative

It is necessary to reduce the input text to a document representative. A document representative will consist of a list of class names which can also be referred to as index terms. Each class name represents a class of words. So assigning a document representative a particular index term implies that one of the original documents *significant* words is contained within a particular class of words. Achieving this will require three steps:

1. Removal of high frequency words
2. Suffix stripping
3. Detection of equivalent stems

#### 2.5.1.1 Removal of Stop words

Zipf's Law [55] states that "the product of the frequency of use of words and the rank order is approximately constant". This essentially means that words which occur very commonly or very rarely in a document have little descriptive value. Common words will occur in too many documents to add any description of value and extremely rare words may possibly be misspellings or uncommon proper nouns. Words which occur with medium frequency are best used for differentiating between documents in a collection. The removal of high frequency or *stop words* is usually a process of comparison of the original document with a *stop list* of words to be removed.

#### 2.5.1.2 Suffix stripping

Suffix stripping or *stemming* is based on the assumption that words with the same stem are referring to the same concept and the index should reflect this. It is usually done using a suffix list and context rules. The context rules are devised in order to avoid mistakenly removing suffixes. But even with the best rules it is inevitable that errors will occur and terms will be conflated when they should not be, however the error rate has been shown [44] to be on the order of less than 5%.

#### 2.5.1.3 Index Term Weighting

Keen and Digger [39] have defined the terms *indexing exhaustivity* and *indexing specificity*. The first is defined as the quantity of different topics indexed and the latter as the precision with which a document is actually indexed. Quantifying these definitions proves to be very difficult. High levels of indexing exhaustivity lead to high recall and low precision. Whereas low levels lead to low recall and high precision. On the other hand high levels of indexing specificity lead to high precision and low recall and low levels lead to low precision and high recall. Index term weighting schemes have been applied such that each index term is assigned a weight proportional to the frequency of its occurrence in the document. The term frequency (tf) can be defined as follows:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

Where $n_{i,j}$ equals the number of occurrences of a term $t_i$ in document $d_j$ divided by the sum of the number of occurrences of all terms in document $d_j$. Index term weighting schemes have also been applied such that each index term is assigned a weight proportional to the frequency of its occurrence in the entire collection of documents. The inverse document frequency (idf) can be defined as follows:

$$idf_i = \log \frac{\mid D \mid}{\mid \{d : t_i \in d\} \mid}$$

Where $\mid D \mid$ is the total number of documents and $\mid \{d : t_i \in d\} \mid$ is the number of documents in which $t_i$ occurs. Salton and Yang [52] combined the tf and idf thus taking into account both the *inter* and *intra* document frequencies, defined as term frequency–inverse document frequency (tf-idf) weighting which is equal to:

$$tf\text{-}idf_{i,j} = tf_{i,j} \times idf_i$$

Using this weighting they were able to draw the following conclusions:

- High frequency index terms have little value in retrieval, whatever the distribution.
- Medium frequency index terms proved most valuable, especially with a skewed distribution.
- Infrequent terms with a skewed distribution have value, but were less valuable than medium frequency terms.
- Very infrequent terms have little value, being only more valuable than high frequency terms.

### 2.5.2 IR Models

An IR model describes a means of communicating an information need and how it may be refined. It specifies the format of a user query and also how that query will be satisfied, that is, how a document will be retrieved that fulfils the user's specific topic of interest. The Boolean, Vector Space, Probabilistic and Language models will be described and analysed.

#### 2.5.2.1 Boolean Model

The boolean model is historically the oldest and most common IR model. It is based on boolean logic and classical sets theory. Documents in a collection are treated as a set of terms which have been extracted from all documents in the collection. Each term is assigned a binary weight, 1 denoting a term's presence in the document and 0 denoting its absence. Queries can then be represented as a boolean expression composed of terms and boolean connectors *AND*, *OR* and *NOT*. In a search using

the boolean model the documents retrieved are those which are *TRUE* for a given query [53]. Boolean expressions can be quite limited and so the model was extended with the addition of term proximity operators. A proximity operator allows the ability to specify that two query terms must be present close to each other in a document. The closeness of terms may be defined in terms of a structural unit (sentence or paragraph for example) or a number of intervening words.

#### 2.5.2.2 Vector Space Model

In the Vector Space Model both document and queries are represented by vectors. A term (typically a word or a phrase) that is present in the document or query is given a non-zero value in the document or query vector. The value corresponds to the term weight discussed in the previous section. For example a document could be represented by the vector:

$$\vec{d_k} = (w_{1,k}, w_{2,k}, \ldots, w_{t,k})$$

The query could be represented by the vector:

$$\vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$$

Each term is represented by a dimension in the vector. The angle between the two vectors is used to compare the user query and the document. The cosine of the angle can be used as it has the property such that identical vectors have a cosine of 1 and orthogonal vectors have a cosine of 0. If tf-idf weighting (as described in the previous section) is used, the document $d_k$ can be compared with query $q$ using the cosine similarity function as shown below:

$$sim(d_k, q) = \cos(\vec{d_k}, \vec{q}) = \frac{\vec{d_k} \cdot \vec{q}}{\mid \vec{d_k} \mid\mid \vec{q} \mid} = \frac{\sum_{i=1}^{t} w_{i,k} * w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,k}^2} * \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

#### 2.5.2.3 Probabilistic Models

A family of models operating on the basis of the *probabilities ranking principle*. That is, the documents returned by a query should be ranked by decreasing probability of relevance to a query. The model estimates the probability of relevance and there have been several methods developed for this process. The models require initial estimated assumptions of relevancy because there are no previously retrieved documents upon which to base the probabilities. The initial estimations are then recursively refined to obtain the final ranking of pages. The Bayes rule[28] can be applied so that odds rather than the probability that the retrieved page are relevant can be used. The odds that a page is relevant is equal to the probability it is relevant divided by the probability than it is not relevant. The models make term independence assumptions as well as taking into account term presence and term absence[51].

#### 2.5.2.4 Language Model

The query a user forms when searching for information will typically consist of key-words they expect to appear in a retrieved document. A language model attempts to model this. A probabilistic model is built for every document in the collection. The probability of a language model for a given document generating an inputted query was defined by Ponte and Croft[49] as:

$$P(Q \mid D) = \prod_{q \in Q} P(q \mid D) = \prod_{q \in Q} \frac{D_q}{\mid D \mid}$$

This probability value is calculated for each candidate document which allows ranking of the results. A practical problem with the function above is that documents missing one or more of the query terms will be assigned a probability of zero. To combat this some smoothing to avoid zero frequencies must be done. Smoothing can be achieved either using *discounting* or *interpolating* methods. Discounting methods involve adding or subtracting a constant, to redistribute the probability mass. Better results [54] are seen by interpolating the language model with the background probability of an unseen word, computed as relative frequency of a word in a large collection.

#### 2.5.2.5 Comparative Analysis

The Boolean model is easily implemented and computationally efficient. Queries defined as boolean expressions can be expressive and provide clarity, however they can be difficult to construct and are all or nothing. Results are not ranked and it provides no weighting of index or query terms. The Boolean model can be said to be more suited to data retrieval rather than information retrieval. In comparison Vector Space models are not binary and allow ranking of results. However they have limited expressive power and are computationally more expensive than the Boolean model. There is the assumption that terms are independent. Because they produce low similarity values, long documents are inadequately represented by Vector Space models. Also the order of terms is lost in the Vector Space model representation. Probabilistic models also assume term independence. The probabilities are based on estimates so prior knowledge is required. Term frequencies within a document and document length are not taken into account. Unlike the Probabilistic models, Language models do not require initial estimates and term frequencies are factored into the results. Current evaluations have shown Language models to be equivalent to Vector Space models, and the Vector Space, Probabilistic, and Language models have all been shown to be superior to the standard Boolean model, a summary of some of the comparable properties of the models can be seen in the table below.

Table 2.1: IR Model properties

| | Standard Boolean | Vector Space | Probabilistic | Language |
|---|---|---|---|---|
| Mathematical Basis | Set Theory | Algebraic | Probabilistic | Probabilistic |
| Ranking of Results | No | Yes | Yes | Yes |
| Binary Term Weights | Yes | No | Yes | No |
| Term Frequency Accounted for | No | Yes | No | Yes |

## 2.6 Summary

This chapter examined the techniques and technologies for the generation of focused content caches. First the prominent algorithms used for IR on the world wide web were explored. Breadth-first crawling used by web search engines was introduced, leading on to focused crawling for the generation of topic specific caches. The classification techniques necessary to determine if a document is relevant to a particular topic and typically used for focused crawling were described and analysed. Indexing which refers to the process of generating a document representative to facilitate search and retrieval was discussed. Finally IR models which specify how a document is retrieved that fulfils a user's specific topic of interest were described and compared.

# Chapter 3

# Open Source Tools

This chapter will describe and analyse open source crawlers, indexers and tool chains, that could be used for the generation of a focused content cache.

## 3.1 Crawlers

Crawlers are automated programs that crawl the web in a methodical fashion and download documents for further processing (typically indexing). As discussed in Subsection 2.3.1 focused crawlers will typically use some sort of classification to determine if a downloaded document is relevant to a specific topic. The following subsections will discuss some of the open source crawlers that are available, and analyse their potential use as focused crawlers.

### 3.1.1 Heritrix

Heritrix [7] is the Internet Archive's web crawler distributed under the GNU Lesser General Public License (LGPL). Written in Java, it has undergone continual development since 2004. It is widely used by several National Libraries. Heritrix stores the crawl data in ARC files. Each ARC file consisting of several URL records along with some metadata, the HyperText Transfer Protocol (HTTP) header and the response. Heritrix supports 4 different types of crawl: Broad, focused, continuous and experimental. A crawl is defined by 3 components, the Scope, the Frontier and the Processor Chains. The Scope determines if a URL should be added or discarded from a crawl. It includes the seed pages used to start a crawl, and rules for determining if a discovered page should be scheduled for download. Effectively a breadth-first, order-of-discovery policy is used for choosing URLs to process. The Frontier keeps track of pages downloaded and those scheduled to be downloaded. It is also responsible for selected the next page to be downloaded. The Processor Chains allow modules to perform specific actions on each download page in turn.

### 3.1.2 Methanol

The Methanol Web Crawling System [15] is a customisable web crawler written in C developed by Emil Romanus distributed under the Internet Systems Consortium (ISC) license. It can handle HTML, Cascading Style Sheets (CSS), image, flash and video files. Methanol's primary component is its web crawler Methabot which has been optimised for speed. The crawler has an extensible module system and supports user defined parsers written in JavaScript for additional file type support. A crawl can be configured for specific file types, each file type has an associated parser and potentially some user

defined attributes. The parser is effectively a script or callback function that is invoked when the crawler encounters a specific file type. The parser is responsible for extracting data from the file and can optionally set the user defined attributes. Methabot supports the Robots Exclusion Standard and is distributed and multi-threaded. Crawled information is stored in a MySQL database.

### 3.1.3 Grub

Grub [6] is a distributed Peer-to-peer (P2P) search crawler platform. Like the SETI@Home deployment model, users can download a Grub client which runs when the user's computer is idle. The client crawls and indexes URLs and sends the information in a compressed format back to the central Grub server. Started in 2000, several versions were released under a closed license. It was released under the open source software licence by Wikia Inc. in 2007, and was used for their open source web search engine Wikia Search. It has achieved limited take up, and it is not a true P2P implementation because of the requirement for a central server which is a bottleneck.

### 3.1.4 Analysis

Both Heritrix and Methanol provide architectures which support the plugging in of modules that could be used for focused crawling. They do not provide focused crawling functionality "out of the box". Grub does not support focused crawling and it is no longer being used by Wikia Search since 2009. Both the Heritrix and Methanol open source projects are active, but Heritrix has more usage and a more frequent release schedule. The latest version of Heritrix was released this year, but Methanol has not been updated in over a year.

## 3.2 Indexers

As discussed in Section 2.5, indexing refers to the process of representing the content to allow fast and accurate IR. The following subsections will discuss some of the open source indexing tools which could be integrated into an FCGS. The tools will be analysed and the IR models (see Subsection 2.5.2) they support specified.

### 3.2.1 Lemur

Lemur [12] has been developed as a result of a partnership between Center for Intelligent Information Retrieval at the University of Massachusetts Amherst and the Language Technologies Institute at Carnegie Mellon University and is available under the Berkeley Software Distribution (BSD) license. Together they have developed the Lemur tool kit, an open source tool kit for the construction of language modeling and IR software. As part of this collaboration they have also build the INDRI search engine. The tool kit has been written in C and C++ and is designed to run in a Unix environment but can also be run in Windows. It can index up to large-scale (terabyte) collections. It has built in support for English, Chinese and Arabic text. It supports the TREC Text, TREC Web, plain text, HTML, eXtensible Markup Language (XML), Portable Document Format (PDF), MBox (Unix mailbox files), Microsoft Word, and Microsoft PowerPoint file formats. Indexing supports both Porter and Krovetz word stemming, retrieval can be achieved using language modeling approaches such as Indri and KL-divergence, as well as Vector Space, tf-idf, Okapi and InQuery.

### 3.2.2 Lucene

Lucene [13] is a text indexing and search library originally developed by Doug Cutting. It is currently available under the Apache Licence 2.0. It is written in Java and so has cross platform support, but it has also been ported to several other languages. It does not provide any HTML parsing functionality, so the text needs first to be parsed from the HTML or any other file format to be indexed before it can be processed using the Lucene API. Lucene provides ranked search results and allows querying by phrase, wildcard, proximity, range and more. Text analysis tools are included which facilitate text normalisation, stopword removal and stemming. Lucene can support tf-idf term weighting and a combination of the Boolean and Vector Space model are used to compute query-document similarity.

### 3.2.3 Xapian

Xapian [27] is open source IR library which using Boolean and Probabilistic modelling available under the GNU General Public License (GPL). Written in C++ with binding to several other languages it also has cross platform support. Several large organisations are using Xapian including One Laptop per Child, Delicious and Debian. Xapian allows the easy addition of search facilities and indexing to applications by developers using its adaptable tool kit. It has the capability to scale to hundreds of millions of documents. It allows phrase and proximity searching, probabilistic ranking and relevance feedback. Boolean querying and wildcard search are supported. It can be combined with Omega to add search engine facilities to an intranet or web site.

### 3.2.4 ht://Dig

The ht://Dig [9] system was developed at the San Diego State University for indexing and searching for information on the campus web servers. It provides indexing and search capabilities under the GNU GPL. The system utilises fuzzy word searching with any combination of the following algorithms: exact, soundex, metaphone, stemming, synonyms, accent stripping, substring and prefix. Only HTML and text files can be searched. A HTML and command line interface is provided for Boolean expression querying. The system is only built to support a single business or university and does not scale to the world wide web.

## 3.3 Information Retrieval Tool Chains

There has been work to integrate some of the crawling and indexing tools mentioned in the previous sections of this chapter to create IR tool chains. The following subsections will describe some of the open source IR tool chains available and finally analyse their suitability for use as an FCGS.

### 3.3.1 Combine Harvesting Robot

Combine Harvesting Robot [1] is a system for harvesting and indexing web content and distributed under the GNU GPL. Initially developed as a general purpose crawler it was modified by adding an automated subject classifier to be used as a focused crawler. An ontology is used for topic definition and term matching. A relevant document will undergo character set normalisation, language identification and text segmentation. A Structured Query Language (SQL) database stores the crawl information, allowing parallel execution of multiple crawlers. Metadata can be extracted from text,

HTML, PDF, PostScript, Microsoft Word, LaTeX and image files using Combine.
Before initiating a crawl, a topic definition must be defined. A topic definitions is a triplets consisting of the term, relevance weight and topic classes. Weights are positive or negative numbers used to indicate term relevance with respect to the topic classes. Single words, phrases or a Boolean expression can make up a term. Topic classes are used to define a subject class hierarchy. A file of seed pages must also be provided.

### 3.3.2 OCCS

The OCCS [42] is an IR tool chain created by Seamus Lawless at the University of Dublin, Trinity College in 2009. It is based upon open source components and was developed with focused crawling in mind. The motivation for the tool chain was the harvesting of educational content from open corpus sources, and facilitating its incorporation into eLearning systems. It utilises Heretrix [7] for crawling, JTCL [10] and Rainbow [22] for classification, NutchWax [19] for indexing and WERA [26] for searching for and viewing of content.

The Heritrix crawler is integrated with JTCL for language guessing and Rainbow for text classification, in order to facilitate focused crawling. If the harvested content is determined by JTCL to be English, Rainbow will utilise the Naive Bayes algorithm to classify the text. Rainbow requires classification training in the relevant topic of interest. It builds a statistical model based on key-word files, ODP categories as well as positive and negative training sets. A relevancy rating which is predefined is used to determine whether content should be added to the repository in the form of ARC files. NutchWax then indexes the ARC files generated by the crawler. The ARC files can then be searched and navigated using WERA.

### 3.3.3 Swish-e

Swish-e [23] is an open source tool chain released under the GNU GPL licence. Primarily an indexer it also includes a web crawling spider. It was originally developed in 1996 by the UC Berkeley Library building on top of Swish developed in 1994 by Kevin Hughes. Capable of retrieving and indexing plain text, email PDF, PostScript, MS Word/Excel/PowerPoint documents, XML and HTML. Built using the GNOME libxml2 parser, it is configurable to allow indexing of sections of website as well restricting indexing to selected metadata tags. Query results are ranking using the basic tf and term weighting can be influence by term location on the page, but tf-idf is also supported. It is widely used and well supported.

### 3.3.4 Nutch

Nutch [18] is an open source web search engine written in Java and released under the Apache Licence 2.0. The architecture allows components to be plugged in and out for activities such as file type parsing, querying, clustering and content retrieval. It provides a crawler for fetching and indexing pages and also a search for responding to user search requests. The crawler can be broken down into a fetcher for downloading content and extracting links, a web database (WebDB) for storing URLs and harvested content and an indexer which generates an index made up of key-words for each downloaded page. Crawling on a scale of a local file system up to the world wide web is supported. Lucene [13] (see Subsection 3.2.2 is used to provide indexing and search capabilities.

Nutch is currently released without any document parser plugins but these are readily available for most common document formats.

### 3.3.5 Comparative Analysis

The open source IR tool chains surveyed provide significant functionality when it comes to crawling and indexing. Table 3.1 displays a summary of the capabilities of all of the open source tool surveyed. The OCCS is probably the most fully featured of the tool chains surveyed having been designed with focused crawling for web content in mind. However as a system, the individual components are tightly coupled, and the process of initiating a crawl requires classification training and crawl parameters which need to be set in a configuration file on disk. This puts significant restrictions on the reuse and refactoring of the tool chain. Ideally the tool chain as a whole and also individual tools in the chain would be invokable, tunable and manageable over the web, but the OCCS is only accessible by logging on to a specific server and manually invoking some scripts. That said it still provides the most potential for the development of an FCGS.

Table 3.1: Summary of Open Source tools

| | Classification | Focused Crawling | IR Model |
|---|---|---|---|
| Heritrix | N/A | ✗ | N/A |
| Methanol | N/A | ✗ | N/A |
| Grub | N/A | ✗ | Boolean/Vector Space |
| Lemur | N/A | N/A | Language/Vector Space |
| Lucene | N/A | N/A | Boolean/Vector Space |
| Xapian | N/A | N/A | Boolean/Probabilistic |
| ht://Dig | N/A | N/A | Language |
| Combine Harvesting Robot | Custom Ontology | ✓ | Boolean/Probabilistic |
| OCCS | Naive Bayes | ✓ | Boolean/Vector Space |
| Swish-e | N/A | ✗ | Language |
| Nutch | N/A | ✗ | Boolean/Vector Space |

N/A – Not Applicable, ✓ – Supported, ✗ – Unsupported

# Chapter 4

# Design

## 4.1 Introduction

The core objectives for the development of an FCGS which would provide an API for generation of focused Open Corpus Physical and Virtual Caches can be summarised as follows:

1. Enable application developers to quickly and easily develop services for the generation of focused content caches.
2. Provide monitoring and management services for administration of content generation.
3. Support the generation of physical and virtual caches.

These goals can be further broken down into a set of high level requirements:

- Define web services for an FCGS that will provide the functionality of:
  - Training
  - Tuning
  - Crawling
  - Indexing
- The services should provide monitoring and management interfaces.
- The interfaces should be flexible, extensible, and easily allow the addition of services and operations.
- Where an order of invocation of the component services exists, that order of invocation should be enforced and verified.
- The services should be invocable from the web and not require any manual configuration or invocation.
- The FCGS should allow the generation and storage of physical and virtual caches.
- The FCGS should use open source software and comply with existing standards where appropriate.

## 4.2 From the OCCS to the FCGS

As a result of the research undertaken for the state of the art in Chapter 3, it was found that there are various partial solutions available that could meet the set of high level requirements mentioned in the previous section. None of the solutions offer full "end to end" generation of focused content as well as providing monitoring and management services that are accessible for use on the world wide

web. The solution that comes closest to meeting the requirements is the OCCS [42]. A high level architecture diagram of the OCCS can be seen in 4.1.

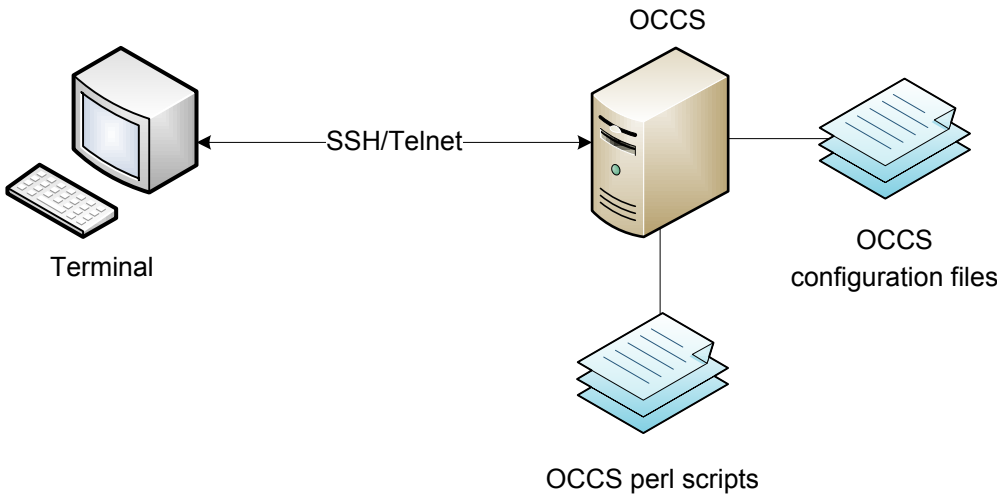



Figure 4.1: Legacy OCCS Architecture

The main gaps between the functionality offered by the OCCS and the high level requirements are as follows:

- The OCCS is only accessible by logging in to a specific server.
- The OCCS tools must be manually invoked using a set of Perl scripts.
- The OCCS tools must be manually configured via a set of configuration files.
- The OCCS does not provide management or monitoring interfaces[1].
- The OCCS does not verify its current state before allowing invocation of a tool[2].

The main impediment to widespread usage of the OCCS is the fact that it is only invocable via a collection of Perl scripts. Any potential user is required to login to the server upon which the OCCS resides, to manually modify some configuration files and then launch certain Perl scripts in a set sequence with further modification of configuration files along the way.

A Service Oriented Architecture (SOA) fits the first high level design requirement of providing Training, Tuning, Crawling and Indexing services. Web services offer the ability for the services to be accessible anywhere on the world wide web. Web services can be thought of as web APIs accessible via HTTP for invoking services that are hosted remotely.

Each tool in the OCCS tool chain would require a web service of its own. Accordingly the functionality provided by the Training, Tuning, Crawling and Indexing tools would be wrapped with the TrainingService, TuningService, CrawlingService and IndexingService web services. The interfaces for these web services would be required to provide operations to:

- Invoke the tool process.
- Monitor the tool process progress.
- Pause/Resume/Terminate the tool process.

[1]The Heritrix web crawler used by the OCCS provides web and Java Management Extensions (JMX) interfaces for management.

[2]The only state maintained by the OCCS are process IDs of the rainbow classifier and the start crawl script in the StateInfo file, the crawl state is available in the Heritrix state.job file as well as via the web and JMX interfaces.

To fulfil the requirement for a separate management interface, each of the tools was analysed to determine their essential management configuration parameters. Out of this analysis it was concluded that three more web services were required, the: TrainingMgmtService, TuningMgmtService and CrawlMgmtService. These management services would provide operations to set management parameters by a manager user. These parameters would be persisted from invocation to invocation of the tool services by normal end users. Collectively all of these web services will from now on will be referred to as the FCGS web services. The FCGS can be seen as being the authors contribution and should not be confused with the OCCS which refers to the preexisting system upon which the FCGS was built. See Figure 4.2 for a high level architecture diagram of the FCGS, it also depicts interactions between a manager as well as an end user with the FCGS.

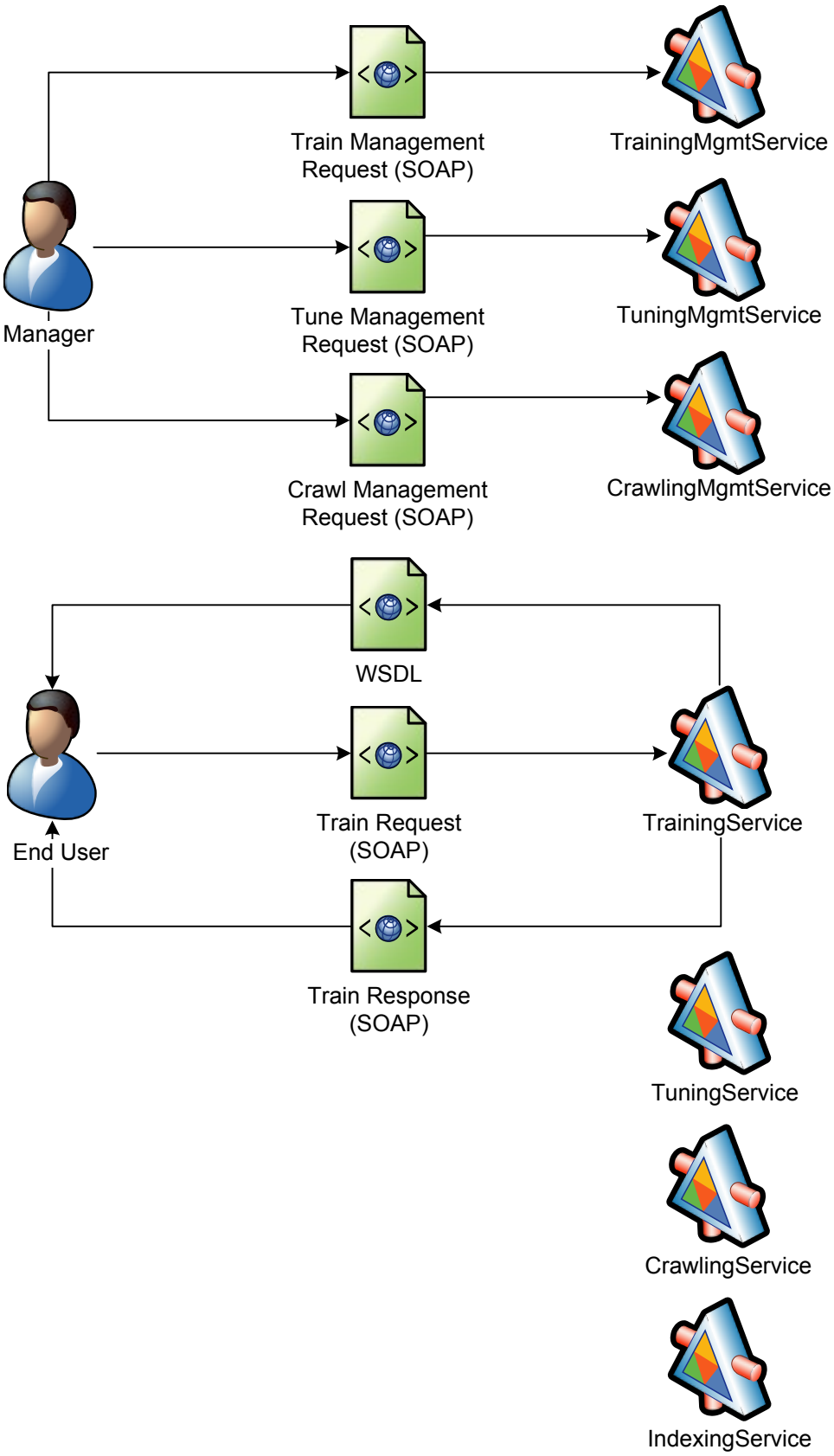



Figure 4.2: FCGS Architecture

## 4.3 FCGS Web Service Interfaces

The interfaces were designed using the top-down approach. This approach mandates that the Web Services Description Language (WSDL) document is composed first, and from that skeleton classes are generated using a code generation tool. The skeleton classes can then be augmented with the appropriate application logic. This approach was taken because it leads to a cleaner design when compared to the bottom-up approach. The bottom-up approach mandates that the implementing class is written first and then a WSDL generation tool is used to expose the class methods as web service operations. The top-down approach was simplified from writing the entire WSDL document from scratch to specifying schemas for the definition of the operation requests and responses. When developing the schemas, close attention was paid to defining elements with the correct type, instead of just using strings for every element. Elements were specified with appropriate occurrence constraints and default values. Where necessary complex types were designed, which also allowed the setting of further restrictions. The schemas were defined with this much attention to detail to allow the potential use of schema-validation of the request parameters on the server side.
The Sun Blueprints XML naming conventions were followed:

- Element names are written in mixed case with the first letter of each internal word capitalised.
- Attribute names are written in mixed case with a lower case first letter.

The interfaces were designed with the second high level requirement in mind. For example it was deemed outside the scope of the dissertation to provide a crawl interface which contained every single Heritrix configurable parameter. In fact it would be undesirable to do so, as this would couple the interface very tightly to the Heritrix crawler. But the crawl interface was designed to be extensible and allow the easy addition of parameters. A concerted effort was made when designing the interfaces that operation and element names would intuitively convey their purpose and meaning. The interfaces were designed to be as simple as possible, whilst still providing enough flexibility to actually be useful and fit for purpose.

