ERCIS
European Research Center for Information Systems

Working Paper No. 20

Implementing the WiPo Architecture

Stahl, F. ■
Godde, A. ■
Hagedorn, B. ■
Köpcke, B. ■
Rehberger, M. ■
Vossen, G. ■

ERCIS – European Research Center for Information Systems

Westfälische
Wilhelms-Universität
Münster

Working Papers

ERCIS – European Research Center for Information Systems
Editors: J. Becker, K. Backhaus, H. L. Grob, T. Hoeren, S. Klein,
H. Kuchen, U. Müller-Funk, U. W. Thonemann, G. Vossen

Working Paper No. 20

Implementing the WiPo Architecture

Florian Stahl, Adrian Godde, Bastian Hagedorn, Bastian Köpcke, Martin Rehberger, Gottfried Vossen

ISSN 1614-7448

cite as: Florian Stahl, Adrian Godde, Bastian Hagedorn, Bastian Köpcke, Martin Rehberger, Gottfried Vossen: Implementing the WiPo Architecture. In: Working Paper No. 20, European Research Center for Information Systems, Eds.: Becker, J. et al. Münster 2014.

Contents

List of Figures

Abstract

Information available online has exploded over the last 20 years. However this lead to the phenomenon known as information overload, making it harder and harder for organisations and individuals to find information that is really relevant to them. We previously have proposed a solution to this problem by presenting a framework that does not answer queries solely based on a pre-assembled index, but based on a subject-specific database that is sourced from the Internet, curated by domain experts, and dynamically generated based on vast user input. In this paper we describe our prototypic implementation of an information platform that delivers high quality information to users, achieved by means of curation.

Keywords

Web in Your Pocket, Information Provisioning, High Quality Information, Curation, Architecture

1 Introduction

Undoubtedly, information available online has exploded over the last 20 years. While this has the advantage that information is made available which previously was not, at the same time, it becomes increasingly harder for organisations and individuals to find data that are really relevant to them. This phenomenon — known as information overload — is not new, however, no satisfying solution has been found so far.

As outlined in [11], early approaches of coping with the plenitude of information on the Web included directories and search engines of different kinds. Out of these, algorithmic search engines which automatically compile huge indices are most successful. Nevertheless, as we argued in [4] there are still cases, where more specific information — generally niche information — is needed that cannot be provided by current search technologies.

Based on this observation, we have proposed a solution to this problem by presenting a framework that does not answer queries solely based on a pre-assembled index, but based on a subject-specific database that is sourced from the Internet, curated by domain experts, and dynamically generated based on vast user input [4]. In this way, completeness, accuracy, quality, and freshness of data can be achieved.

Curation in our understanding refers to the long-term selection, cleansing, enrichment, preservation and retention of data and to provide access to them in a variety of forms. The concept is known, for example, from museums and has gained recognition in the library science where it focuses on preserving data gained through scientific experiments for later usage [2, 8]. As an institution the Digital Curation Centre (DCC[1]) peruse curation in Great Britain as described for instance in [7].

Since our framework is also supposed to make relevant information accessible through a mobile device when no Internet connection is available, we refer to it as Web in your Pocket (WiPo) [4].

To our knowledge, there are only a few other approaches of combining curation and Web Information Retrieval. As outlined in [4], Sanderson et al. [10] suggested to apply a procedure where predefined queries are send to pre-registered services on a nightly basis. Relevant information is harvested and temporarily stored. The retrieved information is then audited by data curators who decide upon its relevance. A similar harvest and curate approach was suggested by Lee et al. [6] who outline ideas to enhance their ContextMiner[2]. However, neither of these is able to make information available off-line on a mobile device.

Data Tamer [12] is similar to our approach in that it is also developed as a system to curate data partially manual. However, it operates more on a database level, particularly on problems of schema integration and entity consolidation, while our approach is more high-level and deals mostly with (textual) information rather than pure data. That said, we use the terms data and information mostly as synonyms throughout this paper and in our previous works.

Having argued how WiPo can help solving the problem of information overload in [5], it was a logical next step to implement a prototype as proof of concept and as a vehicle for practical experiments. We have adopted it to the tourism case described in [4] with the aim of implementing an information platform that delivers high quality data to users. To do so, results are highly user-specific depending on pre-set profiles and user preferences. Further, we wanted to mine information from the Internet and enable curation through an easy to use Web interface as well as enabling off-line availability of selected data.

[1] http://www.dcc.ac.uk/

[2] http://contextminer.org/

In this paper, the basic ideas of WiPo will be recapitulated in Section 2. Then, Section 3 describes the implemented prototype in some detail. Last, the paper will be concluded in Section 4 by summarising key aspects as well as by outlining future work and possible application scenarios.

2 The WiPo Concept

We first introduced the WiPo concept in [4] and the following elaborations are mainly based on this original work. WiPo can be described as process-based approach to information gathering, Web search, and data curation as shown in Figure 1, where thin arrows indicate a relationship between two items and block arrows represent the flow of data within the process. Of course the concept provides for querying the curated database directly (at the top of Figure 1). However, the process of gathering data is somewhat more complex. Starting from the user input our process consists of the following steps:

- Input preparation, resulting in a list of potential sources and pre-filters.
- Source selection, resulting in a list of sources to which pre-filters are applied.
- Data mining executed on the Web, resulting in raw data to which post-filters can be applied which reduce the raw data relevant raw data.
- Curation, where information is checked for quality and interlinked.

The process can be detailed as follows. First, users will supply keywords, a list of relevant links and potentially also documents they already have. Secondly, any given URLs will be crawled and fed into a classifier. Furthermore, input preparation will convert provided documents to a standard file format (e. g., XML) so that generic classifier can be applied and the results are fed into the curation process, too. By doing that, the essence is extracted form URLs and documents in the form of topics or additional keywords. These mined keywords as well as the user-supplied keywords are then used to choose appropriate sources from a list of all available sources (which might also be extended based on user supplied URLs). Furthermore, relevant additional dimensions such as time or location are determined to generate a list of data sources. Pre-filters might also be applied

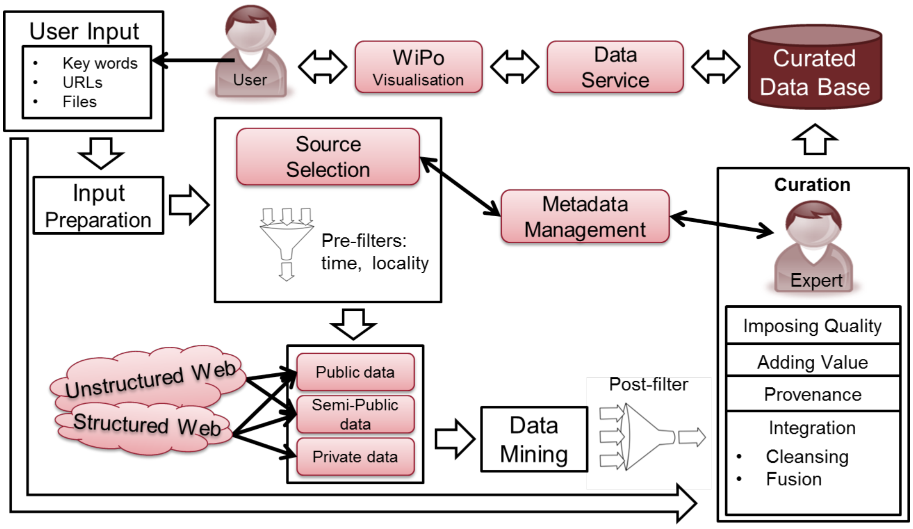


Figure 1: Original WiPo Process

such as when the user specifies the exclusion of certain data or the services purchased excludes private sources.

