Matthias Keller,
Karlsruher Institut für Technologie (KIT),
Decentralized Systems and Network Services Research Group (DSN),
Steinbuch Centre for Computing

March 24, 2015

Technical Report

A B2B Search Engine

Abstract

In this report, we describe a business-to-business search engine that allows searching for potential customers with highly-specific queries. Currently over 1,300,000 pages from more than 65,000 different company Web sites in the D-A-CH area are indexed, but the solution scales to much larger document collections. While the query processing is handled by the search server Elasticsearch, the core contribution of the prototype is a novel interface that allows non-expert users to compose very specific and complex queries in an intuitive way.

Motivation

The B2B search engine described in this report targets a B2B scenario in which a sales person from one company searches for other companies by search criteria that are more specific than just location or the membership of an industrial sector. For example, companies within a certain region that conduct numerical simulations should be retrieved to target sales activities. Currently, this task could be addressed by either using publicly available or commercial address directories or by using traditional search engines such as Google. However, both alternatives fall short:

- Information available in business directories are most likely not specific enough, e.g., the directory does not contain information whether a company's business involves numerical simulation but only the industry sector.
- Current search engines are document-based, which means they return a list of HTML documents and not a list of companies. Whether the returned search results originate from different company Web sites, all from the same company Web site or from Web sites of other organizations than companies does not influence the ranking method. Furthermore, the results cannot be restricted to certain areas.

Even through publicly available company Web sites contain a huge amount of information with great value for actual or potential business partners, current search engines do not allow to access and to exploit these information in an optimal way. They lack the ability to formulate complex queries, e.g., "return all companies, that are located in a region X and the Web sites of which contain the term 'numerical simulation' as well as the term 'aviation'".

Challenges

Beside the focus on company Web sites, the main difference to traditional search engines is the fact that full text indexing and search is combined with querying structured data. Structure is induced in two ways: First, search does not focus on individual documents but on domains, which we currently use as proxy for a company. In order to allow querying, e.g., for a domain that contains a page that fulfils criterion A and another page that fulfils criterion B, we have to implement this hierarchy, i.e. the domain-document

relationship. Second, additional structured data is extracted from each page, e.g., address information or the information whether the page is an imprint page or not. Two main challenges arise from the combination of full text search and structured data querying:

- **Scalability:** On the one hand, indexing and searching full texts, the power of an up-to-date search framework such as Lucene [1] should be available, which implements methods for stemming, stop word filtering, ranking, et cetera. On the other hand, the capabilities of traditional data bases for querying structured data, e.g., with SQL are needed as well. The approach of running a search framework as Lucene parallel to a SQL data base leads to new challenges: The results from both components need to be joined and ranked to build the final result list. However, the fact that this can involve a very large number of documents is clearly performance problem.
- **Usability:** The structure of the data needs to be considered for formulating search queries. Hence, in contrast to current search engines, search queries are much more complex. Still, it should be possible to use the search engine intuitively without deeper knowledge in the field of data based and data processing.

Solution Architecture

The solution is built around the search server Elasticsearch[2], which combines the text-analysis and indexing functionality of Lucene with the capability of indexing and querying structured documents. Elasticsearch provides a RESTful interface for indexing JSON documents that can be queried with a dedicated QueryDSL, which is JSON-based as well. The QueryDSL allows formulating complex, nested queries from a large set of atomic clauses, such as a term queries, range queries or fuzzy queries. Hence, full text search in combination with sub queries on structured data can be handled by a single component. At the same time, Elasticsearch is a distributed search server that offers easy scalability.