## 4.4 FCGS Web Service Operation Workflow

Each of the FCGS web service operations has the same basic structure for as far as workflow is concerned. Figure 4.3 illustrates this workflow. As you can see from the activity diagram, the following activities must be carried out before a response can be returned to the user:

1. Read FCGS properties.
2. Validate request parameters.
3. Query and verify current state of the FCGS.
4. Using request parameters write appropriate OCCS configuration files.
5. Invoke appropriate OCCS script.
6. Get the result of the script invocation.

The design decisions regarding each of the items in the list above will be discussed in the following subsections.

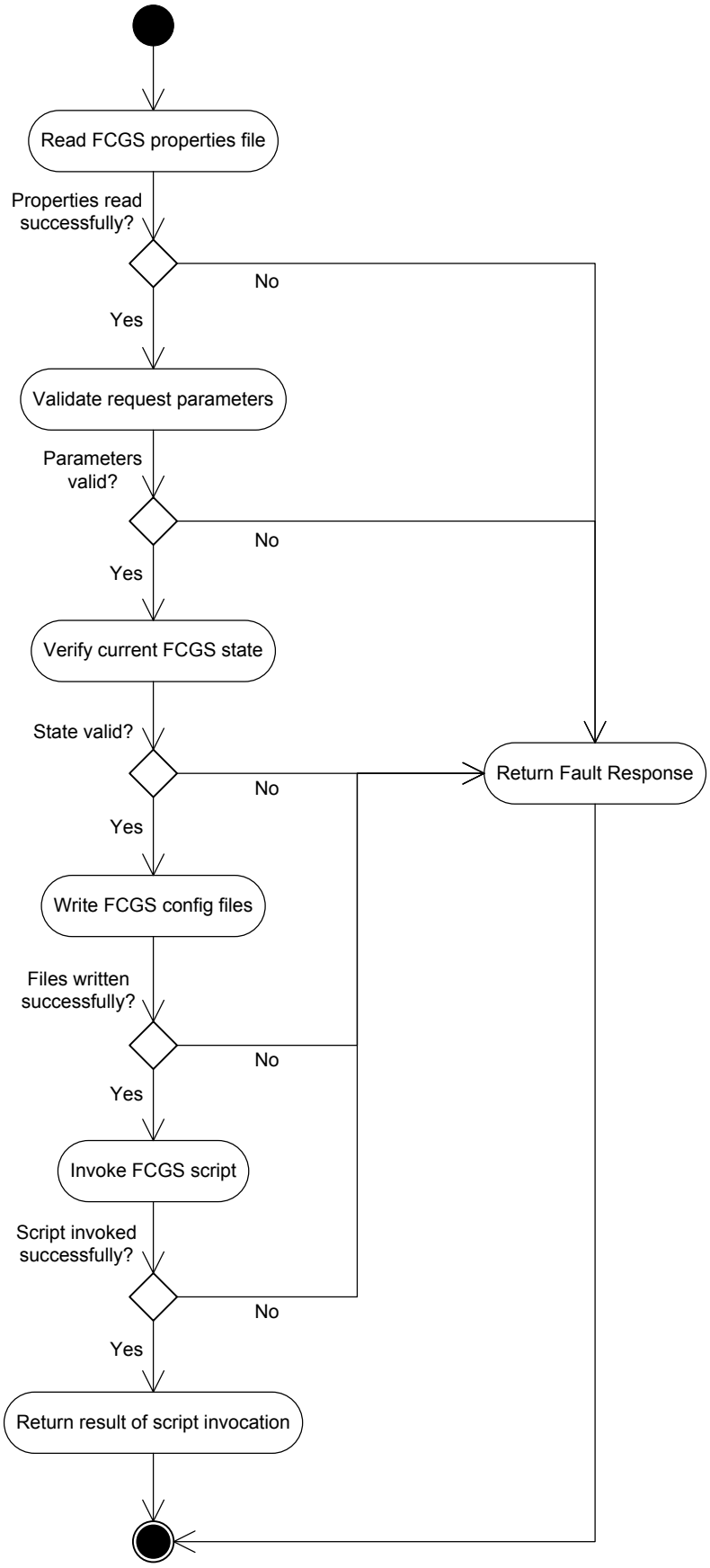



Figure 4.3: Operation Workflow

### 4.4.1 FCGS Properties

There are various properties that the FCGS requires knowledge of in order to function, the location of the OCCS scripts, the location of the OCCS configuration file templates etc. Rather than hard code all of these properties, the FCGS was designed such that each web service would access these properties via a configuration file. This file can easily be expanded in the future with more properties if necessary. See Subsection 5.2.1 for details on how this was implemented.

### 4.4.2 Request Validation

As a result of specifying the schemas for the operation request parameters, using the WSDLs for the services as input, most client application code generation software should ensure that requests are in the expected format and parameters will be of the correct type. However the request parameters still need to be checked to ensure their validity. For example when setting stop words using the tuning service, it must be validated that the stop words received are actual top words for the document classification model. The validation methods were designed to validate all parameters and if any were invalid to return a fault response with an error message for all of the invalid parameters. This was to avoid the user having to make multiple requests in the case that more than one of the request parameters were invalid. The error messages were designed to be informative, listing the invalid parameter, its valid values and an example value if appropriate. See Subsection 5.2.2 for details on how this was implemented.

### 4.4.3 OCCS State

The OCCS tool chain must be invoked in the following sequence:

1. Train
2. Tune (Optional)
3. Crawl
4. Index

Consequently the FCGS must verify that the prerequisite tasks have been completed before invoking any tool of the OCCS. The rules can be summarised as follows:

- To invoke the Tune tool, it must be verified that the Train tool has been run successfully.
- To invoke the Crawl tool, it must be verified that the Train tool has been run successfully, it is not necessary to verify that the Tune tool has been run as it is an optional step in the sequence.
- To invoke the Index tool it must be verified that the Crawl tool has been run successfully.

See Figure 4.4 for an example of the OCCS state verification workflow that takes place when a user wishes to initiate a crawl.

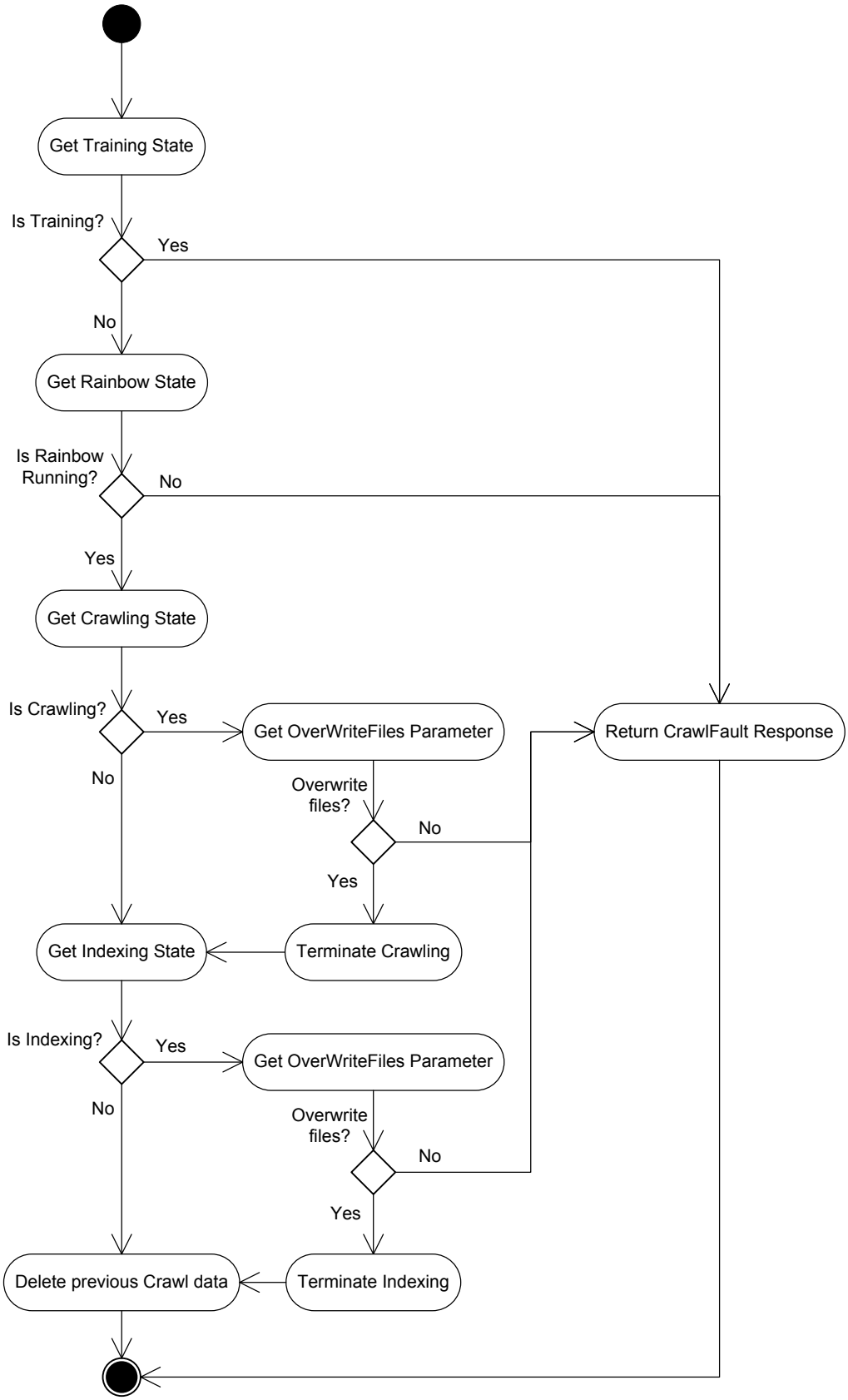



Figure 4.4: Crawl OCCS State Verification Workflow

The tasks in the workflow are as follows:

1. Get the state of the Training tool. Did the tool complete successfully, or is it currently running. If the Training tool did not complete successfully or is currently running a fault response is returned to the user.
2. Check if the Rainbow classifier is currently running. If the rainbow classifier is not currently running a fault response is returned to the user.
3. Get the state of the Crawling tool. If it is currently running and the user had indicated in the request they wish to overwrite the crawling data, the crawling is terminated. If they indicated they did not wish to overwrite the crawling data, a fault response is returned to the user.
4. Get the state of the Indexing tool. If it is currently running and the user had indicated in the request they wish to overwrite the indexing data, the indexing is terminated. If they indicated they did not wish to overwrite the indexing data, a fault response is returned to the user.
5. The previous crawl data is deleted and a new crawling process can be initiated.

The workflows for the train and index operations is quite similar, but are not discussed here for the sake of brevity. The maintenance of the OCCS state is a key factor in this workflow. The existing OCCS maintains the state of the Rainbow classifier through the use of a file and specialised Perl functions for reading and updating the state information in the file. However this was not sufficient for the requirement of workflow detailed previously. For the FCGS it was decided to expand the usage of this file so it would maintain state of the training and indexing tools as well. It would facilitate the querying of their status and their termination if necessary. The approach was taken as the functionality was already present in the OCCS and it only required some minor modifications of the scripts. See Subsection 5.2.3 for details on how this was implemented. The Heritrix crawler maintains its state in its own state file. This state file is also used in the OCCS state verification process. The crawler also exposes management functionality through a JMX interface. The CrawlingService uses this JMX interface as it provides detailed information on the progress of the crawl as well as allowing the pausing and termination of a crawl. See Subsection 5.1.4 for details on how this was implemented.

### 4.4.4 Generation of OCCS Configuration Files

For every invocation of a tool in the OCCS tool chain, it is required that a set of configuration files are manually edited. It was decided for the FCGS these files would be automatically generated through the use of a templating engine. Each tool can require multiple configuration files and can share configuration files with other tools. The shared configuration files were broken into multiple templates. This allows a service of the FCGS to modify its set of parameters in a shared configuration file, which corresponds to one template, all outputs of the templates for the shared file are then combined to form the final configuration file with the correct parameters for that service. This also allows the management services (TrainingMgmtService, TuningMgmtService and CrawlingMgmtService) to set their set of management parameters which are contained in a template of their own (for each configuration file) and these values will persist from invocation to invocation of the tool services (TrainingService, TuningService, CrawlingService and IndexingService). See Subsection 5.2.4 for details on how this was implemented.

### 4.4.5 OCCS Script Invocation

Each of the OCCS tools has an associated Perl script. For the FCGS to work, it was necessary to be able to invoke these scripts and query their status. There are two types of OCCS tool scripts, those that are short running and those that are long running. The short running scripts therefore lend themselves to a request-response type of service, where the service request invokes the script and the service response returns the result of the script invocation. An example of this would be tuning script which returns the classification document model top words. The long running scripts lend themselves to a service with two operations, one to invoke the script and the other to query the status of the script. An example of the would the training, crawling and indexing scripts. Hence the FCGS and the script invocation classes were developed with this in mind. See Subsection 5.2.5 for details on how this was implemented. Once an FCGS tool request is received and the appropriate OCCS script is invoked a response must be returned. In the case of a short running script, this entails returning the output of the invoked script. To facilitate this the output of the invoked script is written to a file, this file is parsed and its contents mapped to the appropriate parameters in the service response message. In the case of a long running script, the status of the script invocation is what would be returned in the response, this information is available in the state information file in the case of the training and indexing tools and via the Heritrix state file and JMX interface in the case of the crawling tool.

## 4.5 Virtual Caches

The OCCS currently only supports the generation of physical content caches. For these caches to be virtual it is only necessary to delete the content downloaded by the crawl process once the indexing process has taken place. The index generated contains the URL of the original content and so the data can be safely deleted. The IndexingService was designed such that once the indexing has completed, if the index operation request parameter *VirtualCache* is set to *true*, then the crawl data content is deleted.

# Chapter 5

# Implementation and Specification

This chapters details the modifications made to the existing OCCS in the development of the FCGS. That is followed by a section describing the implementation details of a typical FCGS web service operation workflow. Technology and deployment are then discussed and finally a full listing of the FCGS API follows.

## 5.1 Modifications to the OCCS

The development of the FCGS required key modifications to the OCCS. These modifications were necessary because of software obsolescence and also because of the requirements of the FCGS. The following subsections will discuss how the modifications to the Perl scripts, the Google Search API the TextCat Processor and finally to the Heritrix crawling software were implemented.

### 5.1.1 Perl Scripts

The existing OCCS Perl scripts are:

- *train_step1.pl* launches training process (step 1).
- *train_step2.pl* launches training process (step 2).
- *fine-tune1.pl* gets the document model top words.
- *fine-tune2.pl* sets the document model stop words.
- *start_crawler.pl* starts the Heritrix crawling process.

As the Google Search API [5] no longer returns ODP categories for search results, the 2-step nature of the OCCS training process is no longer necessary, so the FCGS refactors the train_step1.pl and train_step2.pl into a new *train.pl* script. The OCCS indexing process is a purely manual, the FCGS achieves this with the addition of the *index.pl* script.

In order to verify the prerequisites for each tool invocation, it was necessary to extend the existing OCCS Perl scripts to maintain state. The original Perl scripts maintained some state information in a *StateInfo* file. See below for an example of the original StateInfo file:

```
RAINBOW-PID = 26021
HERITRIX-PID = 26973
```

Listing 5.1: Original StateInfo file

The original file contains two keys and values:

- *RAINBOW-PID* represents the Rainbow classifier process id.
- *HERITRIX-PID* represents the Heritrix crawler process id.

The FCGS expands the StateInfo file with several more keys. See below for an example of the updated StateInfo file:

```
TRAINING-PID = -1
TRAINING-STATUS = FINISHED
TOPIC = HBO_Series_Topic
RAINBOW-PID = 26021
HERITRIX-PID = 26973
INDEXING-PID = 27318
INDEXING-STATUS = RUNNING
```

Listing 5.2: Updated StateInfo file

The additional keys are:

- *TRAINING-PID* represents the training process id.
- *TRAINING-STATUS* represents the training status.
- *TOPIC* represents the Topic parameter inputted in the train request.
- *INDEXING-PID* represents the indexing process id.
- *INDEXING-STATUS* represents the indexing status.

A process id of $-1$ indicates that a process is not running. The FCGS uses the additional state maintained in the StateInfo file to determine which OCCS processes are currently running, and if necessary to terminate them, and also to return the status of the training and indexing processes.

The FCGS adds five new Perl scripts:

- *is_PID_key_running.pl* checks if process ids in the StateInfo file are running.
- *kill_PID_key.pl* terminates process ids in the StateInfo file.
- *clean_training_files.pl* deletes all files generated by the training and tuning processes.
- *clean_heritrix_files.pl* deletes all files generated by the crawling process.
- *clean_index_files.pl* deletes all files generated by the indexing process.

### 5.1.2 Google Search API

The OCCS uses the Google Search API to generate seed URLs from keyword phrases. However this API which had a Simple Object Access Protocol (SOAP) interface was decommissioned in 2009.

It was replaced with the Google Asynchronous JavaScript and XML (AJAX) Search API [5]. The Google AJAX Search API exposes a Representational State Transfer (REST)ful interface that returns JavaScript Object Notation (JSON) encoded results, which allows the embedding of Google Search in web pages and other web applications. The OCCS training script was updated to use the new AJAX API, this involved replacing the use of the module Net::Google [17], with direct calls to the AJAX API using the LWP::UserAgent [14] and JSON [11] modules.

### 5.1.3 TextCat Processor

The OCCS uses the TextCat Processor to determine the language of a downloaded document during the crawling process. The language is hard-coded to *english*, even though the Java Text Categorizing Library (JTCL) library [10] supports 14 other languages. The FCGS modifies the *TextCatProcessor* class to accept a language parameter in the TextCat Processor configuration file: *TextCat_Config_File.txt*. The language is specified by the user in the *crawl* operation request of the CrawlingService. The crawl operation also validates that the language specified is one supported by the JTCL library.

### 5.1.4 Heritrix

The OCCS uses the web crawler Heritrix [7] v1.4.0. This version was released in 2005 and has since been superseded by v1.14.4 which was released in 2010. Version 1.14.4 contains various bug fixes and also provides a complete set of JMX attributes and operations with which to monitor and manage the crawler. Version 1.4.0 only provides only a very limited set of JMX attributes and operations. Further research determined that any useful monitoring or management of the Heritrix crawler would not be possible without migrating to v1.14.4. The migration to v1.14.4 was relatively straightforward only requiring the modification of the crawl job configuration file: *order.xml*. During testing of the migration it was found that Heritrix developed a memory leak when used with Java version 5, this was related to garbage collection, the crawler would run out of heap space within a half hour. Testing Heritrix with Java version 6, showed the memory leak issue to be resolved.

The interaction with JMX interface is handled by the *HeritrixJMXConnection* class, usage of this class requires that the JMX host, port, user name and password are set in the FCGS properties file (see Subsection 5.2.1).

## 5.2 FCGS Web Service Operation Workflow

This section describes the implementation details of a typical FCGS web service operation workflow. The workflow is made up of the following tasks:

1. Read FCGS properties.
2. Validate request parameters.
3. Query and verify current state of the FCGS.
4. Using request parameters write appropriate OCCS configuration files.
5. Invoke appropriate OCCS script.
6. Get the result of the script invocation.

These tasks are discussed in the following subsections.

### 5.2.1 FCGS Properties file

Each invocation of an FCGS web service operation requires knowledge of some global properties. Location of the templates and Perl scripts are examples of such properties. Each operation gains access to these properties via a Singleton class which reads the properties file into memory. The properties file is read into memory only once when the first operation requests an instance of the Singleton class, each subsequent request for the FCGS properties is read from memory. The location of the properties file is configured as a context parameter in the web.xml file for the FCGS web application. See below for a listing of the setting.

```
<context-param>
    <description>Location of the OCCS Web Services configuration file</
        description>
    <param-name>OCCSServiceConfigFile</param-name>
    <param-value>/home/phannon/OCCSService/OCCSService.properties</param-value>
</context-param>
```

Listing 5.3: Context Parameter setting in FCGS application web.xml

See Appendix A for a listing of the FCGS properties file.

### 5.2.2 Validation of Operation Requests

For each request and response message generated by a FCGS web service there is a schema. These schemas are all listed in Appendix C. For each input and output parameter a *type*, *occurrence constraint*, *default value* as well as further restrictions are specified as appropriate. The schemas were designed for the potential use of schema validation. In reality the Java API for XML Web Services (JAX-WS) implementation of schema validation was found to be too unwieldy and did not meet the requirements of the FCGS. Instead the open source Apache Commons Validator [3] package is used for validation of the input parameters for each FCGS web service operation request. This package has built in validators for URLs, email addresses, as well as range and minimum and maximum value validators for the *Integer*, *BigInteger*, *Long* and *Double* Java datatypes. In Section 5.4 the *Valid Values* for each request parameter are listed. There is validation logic which ensures that request parameters conform to these *Valid Values*, in the case where they do not, an appropriate Fault response is returned. The Fault response contains a message detailing the incorrect parameter(s) as well as the valid values accepted. See below for a listing of an example Fault response for a validation error.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Validation error</faultstring>
         <detail>
            <ns2:TrainFault xmlns:ns2="http://training.service/" xmlns:ns3="http:
                //xml.netbeans.org/schema/TrainingParameters">
               <message>Invalid MaxSearchResults. Please enter a value in the
                   range of 1 to 64.</message>
            </ns2:TrainFault>
         </detail>
      </S:Fault>
```

```
   </S:Body>
</S:Envelope>
```

Listing 5.4: Example Validation Error Fault Response

In the example case above, you can see the TrainingService returned a TrainFault because the field *MaxSearchResults* was invalid — its value was not in the range of 1–64.

### 5.2.3 Verification of Current OCCS State

The next subsections will describe how the verification of the OCCS state was actually implemented for the training, tuning, crawling and indexing operations.

#### 5.2.3.1 Training

When the train operation is invoked, it is first necessary to verify that no other OCCS processes are running. The StateInfo file along with the is_PID_key_running.pl script (see Subsection 5.1.1 for details) is used to determine if the training, Rainbow, crawling or indexing processes are currently running. By default the *OverwriteFiles* parameter in the train operation request is set to *false*, in this case, if any of the OCCS processes are running, a TrainFault response is returned to the user. If OverwriteFiles is set to *true*, any of the currently running OCCS processes will be killed using the kill_PID_key.pl script, any previous training data deleted using the clean_training_files.pl script and finally a new training process will be invoked.

#### 5.2.3.2 Tuning

When either the getTopWords or setStopWords operations are invoked it is a prerequisite that the training process has completed successfully, in which case the Rainbow process should be running (as it is only launched once the training process completes). The StateInfo file along with the is_PID_key_running.pl script is used to determine if the Rainbow process is running, if it is not a TuneFault is returned to the user. If the training process is currently running a TuneFault response is returned to the user. By default the *OverwriteFiles* parameter in the getTopWords or setStopWords operations request is set to *false*, in this case, if a crawling or indexing processes are currently running, a TuneFault response is returned to the user. If OverwriteFiles is set to *true*, the crawling or indexing processes will be killed using the kill_PID_key.pl script and Rainbow classifier queried/tuned as per the request.

#### 5.2.3.3 Crawling

When the crawl operation is invoked it is a prerequisite that the training process has completed successfully, in which case the Rainbow process should be running (as it is only launched once the training process completes). The StateInfo file along with the is_PID_key_running.pl script is used to determine if the Rainbow process is running, if it is not a CrawlFault is returned to the user. If the training process is currently running a CrawlFault response is returned to the user. By default the *OverwriteFiles* parameter in the crawl operation request is set to *false*, in this case, if a crawling or indexing processes are currently running, a CrawlFault response is returned to the user. If OverwriteFiles is set to *true*, the crawling or indexing processes will be killed using the kill_PID_key.pl

script, any previous crawl data is deleted using the clean_heritrix_files.pl and a new crawling process launched.

#### 5.2.3.4 Indexing

When the index operation is invoked it is a prerequisite that the crawling process has completed successfully. The status of the crawling process can be determined by parsing the Heritrix crawler state file. The Heritrix crawler outputs its state to a *state.job* file, see below for an example listing.