A major aspect of WiPo is the curation component, which is intended to assure high data quality of the output. In order to achieve this, we suppose that an expert will verify and potentially modify the results of Web crawls prior to their delivery to the user. This expert can in some cases be an algorithm, in others it may an externally provisioned service, perhaps with human involvement. The curation result will be documented in an index, repository, or a catalogue that also comprises meta-data for future updates and enhancements. However, WiPo differs from many other approaches to information gathering from the Web in that it does not try to solve everything automatically, e.g., though unsolicited search engines.

The first prototypical implementation has made some modifications as depicted in Figure 2. The main restriction is that the proof of concept does not yet consider documents supplied by users as well as private Web sources. Pre-filters and post-filters are currently employed when querying the curated database (as part of the ranking) rather than before and after the actual crawl. The tasks we initially envisioned the pre- and post-filter for are currently conducted by curators (human experts) and input preparation is limited to split the query in two parts. First, the database is queried and secondly, the user-supplied URLs are fed into the curation process.

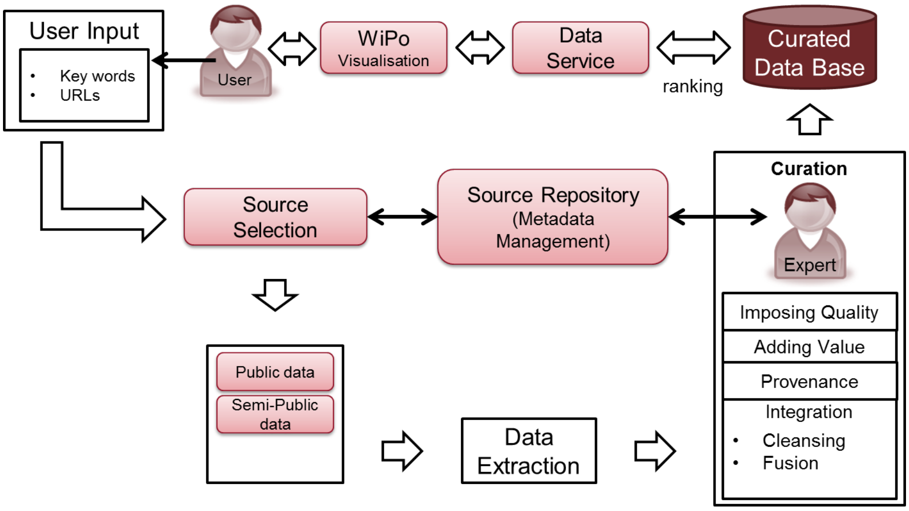


Figure 2: Implemented WiPo Architecture

Furthermore, we provide a Petri net representation of the process flow within WiPo in Figure 3 to illustrate the processing of a (searching) user's input in the WiPo system. However, we restrict this process to the search side and omit actions taken by the curator for simplicity reasons.

Generally speaking, there are three different types of user input, which must be interpreted and processed by the system. First, a user has to be able to open a new account by submitting the necessary information. The system will then create a database entry for the newly registered user.

The second option is to login to an existing account. Therefore, users provide login credentials, which will then be verified by checking them against the stored information in the database. The system keeps track of the logged in users.

The third main type of user input, possibly the most important one, is a query. At first queries are processed by pre-filters, which take place before the actual search on the stored documents. On the one hand the pre-filters generate a query, which is suitable for the used indexing service.

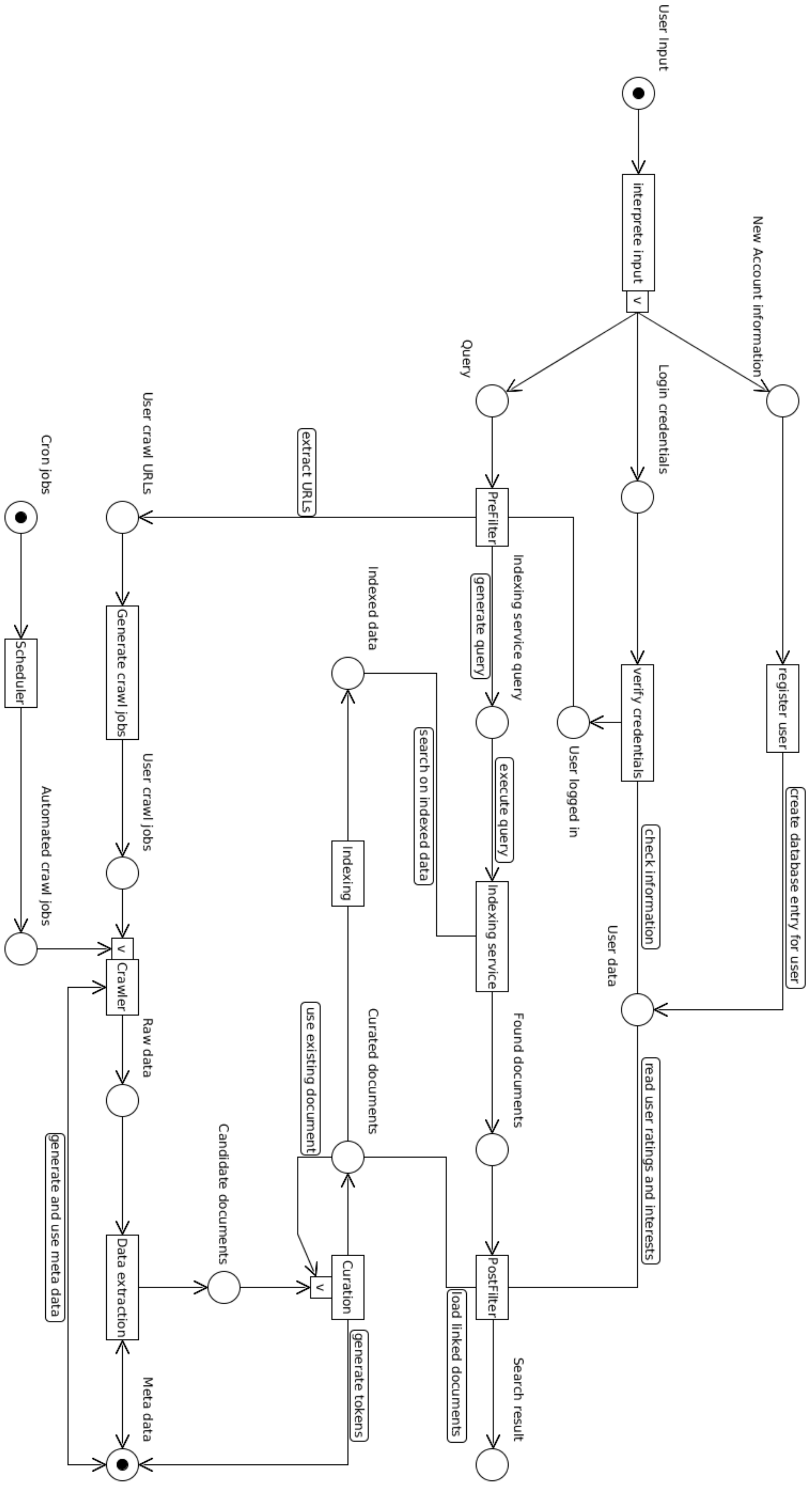


Figure 3: Petri Net Representation of WiPo

On the other hand they validate and extract URLs that were passed to the system along with the user's query. It is necessary for the pre-filters to know whether the user is logged in since some information are used later in the process.