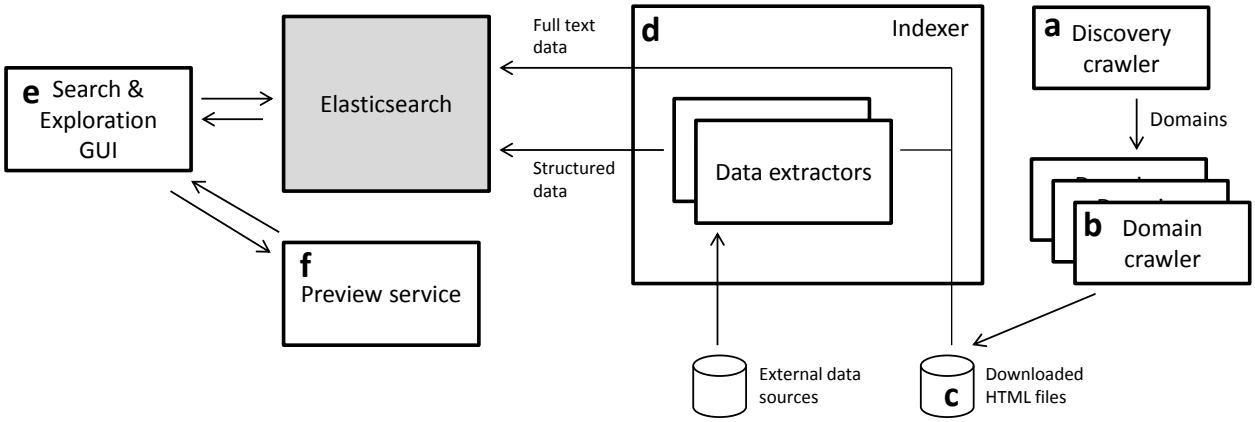

Figure 1. Architecture overview

It is clearly not feasibly to crawl the entire Web in order to identify and index business Web sites. Hence, selected business directory Web sites are crawled in search of domains to be indexed (Discovery crawler component, Figure 1(a)). Additional domains are extracted from company info boxes of the German

[1] http://lucene.apache.org/core/
[2] https://www.elastic.co/

Wikipedia. The domains are passed to Domain crawler components (Figure 1(b)) which download the homepage of each domain and all pages linked by the homepage. A full crawl of each domain (i.e., retrieving all HTML pages of the domain) is not conducted. This allows processing a larger number of domains with the available computing and storage resources. The downloaded raw HTML files are stored temporarily in a NoSQL database (Figure 1(c)). This is mainly due to the fact that the current solution is a development prototype and the crawling process is very time-consuming. Hence, storing downloaded HTML sources previous to indexing them allows avoiding crawling again every time a downstream component was changed and a re-indexing is necessary. The indexer component (Figure 1(d)) assembles the JSON documents and passes them to Elasticsearch. Two document types are generated: 1.) domain documents, one for each indexed domain containing meta data such as the number of indexed pages from this domain and 2.) page documents, one for each HTML page, containing the entire textual content, meta data and extracted structured data. Currently, two extractors for structured data are implemented: An extractor for retrieving German zip codes, which makes use of an external data set of valid zip code / city name combinations and an extractor for recognizing imprint pages. In combination, both extractors allow to predict the location of a company by the addresses found on the imprint page. Additional extractors can be easily added. All downstream components are fully adaptive with respect to the document structure, so that adding further fields of structured data does not require any changes.

While scalability could be achieved by selecting Elasticsearch as core component, making the expressive power of QueryDSL accessible to non-expert users was more challenging. Hence, developing a usable and intuitive GUI was considered as crucial (Figure 1(e)). The developed search interface is Web-based and all logic is implemented with Javascript on the client-side. Only an additional service of creating preview images of Web sites is hosted on the server-side (Figure 1(f)).