```
20100824200659768
Autolaunched
Finished - Maximum number of documents limit hit
true
false
3
0
order.xml
```

Listing 5.5: Example state.job file

The first line contains the crawl Job UID: *20100824200659768*.
The third line contains the crawl job status: *Finished - Maximum number of documents limit hit*.
The fifth line indicates if the crawler is currently running: *false*.
The last line contains the crawl job settings file: *order.xml*.

If an error occurred whilst running the crawl it will be indicated on the status line, and an error message will be appended after the job settings file line. The index operation uses the *StateJobReader* class to verify that the crawling process did not generate an error status and that it is not currently running, otherwise an IndexFault is returned to the user. If this can be verified, it is then necessary to verify that the crawling process generated some ARC files to be indexed. It is first necessary to verify that there are no open ARC files. When a crawling process finishes it can often take a few minutes for the process to close all the open ARC files. If any open ARC files exist, an IndexFault is returned to the user. If no ARC files exist, an IndexFault is returned to the user. If only closed ARC files exist, then the prerequisite that the crawling process completed successfully can be said to have been met.

If the training or crawling processes are currently running an IndexFault response is returned to the user. By default the *OverwriteFiles* parameter in the index operation request is set to *false*, in this case, if an indexing process is currently running, an IndexFault response is returned to the user. If OverwriteFiles is set to *true*, the indexing process will be killed using the kill_PID_key.pl script, any previous index data is deleted using the clean_index_files.pl and a new indexing process launched.

### 5.2.4 Generation of OCCS Configuration Files

The existing OCCS system is based on Perl scripts which are manually invoked and various configuration files. For convenience and to maintain backwards compatibility with the existing OCCS architecture the FCGS continues to use these scripts (with some modification) and configuration files. Using the OCCS requires the configuration files to be manually edited, however the FCGS automatically generates these configuration files with a templating engine. The Java-based open source

Apache Velocity [25] engine was chosen for its proven reliability, speed and ease of use. Listings of all the velocity templates are located in Appendix B.

#### 5.2.4.1 Perl Script Configuration Templates

All of the Perl scripts expect at the very least the existence of the configuration file: *config.txt*. Because this configuration file contains settings for training, tuning, crawling and indexing it has been broken into several templates:

- *config-train-mgmt.txt.vm* contains the training management configuration parameters.
- *config-train.txt.vm* contains the training configuration parameters.
- *config-tune-mgmt.txt.vm* contains the tuning management configuration parameters.
- *config-crawl.txt.vm* contains the crawling configuration parameters.
- *config-index.txt.vm* contains the indexing configuration parameters.

All of above templates combine to produce the final config.txt file using the: *config.txt.vm* template. By convention the template file names end with a .vm extension. Listings for all of the config.txt templates can be seen in Appendix B.1.

#### 5.2.4.2 Training Configuration Templates

The train.pl script expects two configuration files: *keywords.txt* and *pos_categories.txt.vm* (in addition to config.txt). These files contain Google API keyword search phrases and the positive ODP categories respectively. The *keywords.txt* file is generated using the *keywords.txt.vm* template. The *pos_categories.txt.vm* file is generated using the *pos_categories.txt.vm* template. Listings for all of the training templates can be seen in Appendix B.2.

#### 5.2.4.3 Tuning Configuration Templates

The fine-tune2.pl script expects one configuration file: *top-words.txt* (in addition to config.txt). This file contains the top words in the document model generated by the Rainbow classifier. The *top-words.txt* file is generated using the *top-words.txt.vm* template. A listing for the template can be seen in Appendix B.3.

#### 5.2.4.4 Crawling Configuration Templates

The start_crawler.pl script uses one configuration file: *order.xml* (in addition to config.txt). This file allows the configuration of various crawl parameters. The crawling management configuration parameters are contained within the *order1.xml.vm* and *order3.xml.vm* templates. The crawling configuration parameters are contained within the: *order2.xml.vm* template. All of these templates are then combined to produce the final order.xml file using the: *order.xml.vm* template. Listings for all of the crawling templates can be seen in Appendix B.6.

The Heritrix crawler uses two processors for focused crawling. These processors were specifically developed for the OCCS, and were made possible by the fact that the Heritrix [7] crawler allows the addition of custom processors to its processor chain. The language of a downloaded document is detected using the *TextCat processor*. If the TextCat processor detects the document is the configured language, the document is accepted and passed on to the next processor. Rejected documents receive

no further processing. The *Bow processor* uses the Rainbow classifier to determine if the downloaded document is sufficiently similar to the document model generated in the training and tuning process. Documents accepted by the Bow processor are archived and rejected documents are discarded.

The TextCat processor uses one configuration file: *TextCat_Config_File.txt.vm*. This file allows the configuration of the language of the documents the processor will accept in the crawl process. The *TextCat_Config_File.txt* file is generated using the *TextCat_Config_File.txt.vm* template. A listing for the TextCat processor template can be seen in Appendix B.4.

The Bow processor uses one configuration file: *Bow_Config_File.txt*. This file allows the configuration of the cut off value for document similarity to the document model the processor will use to accept or reject documents in the crawl process. The *Bow_Config_File.txt* file is generated using the *Bow_Config_File-train.txt.vm* and *Bow_Config_File-crawl.txt.vm* templates. Listings for the Bow processor templates can be seen in Appendix B.5.

### 5.2.5 Invocation of OCCS Perl Scripts

Invocation of the Perl scripts (described previously in Subsection 5.1.1) from the FCGS, is handled by the *ScriptRunner* class. This utilises Java's *ProcessBuilder* class, which allows the execution of system processes from within a Java program. Optionally a separate thread can be launched to consume the output and error streams of the process and this is handled by the *StreamConsumer* class. The StreamConsumer class allows the output and error streams of a launched process to be redirected to a file. The ScriptRunner class handles:

- Verification of the existence of the script.
- Setting of the working directory.
- Passing of command line arguments.
- Checking the exit value of the completed script.
- Exceptions thrown whilst running the script.

The ScriptRunner class can invoke scripts in 2 ways. The first is for short-running scripts (such as *fine-tune1.pl* or *fine-tune2.pl*), which when invoked will return immediately or in a short period of time. The second is for long-running scripts (such as *train.pl* or *index.pl*) which will run for a long period of time. In the case of short-running scripts, the ScriptRunner class will block and wait for the script to complete before returning the exit value to the calling class. In the case of long-running scripts, the ScriptRunner classes will launch a separate thread for the script process and return immediately to the calling class. The two methods of invocation can be illustrated by the following example. The training process is invoked by the *train* operation of the TrainingService and the status of the training process is queried by invoking the *trainStatus* operation. In contrast, to get the top words of the document model, the *getTopWords* operation of the TuningService is invoked. The getTopWords operation responds with the top words generated as a result of invoking the *fine-tune1.pl* script. In the first case, ScriptRunner launches the long-running train.pl script in a separate thread. In the second case, ScriptRunner launches the short-running fine-tune1.pl script and blocks until it returns.

## 5.3 Technology and Deployment

This section details the technologies used in the development of the FCGS and its deployment.

### 5.3.1 Java

The FCGS was developed using JAX-WS on the Java 6 platform. JAX-WS was chosen because of its ease of use which allows speedy development of web services and web service client applications. The reference implementation of JAX-WS was developed as part of the GlassFish [4] project. GlassFish is an open source Java Enterprise Edition (EE) application server. GlassFish v2 was chosen because of its close integration with the NetBeans [16] development environment. Several open source libraries were used, the Apache Velocity [25] templating engine was used for generating the OCCS configuration files. The Apache Commons Validator [3] package was used for its validation routines. The Apache Commons IO [2] package was used for its stream handling facilities.

### 5.3.2 Perl

The modifications to the OCCS Perl scripts required the use of several Perl modules. The Google AJAX Search API [5] was accessed using the URI::Escape [24], LWP::UserAgent [14] and JSON [11] modules. The Proc::Killfam [21] module was used for terminating the OCCS Perl scripts, it facilitates killing a process id and all its child processes.

### 5.3.3 Deployment

All of the FCGS web services are bundled as one web application which allows easy deployment on the GlassFish application server through the use of a Web application ARchive (WAR) file. The GlassFish server provides a web interface for easy deployment of web applications. The NetBeans [16] development environment provides automated application deployment to GlassFish which was very useful during development of the FCGS.

## 5.4 API Specification

Listings for all of the FCGS web service WSDLs can be seen in Appendix C. This section will describe each web service in detail, documenting all of the operations, their request and response parameters as well as providing example requests and responses and also the various Fault response messages that can be returned in the case of an error.

### 5.4.1 TrainingMgmtService

For the full TrainingMgmtService WSDL specification see Appendix C.1. The TrainingMgmtService has only one operation: *trainMgmt*.

#### 5.4.1.1 trainMgmt operation

The trainMgmt operation is used for setting the management configuration parameters of the TrainingService. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.1: trainMgmt Request Parameters

| Name | Description | Required |
|---|---|---|
| NegativeCategoryCount | The number of Negative ODP Categories generated.<br>Type: xs:int<br>Default: 100<br>Valid Values: An int greater than or equal to 0. | No |
| HttpProxyHost | A string representing the http proxy host.<br>Type: xs:anyURI<br>Default: http://www-proxy.cs.tcd.ie<br>Valid Values: A valid URL. | No |
| HttpProxyPort | A number representing the http proxy port.<br>Type: xs:int<br>Default: 8080<br>Valid Values: An int in the range of 0 to 65535. | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tra="http://trainingMgmt.service/" xmlns:tra1="http://xml.netbeans.org/
    schema/TrainingMgmtParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <tra:trainMgmt>
         <trainMgmtRequest>
            <tra1:NegativeCategoryCount>100</tra1:NegativeCategoryCount>
            <tra1:HttpProxyHost>http://www-proxy.cs.tcd.ie</tra1:HttpProxyHost>
            <tra1:HttpProxyPort>8080</tra1:HttpProxyPort>
         </trainMgmtRequest>
      </tra:trainMgmt>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.6: Example trainMgmt Request

The trainMgmt operation will return with the input parameters set if it was successful, if not a TrainMgmtFault response will be returned, see 5.4.1.2 for an example TrainMgmtFault. The table below describes the trainMgmt response parameters and is followed by a listing of an example response.

Table 5.2: trainMgmt Response Parameters

| Name | Description |
|---|---|
| NegativeCategoryCount | The NegativeCategoryCount set in the request.<br>Type: Integer |
| HttpProxyHost | The HttpProxyHost set in the request.<br>Type: String |
| HttpProxyPort | The HttpProxyPort value set in the request.<br>Type: Integer |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:trainMgmtResponse xmlns:ns2="http://xml.netbeans.org/schema/
          TrainingMgmtParameters" xmlns:ns3="http://trainingMgmt.service/">
         <return>
            <ns2:NegativeCategoryCount>100</ns2:NegativeCategoryCount>
            <ns2:HttpProxyHost>http://www-proxy.cs.tcd.ie</ns2:HttpProxyHost>
            <ns2:HttpProxyPort>8080</ns2:HttpProxyPort>
         </return>
      </ns3:trainMgmtResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.7: Example trainMgmt Response

#### 5.4.1.2 TrainMgmtFault

A TrainMgmtFault response will be returned by all of the TrainingMgmtService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example TrainMgmtFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Validation error</faultstring>
         <detail>
            <ns3:TrainMgmtFault xmlns:ns3="http://trainingMgmt.service/"
                xmlns:ns2="http://xml.netbeans.org/schema/TrainingMgmtParameters"
                >
               <message>Invalid NegativeCategoryCount. Please enter a value
                   greater than or equal to zero.</message>
            </ns3:TrainMgmtFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.8: Example TrainMgmtFault Response

### 5.4.2 TrainingService

For the full TrainingService WSDL specification see Appendix C.2. The TrainingService has 3 operations: *train*, *trainStatus* and *trainTerminate*.

#### 5.4.2.1 train operation

The train operation is used to start a new training process. The training process involves training the Rainbow classifier, which allows it to build a document model. In the subsequent crawl documents can be accepted or rejected based on their similarity to the document model. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.3: train Request Parameters

| Name | Description | Required |
|---|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string<br>Default: None<br>Valid Values: numbers, letters or underscores with maximum length 255. | Yes |
| MaxSearchResults | The number of search results returned by the Google API.<br>Type: xs:int<br>Default: 10<br>Valid Values: an int in the range of 1 to 64. | No |
| KeywordPhrases | Container for one or more KeywordPhrase elements. | Yes |
| KeywordPhrase | A string representing a Google search phrase.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid search string. | Yes |
| PositiveODPCategories | Container for one or more ODPCategory elements. | Yes |
| ODPCategory | A string representing an ODP Category.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid ODP Category | Yes |
| OverwriteFiles | Set to *true* if a currently running training process should be killed and existing training data overwritten, if set to *false* and a training process is currently running a TrainFault will be thrown.<br>Type: xs:boolean<br>Default: false<br>Valid Values: true or false | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tra="http://training.service/" xmlns:tra1="http://xml.netbeans.org/
    schema/TrainingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <tra:train>
         <trainRequest>
            <tra1:Topic>HBO_Series_Topic</tra1:Topic>
            <tra1:MaxSearchResults>10</tra1:MaxSearchResults>
            <tra1:KeywordPhrases>
               <tra1:KeywordPhrase>The Wire</tra1:KeywordPhrase>
```

```
            <tra1:KeywordPhrase>The Sopranos</tra1:KeywordPhrase>
            <tra1:KeywordPhrase>Entourage</tra1:KeywordPhrase>
            <tra1:KeywordPhrase>True Blood</tra1:KeywordPhrase>
            <tra1:KeywordPhrase>Curb Your Enthusiasm</tra1:KeywordPhrase>
            <tra1:KeywordPhrase>Treme</tra1:KeywordPhrase>
            <tra1:KeywordPhrase>Generation Kill</tra1:KeywordPhrase>
         </tra1:KeywordPhrases>
         <tra1:PositiveODPCategories>
            <tra1:ODPCategory>Top/Arts/Television/Networks/Cable/HBO</
               tra1:ODPCategory>
            <tra1:ODPCategory>Top/Arts/Television/Programs/Dramas/Sopranos,
               The</tra1:ODPCategory>
         </tra1:PositiveODPCategories>
         <tra1:OverwriteFiles>true</tra1:OverwriteFiles>
      </trainRequest>
   </tra:train>
</soapenv:Body>
</soapenv:Envelope>
```

Listing 5.9: Example train Request

The train operation request will return immediately and inform the user whether the request to start the training process was successful or not. If the request was unsuccessful, a TrainFault response will be returned, see 5.4.2.4 for an example TrainFault. The *trainStatus* operation should then be used to query the progress of the training process. The table below describes the train response parameters and is followed by a listing of an example response.

Table 5.4: train Response Parameters

| Name | Description |
|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string |
| Status | The current status of the training process.<br>Type: xs:string |
| IsTraining | This is *true* if the training process is running, *false* if it is not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:trainResponse xmlns:ns2="http://training.service/" xmlns:ns3="http://
          xml.netbeans.org/schema/TrainingParameters">
         <return>
            <ns3:Topic>HBO_Series_Topic</ns3:Topic>
            <ns3:Status>RUNNING - Generating Seeds</ns3:Status>
            <ns3:IsTraining>true</ns3:IsTraining>
         </return>
      </ns2:trainResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.10: Example train Response

#### 5.4.2.2 trainStatus operation

The trainStatus operation is used to query the progress of the currently running training process. The trainStatus operation has no particular input parameters, see below for a listing of an example request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tra="http://training.service/">
   <soapenv:Header/>
   <soapenv:Body>
      <tra:trainStatus>
         <trainStatusRequest/>
      </tra:trainStatus>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.11: Example trainStatus Request

The trainStatus operation request will return the status of the currently running training process if successful. If the request failed due to an error, a TrainFault response will be returned, see 5.4.2.4 for an example TrainFault. The table below describes the trainStatus response parameters and is followed by a listing of an example response.

Table 5.5: trainStatus Response Parameters

| Name | Description |
|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string |
| Status | The current status of the training process.<br>Type: xs:string |
| IsTraining | This is *true* if the training process is running, *false* if it is not.<br>Type: xs:boolean |
| IsRainbowRunning | This is *true* if the Rainbow classifier process is running, *false* if it is not.<br>Type: xs:boolean |
| SeedURLs | Container for one or more URL elements. |
| URL | A string representing a seed URL.<br>Type: xs:anyURI |
| NegativeODPCategories | Container for one or more ODPCategory elements. |
| ODPCategory | A string representing an ODP Category.<br>Type: xs:string |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:trainStatusResponse xmlns:ns2="http://training.service/" xmlns:ns3="
          http://xml.netbeans.org/schema/TrainingParameters">
         <return>
            <ns3:Topic>HBO_Series_Topic</ns3:Topic>
            <ns3:Status>RUNNING - Generating Negative training set</ns3:Status>
            <ns3:IsTraining>true</ns3:IsTraining>
            <ns3:IsRainbowRunning>false</ns3:IsRainbowRunning>
            <ns3:SeedURLs>
               <ns3:URL>http://www.hbo.com/the-wire/index.html</ns3:URL>
               <ns3:URL>http://www.guardian.co.uk/media/wire</ns3:URL>
               <ns3:URL>http://en.wikipedia.org/wiki/The_Sopranos</ns3:URL>
               <ns3:URL>http://www.sopranoland.com/</ns3:URL>
               <ns3:URL>http://www.hbo.com/entourage/index.html</ns3:URL>
               <ns3:URL>http://entourageventures.com/</ns3:URL>
               <ns3:URL>http://www.hbo.com/true-blood/index.html</ns3:URL>
               <ns3:URL>http://www.amazon.com/True-Blood-Complete-First-Season/dp
                   /B001FB4W0W</ns3:URL>
               <ns3:URL>http://www.hbo.com/curb-your-enthusiasm/index.html</
                   ns3:URL>
               <ns3:URL>http://epguides.com/CurbYourEnthusiasm/</ns3:URL>
               <ns3:URL>http://www.hbo.com/treme/index.html</ns3:URL>
               <ns3:URL>http://latimesblogs.latimes.com/showtracker/2010/06/treme
                   -did-he-really-jump-do-you-really-care.html</ns3:URL>
               <ns3:URL>http://www.hbo.com/generation-kill/index.html</ns3:URL>
               <ns3:URL>http://www.netflix.com/Movie/Generation_Kill/70089223</
                   ns3:URL>
            </ns3:SeedURLs>
            <ns3:NegativeODPCategories>
               <ns3:ODPCategory>Top/Regional/North_America/United_States/Florida/
                   Counties/Liberty/Science_and_Environment</ns3:ODPCategory>
               <ns3:ODPCategory>Top/Regional/North_America/United_States/Arizona/
                   Localities/P/Page/Travel_and_Tourism/Travel_Services/
                   Tour_Operators</ns3:ODPCategory>
               <ns3:ODPCategory>Top/Health/Pharmacy/Drugs_and_Medications/P/
                   Phosphates</ns3:ODPCategory>
               <ns3:ODPCategory>Top/Society/Issues/Health/Research/Fraud</
                   ns3:ODPCategory>
               <ns3:ODPCategory>Top/World/Deutsch/Regional/Europa/Deutschland/
                   Bremen/Stadt_Bremen/Gesellschaft/Menschen/Jugendliche</
                   ns3:ODPCategory>
               <ns3:ODPCategory>Top/Arts/Music/Composition/Composers/By_Region/
                   Europe/German/Q</ns3:ODPCategory>
            </ns3:NegativeODPCategories>
         </return>
      </ns2:trainStatusResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.12: Example trainStatus Response

#### 5.4.2.3 trainTerminate operation

The trainTerminate operation is used to terminate the currently running training process. The trainTerminate operation has no particular input parameters, see below for a listing of an example request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tra="http://training.service/">
   <soapenv:Header/>
   <soapenv:Body>
      <tra:trainTerminate>
         <trainTerminateRequest/>
      </tra:trainTerminate>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.13: Example trainTerminate Request

The trainTerminate request will return a *Success* parameter if successful. If the request failed due to an error, a TrainFault response will be returned, see 5.4.2.4 for an example TrainFault. The table below describes the trainTerminate response parameters and is followed by a listing of an example response.

Table 5.6: trainTerminate Response Parameters

| Name | Description |
|---|---|
| Success | This is *true* if the training process was successfully terminated, *false* if it was not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:trainTerminateResponse xmlns:ns2="http://training.service/" xmlns:ns3=
          "http://xml.netbeans.org/schema/TrainingParameters">
         <return>
            <ns3:Success>true</ns3:Success>
         </return>
      </ns2:trainTerminateResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.14: Example trainTerminate Response

#### 5.4.2.4 TrainFault

A TrainFault response will be returned by all of the TrainingService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example TrainFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Service error</faultstring>
         <detail>
            <ns2:TrainFault xmlns:ns2="http://training.service/" xmlns:ns3="http:
                //xml.netbeans.org/schema/TrainingParameters">
               <message>The Training service is currently running. Please invoke
                   service with OverwriteFiles set to true to override and start
                   a new training process.</message>
            </ns2:TrainFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.15: Example TrainFault Response

### 5.4.3 TuningMgmtService

For the full TuningMgmtService WSDL specification see Appendix C.3. The TuningMgmtService has only one operation: *tuneMgmt*.

#### 5.4.3.1 tuneMgmt operation

The tuneMgmt operation is used for setting the management configuration parameters of the TuningService. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.7: tuneMgmt Request Parameters

| Name | Description | Required |
|---|---|---|
| TopWordCount | The number of Top Words returned by the Tuning Service getTopWords operation.<br>Type: xs:int<br>Default: 200<br>Valid Values: An int greater than 0. | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tun="http://tuningMgmt.service/" xmlns:tun1="http://xml.netbeans.org/
    schema/TuningMgmtParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <tun:tuneMgmt>
         <tuneMgmtRequest>
            <tun1:TopWordCount>10</tun1:TopWordCount>
         </tuneMgmtRequest>
      </tun:tuneMgmt>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.16: Example tuneMgmt Request

The tuneMgmt operation will return with the input parameters set if it was successful, if not a TuneMgmtFault response will be returned, see 5.4.3.2 for an example TuneMgmtFault. The table below describes the tuneMgmt response parameters and is followed by a listing of an example response.

Table 5.8: tuneMgmt Response Parameters

| Name | Description |
|---|---|
| TopWordCount | The TopWordCount set in the request.<br>Type: xs:int |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:tuneMgmtResponse xmlns:ns2="http://xml.netbeans.org/schema/
          TuningMgmtParameters" xmlns:ns3="http://tuningMgmt.service/">
         <return>
            <ns2:TopWordCount>10</ns2:TopWordCount>
         </return>
      </ns3:tuneMgmtResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.17: Example tuneMgmt Response

#### 5.4.3.2 TuneMgmtFault

A TuneMgmtFault response will be returned by all of the TuningMgmtService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example TuneMgmtFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Validation error</faultstring>
         <detail>
            <ns3:TuneMgmtFault xmlns:ns3="http://tuningMgmt.service/" xmlns:ns2="
                http://xml.netbeans.org/schema/TuningMgmtParameters">
               <message>Invalid TopWordCount. Please enter a value greater than
                   zero.</message>
            </ns3:TuneMgmtFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.18: Example TuneMgmtFault Response

### 5.4.4 TuningService

For the full TuningService WSDL specification see Appendix C.4. The TuningService has 3 operations: *getTopWords*, *setStopWords* and *rainbowTerminate*.

#### 5.4.4.1 getTopWords operation

The getTopWords operation returns a configurable number of the top words for the document model generated by the Rainbow classifier once the training process has been completed. The number of top words returned in configured by using the *tuneMgmt* operation of the TuningMgmtService. The getTopWords operation can only be run after the training process has completed successfully. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.9: getTopWords Request Parameters

| Name | Description | Required |
|---|---|---|
| OverwriteFiles | Set to *true* if a currently running crawling or indexing process should be killed and existing crawling or indexing data overwritten, if set to *false* and a crawling or indexing process is currently running a TuneFault will be thrown.<br>Type: xs:boolean<br>Default: false<br>Valid Values: true or false | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tun="http://tuning.service/" xmlns:tun1="http://xml.netbeans.org/schema
    /TuningParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <tun:getTopWords>
         <getTopWordsRequest>
            <tun1:OverwriteFiles>true</tun1:OverwriteFiles>
         </getTopWordsRequest>
      </tun:getTopWords>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.19: Example getTopWords Request

The getTopWords operation request will return the document model top words if successful. If the request was unsuccessful, a TuneFault response will be returned, see 5.4.4.4 for an example TuneFault. The table below describes the getTopWords response parameters and is followed by a listing of an example response.

Table 5.10: getTopWords Response Parameters

| Name | Description |
|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string |
| TopWords | Container for one or more TopWord elements. |
| TopWord | A string representing a top word.<br>Type: xs:string |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:getTopWordsResponse xmlns:ns2="http://xml.netbeans.org/schema/
          TuningParameters" xmlns:ns3="http://tuning.service/">
         <return>
            <ns2:Topic>HBO_Series_Topic</ns2:Topic>
            <ns2:TopWords>
               <ns2:TopWord>episode</ns2:TopWord>
               <ns2:TopWord>tv</ns2:TopWord>
               <ns2:TopWord>season</ns2:TopWord>
               <ns2:TopWord>show</ns2:TopWord>
               <ns2:TopWord>blood</ns2:TopWord>
               <ns2:TopWord>true</ns2:TopWord>
               <ns2:TopWord>episodes</ns2:TopWord>
               <ns2:TopWord>ago</ns2:TopWord>
               <ns2:TopWord>curb</ns2:TopWord>
               <ns2:TopWord>series</ns2:TopWord>
            </ns2:TopWords>
         </return>
      </ns3:getTopWordsResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.20: Example getTopWords Response

#### 5.4.4.2 setStopWords operation

If one or more of the top words returned by the *getTopWords* should in fact be stop words, the setTopWords operation can be used to update the document model so this is the case. As is the case for the *getTopWords* operation, the setStopWords operation can only be run after the training process has completed successfully. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.11: setStopWords Request Parameters

| Name | Description | Required |
|---|---|---|
| OverwriteFiles | Set to *true* if a currently running crawling or indexing process should be killed and existing crawling or indexing data overwritten, if set to *false* and a crawling or indexing process is currently running a TuneFault will be thrown.<br>Type: xs:boolean<br>Default: false<br>Valid Values: true or false | No |
| StopWords | Container for one or more StopWord elements. | Yes |
| StopWord | A string representing a stop word.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid top word returned by the getTopWords operation. | Yes |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tun="http://tuning.service/" xmlns:tun1="http://xml.netbeans.org/schema
    /TuningParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <tun:setStopWords>
         <setStopWordsRequest>
            <tun1:OverwriteFiles>false</tun1:OverwriteFiles>
            <tun1:StopWords>
               <tun1:StopWord>ago</tun1:StopWord>
            </tun1:StopWords>
         </setStopWordsRequest>
      </tun:setStopWords>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.21: Example setStopWords Request

The setStopWords operation request will return the document model top words and stop words if successful. If the request was unsuccessful, a TuneFault response will be returned, see 5.4.4.4 for an example TuneFault. The table below describes the setStopWords response parameters and is followed by a listing of an example response.

Table 5.12: setStopWords Response Parameters

| Name | Description |
|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string |
| TopWords | Container for one or more TopWord elements. |
| TopWord | A string representing a top word.<br>Type: xs:string |
| StopWords | Container for one or more StopWord elements. |
| StopWord | A string representing a stop word.<br>Type: xs:string |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:setStopWordsResponse xmlns:ns2="http://xml.netbeans.org/schema/
          TuningParameters" xmlns:ns3="http://tuning.service/">
         <return>
            <ns2:Topic>HBO_Series_Topic</ns2:Topic>
            <ns2:TopWords>
               <ns2:TopWord>episode</ns2:TopWord>
               <ns2:TopWord>tv</ns2:TopWord>
               <ns2:TopWord>season</ns2:TopWord>
               <ns2:TopWord>show</ns2:TopWord>
               <ns2:TopWord>blood</ns2:TopWord>
               <ns2:TopWord>true</ns2:TopWord>
               <ns2:TopWord>episodes</ns2:TopWord>
               <ns2:TopWord>curb</ns2:TopWord>
               <ns2:TopWord>series</ns2:TopWord>
            </ns2:TopWords>
            <ns2:StopWords>
               <ns2:StopWord>ago</ns2:StopWord>
            </ns2:StopWords>
         </return>
      </ns3:setStopWordsResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.22: Example setStopWords Response

#### 5.4.4.3 rainbowTerminate operation

The rainbowTerminate operation is used to terminate the currently running Rainbow classification process. The rainbowTerminate operation has no particular input parameters, see below for a listing of an example request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tun="http://tuning.service/">
   <soapenv:Header/>
   <soapenv:Body>
      <tun:rainbowTerminate>
         <rainbowTerminateRequest/>
      </tun:rainbowTerminate>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.23: Example rainbowTerminate request

The rainbowTerminate request will return a *Success* parameter if successful. If the request failed due to an error, a TuneFault response will be returned, see 5.4.4.4 for an example TuneFault. The table below describes the rainbowTerminate response parameters and is followed by a listing of an example response.

Table 5.13: rainbowTerminate Response Parameters

| Name | Description |
|---|---|
| Success | This is *true* if the Rainbow classifier process was successfully terminated, *false* if it was not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:rainbowTerminateResponse xmlns:ns2="http://xml.netbeans.org/schema/
          TuningParameters" xmlns:ns3="http://tuning.service/">
         <return>
            <ns2:Success>true</ns2:Success>
         </return>
      </ns3:rainbowTerminateResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.24: Example rainbowTerminate response

#### 5.4.4.4 TuneFault

A TuneFault response will be returned by all of the TuningService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example TuneFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Service error</faultstring>
         <detail>
            <ns3:TuneFault xmlns:ns3="http://tuning.service/" xmlns:ns2="http://
                xml.netbeans.org/schema/TuningParameters">
               <message>The Training service is currently running. Please wait
                   until it is finished before running the Tuning Service.</
                   message>
            </ns3:TuneFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.25: Example TuneFault Response

### 5.4.5 CrawlingMgmtService

For the full CrawlingMgmtService WSDL specification see Appendix C.5. The CrawlingMgmtService has only one operation: *crawlMgmt*.

#### 5.4.5.1 crawlMgmt operation

The crawlMgmt operation is used for setting the management configuration parameters of the CrawlingService. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.14: crawlMgmt Request Parameters

| Name | Description | Required |
|---|---|---|
| MaxToeThreads | Maximum number of threads crawler will use to process Uniform Resource Identifiers (URIs).<br>Type: xs:integer<br>Default: 90<br>Valid Values: An integer greater than 0. | No |
| MaxLinkHops | Maximum number of links followed to get to the current URI.<br>Type: xs:integer<br>Default: 25<br>Valid Values: An integer greater than 0. | No |
| MaxTransHops | Maximum number of non link hops at the end of the current URI's Discovery path.<br>Type: xs:integer<br>Default: 5<br>Valid Values: An integer greater than 0. | No |
| HttpProxyHost | A string representing the IP address of the http proxy host.<br>Type: xs:string<br>Default: 134.226.32.57<br>Valid Values: A valid IP address. | No |
| HttpProxyPort | A number representing the http proxy port.<br>Type: xs:int<br>Default: 8080<br>Valid Values: An int in the range of 0 to 65535. | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawlingMgmt.service/" xmlns:craw1="http://xml.netbeans.
    org/schema/CrawlingMgmtParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawlMgmt>
         <crawlMgmtRequest>
            <craw1:MaxToeThreads>90</craw1:MaxToeThreads>
            <craw1:MaxLinkHops>25</craw1:MaxLinkHops>
            <craw1:MaxTransHops>5</craw1:MaxTransHops>
            <craw1:HttpProxyHost>134.226.32.57</craw1:HttpProxyHost>
            <craw1:HttpProxyPort>8080</craw1:HttpProxyPort>
         </crawlMgmtRequest>
      </craw:crawlMgmt>
   </soapenv:Body>
```