The extracted URLs are prepared for the crawler and get wrapped in so-called *CrawlJobs*. Besides these *UserCrawlJobs* the crawler can execute automated *CrawlJobs*, which are generated by a scheduler to update previously crawled data or to gather new information for our databases. The crawler collects raw data from the Internet whereby it uses and generates various types of meta data. Afterwards, the raw data is processed and data extracted, transforming it to a candidate document that needs to be curated. Curation can either use candidate documents, previously curated documents, or a combination thereof to create new or update existing curated documents. Moreover, meta data is generated during these steps. The newly curated documents are then indexed by the indexing service, which is necessary to make the curated data available for the users of the WiPo system.

Simultaneously, the prepared query is executed by the indexing service, which searches on its indexed data. The found documents are then processed by a number of post-filters, which sort the results by users' interests and their previously made document ratings. At last the process ends with the presentation of the search result.

3 Architecture

The WiPo architecture (depicted in Figure 4) follows the client server principle, where the client (i. e. for search and access of information) is currently implemented as a browser (1) which retrieves Web pages (result) from the WiPo Server Web GUI (3), but could also be an arbitrary application (2).

The WiPo-Server has a unique system architecture with a rather complex infrastructure which has been designed to provide the functionality outlined in Section 2. It is realised based on Linux and has been implemented and tested on Fedora 19. However, every other Linux distribution can also be used with some minor adjustments.

The server infrastructure consists of a number of modules which can be accessed through the Web interface (4) which is assisted by user management (5). The core component of WiPo is the curated database (11) around which all other functionality has been developed. In the following section, we will elaborate on these modules in more detail. Namely these are: data collection — including scheduler, crawler, data extraction, and the meta database — (9); the candidate database (10); curation (7); as well as search (6) and indexing functionality (8);

3.1 User Interface

The graphical user interface (GUI) is the entry point for both searchers and curators. In both cases it acts as a means to dynamically interact with the rather complex underlying infrastructure of WiPo. Basically, any application capable of communicating with a RESTful Web-Service can be used as a front-end to WiPo (Figure 4:2). Exemplary, this was realised as a Web GUI (Figure 4:1,3). On the browser side the standard tool set of HTML and javascript was used simplified through the use of the following libraries and frameworks: jquery[3], jqueryui[4], and bootstrap[5].

[3] http://www.jquery.com

[4] http://www.jqeryui.com

[5] http://getbootstrap.com/

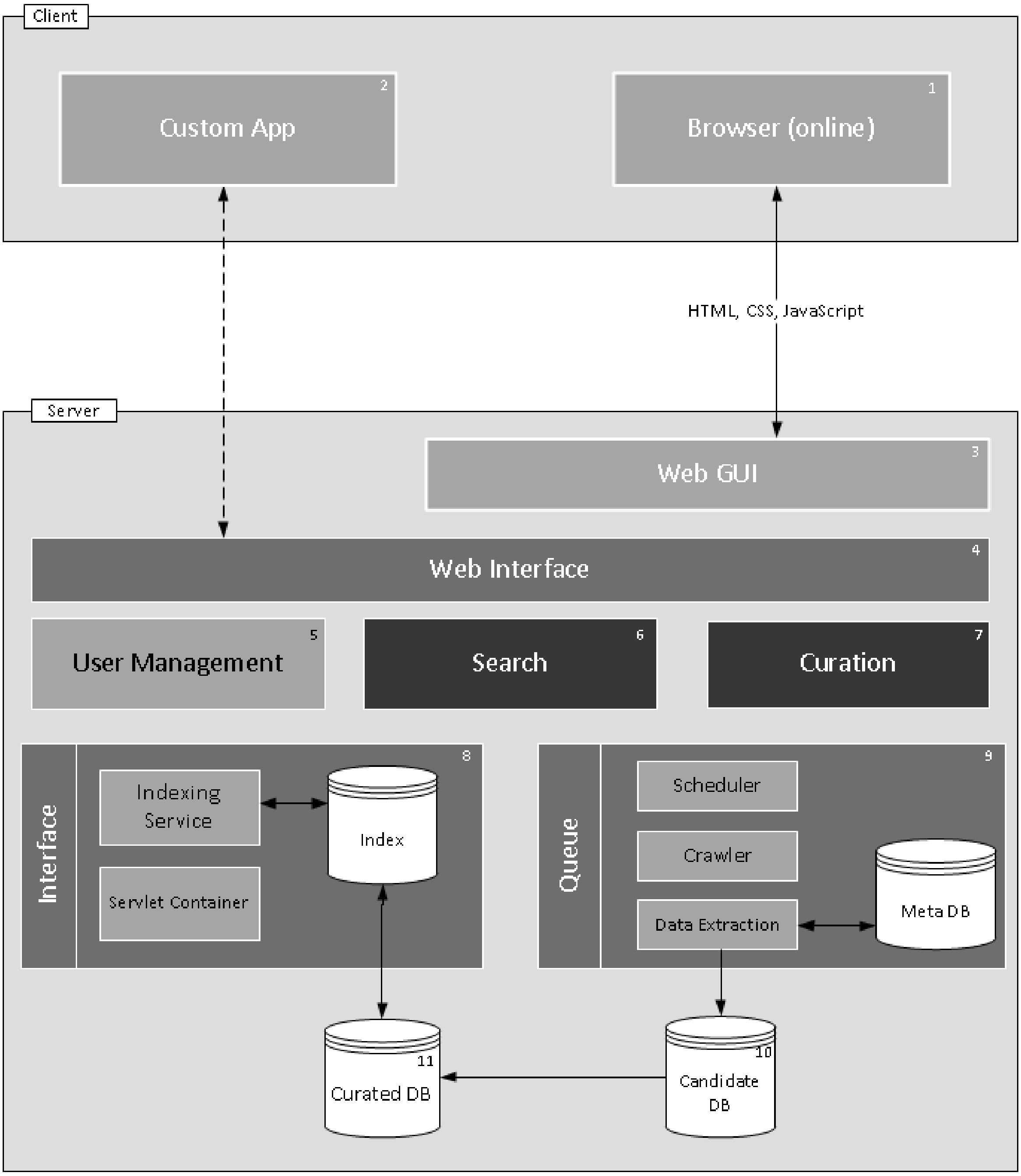


Figure 4: WiPo Module Overview (numbers are explained in the text)

This was in particular helpful for the complex task of curation. To allow for the GUI and the actual core system of WiPo to reside on different machines — as heavy load on the back-end can be expected for full crawls — a PHP[6]-Proxy was implemented using the kohana-framework[7] and the PHP implementation of `libcurl` to ensure compliance with the same origin policy enforced by javascript. The proxy provides two main functionalities, first, signing in users and secondly, forwarding *GET*, *POST*, and *DELETE* request to the interface. An example of the curator GUI is presented in Figure 5 in Section 3.5.

3.2 Web Interface

The Web interface (Figure 4:4) can be seen as the glue between the client side (GUI and proxy respectively) and the WiPo core system. It is this component of the system that allows arbitrary software clients — as long as they can make use of the common http protocol — to use it and enables for instance app development. Its main task is to delegate requests from clients to the according internal WiPo module. Thus, it is one of the most important parts of the WiPo architecture.