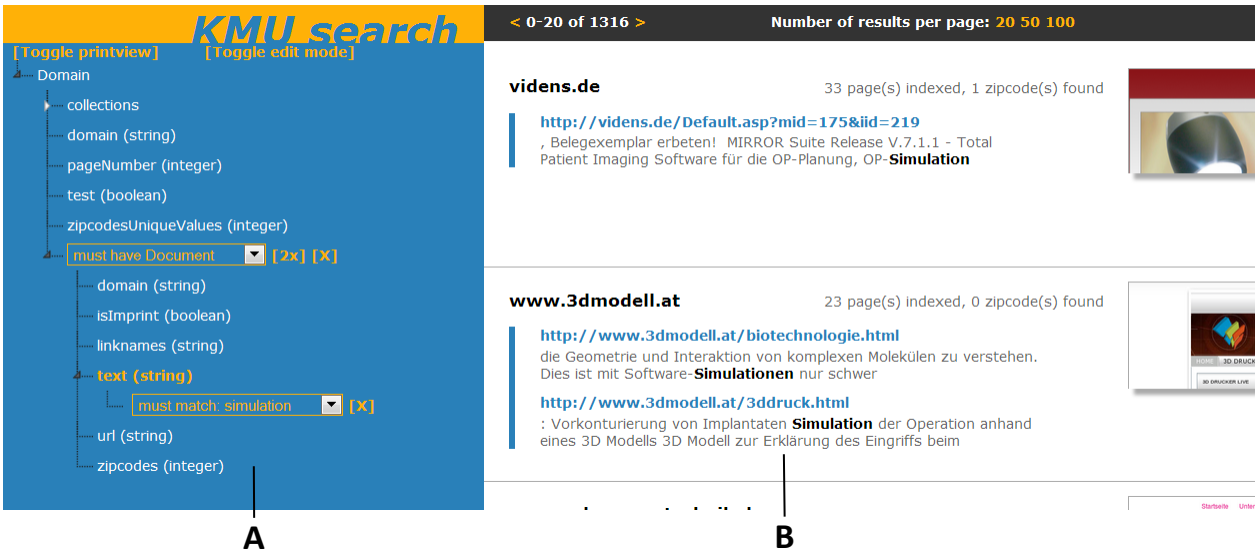

Figure 2. Search GUI screenshot

Search Assistant

Full text search queries typically have a relatively simple structure, i.e., keywords combined by AND/OR clauses. In contrast, querying and accessing structured data requires more powerful language features. The

proprietary query language of Elasticsearch, QueryDSL provides the expressiveness to formulate nested queries on structured data, while, at the same time, the full text search features of Lucene can be employed on each field. A typical search query that can be expressed in QueryDSL would be, e.g., "return all domains that have a page on which the keyword 'simulation' appears and a page that is an imprint and on which the zip code starting with the digit 7 appears". However, in QueryDSL, statements like this require long JSON expressions that clearly cannot be intuitively composed by non-experts. The purpose of the query builder GUI is to provide an abstraction for enabling non-expert users composing complex queries. This involves two aspects: 1) the structure of the data must be communicated to users in a way that they can easily learn *what* can be queried, and 2) the query builder must support users in *how* to query this structure. Both aspects could be implemented with separate GUI elements, one that shows the data structure, from which fields can be dragged to a second GUI element that represents the query. However, the query is based on the data structure and to provide an intuitive visualization the hierarchical relations of the queried fields should be represented as well. From these considerations, we concluded that the structure of the query and the document structure are closely related and decided to develop a combined GUI element that represent the document structure as well as the query structure. The query builder GUI component is shown in Figure 2 (A). If a query has not been composed yet, the query builder visualizes the document structure. By clicking on any field, a menu opens that shows the possible atomic queries (e.g., string match, fuzzy match, etc.) that fit the data type of the field. Once an atomic query is added, the query structure visually overlays the document structure in the query builder. In the example (Figure 2), an atomic query is defined on the "text"-field of the page document type. Since we want to retrieve domains instead of individual pages, this atomic query is interpreted as "domains that have child pages that have a text field that matches 'simulation'" and this semantics is visualized through annotations and highlighting in the query builder. The page subtree can be duplicated to define queries for domains that have one page that matches an expression and a second page the matches another expression. Once the query has been changed, the results on the right-hand side are updated (Figure 2 (B)). The presentation of the search results includes domain statistics such as the number of indexed pages, a screenshot of the homepage and short snippets extracted from the pages that match the query.

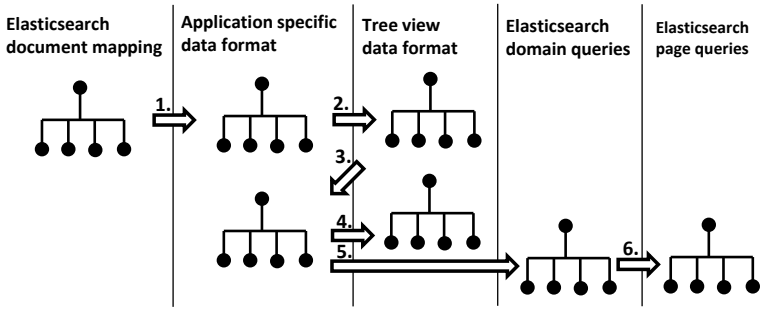

Figure 3. The query builder implementations requires several transformation of tree-structured data

The largest part of the complexity of the implementation results from the necessity to deal with multiple different tree-structured data formats (Figure 3). First the current document structures of both, the domain document and the page document type are retrieved from Elasticsearch and combined into a dedicated application specific data format (1.). On the one hand, this data format is a more compact representation of the document structure and on the other hand allows the attachment of queries to leaf nodes (which

represent fields). Based on this primary data format, the data format is generated that is required by the tree view GUI element of the query wizard (2). User interactions and manipulations with the query builder result in updates of the application specific data format (3.), which trigger an update of the tree view (4.) as well as translating the applications specific data structure in a valid QueryDSL expression (5.). First, the domain document type is queried to retrieve a list of matching domains. For querying details about the matching pages, a corresponding query has to be generated separately for each matching domain (6.)