```
</soapenv:Envelope>
```

Listing 5.26: Example crawlMgmt request

The crawlMgmt operation will return with the input parameters set if it was successful, if not a CrawlMgmtFault response will be returned, see 5.4.5.2 for an example CrawlMgmtFault. The table below describes the crawlMgmt response parameters and is followed by a listing of an example response.

Table 5.15: crawlMgmt Response Parameters

| Name | Description |
|---|---|
| MaxToeThreads | The MaxToeThreads set in request.<br>Type: xs:integer |
| MaxLinkHops | The MaxLinkHops set in request.<br>Type: xs:integer |
| MaxTransHops | The MaxTransHops set in request.<br>Type: xs:integer |
| HttpProxyHost | The HttpProxyHost set in request.<br>Type: xs:string |
| HttpProxyPort | The HttpProxyPort set in request.<br>Type: xs:int |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlMgmtResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingMgmtParameters" xmlns:ns3="http://crawlingMgmt.service/">
         <return>
            <ns2:MaxToeThreads>50</ns2:MaxToeThreads>
            <ns2:MaxLinkHops>20</ns2:MaxLinkHops>
            <ns2:MaxTransHops>3</ns2:MaxTransHops>
            <ns2:HttpProxyHost>134.226.32.57</ns2:HttpProxyHost>
            <ns2:HttpProxyPort>8080</ns2:HttpProxyPort>
         </return>
      </ns3:crawlMgmtResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.27: Example crawlMgmt response

#### 5.4.5.2 CrawlMgmtFault

A CrawlMgmtFault response will be returned by all of the CrawlingMgmtService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example CrawlMgmtFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Validation error</faultstring>
         <detail>
            <ns3:CrawlMgmtFault xmlns:ns3="http://crawlingMgmt.service/"
                xmlns:ns2="http://xml.netbeans.org/schema/CrawlingMgmtParameters"
                >
               <message>Invalid MaxToeThreads. Please enter a value greater than
                   zero.</message>
            </ns3:CrawlMgmtFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.28: Example CrawlMgmtFault response

### 5.4.6 CrawlingService

For the full CrawlingService WSDL specification see Appendix C.6. The CrawlingService has 6 operations: *crawl*, *crawlPause*, *crawlResume*, *crawlStatus*, *crawlTerminate* and *crawlerShutdown*.

#### 5.4.6.1 crawl operation

The crawl operation is used to start a new crawling process. It can only be run after the training process and optionally the tuning process have completed successfully. It can be set to crawl continuously, or it can be set to crawl until a certain number or documents or amount of data has been downloaded, or a certain amount of time has passed. The crawler uses a language classifier which will only accept a downloaded document if it is determined to be the configured language. The Rainbow classifier will then be used to determine if a downloaded document should be accepted or rejected based on its similarity to the document model generated during the training and tuning processes. The tables below and on the following page describe the request parameters and are followed by a listing of an example request.

Table 5.16: crawl Request Parameters

| Name | Description | Required |
|---|---|---|
| JobName | A string respresenting the crawl job name.<br>Type: xs:string<br>Default: None<br>Valid Values: Any alphanumeric or underscore characters up to a maximum length of 38. | Yes |
| Organisation | A string respresenting the organization running a crawl.<br>Type: xs:string<br>Default: None<br>Valid Values: a non-empty string. | Yes |
| OrganisationURL | A string respresenting the website that webmasters can go to to view information on the organization or person running a crawl.<br>Type: xs:anyURI<br>Default: None<br>Valid Values: a valid URL. | Yes |
| EmailAddress | A string respresenting the email address of the person or organisation running a crawl.<br>Type: xs:string<br>Default: None<br>Valid Values: a valid email address. | Yes |
| MaxBytesDownload | Stop crawl after a fixed number of bytes have been downloaded. 0 means unlimited.<br>Type: xs:long<br>Default: 0<br>Valid Values: A long greater than 0. | No |
| MaxDocumentDownload | Stop crawl after downloading a fixed number of documents. 0 means unlimited.<br>Type: xs:long<br>Default: 0<br>Valid Values: A long greater than 0. | No |

Table 5.17: crawl Request Parameters, continued...

| Name | Description | Required |
|---|---|---|
| MaxTimeSec | Stop crawl after a certain number of seconds have elapsed. 0 means unlimited.<br>Type: xs:long<br>Default: 0<br>Valid Values: A long greater than 0. | No |
| MaxPathDepth | Maximum path depth of the current URI's Discovery path.<br>Type: xs:integer<br>Default: 20<br>Valid Values: An integer greater than 0. | No |
| CutOff | The cut off value above which the Rainbow classifier will accept a document.<br>Type: xs:double<br>Default: 0.95<br>Valid Values: A double value in the range of 0.0 to 1.0 | No |
| Language | The language of documents the text classifier will accept.<br>Type: xs:string<br>Default: english<br>Valid Values: albanian, danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, polish, slovakian, slovenian, spanish, or swedish. | No |
| OverwriteFiles | Set to *true* if a currently running crawling/indexing process should be killed and crawling/indexing data overwritten, if set to *false* and a crawling/indexing process is currently running a CrawlFault will be thrown.<br>Type: xs:boolean<br>Default: false<br>Valid Values: true or false | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawling.service/" xmlns:craw1="http://xml.netbeans.org/
    schema/CrawlingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawl>
         <crawlRequest>
            <craw1:JobName>HBO_series_job</craw1:JobName>
            <craw1:Organisation>TCD</craw1:Organisation>
            <craw1:OrganisationURL>http://www.tcd.ie</craw1:OrganisationURL>
            <craw1:EmailAddress>phannon@tcd.ie</craw1:EmailAddress>
            <craw1:MaxBytesDownload>0</craw1:MaxBytesDownload>
            <craw1:MaxDocumentDownload>5000</craw1:MaxDocumentDownload>
            <craw1:MaxTimeSec>0</craw1:MaxTimeSec>
            <craw1:MaxPathDepth>10</craw1:MaxPathDepth>
            <craw1:CutOff>0.97</craw1:CutOff>
            <craw1:Language>english</craw1:Language>
            <craw1:OverwriteFiles>false</craw1:OverwriteFiles>
         </crawlRequest>
      </craw:crawl>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.29: Example crawl request

The crawl operation request will return a *JobUID* if successful. This uniquely identifies the crawl job and is used as a request parameter for the *crawlPause*, *crawlResume*, *crawlStatus*, and *crawlTerminate* operations. If the request was unsuccessful, a CrawlFault response will be returned, see 5.4.6.7 for an example CrawlFault. The table below describes the crawl response parameters and is followed by a listing of an example response.

Table 5.18: crawl Response Parameters

| Name | Description |
|---|---|
| JobName | The JobName set in request.<br>Type: xs:string |
| JobUID | The Job UID of the newly created crawl job.<br>Type: xs:string |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingParameters" xmlns:ns3="http://crawling.service/">
         <return>
            <ns2:JobName>HBO_series_job</ns2:JobName>
            <ns2:JobUID>20100822123845061</ns2:JobUID>
         </return>
      </ns3:crawlResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.30: Example crawl response

#### 5.4.6.2 crawlPause operation

The crawlPause operation is used to pause the crawl process for a given *JobUID*. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.19: crawlPause Request Parameters

| Name | Description | Required |
|---|---|---|
| JobUID | The job UID of the crawl job to be paused.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid Job UID. | Yes |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawling.service/" xmlns:craw1="http://xml.netbeans.org/
    schema/CrawlingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawlPause>
         <crawlPauseRequest>
            <craw1:JobUID>20100822123845061</craw1:JobUID>
         </crawlPauseRequest>
      </craw:crawlPause>
```

```
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.31: Example crawlPause request

The crawlPause operation request will return the current status of the crawl job if successful. If the request was unsuccessful, a CrawlFault response will be returned, see 5.4.6.7 for an example CrawlFault. The table below describes the crawlPause response parameters and is followed by a listing of an example response.

Table 5.20: crawlPause Response Parameters

| Name | Description |
|---|---|
| JobUID | The JobUID in the request.<br>Type: xs:string |
| Status | The status of the crawl job.<br>Type: xs:string |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlPauseResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingParameters" xmlns:ns3="http://crawling.service/">
         <return>
            <ns2:JobUID>20100822123845061</ns2:JobUID>
            <ns2:Status>PAUSING</ns2:Status>
         </return>
      </ns3:crawlPauseResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.32: Example crawlPause response

#### 5.4.6.3 crawlResume operation

The crawlResume operation is used to resume the crawl process for a given *JobUID*. The job will have been previously paused using the *crawlPause* operation. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.21: crawlResume Request Parameters

| Name | Description | Required |
|---|---|---|
| JobUID | The job UID of the crawl job to be resumed.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid Job UID. | Yes |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawling.service/" xmlns:craw1="http://xml.netbeans.org/
    schema/CrawlingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawlResume>
```

```
         <crawlResumeRequest>
            <craw1:JobUID>20100822123845061</craw1:JobUID>
         </crawlResumeRequest>
      </craw:crawlResume>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.33: Example crawlResume request

The crawlResume operation request will return the current status of the crawl job if successful. If the request was unsuccessful, a CrawlFault response will be returned, see 5.4.6.7 for an example CrawlFault. The table below describes the crawlResume response parameters and is followed by a listing of an example response.

Table 5.22: crawlResume Response Parameters

| Name | Description |
|---|---|
| JobUID | The JobUID in the request.<br>Type: xs:string |
| Status | The status of the crawl job.<br>Type: xs:string |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlResumeResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingParameters" xmlns:ns3="http://crawling.service/">
         <return>
            <ns2:JobUID>20100822123845061</ns2:JobUID>
            <ns2:Status>RUNNING</ns2:Status>
         </return>
      </ns3:crawlResumeResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.34: Example crawlResume response

#### 5.4.6.4 crawlStatus operation

The crawlStatus operation is used query the status of the crawl process for a given *JobUID*. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.23: crawlStatus Request Parameters

| Name | Description | Required |
|---|---|---|
| JobUID | The job UID of the crawl job to return the status of.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid Job UID. | Yes |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawling.service/" xmlns:craw1="http://xml.netbeans.org/
    schema/CrawlingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawlStatus>
         <crawlStatusRequest>
            <craw1:JobUID>20100822123845061</craw1:JobUID>
         </crawlStatusRequest>
      </craw:crawlStatus>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.35: Example crawlStatus request

The crawlStatus operation request will return the status as well as detailed information on the progress of the crawl job if successful. If the request was unsuccessful, a CrawlFault response will be returned, see 5.4.6.7 for an example CrawlFault. The table on the following page describes the crawlStatus response parameters and is followed by a listing of an example response.

Table 5.24: crawlStatus Response Parameters

| Name | Description |
|---|---|
| JobUID | The JobUID in the request.<br>Type: xs:string |
| Status | The status of the crawl job.<br>Type: xs:string |
| IsCrawling | This is *true* if the crawler is crawling, *false* if it is not.<br>Type: xs:boolean |
| TotalData | The total data written in bytes.<br>Type: xs:long |
| CrawlTime | The time spent crawling in seconds.<br>Type: xs:long |
| DiscoveredDocuments | The number of URIs discovered to date.<br>Type: xs:long |
| QueuedDocuments | The number of URIs currently queued.<br>Type: xs:long |
| DownloadedDocuments | The number of URIs downloaded to date.<br>Type: xs:long |
| DownloadFailures | The number of URIs that have failed to download.<br>Type: xs:long |
| MemoryUseKB | The amount of memory in kilobytes currently in use by the Java Virtual Machine (JVM).<br>Type: xs:long |
| HeapSizeKB | The current heap size in kilobytes of the JVM.<br>Type: xs:long |
| ActiveThreadCount | The number of toe threads currently busy processing a URI.<br>Type: xs:int |
| CurrentDocumentRate | The current document download rate in documents per second.<br>Type: xs:double |
| AverageDocumentRate | The average document download rate in documents per second.<br>Type: xs:double |
| CurrentKBRate | The current download data rate in kilobytes per second.<br>Type: xs:long |
| AverageKBRate | The average download data rate in kilobytes per second.<br>Type: xs:long |
| Congestion | The congestion ratio is a rough estimate of how much initial capacity, as a multiple of current capacity, would be necessary to crawl the current workload at the maximum rate available.<br>Type: xs:double |
| MaxDepth | The size of the Frontier queue with the largest number of queued URLs.<br>Type: xs:long |
| AverageDepth | The average size of all the Frontier queues.<br>Type: xs:int |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlStatusResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingParameters" xmlns:ns3="http://crawling.service/">
         <return>
            <ns2:JobUID>20100822123845061</ns2:JobUID>
            <ns2:Status>Finished - Maximum number of documents limit hit</
                ns2:Status>
            <ns2:IsCrawling>false</ns2:IsCrawling>
            <ns2:TotalData>83397308</ns2:TotalData>
            <ns2:CrawlTime>90</ns2:CrawlTime>
            <ns2:DiscoveredDocuments>55450</ns2:DiscoveredDocuments>
            <ns2:QueuedDocuments>48037</ns2:QueuedDocuments>
            <ns2:DownloadedDocuments>5050</ns2:DownloadedDocuments>
            <ns2:DownloadFailures>23</ns2:DownloadFailures>
            <ns2:MemoryUseKB>142603</ns2:MemoryUseKB>
            <ns2:HeapSizeKB>190180</ns2:HeapSizeKB>
            <ns2:ActiveThreadCount>0</ns2:ActiveThreadCount>
            <ns2:CurrentDocumentRate>0.0</ns2:CurrentDocumentRate>
            <ns2:AverageDocumentRate>56.11</ns2:AverageDocumentRate>
            <ns2:CurrentKBRate>0</ns2:CurrentKBRate>
            <ns2:AverageKBRate>904</ns2:AverageKBRate>
            <ns2:Congestion>0.0</ns2:Congestion>
            <ns2:MaxDepth>2047</ns2:MaxDepth>
            <ns2:AverageDepth>16</ns2:AverageDepth>
         </return>
      </ns3:crawlStatusResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.36: Example crawlStatus response

#### 5.4.6.5 crawlTerminate operation

The crawlTerminate operation is used to terminate the crawl process for a given *JobUID*. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.25: crawlTerminate Request Parameters

| Name | Description | Required |
|---|---|---|
| JobUID | The job UID of the crawl job to be terminated.<br>Type: xs:string<br>Default: None<br>Valid Values: A valid Job UID. | Yes |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawling.service/" xmlns:craw1="http://xml.netbeans.org/
    schema/CrawlingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawlTerminate>
         <crawlTerminateRequest>
```

```
            <craw1:JobUID>20100822134730801</craw1:JobUID>
         </crawlTerminateRequest>
      </craw:crawlTerminate>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.37: Example crawlTerminate request

The crawlTerminate operation request will return a *Success* parameter if successful. If the request failed due to an error, a CrawlFault response will be returned, see 5.4.6.7 for an example CrawlFault. The table below describes the crawlTerminate response parameters and is followed by a listing of an example response.

Table 5.26: crawlTerminate Response Parameters

| Name | Description |
|---|---|
| JobUID | The JobUID in the request.<br>Type: xs:string |
| Success | This is *true* if the crawl job was successfully terminated, *false* if it was not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlTerminateResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingParameters" xmlns:ns3="http://crawling.service/">
         <return>
            <ns2:JobUID>20100822134730801</ns2:JobUID>
            <ns2:Success>true</ns2:Success>
         </return>
      </ns3:crawlTerminateResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.38: Example crawlTerminate response

#### 5.4.6.6 crawlerShutdown operation

The crawlerShutdown operation will shutdown the crawler process itself and its associated web interface. Any currently running crawl job will first be terminated. The crawlerShutdown operation has no particular input parameters, see below for a listing of an example request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:craw="http://crawling.service/">
   <soapenv:Header/>
   <soapenv:Body>
      <craw:crawlerShutdown>
         <crawlerShutdownRequest/>
      </craw:crawlerShutdown>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.39: Example crawlerShutdown request

The crawlerShutdown request will return a *Success* parameter if successful. If the request failed due to an error, a CrawlFault response will be returned, see 5.4.6.7 for an example CrawlFault. The table below describes the crawlerShutdown response parameters and is followed by a listing of an example response.

Table 5.27: crawlerShutdown Response Parameters

| Name | Description |
|---|---|
| Success | This is *true* if the crawler was successfully shutdown, *false* if it was not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:crawlerShutdownResponse xmlns:ns2="http://xml.netbeans.org/schema/
          CrawlingParameters" xmlns:ns3="http://crawling.service/">
         <return>
            <ns2:Success>true</ns2:Success>
         </return>
      </ns3:crawlerShutdownResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.40: Example crawlerShutdown response

#### 5.4.6.7 CrawlFault

A CrawlFault response will be returned by all of the CrawlingService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example CrawlFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Validation error</faultstring>
         <detail>
            <ns3:CrawlFault xmlns:ns3="http://crawling.service/" xmlns:ns2="http:
                //xml.netbeans.org/schema/CrawlingParameters">
               <message>Invalid JobUID, please enter only 17 digits.</message>
            </ns3:CrawlFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.41: Example CrawlFault response

### 5.4.7 IndexingService

For the full IndexingService WSDL specification see Appendix C.7. The IndexingService has 3 operations: *index*, *indexStatus* and *indexTerminate*.

#### 5.4.7.1 index operation

The index operation will index the ARC files which are output as a result of the crawl process. It can only be run after the crawl process has finished successfully. The table below describes the request parameters and is followed by a listing of an example request.

Table 5.28: index Request Parameters

| Name | Description | Required |
|---|---|---|
| VirtualCache | Set to *true* if crawl ARC files should be deleted after indexing is complete, if set to *false* the ARC files will not be deleted.<br>Type: xs:boolean<br>Default: false<br>Valid Values: true or false | No |
| OverwriteFiles | Set to *true* if a currently running indexing process should be killed and existing index data overwritten, if set to *false* and an indexing process is currently running an IndexFault will be thrown.<br>Type: xs:boolean<br>Default: false<br>Valid Values: true or false | No |

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ind="http://indexing.service/" xmlns:ind1="http://xml.netbeans.org/
    schema/IndexingParameters">
   <soapenv:Header/>
   <soapenv:Body>
      <ind:index>
         <indexRequest>
            <ind1:VirtualCache>true</ind1:VirtualCache>
            <ind1:OverwriteFiles>false</ind1:OverwriteFiles>
         </indexRequest>
      </ind:index>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.42: Example index request

The index operation request will return immediately and inform the user whether the request to start the indexing process was successful or not. If the request was unsuccessful, an IndexFault response will be returned, see 5.4.7.4 for an example IndexFault. The *indexStatus* operation should then be used to query the progress of the indexing process. The table below describes the index response parameters and is followed by a listing of an example response.

Table 5.29: index Response Parameters

| Name | Description |
|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string |
| Status | The current status of the indexing process.<br>Type: xs:string |
| IsIndexing | This is *true* if the indexing process is running, *false* if it is not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:indexResponse xmlns:ns2="http://indexing.service/" xmlns:ns3="http://
          xml.netbeans.org/schema/IndexingParameters">
         <return>
            <ns3:Topic>HBO_Series_Topic</ns3:Topic>
            <ns3:Status>RUNNING</ns3:Status>
            <ns3:IsIndexing>true</ns3:IsIndexing>
         </return>
      </ns2:indexResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.43: Example index response

#### 5.4.7.2 indexStatus operation

The indexStatus operation is used to query the progress of the currently running indexing process. The indexStatus operation has no particular input parameters, see below for a listing of an example request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ind="http://indexing.service/">
   <soapenv:Header/>
   <soapenv:Body>
      <ind:indexStatus>
         <indexStatusRequest/>
      </ind:indexStatus>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.44: Example indexStatus request

The indexStatus operation request will return the status of the currently running indexing process if successful. If the indexStatus request failed due to an error an IndexFault response will be returned,

see 5.4.7.4 for an example IndexFault. The table below describes the indexStatus response parameters and is followed by a listing of an example response.

Table 5.30: indexStatus Response Parameters

| Name | Description |
|---|---|
| Topic | A string describing the topic being searched.<br>Type: xs:string |
| Status | The current status of the indexing process.<br>Type: xs:string |
| IsIndexing | This is *true* if the indexing process is running, *false* if it is not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:indexStatusResponse xmlns:ns2="http://indexing.service/" xmlns:ns3="
          http://xml.netbeans.org/schema/IndexingParameters">
         <return>
            <ns3:Topic>HBO_Series_Topic</ns3:Topic>
            <ns3:Status>FINISHED</ns3:Status>
            <ns3:IsIndexing>false</ns3:IsIndexing>
         </return>
      </ns2:indexStatusResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.45: Example indexStatus response

#### 5.4.7.3 indexTerminate operation

The indexTerminate operation is used to terminate the currently running indexing process. The indexTerminate operation has no particular input parameters, see below for a listing of an example request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ind="http://indexing.service/">
   <soapenv:Header/>
   <soapenv:Body>
      <ind:indexTerminate>
         <indexTerminateRequest/>
      </ind:indexTerminate>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 5.46: Example indexTerminate request

The indexTerminate request will return a *Success* parameter if successful. If the request failed due to an error, an IndexFault response will be returned, see 5.4.7.4 for an example IndexFault. The table below describes the indexTerminate response parameters and is followed by a listing of an example response.

Table 5.31: indexTerminate Response Parameters

| Name | Description |
|---|---|
| Success | This is *true* if the indexing process was successfully terminated, *false* if it was not.<br>Type: xs:boolean |

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:indexTerminateResponse xmlns:ns2="http://indexing.service/" xmlns:ns3=
          "http://xml.netbeans.org/schema/IndexingParameters">
         <return>
            <ns3:Success>true</ns3:Success>
         </return>
      </ns2:indexTerminateResponse>
   </S:Body>
</S:Envelope>
```

Listing 5.47: Example indexTerminate Response

#### 5.4.7.4 IndexFault

An IndexFault response will be returned by all of the IndexingService operations in the case of an error. It contains a *faultstring* indicating the type of error and a *message* which gives further information on the error. See the listing below for an example IndexFault response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Request error</faultstring>
         <detail>
            <ns2:IndexFault xmlns:ns2="http://indexing.service/" xmlns:ns3="http:
                //xml.netbeans.org/schema/IndexingParameters">
               <message>The Indexing process is not currently running</message>
            </ns2:IndexFault>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

Listing 5.48: Example IndexFault response

# Chapter 6

# Evaluation of the FCGS

## 6.1 Introduction

This chapter will discuss how the contribution of the author was evaluated. It will not discuss an evaluation of the actual quality of the content caches generated by the FCGS. The quality of the content caches generated by the OCCS has already been evaluated [42]. The objectives of the research for this thesis were derived from the overall research question which was posed in Chapter 1. These objectives were:

1. Investigate the current state of the art in techniques and technologies of web IR.
2. Analyse and determine a full set of requirements and inter-dependencies for exposing each component of an FCGS as web services.
3. Research and develop prototype web services to demonstrate the feasibility of developing an FCGS using Open Source software.
   - The web services should be self configuring and allow intelligent management.
4. Investigate performance and integration issues that arise in the building of these web services.
5. Determine the feasibility of generating a repository of virtual links instead of physical content.

From these research objectives were derived the following set of evaluation objectives:

1. Determine the feasibility of generating physical and virtual content caches using the FCGS.
2. Evaluate the usability of the FCGS in creating content caches.
3. Evaluate the manageability of the FCGS in creating content caches.
4. Investigate the performance of the FCGS.

In addition to the objectives above, a comparative evaluation could have been undertaken comparing existing focused content generation systems with the FCGS. Currently none of the open source tools surveyed in Chapter 3 are accessible via web services. There are plans to extend the Heritrix [7] crawler with a REST web service API [8], however there are currently only draft interfaces specified and no development has occurred. There are also plans to extend the Nutch [18] crawler/indexer with web service interfaces by various universities and organisations, but there has been no open source software released as of yet. As discussed in Subsection 3.3.5 there are tools available which support the generation of focused content caches, but none support the full "end to end" suite of services necessary and are also web accessible. Thus it can be said that the FCGS is the only open source system of its kind, to provide the services it does that are also web accessible.

Section 6.2 discusses the approach taken to accomplish the first evaluation objective. Section 6.3 describes the procedure followed to evaluate the second and third objectives. Finally Section 6.4 details the steps taken to investigate the last objective.

## 6.2 Demonstrator Client Application

To evaluate the feasibility of generating physical and virtual content caches, it was necessary to build a demonstrator application which utilised the FCGS services to generate both physical and virtual content caches. A simple Java command-line client was developed. Taking user input it invokes each of the FCGS services in turn. First the management operation is invoked, then the tool operation. The tool status operation is used to poll the progress of the tool at 1 minute intervals. See Figure 6.1 for an activity diagram of the demonstrator application. The diagram illustrates the order of invocation of the FCGS services and also which service the activity states are invoking. When invoking the *index* operation (see 5.4.7.1) of the Indexing Service, the user is allowed to specify on the command-line if the cache should be virtual or physical (which causes the request parameter *VirtualCache* to be set to true or false). The demonstrator application was used to create both virtual and physical focused content caches on the topic: *Hilton Hotels*. It was verified that for the virtual cache, only a repository of links was generated and the downloaded content had been deleted. The index generated was then verified by conducting several searches using the NutchWax [19] web application deployed on Apache Tomcat.

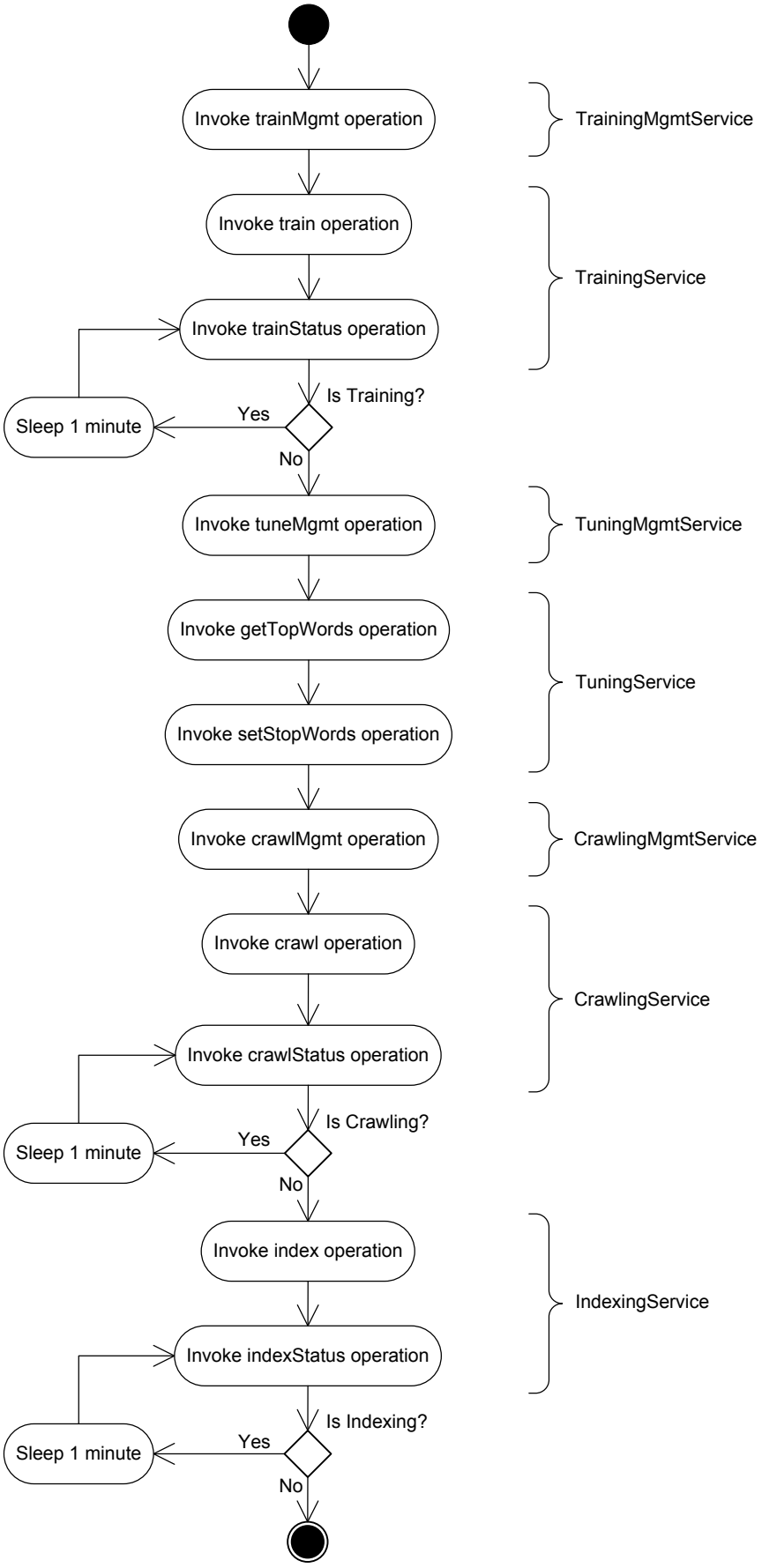



Figure 6.1: Demonstrator Application activity diagram

## 6.3 User Test

In order to evaluate the usability and manageability of the FCGS in creating content caches, a small scale user test was conducted. The users were given the demonstrator client application discussed in Section 6.2 and an abridged version of the API documentation that was presented in Section 5.4. The users were asked to develop a client application using different technologies, and based on this experience to complete a questionnaire.

The questionnaire can be seen in Appendix D.1 and the full set of respondent answers can be seen in Appendix D.2. The questions typically specify a statement to which the user can answer using a 5-point Likert scale, there are also questions where users are given the opportunity to comment on features they value as well as suggest improvements. The Likert scale is the most widely used scale in survey research. It is a type of psychometric response scale, when responding to a Likert questionnaire item, respondents indicate their level of agreement with a statement [43].

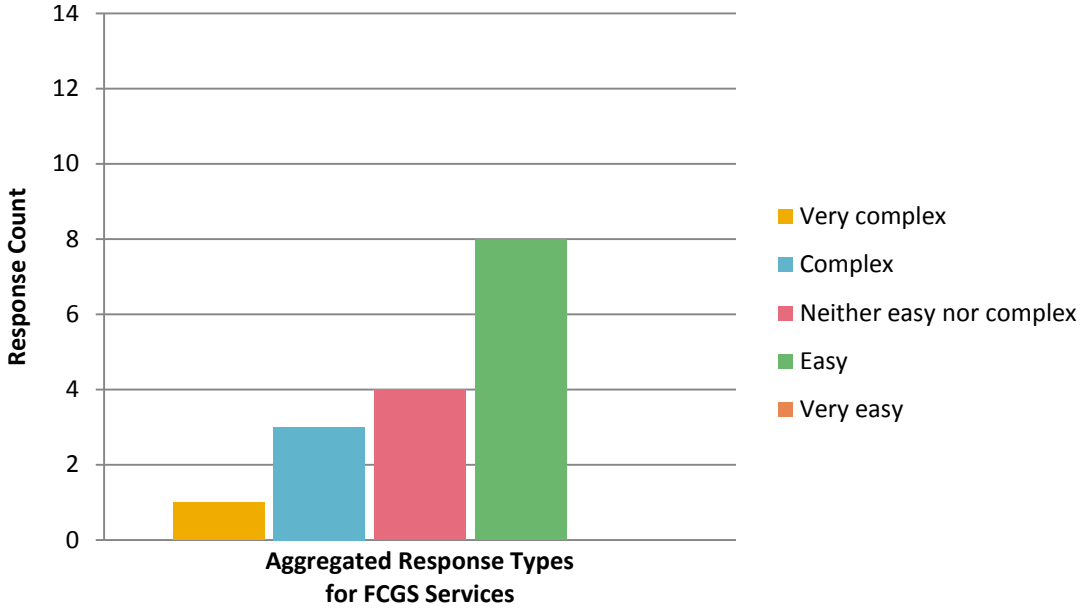
The questionnaire begins with 4 general questions to determine the users competency level with web services and to gather information on the client application developed. These are followed by 31 questions designed to discover the user's experiences using the FCGS services. For each FCGS service API there are questions which address the *Data Requirements*, *Invocation Requirements*, *Error Handling and Messages*, *Efficiency*, *Efficacy*, and *Manageability*. Finally there is space for the user to comment on one thing they like about the service and one thing that could be improved about the service. The following subsections discuss the respondent's answers to these questions, and finishes with a summary of the results.

### 6.3.1 General Questions

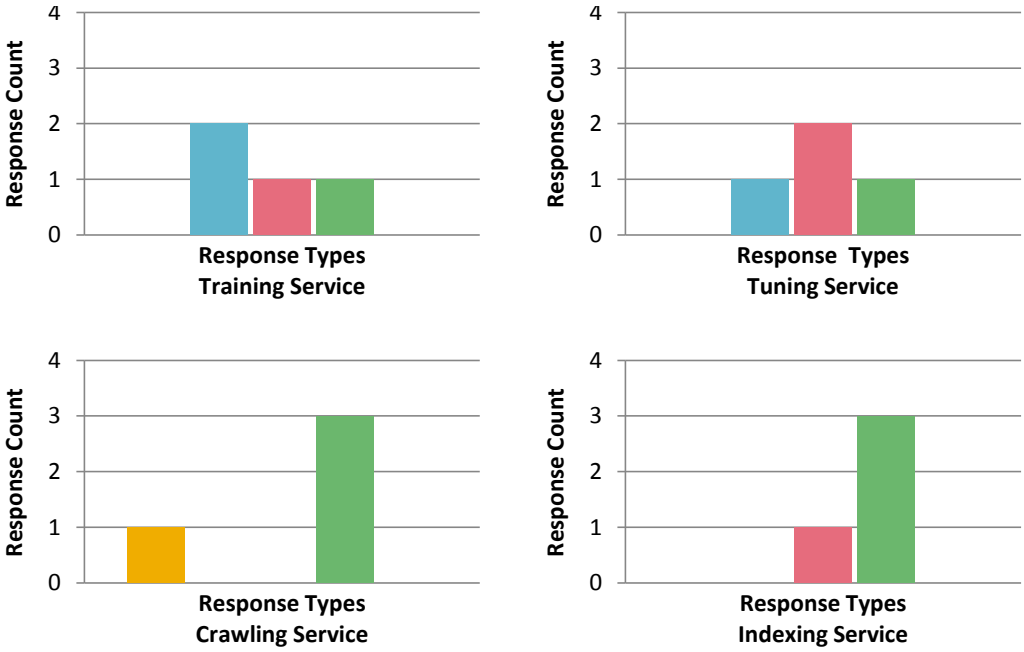
The four respondents were postdoctoral students and research assistants from the Centre for Next Generation Localisation (CNGL) at Trinity College Dublin with a high level of technical ability (see Table D.2 in the appendix). Each of the client applications were developed using different technologies: .NET C#, Java Axis2, Python, and finally manually invoking the web services using the SoapUI tool. Table D.4 in the appendix contains the brief descriptions the users gave of the applications they developed. They were mainly command-line client applications that invoked each service in turn waiting for service to finish (checked by invoking the service status operation) and then invoking the next appropriate service. The clients all generated focused content caches in different subject areas.