With regard to technology, the Web interface has been implemented as Java Servlets using Tomcat[8] as servlet container and the Jersey Framework[9] as a framework for implementing RESTful Web services in Java. Due to this technology choice, actions are triggered by URL calls which are intuitive and easy to read for humans and at the same time easy to parse for computers.

As of now, authentication for the Web interface is done via BasicAuth and all data is sent in plain text (in both directions). This was a good solution for the prototypic implementation described herein but is also something we want to address in the near future.

The Web interface provides the following resources:

- /register - Creation of new users
- /user - Functions for managing users (retrieving and updating account data, deleting users, etc.)
- /administration - Functions for administrating curators (register, retrieve curators, update, and delete a curator)
- /recover - Function to reset lost passwords
- /evernote - Functions to connect to the Evernote notebook program (as sample storage for curated outputs to be taken off-line)
- /curation - Functions to curate content

While most of the items listed are rather self explanatory or run mainly in the background, this is not the case for search and curation. Hence, they will be discussed in more detail.

3.2.1 Search Resource

The search resource enables users to search for documents. The resource expects a GET request with the parameters *search query*, *user defined list of URLs*, and optionally *start*, which is used for pagination. The resource will return a search result which has the form depicted in Listing 1.

[6] http://php.net

[7] http://kohanaframework.org/

[8] http://tomcat.apache.org/tomcat-7.0-doc/index.html

[9] https://jersey.java.net/

Listing 1: Response for a Search

```
{
    "numberOfElements"       : ...,
    "totalNumberOfDocuments": ...,
    "objects"                : [ <Compound Document>, ...]
}
```

numberOfElements represents the number of returned documents, *totalNumberOfDocuments* contains the number of all documents found, and *objects* is an array of so-called compound documents, where a compound document is a list of curated documents that have been linked together by a curator, as shown in Listing 2. An extensive example of a curated document can be seen in Listing 5 in Section 3.5.

Listing 2: Format of Compound Document

```
{
    "title"             : "...",
    "numberOfElements": ...,
    "description"       : "...",
    "documents"         : [ <Curated Document>, ...]
}
```

Beside this functionality, the interface offers methods for rating an article, retrieving articles by id, and retrieving user ratings. Furthermore, search results can be stored to Evernote (as means of making them available off-line), by sending a POST request, which has to include a search result, meaning a compound document precisely. If the user has already registered an Evernote token, then the document will be stored, else the user will be redirected to the Evernote resource's authorisation process and the acquired token will be stored in the user database.

3.2.2 Curation Resource

The main reason for the curation module is to curate documents. This can be done by sending a POST request with a specific form of document as an entity.

With regard to curation it offers special function such as deleting candidate documents or curated documents, black and white listing of URLs (stop URLs from being crawled if they are of bad quality), and retrieval of all blacklisted URLs. Also, returning all open curation task (see Listing 3) is offered by this resource.

Further, the curation resource offers some functions partially similar to search. This includes retrieving an article by id and also search functionality. However, there is no possibility to specify a URL and the result is slightly different, because *objects* contains curated documents rather than compound documents, as presented in Listing 4. This makes sense as curators work on document basis. If they interlink documents these will be retrieved as compound documents after the user's search.

Listing 3: Open Jobs List

```
{
    "openRequests" : [
                        {
                            "content"     : "...",
                            "title"       : "...",
                            "keywords"    : [ "...", ... ],
                            "source"      : "...",
                            "_id"         : "...",
                            "imagelinks"  : [ "...", ... ],
                            "videolinks"  : [ "...", ... ],
                            "description" : "..."
                        },
                        ...
                     ]
}
```

Listing 4: Response for a Curator's Search

```
{
    "numberOfElements"       : ...,
    "totalNumberOfDocuments" : ...,
    "objects"                : [ <Curated Document>, ...]
}
```

3.3 User Management

The User Management (Figure 4:5) consists of three main components: user management, user database, and session managment. User management makes it possible to add, remove and edit users who can authenticate themselves during requests to the Webinterface. Further, it interacts with the underlying user database implemented in PostgreSQL[10] and provides functionality to convert user objects into JSON objects so that the user's data can be easily accessed on the client side. The authentication mechanism (session management) is taken care of by Tomcat[11], specifically over HTTP.

3.4 Data Collection

To obtain data from the Internet (Figure 4:9), we use a crawler, namely Nutch[12], to download Web sites as dump files to the WiPo server. To be able to do that, we have implemented a scheduler daemon that feeds the crawler with jobs. At certain times, it creates so-called *CrawlJobs*, which hold all information the crawler needs to start working. We differentiate between three types of CrawlJobs: *UserCrawlJobs*, *ReCrawlJobs*, and *DataCrawlJobs*, which are all managed in a priority queue.

[10] http://www.postgresql.org/
[11] http://tomcat.apache.org/tomcat-7.0-doc/index.html
[12] https://Nutch.apache.org/

UserCrawlJobs are created when users add one or more URLs to their search and these URLs are unknown to our system. The scheduler continuously waits for incoming URLs and, once received, it creates a *UserCrawlJob* with these URLs as seeds and the highest priority, meaning that *UserCrawlJobs* are executed before any other *CrawlJob* to ensure that users receive the information they are looking for as fast as possible.

ReCrawlJobs are used to update outdated URLs in the WiPo crawler meta database (last crawl time longer ago than a pre-set threshold). Every outdated URL is used as a seed for the crawler and the CrawlJob is added to the scheduler's priority queue with medium priority. *ReCrawlJobs* are created on regular basis (in our experimental setting once a week) using the Quartz framework[13].

DataCrawlJobs are used to enrich the WiPo database with new information. It is the only job with crawl depth greater than 1, meaning that outgoing links from a seed URL are crawled, too, instead of only the seeds themselves, thus extending the body of knowledge. Since this crawl usually needs more time to execute, it has the lowest priority of all CrawlJobs. Similar to *ReCrawlJobs*, *DataCrawlJobs* are also created at given intervals (currently once a month).

Despite influencing the priority in terms of carwling, the crawl type also determines where a document will be displayed in the curator GUI 3.5. In this way curators can attend to user request first, handle re-crawls with lower priority, and if time allows curate new documents.

The scheduler usually sleeps most of the time but wakes up in a given interval - currently every minute - to check if there are one or several emphCrawlJobs in the priority queue. If so, it starts the Crawler, which then creates dump files, which include the data of all URLs crawled. After the successful creation of the dump files by the crawler, the meta data extraction is triggered. First, a parser is used to extract meta information from the given dump file. For example, this can be the fetch time, the last modified time and further relevant meta information. The extracted values are then stored in the metadata database, which is an instance of a postgresql database. The content data extraction follows right after the meta data extraction. Similar to the meta data extraction, the content data extraction uses a parser to create ContentData objects, where the content information is stripped from all meta information and from any mark-up.