### 6.3.2 Data Requirements

The respondents were given the statement "*Understanding the data requirements for the API was:* " and asked to select one of the following: "*Very complex*", "*Complex*", "*Neither easy nor complex*", "*Easy*", or "*Very easy*". A chart displaying the aggregated counts of the Likert scale responses for all of the service APIs can be seen in Figure 6.2a. Charts displaying the counts of the Likert scale responses for the individual services can be seen in Figure 6.2b. The aggregated results chart shows a majority (8) of the respondents indicated that understanding the data requirements for the service APIs was "*Easy*".




(a) Aggregate results

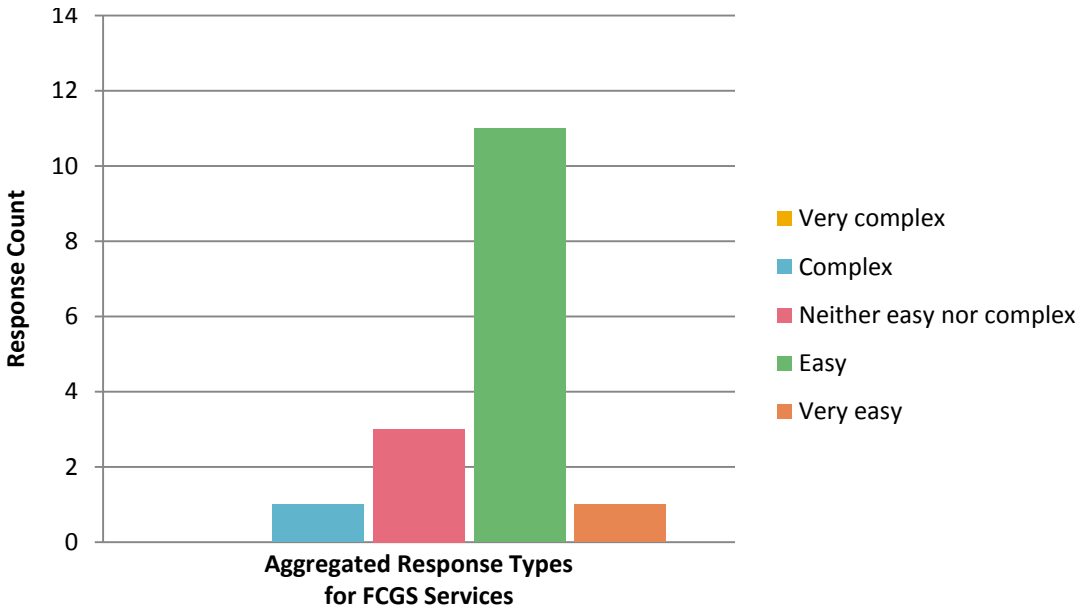



(b) Individual results

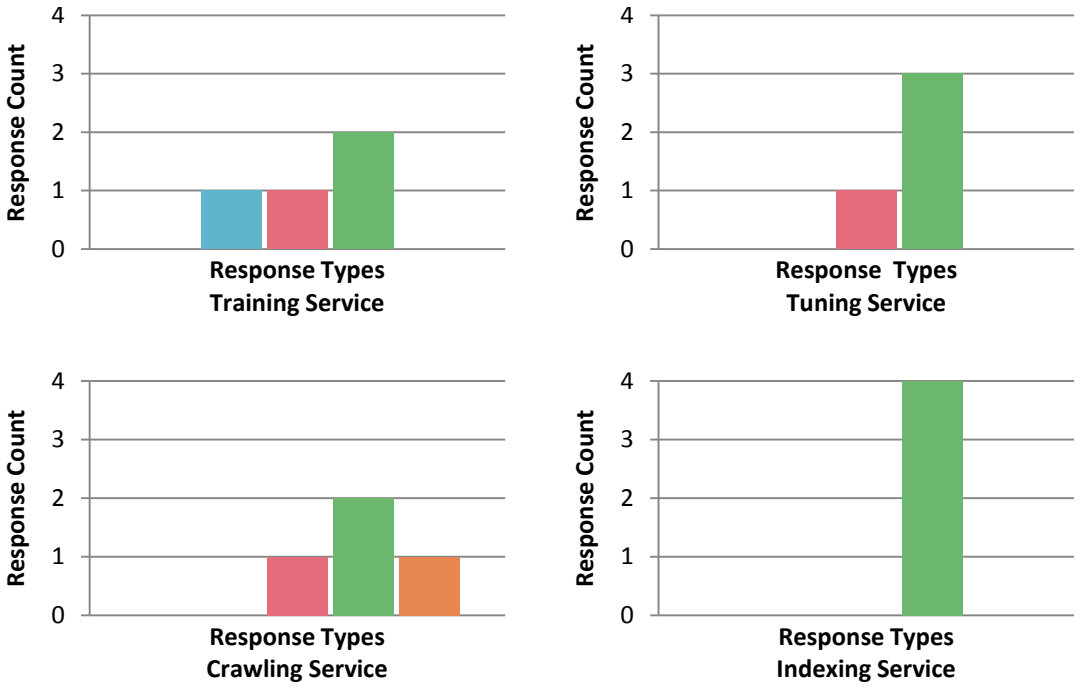
Figure 6.2: Data Requirements questionnaire results

### 6.3.3 Invocation Requirements

The respondents were given the statement "*Understanding the invocation requirements for the API was:* " and asked to select one of the following: "*Very complex*", "*Complex*", "*Neither easy nor complex*", "*Easy*", or "*Very easy*". A chart displaying the aggregated counts of the Likert scale responses for all of the service APIs can be seen in Figure 6.3a. Charts displaying the counts of the Likert scale responses for the individual services can be seen in Figure 6.3b. The aggregated results chart shows a majority (11) of the respondents indicated that understanding the invocation requirements for the service APIs was "*Easy*". One respondent selected "*Complex*" for the Training Service API which could indicate that the documentation for that interface should be improved.




(a) Aggregate results

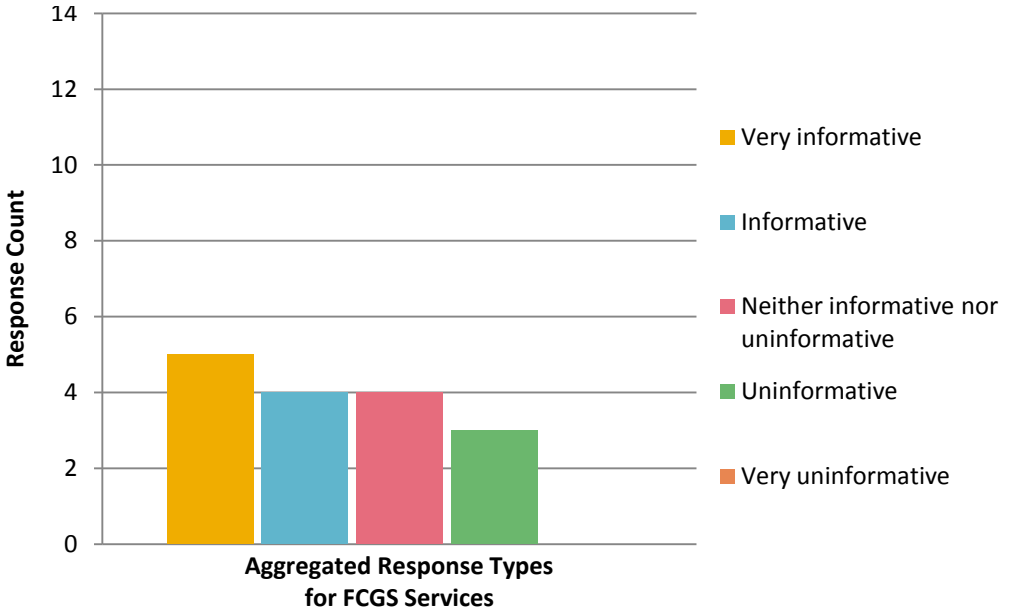



(b) Individual results

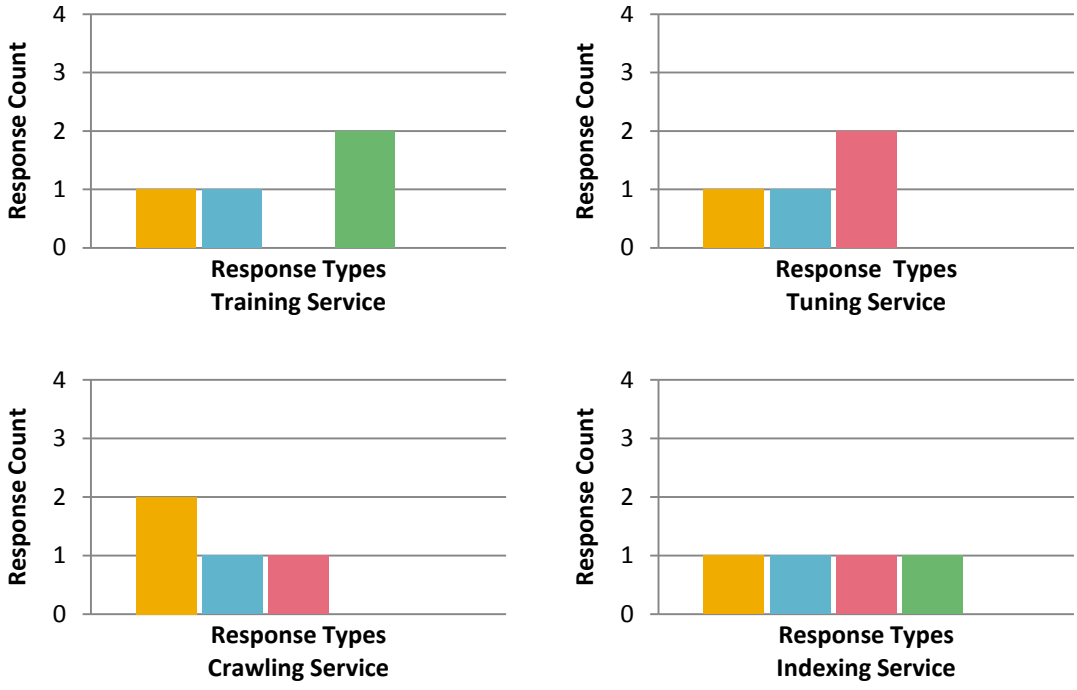
Figure 6.3: Invocation Requirements questionnaire results

### 6.3.4 Error Handling and Messages

The respondents were given the statement "*The error handling and messages produced by the API were:* " and asked to select one of the following: "*Very informative*", "*Informative*", "*Neither informative nor uninformative*", "*Uninformative*", or "*Very uninformative*". A chart displaying the aggregated counts of the Likert scale responses for all of the service APIs can be seen in Figure 6.4a. Charts displaying the counts of the Likert scale responses for the individual services can be seen in Figure 6.4b. The aggregated results chart shows a majority (9) of the respondents indicated that the error handling and messages produced by the service APIs were either "*Very informative*" or "*Informative*". There were 3 responses of "*Uninformative*" indicating that the error handling and messages produced could be improved for the Training and Indexing services.




(a) Aggregate results

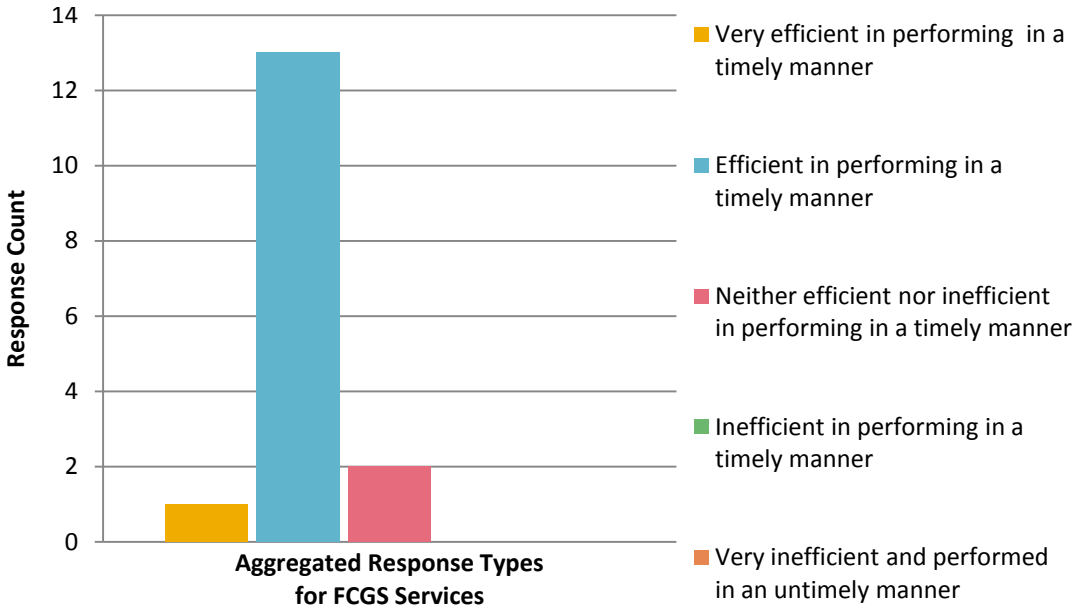



(b) Individual results

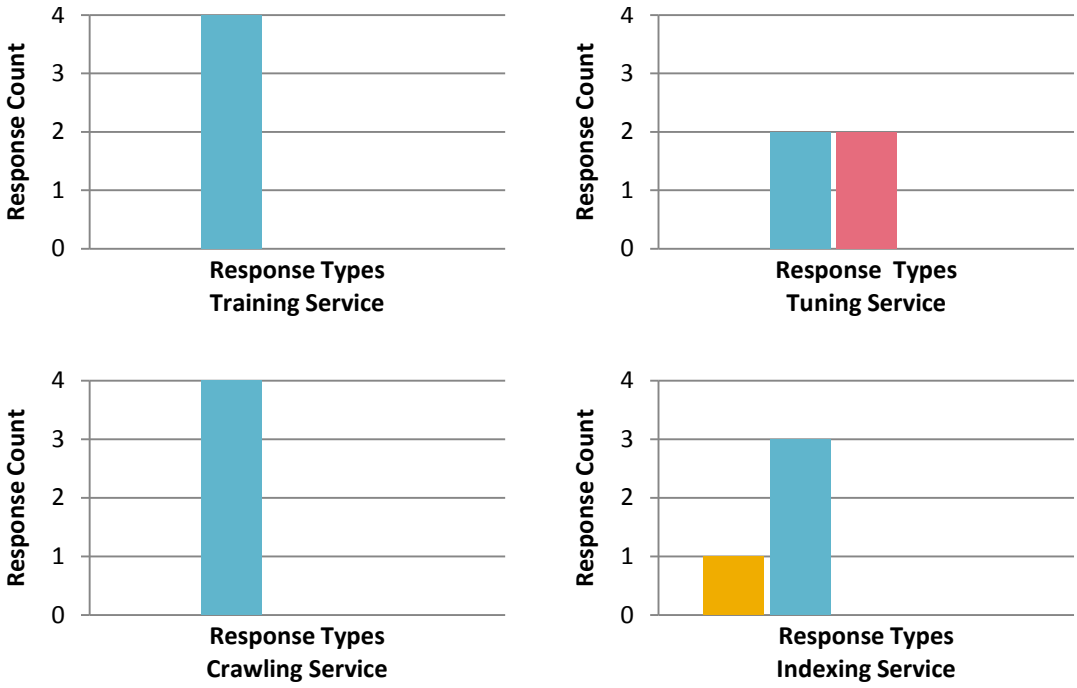
Figure 6.4: Error Handling questionnaire results

### 6.3.5 Efficiency

The respondents were given the statement "*The API was:* " and asked to select one of the following: "*Very efficient in performing in a timely manner*", "*Efficient in performing in a timely manner*", "*Neither efficient nor inefficient in performing in a timely manner*", "*Inefficient in performing in a timely manner*", or "*Very inefficient and performed in an untimely manner*". A chart displaying the aggregated counts of the Likert scale responses for all of the service APIs can be seen in Figure 6.5a. Charts displaying the counts of the Likert scale responses for the individual services can be seen in Figure 6.5b. The aggregated results chart shows a majority (13) of the respondents indicated that the APIs were "*Efficient in performing in a timely manner*".




(a) Aggregate results

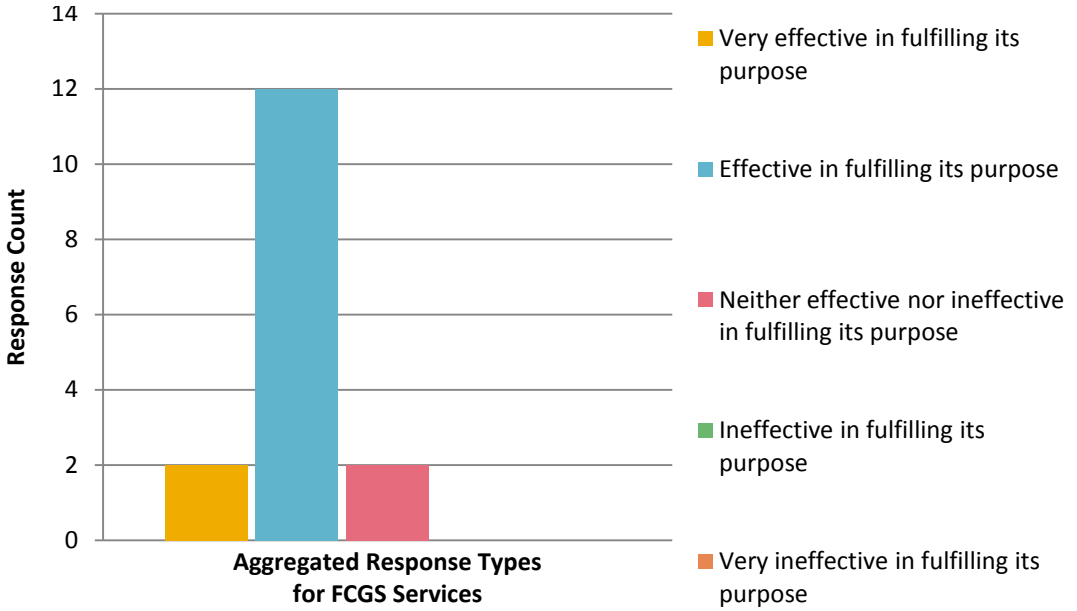



(b) Individual results

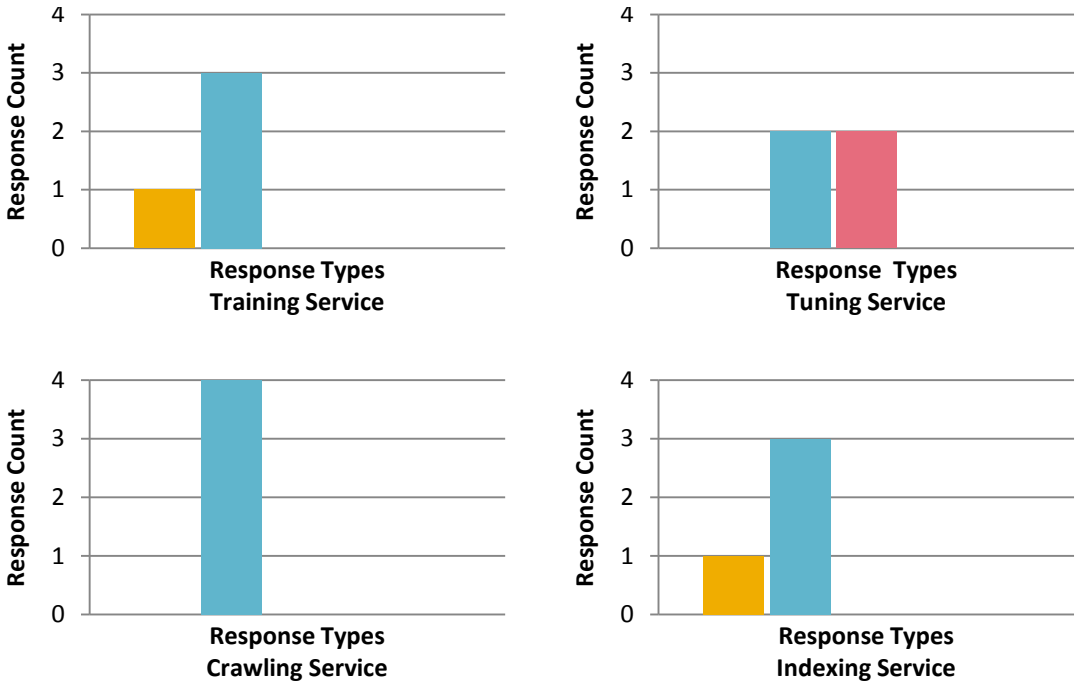
Figure 6.5: Efficiency questionnaire results

### 6.3.6 Efficacy

The respondents were given the statement "*The API was:* " and asked to select one of the following: "*Very effective in fulfilling its purpose*", "*Effective in fulfilling its purpose*", "*Neither effective nor ineffective in fulfilling its purpose*", "*Ineffective in fulfilling its purpose*" or "*Very ineffective in fulfilling its purpose*". A chart displaying the aggregated counts of the Likert scale responses for all of the service APIs can be seen in Figure 6.6a. Charts displaying the counts of the Likert scale responses for the individual services can be seen in Figure 6.6b. The aggregated results chart shows a majority (12) of the respondents indicated that each API was "*Effective in fulfilling its purpose*".




(a) Aggregate results




(b) Individual results

Figure 6.6: Efficacy questionnaire results

### 6.3.7 Manageability

The respondents were given the statement "*The management functionality provided by the Management API was: *" and asked to select one of the following: "*Very sufficient*", "*Sufficient*", "*Neither sufficient nor insufficient*", "*Insufficient*", or "*Very insufficient*". Charts are not presented for these responses because as can be seen in tables: D.10, D.18, and D.26, there were in total 11 responses of "*Sufficient*" and 1 response of "*Neither sufficient nor insufficient*", indicating that overall users found the management APIs provided by the FCGS were sufficient.

### 6.3.8 User Comments

Finally the respondents were asked to comment on what they liked and what could be improved about each FCGS service API.

#### 6.3.8.1 Training Service

The responses to the question "*Please comment on one thing you liked about the Training API*" can be seen in Table D.11 in the appendix. The responses comment on the simplicity of the training interface and the ability to monitor the training as it progresses.

The responses to the question "*Please comment on one thing that could be improved about the Training API*" can be seen in Table D.12 in the appendix. The responses comment on the complexity of the request and response messages as well as the need for detailed documentation.

#### 6.3.8.2 Tuning Service

The responses to the question "*Please comment on one thing you liked about the Tuning API*" can be seen in Table D.19 in the appendix. The responses comment on the responsiveness of the *getStopWords* operation.

The responses to the question "*Please comment on one thing that could be improved about the Tuning API*" can be seen in Table D.20 in the appendix. Again the responses comment on the complexity of the request and response messages as well as the need for detailed documentation.

#### 6.3.8.3 Crawling Service

The responses to the question "*Please comment on one thing you liked about the Crawling API*" can be seen in Table D.27 in the appendix. The responses comment on the ability to pause and resume a crawl, as well as the detailed information provided by the *crawlStatus* operation.

The responses to the question "*Please comment on one thing that could be improved about the Crawling API*" can be seen in Table D.28 in the appendix. The responses comment on the use of an ID for job management as well as the inconsistency of the status messages returned by the FCGS services. This inconsistency arises because the FCGS is responsible for generating the status messages returned by the Training and Indexing services, whereas the status message returned by the Crawling service is a Heritrix generated message. There is also a comment "*Heretrix ran into a problem while downloading one of the files*", this issue was caused as Heritrix went into a retry loop as there was a Domain Name System (DNS) lookup failure for a specific URL. The default configuration of Heritrix is such that if there is a DNS lookup failure, the thread will snooze for 900 seconds and retry up to a maximum of 30 times. Therefore the thread could be snoozed for a maximum of 7.5 hours.

Whilst not a major issue, as Heritrix has a total of 90 (configurable via the *MaxToeThreads* parameter of the *crawlMgmt* operation, see 5.4.5.1) threads to process URLs, and can continue crawling even if a thread is snoozing. However the Crawling Service *crawlStatus* operation should be extended to include information on threads that are snoozing due to DNS lookup failures or for other reasons.

#### 6.3.8.4 Indexing service

The user responses to the question "*Please comment on one thing you liked about the Indexing API*" can be seen in Table D.34 in the appendix. The user responses comment on consistency of the FCGS service interfaces, and how learning to use one of the services aids in the understanding of how to use the other services.

The user responses to the question "*Please comment on one thing that could be improved about the Indexing API*" can be seen in Table D.35 in the appendix. The user responses comment on the lack of detail of the *Status* parameter returned by the *indexStatus* operation and also some inconsistency between it and the *IsIndexing* parameter (for details of these parameters see 5.4.7.2). The inconsistency issues arose as two users were attempting to use the Indexing service at the same time, and currently the FCGS is only designed to be invoked by one user at a time. The detail of the status message was constrained as the indexing tool used does not expose any appropriate information on the status of the indexing process it is performing.

### 6.3.9 Summary

Whilst the user test was only done with four users, which is too small a sample size to do any meaningful statistical analysis, the responses are indicative that:

- Understanding the data and invocation requirements of the FCGS services is easy.
- The error handling and messages produced by the FCGS services are informative.
- The FCGS services are efficient in performing in a timely manner.
- The FCGS services are effective in fulfilling their purpose.
- The management functionality provided by the FCGS services is sufficient.

The comments given by the users were generally favourable, whilst highlighting important areas for improvement. The need for extensive and informative API documentation was also brought to light.

## 6.4 Performance Testing

Performance testing of the Training, Crawling and Indexing services was undertaken to get a baseline of the performance of the FCGS. The Tuning service was not performance tested as the *getTopWords* and *setStopWords* operations are short running and return immediately and can be invoked any number of times, to allow further tuning of the document model generated by the Rainbow classifier. The following subsections discuss the results of the performance testing for the Training, Crawling and Indexing services. Testing of the Heritrix [7] crawler that was necessary as a result of the discovery of a memory leak is discussed and finally a summary of the performance testing results is presented.

### 6.4.1 Training Service

The *train* operation (see 5.4.2.1) was invoked with varying numbers of keyword phrases and positive ODP categories to assess how they affected the training time. These are the parameters that a user would typically alter from invocation to invocation of the Training Service. The number of negative ODP categories was fixed at the default. The topic selected was *Television Shows*, and appropriate keyword phrases and positive ODP categories were chosen. Five tests were run, starting with one keyword phrase and one positive ODP category increasing up to fifty keyword phrases and fifty ODP categories. The Training Service performance testing results are displayed in Table 6.1. A graph showing how the training time varies with the number of keyword phrases and ODP categories input can be seen in Figure 6.7. The graph shows that varying the number of keyword phrases and ODP categories has little effect on the time taken for training. The time taken for one keyword phrase and ODP category was almost the same as the time taken for fifty keyword phrases and ODP categories (35 minutes vs. 38 minutes). In fact most of the training time is spent generating the negative training set, which can be affected by varying the number of negative categories (see 5.4.1.1). Reducing the number of negative categories from the default value was not examined, even though it could have reduced the training time, as it would result in an inferior document model being generated.

| Keyword Phrases / Positive ODP Categories | Train Time (minutes) |
|---:|---:|
| 1 | 35.10 |
| 5 | 49.52 |
| 10 | 40.68 |
| 20 | 36.67 |
| 50 | 38.45 |

Table 6.1: Training Service performance testing results

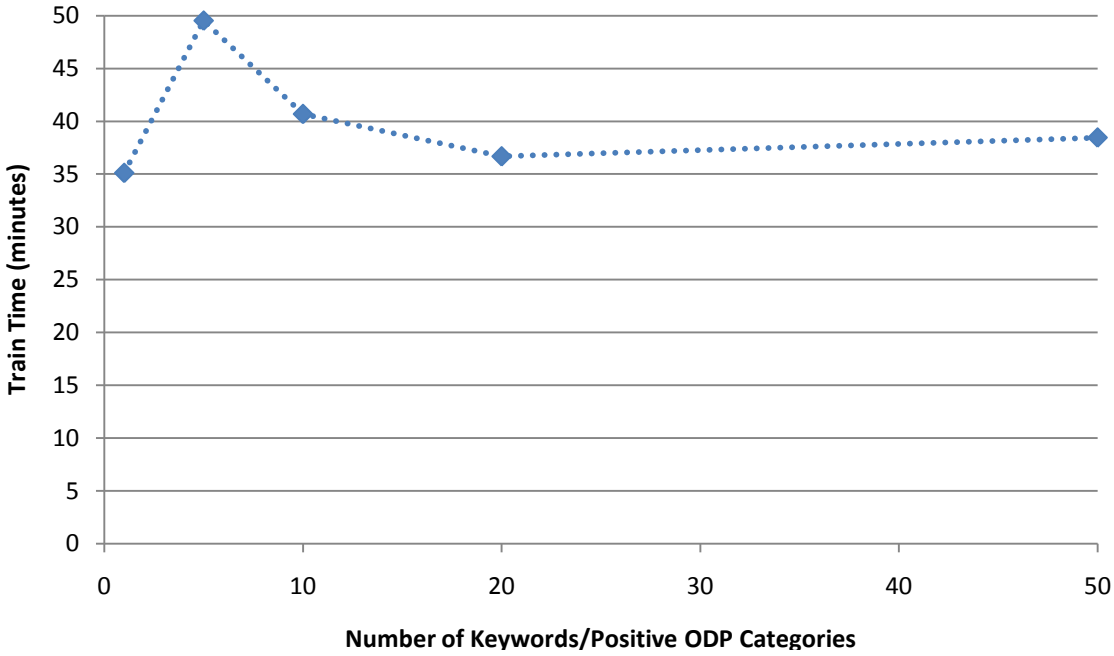



Figure 6.7: Graph of Training Service performance testing results

### 6.4.2 Crawling Service

The performance tests of the Crawling Service were performed using the document model generated after training with the fifty keyword phrases and ODP categories selected in the Training Service performance tests. The *crawl* operation (see 5.4.6.1) was invoked five times, varying the maximum documents to download (which can be configured by setting the *MaxDocumentDownload* parameter) from 1,000 documents up to 100,000 documents. The amount of documents actually downloaded are slightly higher than the MaxDocumentDownload value set, as once the limit is reached all the active threads are allowed to finish the documents in process. The Crawling Service performance testing results are displayed in Table 6.2. This table also shows the total amount of raw data crawled for each test in Gibibytes (GiBs), ranging from 0.04 to 3.65 GiB. A graph showing how the crawl time varies with the number of documents downloaded can be seen in Figure 6.8. The graph shows a linear increase from half a minute for 1,069 documents up to 40 minutes for 100,069 documents.

| Documents Downloaded | Crawl Time (minutes) | Total Data (GiB) |
|---:|---:|---:|
| 1,069 | 0.52 | 0.04 |
| 10,079 | 3.70 | 0.28 |
| 25,077 | 9.57 | 0.88 |
| 50,067 | 19.15 | 2.12 |
| 100,069 | 39.87 | 3.65 |

Table 6.2: Crawling Service performance testing results

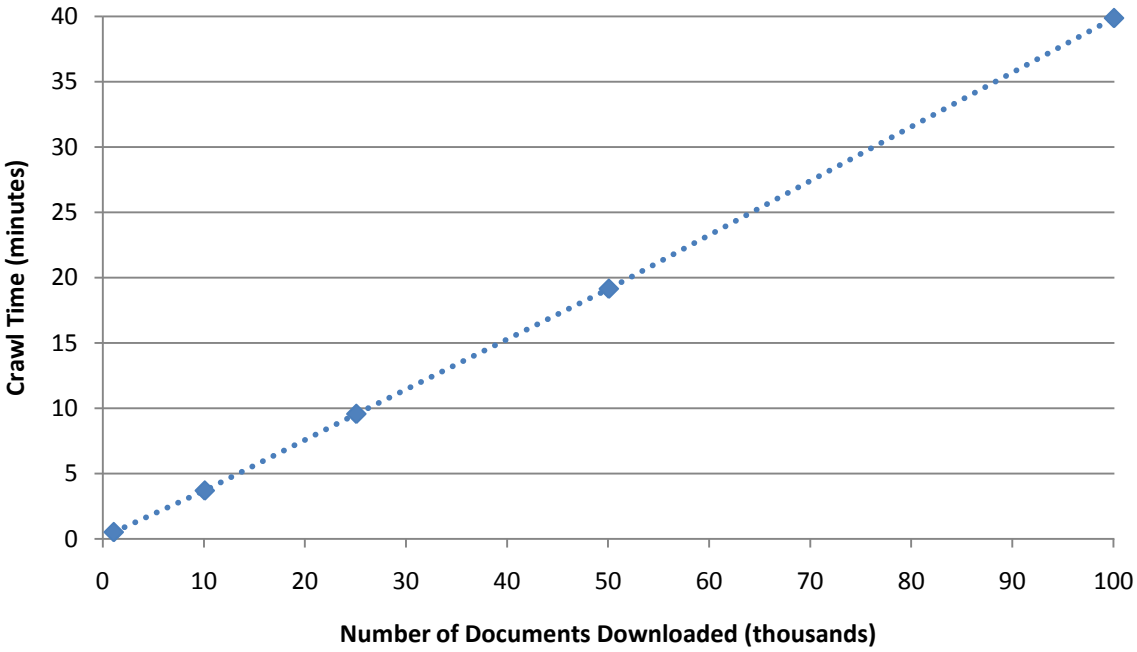



Figure 6.8: Graph of Crawling Service performance testing results

### 6.4.3 Indexing Service

After each performance test for the Crawling Service was performed, the output generated by the crawl was then indexed. The *index* operation (see 5.4.7.1) was invoked five times, varying the number of downloaded documents from 1,000 up to 100,000 documents. The Indexing Service performance testing results are displayed in Table 6.3. A graph showing how the indexing time varies with the number of documents downloaded can be seen in Figure 6.9. The graph shows a linear increase from 1 minute for 1,069 documents up to 2 hours, 13 minutes for 100,069 documents.

| Documents Downloaded | Index Time (minutes) |
|---:|---:|
| 1,069 | 0.92 |
| 10,079 | 9.92 |
| 25,077 | 27.15 |
| 50,067 | 61.33 |
| 100,069 | 133.52 |

Table 6.3: Indexing Service performance testing results

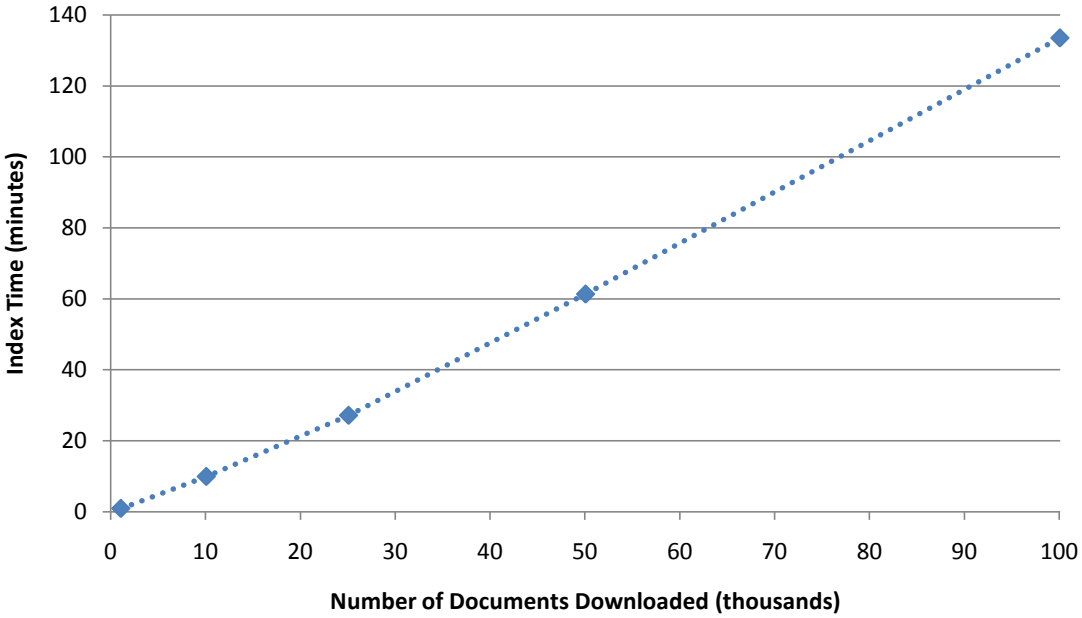



Figure 6.9: Graph of Indexing Service performance testing results

### 6.4.4 Heritrix Memory Leak

As discussed in Subsection 5.1.4, during implementation of the FCGS, it was discovered that the Heritrix [7] crawler consumed all of the JVM heap space and start logging *out of memory exceptions* within a half hour of a crawl being initiated. Once the heap space is consumed the crawler cannot continue and Heritrix automatically pauses the crawl. Several days were spent investigating if the issue was due to a misconfiguration or due to the modifications to the TextCat or Bow Processors. With the help of the Heritrix online user forum, the issue was resolved after upgrading the Java runtime environment from version 5 to version 6. The cause of memory leak is thought to have been an issue with the garbage collection in Java version 5. To ensure the memory leak issue was indeed resolved, several performance tests of the Heritrix crawler were run. It was not possible to reproduce the out of memory exceptions after running several tests, with the longest crawl being run for over eight hours, with over $600,000$ documents downloaded and over 30 GiB of total data crawled.

### 6.4.5 Summary

The performance testing on the Training Service demonstrates that increasing the number of keyword phrases and positive ODP categories selected from one up to fifty has little effect on the length of the training time. It typically takes approximately forty minutes to train the classifier. How long the classifier is tuned is entirely up to the user, as the user can continue to query the top words and to set stop words of the document model for as long as they wish. Both the Crawling and Indexing services show a linear increase in time taken for crawling or indexing depending on the number of documents downloaded. Finally the FCGS should only be deployed on the Java 6 platform or greater, as previous versions have a garbage collection issue which leads to the crawler's heap space being entirely consumed within a half hour of starting a crawl.

# Chapter 7

# Conclusions

This chapter discusses the goals and objectives this thesis set out to accomplish. It describes how these goals and objectives were achieved, and the major hurdles that were overcome. Finally the chapter outlines the future work that could be carried out to augment the research detailed in this thesis.

## 7.1 Objectives and Achievements

Referring back to the research question posed in Chapter 1 of this thesis: *(i) How can an FCGS be developed with loosely coupled components, that is exposed as web services, to allow intelligent management for auto-configuration, allowing each component to be invocable, tunable and customisable and (ii) how can such a system be engineered to generate a virtual cache of linked content, rather than actual harvested content*. From this question the following research objectives were derived:

1. Investigate the current state of the art in techniques and technologies of web IR.
2. Analyse and determine a full set of requirements and inter-dependencies for exposing each component of an FCGS as web services.
3. Research and develop prototype web services to demonstrate the feasibility of developing an FCGS using Open Source software.
   - The web services should be self configuring and allow intelligent management.
4. Investigate performance and integration issues that arise in the building of these web services.
5. Determine the feasibility of generating a repository of virtual links instead of physical content.

To address the first and second objectives, it was necessary to examine the techniques and technologies for the generation of focused content caches. A comprehensive review of the state of the art in web IR was undertaken. Focused crawling, document classification and finally indexing were examined. Open source tools and tool chains providing these services were surveyed. An understanding of these techniques and technologies allowed the formulation of requirements for each component of an FCGS as well as elucidating the inter-dependencies between these components. The survey of tools and tool chains identified the OCCS [42] as the best basis for building an FCGS that was Open Source and could be exposed as a web service API. However the OCCS was not developed with an SOA in mind, its components are manually invoked by executing Perl scripts and require the manual modification of configuration files. These scripts and configuration files are inaccessible to the world

wide web, as the system can only be accessed by logging in to a specific server with the requisite privileges.

In fulfilling the third objective it was not just sufficient to wrap the OCCS with web services. The OCCS components have an order of execution, training before crawling, crawling before indexing, etc. but the responsibility for the verification of this state is left up to the user. The FCGS web services were developed to verify and inform the user in the event that the system is not in the required state, if the user attempts to index before they have trained the classifier for example.

In order to satisfy the fourth objective, it was necessary to ensure that the FCGS web services were able to appropriately configure the OCCS components before invoking any of the Perl scripts. This was achieved through the use of a templating engine. Templates of the configuration files were created, so custom versions of the files could be generated on the fly when a user made an FCGS web service request. It was also necessary that the FCGS expose a management interface to allow a manager to set parameters which would be preserved from invocation to invocation of the FCGS services.

In developing the FCGS it was found that the existing version of the Heritrix [7] crawler used by the OCCS was insufficient to meet the requirements of the FCGS, it was necessary to migrate to a newer version which provides a JMX interface that allows the monitoring and management of the crawler. During this migration it was discovered that the crawler was consuming all of its heap space in half an hour, and so was unable to continue crawling. This issue was resolved with an upgrade of the Java platform. In addition, the Google API utilised by the OCCS for training was decommissioned and no longer available for use. It was necessary to refactor the OCCS scripts to utilise the replacement API provided by Google.

In order to achieve the final objective the Indexing Service of the FCGS was developed such that a request parameter allows the user to specify if a virtual or physical content cache should be generated. The Indexing Service then deletes any content downloaded after the indexing process is complete, if the user specifies a virtual content cache. In the case of the user specifying a physical content cache, the content is not deleted.

In order to verify that the objectives were met, a thorough evaluation of the FCGS was undertaken. This evaluation took the form of the development of a demonstrator application, a user test, and finally system performance tests. The success of the demonstrator and performance tests clearly demonstrate the feasibility of providing a system for the generation of focused content caches that is web accessible, that is tunable and customisable and that provides interfaces for monitoring and management. The feedback from the user test (in the form of a questionnaire) indicates that the data and invocation requirements of FCGS are easily understood, the error handling and messages produced by the FCGS are informative, the FCGS web services perform in a timely manner and are fit for purpose and finally the management functionality provided by the FCGS is sufficient. This is a significant contribution to the state of the art in terms of research and technology. The ability to generate focused content caches is now available to anyone with the means to develop a web service client. However whilst the FCGS has the potential to be used by a wide audience, it still requires a high level of technical expertise and understanding of web IR in general to use.

## 7.2 Future Work

This section discusses the areas where the research detailed in this thesis could be extended.

### 7.2.1 Concurrency

The OCCS upon which the FCGS was built only supports one user at a time. Due to the thesis time constraints the FCGS also has this restriction. The interfaces and architecture were designed to be flexible so the changes required to allow concurrent use could be integrated as easily as possible.

### 7.2.2 Migration from Perl to Java

The Training and Indexing Perl scripts could potentially be rewritten in Java. This would allow much finer grain control over the training and indexing processes. For example the FCGS is restricted to starting or terminating a training or indexing process. If the functionality provided by the Perl scripts was migrated to Java, a user could potentially pause or resume a training process or reconfigure an already running process. It would also allow status operations to return more detailed information when informing the user of the status of an ongoing training or indexing process.

### 7.2.3 Large Scale User Test

The user test conducted for this thesis was very limited, with only four users. A far more thorough evaluation of the FCGS would be possible with a large scale user test. A large scale test would undoubtedly provide invaluable feedback which could only assure the FCGS of reaching a wider audience.

# Bibliography


[1] Combine Harvesting Robot is an open system for crawling internet resources. http://combine.it.lth.se/. Retrieved September 10, 2010.

[2] Commons IO is a library of utilities to assist with developing IO functionality. http://commons.apache.org/io/. Retrieved September 10, 2010.

[3] Commons Validator is a Java-based package which speeds development and maintenance of validation rules. http://commons.apache.org/validator. Retrieved September 10, 2010.

[4] GlassFish an open source Java EE application server. https://glassfish.dev.java.net/. Retrieved September 10, 2010.

[5] Google AJAX Search API exposes a RESTful interface that returns JSON encoded results, which allows the embedding of Google Search in web pages and other web applications. http://code.google.com/apis/ajaxsearch. Retrieved September 10, 2010.

[6] Grub, the distributed web crawling system and search engine. http://www.grub.org. Retrieved September 10, 2010.

[7] Heritrix, the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project. http://crawler.archive.org/. Retrieved September 10, 2010.

[8] Heritrix web services, a draft REST API for the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project. https://webarchive.jira.com/wiki/display/Heritrix/Heritrix+2+Web+Services. Retrieved September 10, 2010.

[9] The ht://Dig system is a complete indexing and searching system for a domain or intranet. http://www.htdig.org/. Retrieved September 10, 2010.

[10] Java Text Categorizing Library, a library developed for language guessing. http://textcat.sourceforge.net/. Retrieved September 10, 2010.

[11] JSON, a Perl module which provides a (JavaScript Object Notation) encoder/decoder. http://search.cpan.org/~makamaka/JSON-2.22. Retrieved September 10, 2010.

[12] The Lemur Toolkit is an open-source suite of tools designed to facilitate research in language modeling and information retrieval. http://www.lemurproject.org/. Retrieved September 10, 2010.

[13] Lucene provides Java-based indexing and search technology. http://lucene.apache.org/. Retrieved September 10, 2010.

[14] LWP::UserAgent, a Perl module which provides a Web user agent class. http://search.cpan.org/~gaas/libwww-perl-5.836/lib/LWP/UserAgent.pm. Retrieved September 10, 2010.

[15] Methanol is a fully customizable web crawling system. http://metha-sys.org/. Retrieved September 10, 2010.

[16] NetBeans is an integrated development environment (IDE) for developing with Java, JavaScript, PHP, Python, Ruby, Groovy, C, C++, Scala, Clojure, and others. http://netbeans.org/. Retrieved September 10, 2010.

[17] Net::Google, a Perl module which provides simple OOP-ish interface to the Google SOAP API. http://search.cpan.org/dist/Net-Google. Retrieved September 10, 2010.

[18] Nutch, open source web-search software. http://lucene.apache.org/nutch/. Retrieved September 10, 2010.

[19] NutchWAX (Nutch + Web Archive eXtensions) searches web archive collections. http://archive-access.sourceforge.net/projects/nutchwax/. Retrieved September 10, 2010.

[20] Open Directory Project, the largest human-edited directory of the web. http://www.dmoz.org/. Retrieved September 10, 2010.

[21] Proc::Killfam - a Perl module to kill a list of pids, and all their sub-children. http://search.cpan.org/~durist/Proc-ProcessTable-0.45/Killfam.pm. Retrieved September 10, 2010.

[22] Rainbow, a program that performs statistical text classification. http://www.cs.cmu.edu/~mccallum/bow/rainbow/. Retrieved September 10, 2010.

[23] Swish-e, a Simple Web Indexing System for Humans - Enhanced. http://swish-e.org/. Retrieved September 10, 2010.

[24] URI::Escape - a Perl module to escape and unescape unsafe characters. http://search.cpan.org/~gaas/URI-1.54/URI/Escape.pm. Retrieved September 10, 2010.

[25] Velocity is a simple yet powerful Java-based template engine that renders data from plain Java objects to text, XML, email, SQL, Post Script, HTML and more. http://velocity.apache.org/. Retrieved September 10, 2010.

[26] WERA (Web ARchive Access) is a freely available solution for searching and navigating archived web document collections. http://archive-access.sourceforge.net/projects/wera/. Retrieved September 10, 2010.

[27] Xapian, a toolkit which allows developers to easily add advanced indexing and search facilities to their own applications. http://xapian.org/. Retrieved September 10, 2010.

[28] Thomas Bayes. An essay towards solving a Problem in the Doctrine of Chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763.

[29] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[30] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[31] Brian D. Davison. Topical locality in the Web. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 272–279, New York, NY, USA, 2000. ACM.

[32] P.M.E. De Bra and R.D.J. Post. Information retrieval in the World-Wide Web: Making client-based searching feasible. *Computer Networks and ISDN Systems*, 27(2):183–192, 1994. Selected Papers of the First World-Wide Web Conference.

[33] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM.

[34] E. Garfield. Citation analysis as a tool in journal evaluation. *Science*, 178(60):471–479, November 1972.

[35] C. W. Gini. Variability and Mutability, contribution to the study of statistical distributions and relations. *Studi Economico-Giuridici della R. Universita de Cagliari*, 1912. Reviewed in: Light, R.J., Margolin, B.H.: An Analysis of Variance for Categorical Data. *J. American Statistical Association*, Vol. 66 pp. 534-544 (1971).

[36] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

[37] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the deep web. *Commun. ACM*, 50(5):94–101, 2007.

[38] Michael Hersovici, Michal Jacovi, Yoelle S. Maarek, Dan Pelleg, Menachem Shtalhaim, and Sigalit Ur. The shark-search algorithm. An application: tailored Web site mapping. *Computer Networks and ISDN Systems*, 30(1-7):317–326, 1998. Proceedings of the Seventh International World Wide Web Conference.

[39] E. M. Keen and J. A. Digger. *Report of an Information Science Index Languages Test*. Aberystwyth College of Librarianship, Aberystwyth, Wales, 1972.

[40] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[41] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

[42] Seamus Lawless. *Leveraging Content from Open Corpus Sources for Technology Enhanced Learning*. PhD thesis, University of Dublin, Trinity College, 2009.

[43] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[44] Julie B. Lovins. Error evaluation for stemming algorithms as clustering algorithms. *Journal of the American Society for Information Science*, 22(1):28–40, 1971.

[45] Filippo Menczer and Richard K. Belew. Adaptive Retrieval Agents: Internalizing Local Context and Scaling up to the Web. *Machine Learning*, 39(2):203–242, May 2000.

[46] Sougata Mukherjea. WTMS: a system for collecting and analyzing topic-specific Web information. *Computer Networks*, 33(1-6):457–471, 2000.

[47] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[48] Gabriel Pinski and Francis Narin. Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of physics. *Inf. Process. Manage.*, 12(5):297–312, 1976.

[49] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281, New York, NY, USA, 1998. ACM.

[50] Jason D. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In *ICML*, pages 616–623. AAAI Press, 2003.

[51] S. E. Robertson and Karen Spärck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[52] Gerard Salton and C.S. Yang. On the Specification of Term Values in Automatic Indexing. *Journal of Documentation*, 29:351–372, 1973.

[53] C. J. van Rijsbergen. *Information retrieval*. Butterworths, London, England, second edition, 1979.

[54] Chengxiang Zhai and John Lafferty. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342, New York, NY, USA, 2001. ACM.

[55] George K. Zipf. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, Cambridge MA, 1949.

# Appendix A

# FCGS Web Services Properties file

```
# Path where generated config files such as keywords.txt and pos_categories.txt
    are located
GeneratedConfigFilesPath = /home/phannon/OCCSService/base_occs_crawler/

# Path where Velocity template files are located
TemplateFilesPath = /home/phannon/OCCSService/templates/

# Path where perl scripts such as train_step1.pl and start_crawler.pl are located
PerlScriptsPath = /home/phannon/OCCSService/base_occs_crawler/base_crawler/

# Path where Heritrix configuration files, such as order.xml and seeds.txt are
    located
HeritrixConfigFilesPath = /home/phannon/OCCSService/base_occs_crawler/
    heritrix_files/

# Path where the BOW config file: Bow_Config_File.txt is located
HeritrixBinPath = /home/phannon/heritrix-1.14.4/bin/

# Heritrix JMX Host
HeritrixJMXHost = 134.226.35.161

# Hertrix JMX Port
HeritrixJMXPort = 8849

# Heritrix JMX Username
HeritrixJMXUsername = controlRole

# Heritrix JMX Password
HeritrixJMXPassword = ***********
```

Listing A.1: OCCSService.properties

# Appendix B

# Velocity Templates

## B.1 Perl Script Configuration Templates

```
Path = /home/phannon/OCCSService/base_occs_crawler

BowPath = /usr/local/bin

HeritrixPath = /home/phannon/heritrix-1.14.4/bin

XMLFile-Configure-Script = /home/slawless/OAI-XMLFile-2.1/XMLFile/

Negative-Category-Count = $negativeCategoryCount

KEYWORD-FLAG = 1

ODP-FLAG = 1

OAI-FLAG = 0

Google-License-Key = 6RUbjkFQFHKHjTOkfkMo2D/38HhtCb7Y

Host = 134.226.35.161

Proxy = ${httpProxyHost}:${httpProxyPort}

Port = 5555

User = phannon

Pass = *******
```

Listing B.1: config-train-mgmt.txt.vm

```
Topic = $topic

Max-Search-Results = $maxSearchResults
```

Listing B.2: config-train.txt.vm

```
topN = $topWordCount
```

Listing B.3: config-tune-mgmt.txt.vm

```
JobName = $jobName
```

Listing B.4: config-crawl.txt.vm

```
VIRTUAL-CACHE-FLAG = $virtualCacheFlag
```

Listing B.5: config-index.txt.vm

```
#include( "config-train.txt" )

#include( "config-train-mgmt.txt" )

#include( "config-tune-mgmt.txt" )

#include( "config-crawl.txt" )

#include( "config-index.txt" )
```

Listing B.6: config.txt.vm

## B.2 Training Configuration Templates

```
#foreach( $name in $keywordPhraseList )
$name
#end
```

Listing B.7: keywords.txt.vm

```
#foreach( $name in $posCategoriesList )
$name
#end
```

Listing B.8: pos_categories.txt.vm

## B.3 Tuning Configuration Templates

```
#foreach( $topWord in $topWordList )
#if( $topWord.isStopWord() )
$topWord.getValue()     [1]
#else
$topWord.getValue()     [0]
#end
#end
```

Listing B.9: top-words.txt.vm

## B.4 TextCat Processor Configuration Templates

```
TextCatLog_File_Path = /home/phannon/OCCSService/base_occs_crawler/textcatlogfile
    .txt

Language = $language
```

Listing B.10: TextCat_Config_File.txt.vm

## B.5 Bow Processor Configuration Templates

```
Topic = $topic
```

Listing B.11: Bow_Config_File-train.txt.vm

```
#include( "Bow_Config_File-train.txt" )

CutOff = $cutOff

Host = 134.226.35.161

Port = 5555

BowLog_File_Path = /home/phannon/OCCSService/base_occs_crawler/bowlogfile.txt

BowLogContents_File_Path = /home/phannon/OCCSService/base_occs_crawler/
    bowlogcontents.txt
```

Listing B.12: Bow_Config_File-crawl.txt.vm

## B.6 Heritrix Configuration Templates

```
    <integer name="max-toe-threads">${maxToeThreads}</integer>
    <integer name="recorder-out-buffer-bytes">4096</integer>
    <integer name="recorder-in-buffer-bytes">65536</integer>
    <integer name="bdb-cache-percent">0</integer>
    <newObject name="scope" class="org.archive.crawler.scope.BroadScope">
      <boolean name="enabled">true</boolean>
      <string name="seedsfile">seeds.txt</string>
      <integer name="max-link-hops">${maxLinkHops}</integer>
      <integer name="max-trans-hops">${maxTransHops}</integer>
      <newObject name="exclude-filter" class="org.archive.crawler.filter.OrFilter
          ">
        <boolean name="enabled">true</boolean>
        <boolean name="if-matches-return">true</boolean>
        <map name="filters">
          <newObject name="removeNonTextPages" class="org.archive.crawler.filter.
              URIRegExpFilter">
            <boolean name="enabled">true</boolean>
```

```
          <boolean name="if-match-return">true</boolean>
          <string name="regexp">.*(?i)\.(a|ai|aif|aifc|aiff|asc|bcpio|bin|bz2|c
              |cdf|cgi|cgm|class|cpio|cpp?|cpt|csh|css|cxx|dcr|dif|dir|djv|djvu
              |dll|dmg|dms|dtd|dv|dvi|dxr|eps|etx|exe|ez|gram|grxml|gtar|h|hdf|
              hqx|ice|ics|ief|ifb|iges|igs|iso|jnlp|jp2|js|kar|lha|lzh|m3u|mac|
              man|mathml|me|mesh|mif|ms|msh|mxu|nc|o|oda|ogg|pbm|pct|pdb|pgm|
              pgn|pl|pnm|pnt|pntg|ppm|py|qt|qti|qtif|ra|ram|ras|rdf|rgb|rm|roff
              |rpm|rtx|s|sgm|sgml|sh|shar|silo|sit|skd|skm|skp|skt|smi|smil|snd
              |so|spl|src|srpm|sv4cpio|sv4crc|swf|t|tar|tcl|tex|texi|texinfo|
              tgz|tr|tsv|ustar|vcd|vrml|vxml|wav|wbmp|wbxml|wml|wmlc|wmls|wmlsc
              |wrl|xbm|xht|xpm|xsl|xslt|xwd|xyz|z|zip)$</string>
        </newObject>
```

Listing B.13: order1.txt.vm

```
        <newObject name="pathdepth" class="org.archive.crawler.filter.
            PathDepthFilter">
          <boolean name="enabled">true</boolean>
          <integer name="max-path-depth">${maxPathDepth}</integer>
          <boolean name="path-less-or-equal-return">false</boolean>
        </newObject>
        <newObject name="pathologicalpath" class="org.archive.crawler.filter.
            PathologicalPathFilter">
          <boolean name="enabled">true</boolean>
          <integer name="repetitions">3</integer>
        </newObject>
      </map>
    </newObject>
  </newObject>
  <map name="http-headers">
    <string name="user-agent">Mozilla/5.0 (compatible; heritrix/1.14.4 +${
        organisationURL})</string>
    <string name="from">${emailAddress}</string>
  </map>
  <newObject name="robots-honoring-policy" class="org.archive.crawler.datamodel
      .RobotsHonoringPolicy">
    <string name="type">classic</string>
    <boolean name="masquerade">false</boolean>
    <text name="custom-robots"></text>
    <stringList name="user-agents">
    </stringList>
  </newObject>
  <newObject name="frontier" class="org.archive.crawler.frontier.BdbFrontier">
    <float name="delay-factor">5.0</float>
    <integer name="max-delay-ms">5000</integer>
    <integer name="min-delay-ms">500</integer>
    <integer name="max-retries">30</integer>
    <long name="retry-delay-seconds">900</long>
    <integer name="preference-embed-hops">1</integer>
    <integer name="total-bandwidth-usage-KB-sec">0</integer>
    <integer name="max-per-host-bandwidth-usage-KB-sec">0</integer>
    <string name="force-queue-assignment"></string>
    <boolean name="pause-at-finish">false</boolean>
    <boolean name="hold-queues">true</boolean>
```

```
    <integer name="balance-replenish-amount">3000</integer>
    <long name="queue-total-budget">-1</long>
    <string name="cost-policy">org.archive.crawler.frontier.
        ZeroCostAssignmentPolicy</string>
  </newObject>
  <map name="uri-canonicalization-rules">
    <newObject name="Lowercase" class="org.archive.crawler.url.canonicalize.
        LowercaseRule">
      <boolean name="enabled">true</boolean>
    </newObject>
    <newObject name="Userinfo" class="org.archive.crawler.url.canonicalize.
        StripUserinfoRule">
      <boolean name="enabled">true</boolean>
    </newObject>
    <newObject name="WWW[0-9]*" class="org.archive.crawler.url.canonicalize.
        StripWWWNRule">
      <boolean name="enabled">true</boolean>
    </newObject>
    <newObject name="SessionIDs" class="org.archive.crawler.url.canonicalize.
        StripSessionIDs">
      <boolean name="enabled">true</boolean>
    </newObject>
    <newObject name="SessionCFIDs" class="org.archive.crawler.url.canonicalize.
        StripSessionCFIDs">
      <boolean name="enabled">true</boolean>
    </newObject>
    <newObject name="QueryStrPrefix" class="org.archive.crawler.url.
        canonicalize.FixupQueryStr">
      <boolean name="enabled">true</boolean>
    </newObject>
  </map>
  <map name="pre-fetch-processors">
    <newObject name="Preselector" class="org.archive.crawler.prefetch.
        Preselector">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
      <boolean name="recheck-scope">true</boolean>
      <boolean name="block-all">false</boolean>
      <string name="block-by-regexp"></string>
    </newObject>
    <newObject name="Preprocessor" class="org.archive.crawler.prefetch.
        PreconditionEnforcer">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
      <integer name="ip-validity-duration-seconds">21600</integer>
      <integer name="robot-validity-duration-seconds">86400</integer>
    </newObject>
  </map>
```