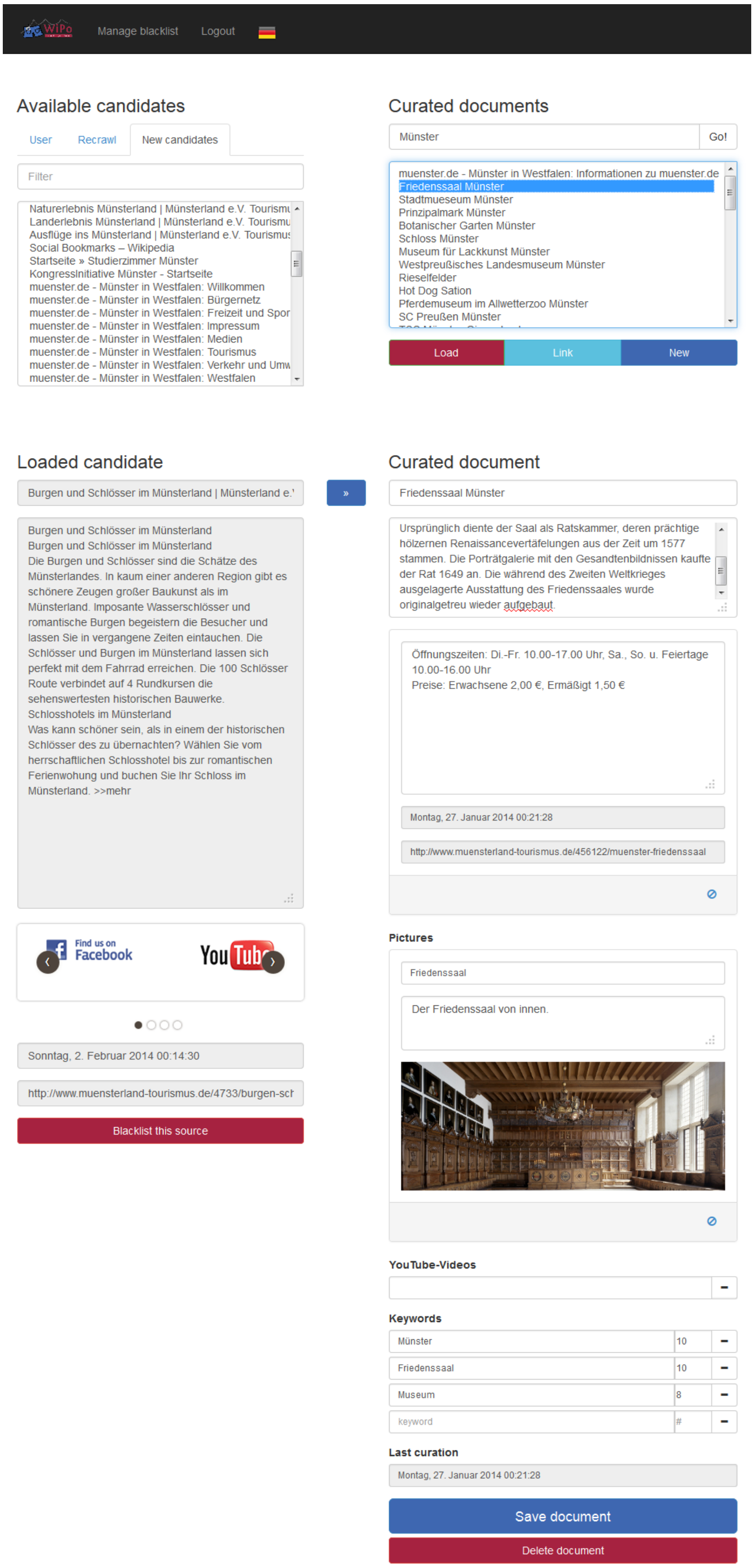
The most important part of the content data extraction is the text extraction from HTML because it's a difficult challenge to filter text from any HTML page, whether or not it meets the W3C standard. We use Boilerpipe[14] to fulfil this task. Finally, before saving the created ContentData objects in the candidate database (Figure 4:10), it is checked for validity, i. e., that all main information such as title or content is present.

3.5 Curation

The curation module (Figure 4:7) connects the curator interface (and thus the curator through the curator GUI, see Figure 5) and the curated databases (Figure 4:11) as well as the candidate database (Figure 4:10). Once data has been collected by the crawler it is stored in the candidate database. These documents are then reviewed by curators and transformed into curated documents. The curation module presents entries from the candidate database to curators who check them to ensure that they meet the necessary quality requirements. Also, curators can combine a number of candidates to a single curated document or extend an already existing curated document with contents from a candidate. Furthermore, they may recognise meaningful paragraphs and extract only these into a curated document. In addition curators can reference other curated documents that are relevant in the context and add tags that describe the document. This illustrates well the possibility of curators to combine content from several sources.

[13] `http://quartz-scheduler.org/`

[14] `http://code.google.com/p/boilerpipe/`



Available candidates

Naturerlebnis Münsterland | Münsterland e.V. Tourismu
Landerlebnis Münsterland | Münsterland e.V. Tourismu
Ausflüge ins Münsterland | Münsterland e.V. Tourismu
Social Bookmarks – Wikipedia
Startseite » Studierzimmer Münster
Kongressinitiative Münster - Startseite
muenster.de - Münster in Westfalen: Willkommen
muenster.de - Münster in Westfalen: Bürgernetz
muenster.de - Münster in Westfalen: Freizeit und Spor
muenster.de - Münster in Westfalen: Impressum
muenster.de - Münster in Westfalen: Medien
muenster.de - Münster in Westfalen: Tourismus
muenster.de - Münster in Westfalen: Verkehr und Umw
muenster.de - Münster in Westfalen: Westfalen

Curated documents

Münster

muenster.de - Münster in Westfalen: Informationen zu muenster.de
Friedenssaal Münster
Stadtmuseum Münster
Prinzipalmark Münster
Botanischer Garten Münster
Schloss Münster
Museum für Lackkunst Münster
Westpreußisches Landesmuseum Münster
Rieselfelder
Hot Dog Sation
Pferdemuseum im Allwetterzoo Münster
SC Preußen Münster

Loaded candidate

Burgen und Schlösser im Münsterland | Münsterland e.'

Burgen und Schlösser im Münsterland
Burgen und Schlösser im Münsterland
Die Burgen und Schlösser sind die Schätze des Münsterlandes. In kaum einer anderen Region gibt es schönere Zeugen großer Baukunst als im Münsterland. Imposante Wasserschlösser und romantische Burgen begeistern die Besucher und lassen Sie in vergangene Zeiten eintauchen. Die Schlösser und Burgen im Münsterland lassen sich perfekt mit dem Fahrrad erreichen. Die 100 Schlösser Route verbindet auf 4 Rundkursen die sehenswertesten historischen Bauwerke.
Schlosshotels im Münsterland
Was kann schöner sein, als in einem der historischen Schlösser des zu übernachten? Wählen Sie vom herrschaftlichen Schlosshotel bis zur romantischen Ferienwohung und buchen Sie Ihr Schloss im Münsterland. >>mehr

Sonntag, 2. Februar 2014 00:14:30

http://www.muensterland-tourismus.de/4733/burgen-sc

Curated document

Friedenssaal Münster

Ursprünglich diente der Saal als Ratskammer, deren prächtige hölzernen Renaissancevertäfelungen aus der Zeit um 1577 stammen. Die Porträtgalerie mit den Gesandtenbildnissen kaufte der Rat 1649 an. Die während des Zweiten Weltkrieges ausgelagerte Ausstattung des Friedenssaales wurde originalgetreu wieder aufgebaut.

Öffnungszeiten: Di.-Fr. 10.00-17.00 Uhr, Sa., So. u. Feiertage 10.00-16.00 Uhr
Preise: Erwachsene 2,00 €, Ermäßigt 1,50 €

Montag, 27. Januar 2014 00:21:28

http://www.muensterland-tourismus.de/456122/muenster-friedenssaal

Pictures

Friedenssaal

Der Friedenssaal von innen.

YouTube-Videos

Keywords

Münster	10
Friedenssaal	10
Museum	8
keyword	#

Last curation

Montag, 27. Januar 2014 00:21:28

Figure 5: Screenshot of the Curation Interface

Subsequently, the entries are moved into the curated database. Both the candidate as well as the curated database are implemented using mongoDB[15]. Even though mongoDB is schema free, candidates and curated documents each follow a rough structure, however, the schema-free-ness allows maximum flexibility for individual documents in particular with regard to the number and types of sections in a curated document.

During the transition from candidates to curated documents meta data (such as re-crawl intervals etc.) will be generated. Further, the curation module provides all other functionality described in the curation interface section (Section 3.2.2).

The JSON object, which is created on the client side during the curation process, has a complex structure, since it has to contain a lot of meta information about the extracted text and about the new text segments. This object is then handed to the interface and piped to the curation module, which performs the necessary computation of meta data, re-modelling, configuration of the curated document and storing both meta data and curated documents in the respective databases. For illustrative purposes the JSON schema of such a curated document is presented in Listing 5.

A particularly important part of a curated document are the segments (selections made by curators). They are identified by so-called tokens. In order to automatically identify them after re-crawling a source, the tokens must be as unique as possible and at the same time effectively processable. We chose a length of 100 characters, which makes it highly unlikely that the token does not have a unique string representation while it still allows for fairly quick processing. However, it has to be stated that we selected it intuitively.

In some (not even rare) cases, it can be possible that there aren't enough chars, in front of or behind the chosen text segment to extract a complete token of 100 chars. For example, the segment starts at offset 30, then the *begin_token* will consist of 30 chars.

The final document can then be stored in the curated database. If an already curated document forms the basis of the new one, then the document will be updated. Otherwise a new document will be created.

3.6 Update Cycle

Generally, it is important to reduce the curators' workload as much as possible, as curation is a time consuming task and human resources are expensive and limited. Whenever new content has been found by the crawler, it has to be analysed by a curator because (as of today and with the aspiration of very high quality) only humans have the ability to reliably determine the quality of an extracted text. This is the case for both new content demanded by users (through adding an url to their search) as well as for content retrieved by the crawler, the latter having a lower priority in regard to processing. However, in order to allow for automatic propagation of changes on a source, curators have the option to chose what happens, when the source of a curated document changes. The options are *keep*, i. e. the text remains in the curated database as is, even if the underlying source changes, *notify*, i. e. the curator will see the document again, once it has changed and can decide what to do then, and *auto*, i. e. the changes will be propagated unchecked. The exact way of how changes are propagate is illustrated by set of rules, depicted in Figure 6.

When re-crawling and auto-updating texts, determining the tokens is an important task. If a URL is re-crawled , all collected meta data for this source is loaded. Obviously, a new parsed text is created during the re-crawl, which is compared to the old parsed text by means of hash comparison.

[15] https://www.mongodb.org/

Listing 5: Curated Document in JSON format

```
{
    "_id": ...,
    "title": ...,
    "description": ...,
    "keywords": ["keyword1", "keyword2", ...],
    "keyvalues":[weight1, weight2, ...],
    "images":[{"src":"http://example.com/image",
               "title":"...",
               "description":"..."
              },
              ...
             ],
    "videos":[{"src":""http://example.com/video",
               "title":"...",
               "description":"..."
              },
              ...
             ],
    "category": "...",
    "sources": ["",...,""],
    "content": [{"subId":1,
                 "text":...,
                 "timestamp":...,
                 "source":...,
                 "notify":...,
                 "autoUpdate":...
                },
                {
                 "subId":...,
                 "text":...,
                 "timestamp":...,
                 "source":...,
                 "notify":...,
                 "autoUpdate":...
                },
                                 ...
               ],
    "links": ["",...,""],
    "curators":["",...,""],
    "timestamp_s":...
}
```

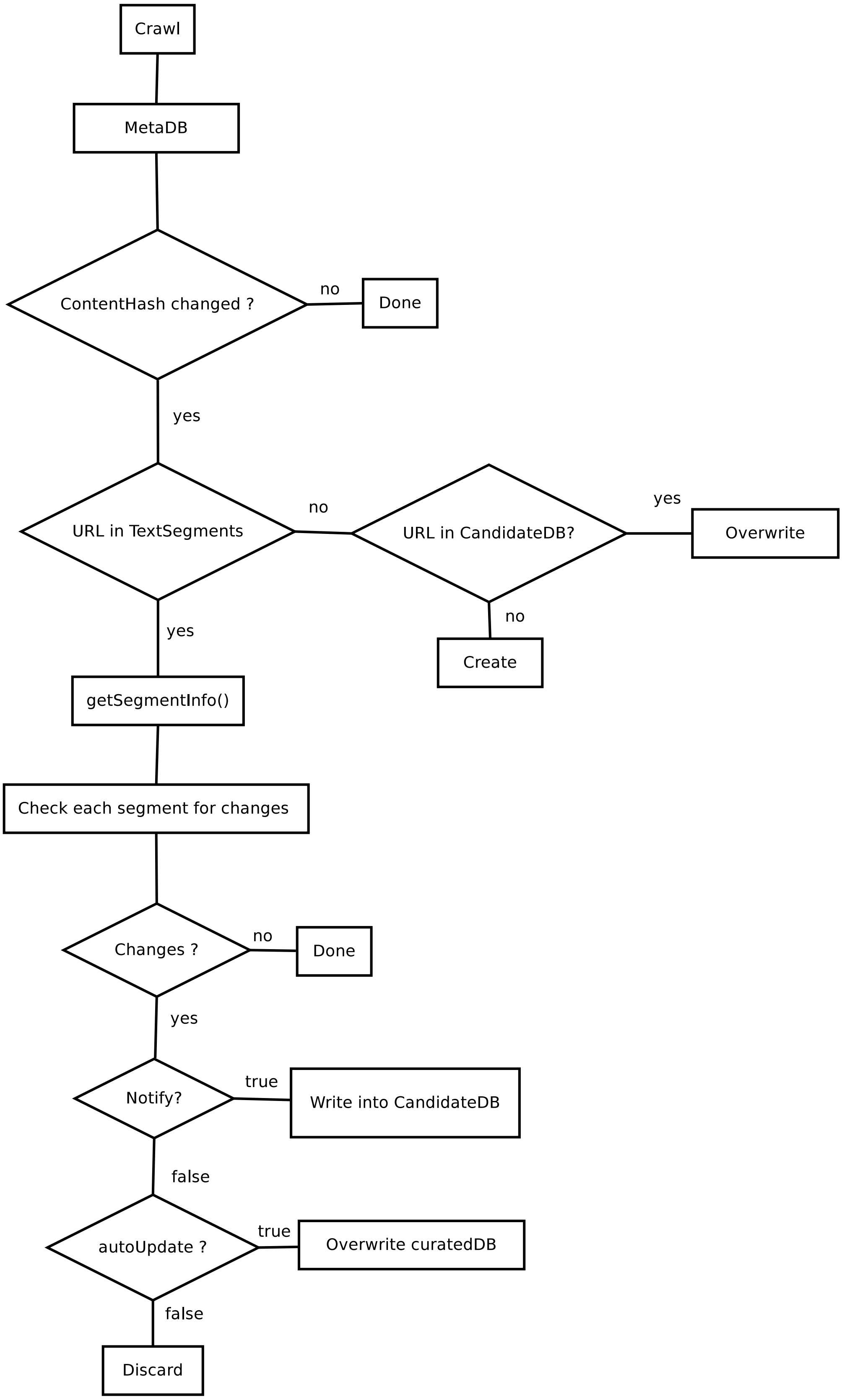


Figure 6: Update Cycle

If the hash values are identical nothing has changed and no action needs to be taken. However, if the hash values are not equal, then something in the text has changed and it has to be checked whether these changes are inside of a text segment marked by tokens.