Listing B.14: order2.txt.vm

```
<map name="fetch-processors">
  <newObject name="DNS" class="org.archive.crawler.fetcher.FetchDNS">
    <boolean name="enabled">true</boolean>
    <map name="filters">
    </map>
    <boolean name="accept-non-dns-resolves">false</boolean>
  </newObject>
  <newObject name="HTTP" class="org.archive.crawler.fetcher.FetchHTTP">
    <boolean name="enabled">true</boolean>
    <map name="filters">
    </map>
    <map name="midfetch-filters">
    </map>
    <integer name="timeout-seconds">1200</integer>
    <integer name="sotimeout-ms">20000</integer>
    <long name="max-length-bytes">0</long>
    <string name="load-cookies-from-file"></string>
    <string name="save-cookies-to-file"></string>
    <string name="trust-level">open</string>
    <stringList name="accept-headers">
    </stringList>
    <string name="http-proxy-host">${httpProxyHost}</string>
    <string name="http-proxy-port">${httpProxyPort}</string>
    <string name="default-encoding">ISO-8859-1</string>
    <boolean name="send-connection-close">true</boolean>
    <boolean name="send-referer">true</boolean>
    <boolean name="send-range">false</boolean>
  </newObject>
</map>
<map name="extract-processors">
  <newObject name="TextCatProcessor" class="org.metacombine.languagemodule.
      TextCatProcessor">
    <boolean name="enabled">true</boolean>
    <map name="filters">
    </map>
  </newObject>
  <newObject name="BowProcessor" class="org.metacombine.crawlmodule.
      BowProcessor">
    <boolean name="enabled">true</boolean>
    <map name="filters">
    </map>
  </newObject>
  <newObject name="ExtractorHTTP" class="org.archive.crawler.extractor.
      ExtractorHTTP">
    <boolean name="enabled">true</boolean>
    <map name="filters">
    </map>
  </newObject>
  <newObject name="ExtractorHTML" class="org.archive.crawler.extractor.
      ExtractorHTML">
    <boolean name="enabled">true</boolean>
    <map name="filters">
    </map>
  </newObject>
```

```
    <newObject name="ExtractorCSS" class="org.archive.crawler.extractor.
        ExtractorCSS">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
    </newObject>
    <newObject name="ExtractorJS" class="org.archive.crawler.extractor.
        ExtractorJS">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
    </newObject>
    <newObject name="ExtractorSWF" class="org.archive.crawler.extractor.
        ExtractorSWF">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
    </newObject>
  </map>
  <map name="write-processors">
    <newObject name="Archiver" class="org.archive.crawler.writer.
        ARCWriterProcessor">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
      <boolean name="compress">true</boolean>
      <string name="prefix">IAH</string>
      <string name="suffix">${HOSTNAME_ADMINPORT}</string>
      <integer name="max-size-bytes">70000000</integer>
      <stringList name="path">
        <string>arcs</string>
      </stringList>
      <integer name="pool-max-active">5</integer>
      <integer name="pool-max-wait">300000</integer>
      <long name="total-bytes-to-write">0</long>
    </newObject>
  </map>
  <map name="post-processors">
    <newObject name="Updater" class="org.archive.crawler.postprocessor.
        CrawlStateUpdater">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
    </newObject>
    <newObject name="LinksScoper"
          class="org.archive.crawler.postprocessor.LinksScoper">
      <boolean name="enabled">true</boolean>
      <map name="filters">
      </map>
      <boolean name="seed-redirects-new-seed">true</boolean>
      <boolean name="override-logger">false</boolean>
      <map name="scope-rejected-url-filters">
      </map>
    </newObject>
```

```
      <newObject name="Scheduler"
            class="org.archive.crawler.postprocessor.FrontierScheduler">
        <boolean name="enabled">true</boolean>
      </newObject>
    </map>
    <map name="loggers">
      <newObject name="crawl-statistics" class="org.archive.crawler.admin.
          StatisticsTracker">
        <integer name="interval-seconds">20</integer>
      </newObject>
    </map>
    <string name="recover-path"></string>
    <boolean name="recover-retain-failures">false</boolean>
    <newObject name="credential-store" class="org.archive.crawler.datamodel.
        CredentialStore">
      <map name="credentials">
      </map>
    </newObject>
  </controller>
</crawl-order>
```

Listing B.15: order3.txt.vm

```
<?xml version="1.0" encoding="UTF-8"?><crawl-order xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="heritrix_settings.
    xsd">
  <meta>
    <name>${jobName}</name>
    <description>Default Profile</description>
    <operator>Admin</operator>
    <organization>${organisation}</organization>
    <audience></audience>
    <date>${date}</date>
  </meta>
  <controller>
    <string name="settings-directory">settings</string>
    <string name="disk-path"></string>
    <string name="logs-path">logs</string>
    <string name="checkpoints-path">checkpoints</string>
    <string name="state-path">state</string>
    <string name="scratch-path">scratch</string>
    <long name="max-bytes-download">${maxBytesDownload}</long>
    <long name="max-document-download">${maxDocumentDownload}</long>
    <long name="max-time-sec">${maxTimeSec}</long>
#include( "order1.xml.txt","order2.xml.txt","order3.xml.txt" )
```

Listing B.16: order.txt.vm

# Appendix C

# FCGS Web Service WSDLs

## C.1 TrainingMgmtService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://trainingMgmt.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    trainingMgmt.service/" name="TrainingMgmtService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                TrainingMgmtParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.
                ie:8080/OCCSService/TrainingMgmtService?xsd=1"></xsd:import>
        </xsd:schema>
        <xsd:schema>
            <xsd:import namespace="http://trainingMgmt.service/" schemaLocation="
                http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/TrainingMgmtService?
                xsd=2"></xsd:import>
        </xsd:schema>
    </types>
    <message name="trainMgmt">
        <part name="parameters" element="tns:trainMgmt"></part>
    </message>
    <message name="trainMgmtResponse">
        <part name="parameters" element="tns:trainMgmtResponse"></part>
    </message>
    <message name="TrainMgmtFault">
        <part name="fault" element="tns:TrainMgmtFault"></part>
    </message>
    <portType name="TrainingMgmt">
        <operation name="trainMgmt">
            <input message="tns:trainMgmt"></input>
            <output message="tns:trainMgmtResponse"></output>
            <fault message="tns:TrainMgmtFault" name="TrainMgmtFault"></fault>
        </operation>
    </portType>
    <binding name="TrainingMgmtPortBinding" type="tns:TrainingMgmt">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
            document"></soap:binding>
```

```
        <operation name="trainMgmt">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="TrainMgmtFault">
                <soap:fault name="TrainMgmtFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="TrainingMgmtService">
        <port name="TrainingMgmtPort" binding="tns:TrainingMgmtPortBinding">
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                TrainingMgmtService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.1: TrainingMgmtService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified" version="1.0" targetNamespace="http://xml.netbeans.org/schema/
    TrainingMgmtParameters">
    <xs:complexType name="TrainMgmtInputType">
        <xs:sequence>
            <xs:element name="NegativeCategoryCount" type="xs:int" default="100">
                </xs:element>
            <xs:element name="HttpProxyHost" type="xs:anyURI" default="http://www
                -proxy.cs.tcd.ie"></xs:element>
            <xs:element name="HttpProxyPort" type="xs:int" default="8080"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TrainMgmtOutputType">
        <xs:sequence>
            <xs:element name="NegativeCategoryCount" type="xs:int" default="100">
                </xs:element>
            <xs:element name="HttpProxyHost" type="xs:anyURI" default="http://www
                -proxy.cs.tcd.ie"></xs:element>
            <xs:element name="HttpProxyPort" type="xs:int" default="8080"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.2: TrainingMgmtService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://trainingMgmt.service/" xmlns:ns1="http://xml.
    netbeans.org/schema/TrainingMgmtParameters" xmlns:xs="http://www.w3.org/2001/
    XMLSchema" version="1.0" targetNamespace="http://trainingMgmt.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/TrainingMgmtParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
        TrainingMgmtService?xsd=1"></xs:import>
    <xs:element name="TrainMgmtFault" nillable="true" type="tns:faultBean"></
        xs:element>
    <xs:element name="trainMgmt" type="tns:trainMgmt"></xs:element>
    <xs:element name="trainMgmtResponse" type="tns:trainMgmtResponse"></
        xs:element>
    <xs:complexType name="trainMgmt">
        <xs:sequence>
            <xs:element name="trainMgmtRequest" type="ns1:TrainMgmtInputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="trainMgmtResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:TrainMgmtOutputType" minOccurs="0
                "></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="faultBean">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.3: TrainingMgmtService2.xsd

## C.2 TrainingService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://training.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    training.service/" name="TrainingService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                TrainingParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.
                ie:8080/OCCSService/TrainingService?xsd=1"></xsd:import>
        </xsd:schema>
        <xsd:schema>
            <xsd:import namespace="http://training.service/" schemaLocation="
                http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/TrainingService?xsd=2
                "></xsd:import>
```

```
    </xsd:schema>
</types>
<message name="train">
    <part name="parameters" element="tns:train"></part>
</message>
<message name="trainResponse">
    <part name="parameters" element="tns:trainResponse"></part>
</message>
<message name="TrainFault">
    <part name="fault" element="tns:TrainFault"></part>
</message>
<message name="trainStatus">
    <part name="parameters" element="tns:trainStatus"></part>
</message>
<message name="trainStatusResponse">
    <part name="parameters" element="tns:trainStatusResponse"></part>
</message>
<message name="trainTerminate">
    <part name="parameters" element="tns:trainTerminate"></part>
</message>
<message name="trainTerminateResponse">
    <part name="parameters" element="tns:trainTerminateResponse"></part>
</message>
<portType name="Training">
    <operation name="train">
        <input message="tns:train"></input>
        <output message="tns:trainResponse"></output>
        <fault message="tns:TrainFault" name="TrainFault"></fault>
    </operation>
    <operation name="trainStatus">
        <input message="tns:trainStatus"></input>
        <output message="tns:trainStatusResponse"></output>
        <fault message="tns:TrainFault" name="TrainFault"></fault>
    </operation>
    <operation name="trainTerminate">
        <input message="tns:trainTerminate"></input>
        <output message="tns:trainTerminateResponse"></output>
        <fault message="tns:TrainFault" name="TrainFault"></fault>
    </operation>
</portType>
<binding name="TrainingPortBinding" type="tns:Training">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
        document"></soap:binding>
    <operation name="train">
        <soap:operation soapAction=""></soap:operation>
        <input>
            <soap:body use="literal"></soap:body>
        </input>
        <output>
            <soap:body use="literal"></soap:body>
        </output>
        <fault name="TrainFault">
            <soap:fault name="TrainFault" use="literal"></soap:fault>
        </fault>
```

```
        </operation>
        <operation name="trainStatus">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="TrainFault">
                <soap:fault name="TrainFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="trainTerminate">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="TrainFault">
                <soap:fault name="TrainFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="TrainingService">
        <port name="TrainingPort" binding="tns:TrainingPortBinding">
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                TrainingService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.4: TrainingService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified" version="1.0" targetNamespace="http://xml.netbeans.org/schema/
    TrainingParameters">
    <xs:complexType name="TrainStatusInputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="TrainStatusOutputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element name="Status" type="xs:string"></xs:element>
            <xs:element name="IsTraining" type="xs:boolean"></xs:element>
            <xs:element name="IsRainbowRunning" type="xs:boolean"></xs:element>
            <xs:element name="SeedURLs">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="URL" type="xs:anyURI" minOccurs="0"
                            maxOccurs="unbounded"></xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="NegativeODPCategories">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ODPCategory" type="xs:string" minOccurs
                            ="0" maxOccurs="unbounded"></xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TrainTerminateInputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="TrainTerminateOutputType">
        <xs:sequence>
            <xs:element name="Success" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TrainInputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element name="MaxSearchResults" type="xs:int" default="10"></
                xs:element>
            <xs:element name="KeywordPhrases">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="KeywordPhrase" type="xs:string"
                            maxOccurs="unbounded"></xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="PositiveODPCategories">
                <xs:complexType>
```

```
                    <xs:sequence>
                        <xs:element name="ODPCategory" type="xs:string" maxOccurs
                            ="unbounded"></xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="OverwriteFiles" type="xs:boolean" default="false"><
                /xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TrainOutputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element name="Status" type="xs:string"></xs:element>
            <xs:element name="IsTraining" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.5: TrainingService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://training.service/" xmlns:ns1="http://xml.netbeans.
    org/schema/TrainingParameters" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="http://training.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/TrainingParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
        TrainingService?xsd=1"></xs:import>
    <xs:element name="TrainFault" nillable="true" type="tns:faultBean"></
        xs:element>
    <xs:element name="train" type="tns:train"></xs:element>
    <xs:element name="trainResponse" type="tns:trainResponse"></xs:element>
    <xs:element name="trainStatus" type="tns:trainStatus"></xs:element>
    <xs:element name="trainStatusResponse" type="tns:trainStatusResponse"></
        xs:element>
    <xs:element name="trainTerminate" type="tns:trainTerminate"></xs:element>
    <xs:element name="trainTerminateResponse" type="tns:trainTerminateResponse"><
        /xs:element>
    <xs:complexType name="trainStatus">
        <xs:sequence>
            <xs:element name="trainStatusRequest" type="ns1:TrainStatusInputType"
                 minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="trainStatusResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:TrainStatusOutputType" minOccurs=
                "0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="faultBean">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"></
                xs:element>
```

```
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="trainTerminate">
        <xs:sequence>
            <xs:element name="trainTerminateRequest" type="
                ns1:TrainTerminateInputType" minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="trainTerminateResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:TrainTerminateOutputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="train">
        <xs:sequence>
            <xs:element name="trainRequest" type="ns1:TrainInputType" minOccurs="
                0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="trainResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:TrainOutputType" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.6: TrainingService2.xsd

## C.3 TuningMgmtService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://tuningMgmt.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    tuningMgmt.service/" name="TuningMgmtService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                TuningMgmtParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.
                ie:8080/OCCSService/TuningMgmtService?xsd=1"></xsd:import>
        </xsd:schema>
        <xsd:schema>
            <xsd:import namespace="http://tuningMgmt.service/" schemaLocation="
                http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/TuningMgmtService?xsd
                =2"></xsd:import>
        </xsd:schema>
    </types>
    <message name="tuneMgmt">
        <part name="parameters" element="tns:tuneMgmt"></part>
    </message>
```

```
    <message name="tuneMgmtResponse">
        <part name="parameters" element="tns:tuneMgmtResponse"></part>
    </message>
    <message name="TuneMgmtFault">
        <part name="fault" element="tns:TuneMgmtFault"></part>
    </message>
    <portType name="TuningMgmt">
        <operation name="tuneMgmt">
            <input message="tns:tuneMgmt"></input>
            <output message="tns:tuneMgmtResponse"></output>
            <fault message="tns:TuneMgmtFault" name="TuneMgmtFault"></fault>
        </operation>
    </portType>
    <binding name="TuningMgmtPortBinding" type="tns:TuningMgmt">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
            document"></soap:binding>
        <operation name="tuneMgmt">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="TuneMgmtFault">
                <soap:fault name="TuneMgmtFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="TuningMgmtService">
        <port name="TuningMgmtPort" binding="tns:TuningMgmtPortBinding">
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                TuningMgmtService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.7: TuningMgmtService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified" version="1.0" targetNamespace="http://xml.netbeans.org/schema/
    TuningMgmtParameters">
    <xs:complexType name="TuneMgmtInputType">
        <xs:sequence>
            <xs:element name="topWordCount" type="xs:int" default="200"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TuneMgmtOutputType">
        <xs:sequence>
            <xs:element name="topWordCount" type="xs:int" default="200"></
                xs:element>
        </xs:sequence>
```

```
    </xs:complexType>
</xs:schema>
```

Listing C.8: TuningMgmtService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://tuningMgmt.service/" xmlns:ns1="http://xml.netbeans.
    org/schema/TuningMgmtParameters" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="http://tuningMgmt.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/TuningMgmtParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
        TuningMgmtService?xsd=1"></xs:import>
    <xs:element name="TuneMgmtFault" nillable="true" type="tns:faultBean"></
        xs:element>
    <xs:element name="tuneMgmt" type="tns:tuneMgmt"></xs:element>
    <xs:element name="tuneMgmtResponse" type="tns:tuneMgmtResponse"></xs:element>
    <xs:complexType name="tuneMgmt">
        <xs:sequence>
            <xs:element name="tuneMgmtRequest" type="ns1:TuneMgmtInputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="tuneMgmtResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:TuneMgmtOutputType" minOccurs="0"
                ></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="faultBean">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.9: TuningMgmtService2.xsd

## C.4 TuningService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://tuning.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    tuning.service/" name="TuningService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                TuningParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080
                /OCCSService/TuningService?xsd=1"></xsd:import>
        </xsd:schema>
```

```
        <xsd:schema>
            <xsd:import namespace="http://tuning.service/" schemaLocation="http:
                //kdeg-vm-1.cs.tcd.ie:8080/OCCSService/TuningService?xsd=2"></
                xsd:import>
        </xsd:schema>
    </types>
    <message name="getTopWords">
        <part name="parameters" element="tns:getTopWords"></part>
    </message>
    <message name="getTopWordsResponse">
        <part name="parameters" element="tns:getTopWordsResponse"></part>
    </message>
    <message name="TuneFault">
        <part name="fault" element="tns:TuneFault"></part>
    </message>
    <message name="setStopWords">
        <part name="parameters" element="tns:setStopWords"></part>
    </message>
    <message name="setStopWordsResponse">
        <part name="parameters" element="tns:setStopWordsResponse"></part>
    </message>
    <message name="rainbowTerminate">
        <part name="parameters" element="tns:rainbowTerminate"></part>
    </message>
    <message name="rainbowTerminateResponse">
        <part name="parameters" element="tns:rainbowTerminateResponse"></part>
    </message>
    <portType name="Tuning">
        <operation name="getTopWords">
            <input message="tns:getTopWords"></input>
            <output message="tns:getTopWordsResponse"></output>
            <fault message="tns:TuneFault" name="TuneFault"></fault>
        </operation>
        <operation name="setStopWords">
            <input message="tns:setStopWords"></input>
            <output message="tns:setStopWordsResponse"></output>
            <fault message="tns:TuneFault" name="TuneFault"></fault>
        </operation>
        <operation name="rainbowTerminate">
            <input message="tns:rainbowTerminate"></input>
            <output message="tns:rainbowTerminateResponse"></output>
            <fault message="tns:TuneFault" name="TuneFault"></fault>
        </operation>
    </portType>
    <binding name="TuningPortBinding" type="tns:Tuning">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
            document"></soap:binding>
        <operation name="getTopWords">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
```

```
            </output>
            <fault name="TuneFault">
                <soap:fault name="TuneFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="setStopWords">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="TuneFault">
                <soap:fault name="TuneFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="rainbowTerminate">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="TuneFault">
                <soap:fault name="TuneFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="TuningService">
        <port name="TuningPort" binding="tns:TuningPortBinding">
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                TuningService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.10: TuningService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://xml.netbeans.org/schema/TuningParameters" xmlns:xs="
    http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" version="1.0
    " targetNamespace="http://xml.netbeans.org/schema/TuningParameters">
    <xs:element name="StopWords">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="StopWord" type="xs:string" minOccurs="0"
                    maxOccurs="unbounded"></xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="TopWords">
        <xs:complexType>
```

```
            <xs:sequence>
                <xs:element name="TopWord" type="xs:string" minOccurs="0"
                    maxOccurs="unbounded"></xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="SetStopWordsInputType">
        <xs:sequence>
            <xs:element name="OverwriteFiles" type="xs:boolean" default="false"><
                /xs:element>
            <xs:element ref="tns:StopWords"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="SetStopWordsOutputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element ref="tns:TopWords"></xs:element>
            <xs:element ref="tns:StopWords"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="GetTopWordsInputType">
        <xs:sequence>
            <xs:element name="OverwriteFiles" type="xs:boolean" default="false"><
                /xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="GetTopWordsOutputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element ref="tns:TopWords"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RainbowTerminateInputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="RainbowTerminateOutputType">
        <xs:sequence>
            <xs:element name="Success" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.11: TuningService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://tuning.service/" xmlns:ns1="http://xml.netbeans.org/
    schema/TuningParameters" xmlns:xs="http://www.w3.org/2001/XMLSchema" version=
    "1.0" targetNamespace="http://tuning.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/TuningParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/TuningService
        ?xsd=1"></xs:import>
    <xs:element name="TuneFault" nillable="true" type="tns:faultBean"></
        xs:element>
    <xs:element name="getTopWords" type="tns:getTopWords"></xs:element>
```

```
    <xs:element name="getTopWordsResponse" type="tns:getTopWordsResponse"></
        xs:element>
    <xs:element name="rainbowTerminate" type="tns:rainbowTerminate"></xs:element>
    <xs:element name="rainbowTerminateResponse" type="
        tns:rainbowTerminateResponse"></xs:element>
    <xs:element name="setStopWords" type="tns:setStopWords"></xs:element>
    <xs:element name="setStopWordsResponse" type="tns:setStopWordsResponse"></
        xs:element>
    <xs:complexType name="setStopWords">
        <xs:sequence>
            <xs:element name="setStopWordsRequest" type="
                ns1:SetStopWordsInputType" minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="setStopWordsResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:SetStopWordsOutputType" minOccurs
                ="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="faultBean">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="getTopWords">
        <xs:sequence>
            <xs:element name="getTopWordsRequest" type="ns1:GetTopWordsInputType"
                 minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="getTopWordsResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:GetTopWordsOutputType" minOccurs=
                "0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="rainbowTerminate">
        <xs:sequence>
            <xs:element name="rainbowTerminateRequest" type="
                ns1:RainbowTerminateInputType" minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="rainbowTerminateResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:RainbowTerminateOutputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.12: TuningService2.xsd

## C.5 CrawlingMgmtService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://crawlingMgmt.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    crawlingMgmt.service/" name="CrawlingMgmtService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                CrawlingMgmtParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.
                ie:8080/OCCSService/CrawlingMgmtService?xsd=1"></xsd:import>
        </xsd:schema>
        <xsd:schema>
            <xsd:import namespace="http://crawlingMgmt.service/" schemaLocation="
                http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/CrawlingMgmtService?
                xsd=2"></xsd:import>
        </xsd:schema>
    </types>
    <message name="crawlMgmt">
        <part name="parameters" element="tns:crawlMgmt"></part>
    </message>
    <message name="crawlMgmtResponse">
        <part name="parameters" element="tns:crawlMgmtResponse"></part>
    </message>
    <message name="CrawlMgmtFault">
        <part name="fault" element="tns:CrawlMgmtFault"></part>
    </message>
    <portType name="CrawlingMgmt">
        <operation name="crawlMgmt">
            <input message="tns:crawlMgmt"></input>
            <output message="tns:crawlMgmtResponse"></output>
            <fault message="tns:CrawlMgmtFault" name="CrawlMgmtFault"></fault>
        </operation>
    </portType>
    <binding name="CrawlingMgmtPortBinding" type="tns:CrawlingMgmt">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
            document"></soap:binding>
        <operation name="crawlMgmt">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlMgmtFault">
                <soap:fault name="CrawlMgmtFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="CrawlingMgmtService">
        <port name="CrawlingMgmtPort" binding="tns:CrawlingMgmtPortBinding">
```

```
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                CrawlingMgmtService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.13: CrawlingMgmtService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified" version="1.0" targetNamespace="http://xml.netbeans.org/schema/
    CrawlingMgmtParameters">
    <xs:complexType name="CrawlMgmtInputType">
        <xs:sequence>
            <xs:element name="MaxToeThreads" type="xs:integer" default="90"></
                xs:element>
            <xs:element name="MaxLinkHops" type="xs:integer" default="25"></
                xs:element>
            <xs:element name="MaxTransHops" type="xs:integer" default="5"></
                xs:element>
            <xs:element name="HttpProxyHost" type="xs:string" default="
                134.226.32.57"></xs:element>
            <xs:element name="HttpProxyPort" type="xs:int" default="8080"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlMgmtOutputType">
        <xs:sequence>
            <xs:element name="MaxToeThreads" type="xs:integer" default="90"></
                xs:element>
            <xs:element name="MaxLinkHops" type="xs:integer" default="25"></
                xs:element>
            <xs:element name="MaxTransHops" type="xs:integer" default="5"></
                xs:element>
            <xs:element name="HttpProxyHost" type="xs:string" default="
                134.226.32.57"></xs:element>
            <xs:element name="HttpProxyPort" type="xs:int" default="8080"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.14: CrawlingMgmtService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://crawlingMgmt.service/" xmlns:ns1="http://xml.
    netbeans.org/schema/CrawlingMgmtParameters" xmlns:xs="http://www.w3.org/2001/
    XMLSchema" version="1.0" targetNamespace="http://crawlingMgmt.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/CrawlingMgmtParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
        CrawlingMgmtService?xsd=1"></xs:import>
    <xs:element name="CrawlMgmtFault" nillable="true" type="tns:faultBean"></
        xs:element>
```

```
    <xs:element name="crawlMgmt" type="tns:crawlMgmt"></xs:element>
    <xs:element name="crawlMgmtResponse" type="tns:crawlMgmtResponse"></
        xs:element>
    <xs:complexType name="crawlMgmt">
        <xs:sequence>
            <xs:element name="crawlMgmtRequest" type="ns1:CrawlMgmtInputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="crawlMgmtResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:CrawlMgmtOutputType" minOccurs="0
                "></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="faultBean">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.15: CrawlingMgmtService2.xsd

## C.6 CrawlingService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://crawling.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    crawling.service/" name="CrawlingService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                CrawlingParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.
                ie:8080/OCCSService/CrawlingService?xsd=1"></xsd:import>
        </xsd:schema>
        <xsd:schema>
            <xsd:import namespace="http://crawling.service/" schemaLocation="
                http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/CrawlingService?xsd=2
                "></xsd:import>
        </xsd:schema>
    </types>
    <message name="crawl">
        <part name="parameters" element="tns:crawl"></part>
    </message>
    <message name="crawlResponse">
        <part name="parameters" element="tns:crawlResponse"></part>
    </message>
    <message name="CrawlFault">
        <part name="fault" element="tns:CrawlFault"></part>
```

```
    </message>
    <message name="crawlStatus">
        <part name="parameters" element="tns:crawlStatus"></part>
    </message>
    <message name="crawlStatusResponse">
        <part name="parameters" element="tns:crawlStatusResponse"></part>
    </message>
    <message name="crawlPause">
        <part name="parameters" element="tns:crawlPause"></part>
    </message>
    <message name="crawlPauseResponse">
        <part name="parameters" element="tns:crawlPauseResponse"></part>
    </message>
    <message name="crawlResume">
        <part name="parameters" element="tns:crawlResume"></part>
    </message>
    <message name="crawlResumeResponse">
        <part name="parameters" element="tns:crawlResumeResponse"></part>
    </message>
    <message name="crawlTerminate">
        <part name="parameters" element="tns:crawlTerminate"></part>
    </message>
    <message name="crawlTerminateResponse">
        <part name="parameters" element="tns:crawlTerminateResponse"></part>
    </message>
    <message name="crawlerShutdown">
        <part name="parameters" element="tns:crawlerShutdown"></part>
    </message>
    <message name="crawlerShutdownResponse">
        <part name="parameters" element="tns:crawlerShutdownResponse"></part>
    </message>
    <portType name="Crawling">
        <operation name="crawl">
            <input message="tns:crawl"></input>
            <output message="tns:crawlResponse"></output>
            <fault message="tns:CrawlFault" name="CrawlFault"></fault>
        </operation>
        <operation name="crawlStatus">
            <input message="tns:crawlStatus"></input>
            <output message="tns:crawlStatusResponse"></output>
            <fault message="tns:CrawlFault" name="CrawlFault"></fault>
        </operation>
        <operation name="crawlPause">
            <input message="tns:crawlPause"></input>
            <output message="tns:crawlPauseResponse"></output>
            <fault message="tns:CrawlFault" name="CrawlFault"></fault>
        </operation>
        <operation name="crawlResume">
            <input message="tns:crawlResume"></input>
            <output message="tns:crawlResumeResponse"></output>
            <fault message="tns:CrawlFault" name="CrawlFault"></fault>
        </operation>
        <operation name="crawlTerminate">
            <input message="tns:crawlTerminate"></input>
```

```
            <output message="tns:crawlTerminateResponse"></output>
            <fault message="tns:CrawlFault" name="CrawlFault"></fault>
        </operation>
        <operation name="crawlerShutdown">
            <input message="tns:crawlerShutdown"></input>
            <output message="tns:crawlerShutdownResponse"></output>
            <fault message="tns:CrawlFault" name="CrawlFault"></fault>
        </operation>
    </portType>
    <binding name="CrawlingPortBinding" type="tns:Crawling">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
            document"></soap:binding>
        <operation name="crawl">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlFault">
                <soap:fault name="CrawlFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="crawlStatus">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlFault">
                <soap:fault name="CrawlFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="crawlPause">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlFault">
                <soap:fault name="CrawlFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="crawlResume">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
```

```
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlFault">
                <soap:fault name="CrawlFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="crawlTerminate">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlFault">
                <soap:fault name="CrawlFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="crawlerShutdown">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="CrawlFault">
                <soap:fault name="CrawlFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="CrawlingService">
        <port name="CrawlingPort" binding="tns:CrawlingPortBinding">
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                CrawlingService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.16: CrawlingService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?><!-- Published by JAX-WS RI at http://jax-
    ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.3-hudson-757-SNAPSHOT. -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified" version="1.0" targetNamespace="http://xml.netbeans.org/schema/
    CrawlingParameters">
    <xs:complexType name="CrawlResumeInputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlResumeOutputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
```

```
            <xs:element name="Status" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlerShutdownInputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlerShutdownOutputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlStatusInputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlStatusOutputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
            <xs:element name="Status" type="xs:string"></xs:element>
            <xs:element name="IsCrawling" type="xs:boolean"></xs:element>
            <xs:element name="TotalData" type="xs:long"></xs:element>
            <xs:element name="CrawlTime" type="xs:long"></xs:element>
            <xs:element name="DiscoveredDocuments" type="xs:long"></xs:element>
            <xs:element name="QueuedDocuments" type="xs:long"></xs:element>
            <xs:element name="DownloadedDocuments" type="xs:long"></xs:element>
            <xs:element name="DownloadFailures" type="xs:long"></xs:element>
            <xs:element name="MemoryUseKB" type="xs:long"></xs:element>
            <xs:element name="HeapSizeKB" type="xs:long"></xs:element>
            <xs:element name="ActiveThreadCount" type="xs:int"></xs:element>
            <xs:element name="CurrentDocumentRate" type="xs:double"></xs:element>
            <xs:element name="AverageDocumentRate" type="xs:double"></xs:element>
            <xs:element name="CurrentKBRate" type="xs:long"></xs:element>
            <xs:element name="AverageKBRate" type="xs:long"></xs:element>
            <xs:element name="Congestion" type="xs:double"></xs:element>
            <xs:element name="MaxDepth" type="xs:long"></xs:element>
            <xs:element name="AverageDepth" type="xs:int"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlTerminateInputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlTerminateOutputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
            <xs:element name="Success" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlInputType">
        <xs:sequence>
            <xs:element name="JobName" type="xs:string"></xs:element>
            <xs:element name="Organisation" type="xs:string"></xs:element>
            <xs:element name="OrganisationURL" type="xs:anyURI"></xs:element>
            <xs:element name="EmailAddress" type="xs:string"></xs:element>
```

```
            <xs:element name="MaxBytesDownload" type="xs:long" default="0"></
                xs:element>
            <xs:element name="MaxDocumentDownload" type="xs:long" default="0"></
                xs:element>
            <xs:element name="MaxTimeSec" type="xs:long" default="0"></xs:element
                >
            <xs:element name="MaxPathDepth" type="xs:integer" default="20"></
                xs:element>
            <xs:element name="CutOff" type="xs:double" default="0.95"></
                xs:element>
            <xs:element name="Language" type="xs:string" default="english"></
                xs:element>
            <xs:element name="OverwriteFiles" type="xs:boolean" default="false
                "></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlOutputType">
        <xs:sequence>
            <xs:element name="JobName" type="xs:string"></xs:element>
            <xs:element name="JobUID" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlPauseInputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CrawlPauseOutputType">
        <xs:sequence>
            <xs:element name="JobUID" type="xs:string"></xs:element>
            <xs:element name="Status" type="xs:string"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.17: CrawlingService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?><!-- Published by JAX-WS RI at http://jax-
    ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.3-hudson-757-SNAPSHOT. -->
<xs:schema xmlns:tns="http://crawling.service/" xmlns:ns1="http://xml.netbeans.
    org/schema/CrawlingParameters" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="http://crawling.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/CrawlingParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
        CrawlingService?xsd=1"></xs:import>
    <xs:element name="CrawlFault" nillable="true" type="tns:faultBean"></
        xs:element>
    <xs:element name="crawl" type="tns:crawl"></xs:element>
    <xs:element name="crawlPause" type="tns:crawlPause"></xs:element>
    <xs:element name="crawlPauseResponse" type="tns:crawlPauseResponse"></
        xs:element>
    <xs:element name="crawlResponse" type="tns:crawlResponse"></xs:element>
    <xs:element name="crawlResume" type="tns:crawlResume"></xs:element>
```

```
<xs:element name="crawlResumeResponse" type="tns:crawlResumeResponse"></
    xs:element>
<xs:element name="crawlStatus" type="tns:crawlStatus"></xs:element>
<xs:element name="crawlStatusResponse" type="tns:crawlStatusResponse"></
    xs:element>
<xs:element name="crawlTerminate" type="tns:crawlTerminate"></xs:element>
<xs:element name="crawlTerminateResponse" type="tns:crawlTerminateResponse
    "></xs:element>
<xs:element name="crawlerShutdown" type="tns:crawlerShutdown"></xs:element>
<xs:element name="crawlerShutdownResponse" type="tns:crawlerShutdownResponse
    "></xs:element>
<xs:complexType name="crawlResume">
    <xs:sequence>
        <xs:element name="crawlResumeRequest" type="ns1:CrawlResumeInputType"
             minOccurs="0"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="crawlResumeResponse">
    <xs:sequence>
        <xs:element name="return" type="ns1:CrawlResumeOutputType" minOccurs
            ="0"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="faultBean">
    <xs:sequence>
        <xs:element name="message" type="xs:string" minOccurs="0"></
            xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="crawlerShutdown">
    <xs:sequence>
        <xs:element name="crawlerShutdownRequest" type="
            ns1:CrawlerShutdownInputType" minOccurs="0"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="crawlerShutdownResponse">
    <xs:sequence>
        <xs:element name="return" type="ns1:CrawlerShutdownOutputType"
            minOccurs="0"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="crawlStatus">
    <xs:sequence>
        <xs:element name="crawlStatusRequest" type="ns1:CrawlStatusInputType"
             minOccurs="0"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="crawlStatusResponse">
    <xs:sequence>
        <xs:element name="return" type="ns1:CrawlStatusOutputType" minOccurs
            ="0"></xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="crawlTerminate">
```

```
        <xs:sequence>
            <xs:element name="crawlTerminateRequest" type="
                ns1:CrawlTerminateInputType" minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="crawlTerminateResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:CrawlTerminateOutputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="crawl">
        <xs:sequence>
            <xs:element name="crawlRequest" type="ns1:CrawlInputType" minOccurs
                ="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="crawlResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:CrawlOutputType" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="crawlPause">
        <xs:sequence>
            <xs:element name="crawlPauseRequest" type="ns1:CrawlPauseInputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="crawlPauseResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:CrawlPauseOutputType" minOccurs
                ="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.18: CrawlingService2.xsd

## C.7 IndexingService WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    " xmlns:tns="http://indexing.service/" xmlns:xsd="http://www.w3.org/2001/
    XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://
    indexing.service/" name="IndexingService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://xml.netbeans.org/schema/
                IndexingParameters" schemaLocation="http://kdeg-vm-1.cs.tcd.
                ie:8080/OCCSService/IndexingService?xsd=1"></xsd:import>
        </xsd:schema>
```

```
        <xsd:schema>
            <xsd:import namespace="http://indexing.service/" schemaLocation="
                http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/IndexingService?xsd=2
                "></xsd:import>
        </xsd:schema>
    </types>
    <message name="index">
        <part name="parameters" element="tns:index"></part>
    </message>
    <message name="indexResponse">
        <part name="parameters" element="tns:indexResponse"></part>
    </message>
    <message name="IndexFault">
        <part name="fault" element="tns:IndexFault"></part>
    </message>
    <message name="indexStatus">
        <part name="parameters" element="tns:indexStatus"></part>
    </message>
    <message name="indexStatusResponse">
        <part name="parameters" element="tns:indexStatusResponse"></part>
    </message>
    <message name="indexTerminate">
        <part name="parameters" element="tns:indexTerminate"></part>
    </message>
    <message name="indexTerminateResponse">
        <part name="parameters" element="tns:indexTerminateResponse"></part>
    </message>
    <portType name="Indexing">
        <operation name="index">
            <input message="tns:index"></input>
            <output message="tns:indexResponse"></output>
            <fault message="tns:IndexFault" name="IndexFault"></fault>
        </operation>
        <operation name="indexStatus">
            <input message="tns:indexStatus"></input>
            <output message="tns:indexStatusResponse"></output>
            <fault message="tns:IndexFault" name="IndexFault"></fault>
        </operation>
        <operation name="indexTerminate">
            <input message="tns:indexTerminate"></input>
            <output message="tns:indexTerminateResponse"></output>
            <fault message="tns:IndexFault" name="IndexFault"></fault>
        </operation>
    </portType>
    <binding name="IndexingPortBinding" type="tns:Indexing">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="
            document"></soap:binding>
        <operation name="index">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
```

```
            </output>
            <fault name="IndexFault">
                <soap:fault name="IndexFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="indexStatus">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="IndexFault">
                <soap:fault name="IndexFault" use="literal"></soap:fault>
            </fault>
        </operation>
        <operation name="indexTerminate">
            <soap:operation soapAction=""></soap:operation>
            <input>
                <soap:body use="literal"></soap:body>
            </input>
            <output>
                <soap:body use="literal"></soap:body>
            </output>
            <fault name="IndexFault">
                <soap:fault name="IndexFault" use="literal"></soap:fault>
            </fault>
        </operation>
    </binding>
    <service name="IndexingService">
        <port name="IndexingPort" binding="tns:IndexingPortBinding">
            <soap:address location="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
                IndexingService"></soap:address>
        </port>
    </service>
</definitions>
```