At first, it is tried to find the tokens at the position which is given by the stored offsets (meta data). If this position has changed, the parsed text must be searched for the string representations of the tokens. If one of the tokens cannot be found, an automated update is not possible. Then the text is marked as *conflicted* and handed to the curator for review.

It is possible that a token is not unique. This means that every occurance of a begin-end token pair in the text must be identified. In the case that a begin-end token pair occurs multiple times, the text segments between the found tokens must be compared with the old text segment which is supposed to be updated. The result of the comparison can either be that a) one text segment exactly matches the old one and nothing has to be changed. This is easily computable using hash values or b) none of the found text segments exactly matches the old one. In the latter case, it has to be decided which one matches the old segment the most. We can expect that the extracted information only corresponds to the original curated data if they have a certain degree of similarity. This similarity is computed using the *Letter-Pair-Similarity-Algorithm* by Simon White[16] an implementation of the Dice Coefficient [3]. Then the most similar segment is chosen. If no segment has a similiarity of a satisfying degree the new information is marked as *conflicted* and handed to the curator for review.

If a text segment has been found and the curator has authorized automated updating, the entry in the curated database is updated. But if the segment cannot be found or the curator prohibited the automated update function, the updated source is pushed into the candidate database and has to be reviewed by the curator. Changes only to the position but not the text of a segment are propagated to the meta database.

3.7 Search Module

The search module (Figure 4:6) is another important part of the WiPo project. It grants users access to curated documents and thus represents the central element for interacting with the system. It receives the user input from the interface and processes the queries in an appropriate way, in order to ensure it fulfils the requirements of the underlying indexing service (Figure 4:8). This first step is called pre-filtering. The search index then retrieves relevant documents from the curated database. Additionally, the documents found by the indexing service are compared to a user's interests and their former ratings to assure the best possible result set for each query. This is referred to as post-filtering. We use a system of filters which relies on modularity and simplicity to implement this structure. Once the document stack is compiled it is returned to the Web interface. The internal flow of the data search module is presented in Figure 7.

3.7.1 Pre-Filters

The pre-filters, which prepare a user's query for the indexing service, follow the composite design pattern. Thus, it is guaranteed that new filters can be added in an easy way to existing ones without affecting the structure of the filtering system.

Currently, there are two kinds of pre-filters implemented. On the one hand the URLs, which were stated by the user, are send to the scheduler (Figure 4:9) , on the other hand, a parser checks the

[16] http://www.catalysoft.com/articles/StrikeAMatch.html

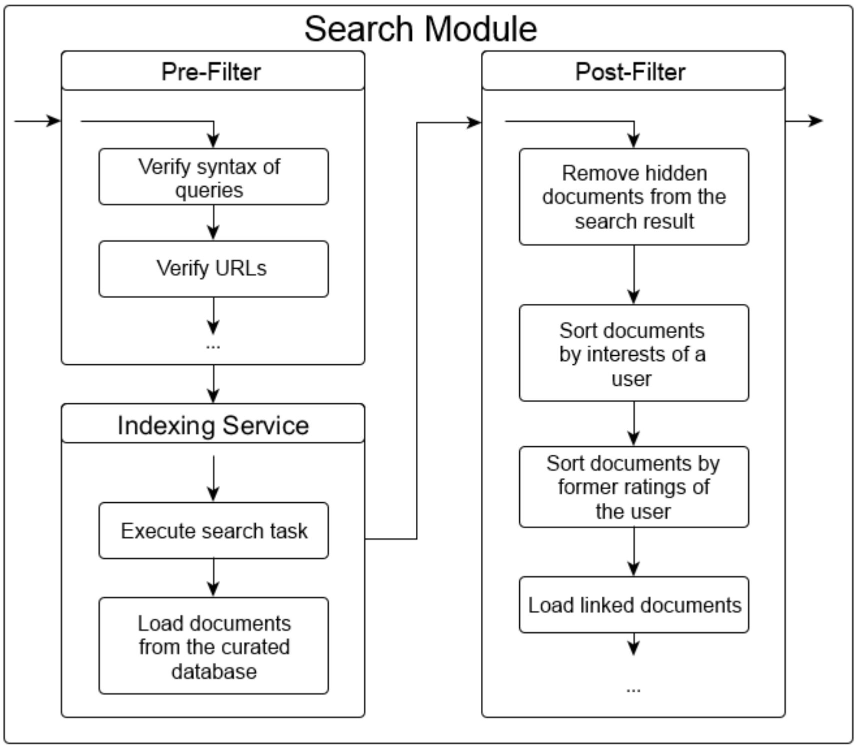

Search Module
Pre-Filter
Verify syntax of queries
Verify URLs
...
Indexing Service
Execute search task
Load documents from the curated database
Post-Filter
Remove hidden documents from the search result
Sort documents by interests of a user
Sort documents by former ratings of the user
Load linked documents
...

Figure 7: Process of a Search Request

syntax of the query string. It is possible to add new filters to the system, e.g. to include files in the search process.

To simplify the processing of user queries, the input has to be restricted by a determined syntax, the WiPo Query Language. It follows a specified LL(1) grammar (for further details on this subject see [1]). This implies, that user queries are subject to some restrictions. Currently, only words and logical operators are allowed as well as blocks thereof. However, currently nesting of blocks is not allowed.

3.7.2 Indexing Service

The task of the indexing service (Figure 4:8) is to make the entries of the curated database easily searchable. We used Solr[17] to enable a configurable powerful search. It is implemented in a way that Solr keeps track of database ids which can then be requested. Beside the Web interface, Solr is the second component that runs as servlet through Tomcat[18].

Since the indexing service needs to be fed with the entries from the curated database, we make use of the implementation of MongoConnector[19]. This daemon keeps the index of Solr in sync with the tables in mongoDB. A priori it is defined, which fields of a JSON object stored in MongoDB can be indexed by Solr. It is speciefied which types the fields are of and how the input for this fields will be treated by Solr. once started, the connector keeps syncing the whole content of the database to the index

[17] https://lucene.apache.org/Solr/

[18] http://tomcat.apache.org/

[19] http://blog.mongodb.org/post/29127828146/introducing-mongo-connector

3.7.3 Post-Filters

The group of so-called post-filters is also implemented using the composite design pattern. Post-filters are responsible for personalizing the search results to match users' interests and his former ratings of documents.

The current procedure is as follows: At first, those documents that were hidden by the user before (i.e. received a rating of 0) are removed from the answer of the indexing service. Afterwards, the user's ratings are loaded from the user database and are added to the matching documents. Next, the result is sorted by the user's interests using the cosine distance (described for instance in [9]) between the interests and the tags of the documents. The ordering by rating takes place after that, since an explicit rating is more relevant than an implicit rating through user's interests. This approach ensures that a document, which received a high rating before, is displayed before a document with a lower rating even if the user's interests might coincide more with the inferior rated document. The last step is to load the linked documents for all documents in the result set. Thus, the user gets more information than their mere query would have discovered.

Owing to the composite design pattern, it is as simple to add new features to the post-filters as it is for the pre-filters. Nevertheless, it has to be kept in mind that the order of applying post-filters is crucial.