Listing C.19: IndexingService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified" version="1.0" targetNamespace="http://xml.netbeans.org/schema/
    IndexingParameters">
    <xs:complexType name="IndexStatusInputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="IndexStatusOutputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element name="Status" type="xs:string"></xs:element>
            <xs:element name="IsIndexing" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
```

```
    <xs:complexType name="IndexTerminateInputType">
        <xs:sequence></xs:sequence>
    </xs:complexType>
    <xs:complexType name="IndexTerminateOutputType">
        <xs:sequence>
            <xs:element name="Success" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="IndexInputType">
        <xs:sequence>
            <xs:element name="VirtualCache" type="xs:boolean" default="false"></
                xs:element>
            <xs:element name="OverwriteFiles" type="xs:boolean" default="false"><
                /xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="IndexOutputType">
        <xs:sequence>
            <xs:element name="Topic" type="xs:string"></xs:element>
            <xs:element name="Status" type="xs:string"></xs:element>
            <xs:element name="IsIndexing" type="xs:boolean"></xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.20: IndexingService1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://indexing.service/" xmlns:ns1="http://xml.netbeans.
    org/schema/IndexingParameters" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" targetNamespace="http://indexing.service/">
    <xs:import namespace="http://xml.netbeans.org/schema/IndexingParameters"
        schemaLocation="http://kdeg-vm-1.cs.tcd.ie:8080/OCCSService/
        IndexingService?xsd=1"></xs:import>
    <xs:element name="IndexFault" nillable="true" type="tns:faultBean"></
        xs:element>
    <xs:element name="index" type="tns:index"></xs:element>
    <xs:element name="indexResponse" type="tns:indexResponse"></xs:element>
    <xs:element name="indexStatus" type="tns:indexStatus"></xs:element>
    <xs:element name="indexStatusResponse" type="tns:indexStatusResponse"></
        xs:element>
    <xs:element name="indexTerminate" type="tns:indexTerminate"></xs:element>
    <xs:element name="indexTerminateResponse" type="tns:indexTerminateResponse"><
        /xs:element>
    <xs:complexType name="indexStatus">
        <xs:sequence>
            <xs:element name="indexStatusRequest" type="ns1:IndexStatusInputType"
                 minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="indexStatusResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:IndexStatusOutputType" minOccurs=
                "0"></xs:element>
```

```
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="faultBean">
        <xs:sequence>
            <xs:element name="message" type="xs:string" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="indexTerminate">
        <xs:sequence>
            <xs:element name="indexTerminateRequest" type="
                ns1:IndexTerminateInputType" minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="indexTerminateResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:IndexTerminateOutputType"
                minOccurs="0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="index">
        <xs:sequence>
            <xs:element name="indexRequest" type="ns1:IndexInputType" minOccurs="
                0"></xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="indexResponse">
        <xs:sequence>
            <xs:element name="return" type="ns1:IndexOutputType" minOccurs="0"></
                xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Listing C.21: IndexingService2.xsd

# Appendix D

# Evaluation

## D.1 Example Questionnaire

**General Questions**

Question 1 - Please enter your name:

_____

Question 2 - What is your level of competency with Web Services and Web Services technologies?

- ❑ Very high
- ❑ High
- ❑ Neither high nor low
- ❑ Low
- ❑ Very low

Question 3 - Please list the technologies you used to develop your client application:

_____

_____

_____

Question 4 - In a few lines please describe the client application you developed:

_____

_____

_____

**Training Service Questions**

Question 5 - Understanding the data requirements for the Training API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 6 - Understanding the invocation requirements for the Training API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 7 - The error handling and messages produced by the Training API were:

- ❑ Very informative
- ❑ Informative
- ❑ Neither informative nor uninformative
- ❑ Uninformative
- ❑ Very uninformative

Question 8 - The Training API was:

- ❑ Very efficient in performing in a timely manner
- ❑ Efficient in performing in a timely manner
- ❑ Neither efficient nor inefficient in performing in a timely manner
- ❑ Inefficient in performing in a timely manner
- ❑ Very inefficient and performed in an untimely manner

Question 9 - The Training API was:

- ❑ Very effective in fulfilling its purpose
- ❑ Effective in fulfilling its purpose
- ❑ Neither effective nor ineffective in fulfilling its purpose
- ❑ Ineffective in fulfilling its purpose
- ❑ Very ineffective in fulfilling its purpose

Question 10 - The management functionality provided by the Training Mgmt API was:

- ❑ Very sufficient
- ❑ Sufficient
- ❑ Neither sufficient nor insufficient
- ❑ Insufficient
- ❑ Very insufficient

Question 11 - Please comment on one thing you liked about the Training API:

Question 12 - Please comment on one thing that could be improved about the Training API:

_____

_____

_____

**Tuning Service Questions**

Question 13 - Understanding the data requirements for the Tuning API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 14 - Understanding the invocation requirements for the Tuning API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 15 - The error handling and messages produced by the Tuning API were:

- ❑ Very informative
- ❑ Informative
- ❑ Neither informative nor uninformative
- ❑ Uninformative
- ❑ Very uninformative

Question 16 - The Tuning API was:

- ❑ Very efficient in performing in a timely manner
- ❑ Efficient in performing in a timely manner
- ❑ Neither efficient nor inefficient in performing in a timely manner
- ❑ Inefficient in performing in a timely manner
- ❑ Very inefficient and performed in an untimely manner

Question 17 - The Tuning API was:

- ❑ Very effective in fulfilling its purpose
- ❑ Effective in fulfilling its purpose
- ❑ Neither effective nor ineffective in fulfilling its purpose
- ❑ Ineffective in fulfilling its purpose
- ❑ Very ineffective in fulfilling its purpose

Question 18 - The management functionality provided by the Tuning Mgmt API was:

- ❑ Very sufficient
- ❑ Sufficient
- ❑ Neither sufficient nor insufficient
- ❑ Insufficient
- ❑ Very insufficient

Question 19 - Please comment on one thing you liked about the Tuning API:

_____

_____

_____

Question 20 - Please comment on one thing that could be improved about the Tuning API:

_____

_____

_____

**Crawling Service Questions**

Question 21 - Understanding the data requirements for the Crawling Service API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 22 - Understanding the invocation requirements for the Crawling API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 23 - The error handling and messages produced by the Crawling API were:

- ❑ Very informative
- ❑ Informative
- ❑ Neither informative nor uninformative
- ❑ Uninformative
- ❑ Very uninformative

Question 24 - The Crawling API was:

- ❑ Very efficient in performing in a timely manner
- ❑ Efficient in performing in a timely manner
- ❑ Neither efficient nor inefficient in performing in a timely manner
- ❑ Inefficient in performing in a timely manner
- ❑ Very inefficient and performed in an untimely manner

Question 25 - The Crawling API was:

- ❑ Very effective in fulfilling its purpose
- ❑ Effective in fulfilling its purpose
- ❑ Neither effective nor ineffective in fulfilling its purpose
- ❑ Ineffective in fulfilling its purpose
- ❑ Very ineffective in fulfilling its purpose

Question 26 - The management functionality provided by the Crawling Mgmt API was:

- ❑ Very sufficient
- ❑ Sufficient
- ❑ Neither sufficient nor insufficient
- ❑ Insufficient
- ❑ Very insufficient

Question 27 - Please comment on one thing you liked about the Crawling Service API:

_____

_____

_____

Question 28 - Please comment on one thing that could be improved about the Crawling Service API:

_____

_____

_____

**Indexing Service Questions**

Question 29 - Understanding the data requirements for the Indexing Service API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 30 - Understanding the invocation requirements for the Indexing API was:

- ❑ Very easy
- ❑ Easy
- ❑ Neither easy nor complex
- ❑ Complex
- ❑ Very complex

Question 31 - The error handling and messages produced by the Indexing API were:

- ❑ Very informative
- ❑ Informative
- ❑ Neither informative nor uninformative
- ❑ Uninformative
- ❑ Very uninformative

Question 32 - The Indexing API was:

- ❑ Very efficient in performing in a timely manner
- ❑ Efficient in performing in a timely manner
- ❑ Neither efficient nor inefficient in performing in a timely manner
- ❑ Inefficient in performing in a timely manner
- ❑ Very inefficient and performed in an untimely manner

Question 33 - The Indexing API was:

- ❑ Very effective in fulfilling its purpose
- ❑ Effective in fulfilling its purpose
- ❑ Neither effective nor ineffective in fulfilling its purpose
- ❑ Ineffective in fulfilling its purpose
- ❑ Very ineffective in fulfilling its purpose

Question 34 - Please comment on one thing you liked about the Indexing Service API:

_____

_____

_____

Question 35 - Please comment on one thing that could be improved about the Indexing Service API:

_____

_____

_____

## D.2 Questionnaire Answers

**General Questions**

Question 1 - Please enter your name:

Table D.1: Question 1 - Answers

| |
|---|
| User 1 |
| User 2 |
| User 3 |
| User 4 |

Question 2 - What is your level of competency with Web Services and Web Services technologies?

Table D.2: Question 2 - Answers

| Very high | High | Neither high nor low | Low | Very Low |
|---|---|---|---|---|
| ✓ | | | | |
| ✓ | | | | |
| | ✓ | | | |
| | ✓ | | | |

Question 3 - Please list the technologies you used to develop your client application:

Table D.3: Question 3 - Answers

| |
|---|
| python 2.7 OSX Suds ( https://fedorahosted.org/suds/ ) version: 0.3.9 GA build: R659-20100219 |
| Java, Apache Axis2 |
| I used C# with .NET version 3.5. The web service proxy code was auto-generated by Visual Studio .NET for me. |
| Soap UI 3.0.1 |

Question 4 - In a few lines please describe the client application you developed:

Table D.4: Question 4 - Answers

| |
|---|
| Command line client that takes each stage of the crawl in turn |
| Basic java application that separated the different functionalities of the WS into different functions. Each function, would kick off a task such as training or crawling and then poll the status of that task with a linear backoff |
| The application was a very simple console application. It carried out a training, crawling and indexing. The input keywords were names of HBO TV series. The ODP category was the HBO television channel. The application did not make use of the management services. It kicked off a training and waited for it to complete, kicked off a crawl and waited for it to complete, kicked off an indexing and waited for it to complete. During the crawl, status information about the crawl was printed in the console. |
| I tested each of the services in Turn, as documented. |

**Training Service Questions**

Question 5 - Understanding the data requirements for the Training API was:

Table D.5: Question 5 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | ✓ | | | |
| | | ✓ | | |
| | | | ✓ | |
| | ✓ | | | |

Question 6 - Understanding the invocation requirements for the Training API was:

Table D.6: Question 6 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | | | ✓ | |
| | | ✓ | | |
| | | | ✓ | |
| | ✓ | | | |

Question 7 - The error handling and messages produced by the Training API were:

Table D.7: Question 7 - Answers

| Very informative | Informative | Neither informative nor uninformative | Uninformative | Very uninformative |
|---|---|---|---|---|
| | | | ✓ | |
| | | | ✓ | |
| | ✓ | | | |
| ✓ | | | | |

Question 8 - The Training API was:

Table D.8: Question 8 - Answers

| Very efficient in performing in a timely manner | Efficient in performing in a timely manner | Neither efficient nor inefficient in performing in a timely manner | Inefficient in performing in a timely manner | Very inefficient and performed in an untimely manner |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |

Question 9 - The Training API was:

Table D.9: Question 9 - Answers

| Very effective in fulfilling its purpose | Effective in fulfilling its purpose | Neither effective nor ineffective in fulfilling its purpose | Ineffective in fulfilling its purpose | Very ineffective in fulfilling its purpose |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| ✓ | | | | |
| | ✓ | | | |

Question 10 - The management functionality provided by the Training Mgmt API was:

Table D.10: Question 10 - Answers

| Very sufficient | Sufficient | Neither sufficient nor insufficient | Insufficient | Very insufficient |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |

Question 11 - Please comment on one thing you liked about the Training API:

Table D.11: Question 11 - Answers

| |
|---|
| It was good to be able to specify multiple options |
| |
| The interface was very simple and uncluttered. |
| The ability to monitor the training as it occurs. |

Question 12 - Please comment on one thing that could be improved about the Training API:

Table D.12: Question 12 - Answers

| |
|---|
| I would have liked to be able to use full category names as the topic |
| Schema for messages was too complex and did not seem to conform to any of the recommended styles. This made the auto generated client code difficult to work with. I would have preferred a more document/literal style of messaging. |
| At the time I was slightly confused over the difference between a Positive ODP Category and Topic. A better explanation of the input parameters in the documentation would have been good. I see that has been fixed in the newest version. |
| Not enough Documentation on what the Training API was actually doing. |

**Tuning Service Questions**

Question 13 - Understanding the data requirements for the Tuning API was:

Table D.13: Question 13 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | | ✓ | | |
| | | | ✓ | |
| | | ✓ | | |
| | ✓ | | | |

Question 14 - Understanding the invocation requirements for the Tuning API was:

Table D.14: Question 14 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | | | ✓ | |
| | | | ✓ | |
| | | ✓ | | |
| | | | ✓ | |

Question 15 - The error handling and messages produced by the Tuning API were:

Table D.15: Question 15 - Answers

| Very informative | Informative | Neither informative nor uninformative | Uninformative | Very uninformative |
|---|---|---|---|---|
| | ✓ | | | |
| | | ✓ | | |
| | | ✓ | | |
| ✓ | | | | |

Question 16 - The Tuning API was:

Table D.16: Question 16 - Answers

| Very efficient in performing in a timely manner | Efficient in performing in a timely manner | Neither efficient nor inefficient in performing in a timely manner | Inefficient in performing in a timely manner | Very inefficient and performed in an untimely manner |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | | ✓ | | |
| | | ✓ | | |

Question 17 - The Tuning API was:

Table D.17: Question 17 - Answers

| Very effective in fulfilling its purpose | Effective in fulfilling its purpose | Neither effective nor ineffective in fulfilling its purpose | Ineffective in fulfilling its purpose | Very ineffective in fulfilling its purpose |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | | ✓ | | |
| | | ✓ | | |

Question 18 - The management functionality provided by the Tuning Mgmt API was:

Table D.18: Question 18 - Answers

| Very sufficient | Sufficient | Neither sufficient nor insufficient | Insufficient | Very insufficient |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | | ✓ | | |
| | ✓ | | | |

Question 19 - Please comment on one thing you liked about the Tuning API:

Table D.19: Question 19 - Answers

| |
|---|
| Listing stopwords was very responsive. |
| |
| I didn't make use of the Tuning API so I can't really say. The tuning was optional and I chose not to use it. |
| |

Question 20 - Please comment on one thing that could be improved about the Tuning API:

Table D.20: Question 20 - Answers

| |
|---|
| GetTopWordsInputType does not need to be complex, in my opinion |
| See comments regarding training API |
| I didn't make use of the Tuning API so I can't really say. The tuning was optional and I chose not to use it. |
| Requires knowledge on web crawling. Which is a different set of knowledge than that of utilizing Web Services. Need to explain the steps involved in detail. |

**Crawling Service Questions**

Question 21 - Understanding the data requirements for the Crawling Service API was:

Table D.21: Question 21 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| ✓ | | | | |
| | | | ✓ | |
| | | | ✓ | |
| | | | ✓ | |

Question 22 - Understanding the invocation requirements for the Crawling API was:

Table D.22: Question 22 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | | ✓ | | |
| | | | ✓ | |
| | | | | ✓ |
| | | | ✓ | |

Question 23 - The error handling and messages produced by the Crawling API were:

Table D.23: Question 23 - Answers

| Very informative | Informative | Neither informative nor uninformative | Uninformative | Very uninformative |
|---|---|---|---|---|
| | ✓ | | | |
| | | ✓ | | |
| ✓ | | | | |
| ✓ | | | | |

Question 24 - The Crawling API was:

Table D.24: Question 24 - Answers

| Very efficient in performing in a timely manner | Efficient in performing in a timely manner | Neither efficient nor inefficient in performing in a timely manner | Inefficient in performing in a timely manner | Very inefficient and performed in an untimely manner |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |

Question 25 - The Crawling API was:

Table D.25: Question 25 - Answers

| Very effective in fulfilling its purpose | Effective in fulfilling its purpose | Neither effective nor ineffective in fulfilling its purpose | Ineffective in fulfilling its purpose | Very ineffective in fulfilling its purpose |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |

Question 26 - The management functionality provided by the Crawling Mgmt API was:

Table D.26: Question 26 - Answers

| Very sufficient | Sufficient | Neither sufficient nor insufficient | Insufficient | Very insufficient |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |
| | ✓ | | | |

Question 27 - Please comment on one thing you liked about the Crawling Service API:

Table D.27: Question 27 - Answers

| |
|---|
| Useful to be able to pause and resume. |
| |
| The detailed status information was very useful. It was good to know at what stage the crawl was. I found this the most impressive part of all the services. |
| Liked the motoring service so was able to monitor each job. |

Question 28 - Please comment on one thing that could be improved about the Crawling Service API:

Table D.28: Question 28 - Answers

| |
|---|
| Job ID management could be easier. For example, by including other ways of specifying crawl jobs. |
| The inconsistency in the responses of the Status operations across the different APIs was an issue, some responded with 'Finished' while others responded with 'FINISHED'. So also had additional information in the string. It was not clear which operation was most suitable for polling the status of the services, getStatus and isCrawling both seemed to be appropriate (I assumed isCrawling and it's corresponding operations for other APIs was the most appropriate) |
| Heretrix ran into a problem while downloading one of the files. This problem wasn't visible through the service interface. Maybe the service could detect if a crawl was hanging and kill it after a certain timeout, throwing an error. |
| What happens if I loose my job ID? How can I recover it? |

**Indexing Service Questions**

Question 29 - Understanding the data requirements for the Indexing Service API was:

Table D.29: Question 29 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | | ✓ | | |
| | | | ✓ | |
| | | | ✓ | |
| | | | ✓ | |

Question 30 - Understanding the invocation requirements for the Indexing API was:

Table D.30: Question 30 - Answers

| Very complex | Complex | Neither easy nor complex | Easy | Very easy |
|---|---|---|---|---|
| | | | ✓ | |
| | | | ✓ | |
| | | | ✓ | |
| | | | ✓ | |

Question 31 - The error handling and messages produced by the Indexing API were:

Table D.31: Question 31 - Answers

| Very informative | Informative | Neither informative nor uninformative | Uninformative | Very uninformative |
|---|---|---|---|---|
| | | | ✓ | |
| | | ✓ | | |
| | ✓ | | | |
| ✓ | | | | |

Question 32 - The Indexing API was:

Table D.32: Question 32 - Answers

| Very efficient in performing in a timely manner | Efficient in performing in a timely manner | Neither efficient nor inefficient in performing in a timely manner | Inefficient in performing in a timely manner | Very inefficient and performed in an untimely manner |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| ✓ | | | | |
| | ✓ | | | |

Question 33 - The Indexing API was:

Table D.33: Question 33 - Answers

| Very effective in fulfilling its purpose | Effective in fulfilling its purpose | Neither effective nor ineffective in fulfilling its purpose | Ineffective in fulfilling its purpose | Very ineffective in fulfilling its purpose |
|---|---|---|---|---|
| | ✓ | | | |
| | ✓ | | | |
| ✓ | | | | |
| | ✓ | | | |

Question 34 - Please comment on one thing you liked about the Indexing Service API:

Table D.34: Question 34 - Answers

| |
|---|
| |
| |
| Again the service interface was very simple. The naming of the operations and parameters was very clear so it was obvious how to use the service. I liked how you designed all 4 services in a consistent fashion. There was the same pattern of interaction. Once you learned how to use one, it was obvious how to use the rest. |
| |

Question 35 - Please comment on one thing that could be improved about the Indexing Service API:

Table D.35: Question 35 - Answers

| |
|---|
| Consistency between Status and IsIndexing could be improved. |
| The responses of the status and isIndexing operations seemed to be inconsistent and did not accurately reflect the actual status of the service at all times. |
| Maybe some extra information about how the indexing is progressing could be included in the status responses. The crawling status information was very detailed and I found that a really nice feature. Maybe something similar could be implemented for indexing. |
| |