4 Conclusions and Future Work

In this work we have presented the WiPo approach and the first implementation as a proof of concept of providing high quality information tailored towards a user's needs. So far it fulfils all requirements set a priori with regard to a simple tourism case and similar application scenarios. The prototype enables Web crawling and comprehensive curation through an easy to use Web interface as well as off-line availability of selected data. Also, it allows to integrated different type of content such as text, images and video.

During the implementation of WiPo a number of ideas sparked on how to improve or how to extend the prototype. One particular useful extension would be the inclusion of geo-data. While the data format of curated documents already supports this the user interface lacks this capability which can provide significant added value in a tourism use case. However, this is a challenge for both the curation as well as for the user front-end. Further, for the curation GUI it would be handy to have an interface that makes it easier to include non Web sources, such as external databases.

Regarding the user front-end, an own client would be most useful as it can be tailored to the exact use case and off-line functionality can be improved. At the junction of users and search index, using Solr dictionaries and other means to extend the search will allow for better search results in the near future.

Regarding crawling, a logical next step is to also include other document formats than HTML. Further, improving the data extraction algorithm can assist curators and reduce their burden. To further reduce the work load on curators, it is sensible to implement algorithms that detects the category of a document by its content. This can be seen as the originally envisioned pre and post filters. Once this is done documents could be cluster according to their topics and only documents that match a curator's skills will be displayed to them.

As the infrastructure was build in a way that it can easily be transferred to a Hadoop cluster it also would be interesting to explore how much the performance changes and whether the application is as scalable as we envion it to be.

Last, security was not a main concern for this proof of concept. Thus, we want to better protect user information by means of more advanced encryption.

Besides this, it will be a main task to tailor the rather generally applicable infrastructure to more specific use cases. Depending on what they may be we envision curation to be crowd sourced, and to integrate the search into users' social networks. Both ideas imply significant organisational and development efforts, while at the same time they can make the WiPo approach even more valuable

Acknowledgments

We gratefully acknowledge and thank the students who implemented the prototype, which are besides co-authors Adrian Godde, Bastian Hagedorn, Bastian Köpcke, and Martin Rehberger, in alphabetical order: Rene Elsbernd, Jens Gutsfeld, Tony Hauptmann, Lars Klein, Marius Umlauf, Jonathan Schott-Vaupel, Philipp Sommer, and Alexander Voß.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[2] G. S. Choudhury. Case Study in Data Curation at Johns Hopkins University. *Library Trends*, pages 211–220, 2008.

[3] Lee R. Dice. Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3):pp. 297–302, 1945.

[4] Stuart Dillon, Florian Stahl, and Gottfried Vossen. Towards The Web in Your Pocket: Curated Data as a Service. In Nguyen et al. N.T., editor, *Advanced Methods for Computational Intelligence*, volume 457 of *Studies in Computational Intelligence*, pages 25–34, Berlin, 2012. Springer-Verlag.

[5] Stuart Dillon, Florian Stahl, Gottfried Vossen, and Karyn Rastrick. A Contemporary Approach to Coping with Modern Information Overload. *Communications of the ICISA: An International Journal*, 14(1):1–24, 2013.

[6] C. A. Lee, R. Marciano, and C.-Y. Hou. From harvesting to cultivating. In *Proceedings of the 9th ACM/IEEE-CS joint Conference on Digital Libraries*, page 423, 2009.

[7] P. Lord, A. Macdonald, L. Lyon, and Giaretta David. From Data Deluge to Data Curation. In *Proceedings of the UK e-Science All Hands Meeting*.

[8] C. L. Palmer, S. Allard, and C.-Y. Marlino, M. Data curation education in research centers. In *Proceedings of the 2011 iConference*, pages 738–740, 2011.

[9] A. Rajaraman and J.D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[10] R. Sanderson, J. Harrison, and C. Llewellyn. A curated harvesting approach to establishing a multi-protocol online subject portal: Opening information horizons. In *6th ACM/IEEE-CS Joint Conference on Digital Libraries*, 2006.

[11] Florian Stahl and Gottfried Vossen. From Unreliable Web Search to Information Provisioning based on Curated Data. *EMISA Forum*, 32(2):6–20, 2012.

[12] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, George Beskales, Mitch Cherniack, Stanley B. Zdonik, Alexander Pagan, and Shan Xu. Data Curation at Scale: The Data Tamer System. In *CIDR*. www.cidrdb.org, 2013.

Working Papers, ERCIS

No. 1 Becker, J.; Backhaus, K.; Grob, H. L.; Hoeren, T.; Klein, S.; Kuchen, H.; Müller-Funk, U.; Thonemann, U. W.; Vossen, G.: European Research Center for Information Systems (ERCIS). Gründungsveranstaltung Münster, 12. Oktober 2004. Oktober 2004.

No. 2 Teubner, R. A.: The IT21 Checkup for IT Fitness: Experiences and Empirical Evidence from 4 Years of Evaluation Practice. March 2005.

No. 3 Teubner, R. A.; Mocker, M.: Strategic Information Planning – Insights from an Action Research Project in the Financial Services Industry. June 2005.

No. 4 Vossen, G.; Hagemann, S.: From Version 1.0 to Version 2.0: A Brief History Of the Web. January 2007.

No. 5 Hagemann, S.; Letz, C.; Vossen, G.: Web Service Discovery – Reality Check 2.0. July 2007.

No. 7 Ciechanowicz, P.; Poldner, M.; Kuchen, H.: The Münster Skeleton Library Muesli – A Comprehensive Overview. January 2009.

No. 8 Hagemann, S.; Vossen, G.: Web-Wide Application Customization: The Case of Mashups. April 2010.

No. 9 Majchrzak, T. A.; Jakubiec, A.; Lablans, M.; Ückert, F.: Evaluating Mobile Ambient Assisted Living Devices and Web 2.0 Technology for a Better Social Integration. January 2011.

No. 10 Majchrzak, T. A.; Kuchen, H.: Muggl: The Muenster Generator of Glass-box Test Cases. February 2011.

No. 11 Becker, J.; Beverungen, D.; Delfmann, P.; Räckers, M.: Network e-Volution. November 2011.

No. 12 Teubner R.; Pellengahr A.; Mocker M.: The IT Strategy Divide: Professional Practice and Academic Debate. February 2012.

No. 13 Niehaves B.; Köffer S.; Ortbach K.; Katschewitz S.: Towards an IT Consumerization Theory – A Theory and Practice Review. July 2012.

No. 14 Stahl, F.; Schromm, F.; Vossen, G.: Marketplaces for Data: An Initial Survey. July 2012.

No. 15 Becker, J.; Matzner, M. (Eds.): Promoting Business Process Management Excellence in Russia. March 2013.

No. 16 Teubner, R. A.; Pellengahr, A. R.: State of and Perspectives for IS Strategy Research. A Discussion Paper. April 2013.

No. 17 Teubner, A.; Klein, S.: Münster Information Management Framework. 2014

No. 18 Stahl, F.; Schomm, F.; Vossen, G.: The Data Marketplace Survey Revisited. January 2014.

No. 19 Dillon, S.; Vossen, G.: SaaS Cloud Computing in Small and Medium Enterprises: A Comparison between Germany and New Zealand. April 2014.

ERCIS

European Research Center for Information Systems

ERCIS – European Research Center for Information Systems
Westfälische Wilhelms-Universität Münster
Leonardo-Campus 3 ▪ 48149 Münster ▪ Germany
Tel: +49 (0)251 83-38100 ▪ Fax: +49 (0)251 83-38109
info@ercis.org ▪ http://www.ercis.org/

ISSN 1614-7448