EPJ Web of Conferences **186**, 08002 (2018) https://doi.org/10.1051/epjconf/201818608002
LISA VIII

Toward Binding Database Interfaces with Scientific Papers

Laurent Michel[1,⋆], *Sinan* Acar[2], *Gilles* Landais[1], and *André* Schaaff[1]

[1] *Observatoire astronomique de Strasbourg, Université de Strasbourg, CNRS, UMR 7550, 11 rue de l'Université, F-67000 Strasbourg, France*
[2] *UTBM, Rue de Leupe, 90400 Sevenans, France*

Abstract. Despite a large variety of facilities helping to either select or manipulate data from Web interfaces, it remains difficult to provide users with relevant scientific or technical annotations for those data. Introducing such content by hand into a Web interface is a tedious job with a risk of providing incomplete or inadequate content. To overcome this difficulty, we are exploring the possibility of using the names of exposed quantities to index a text corpus. This index can be used to show the most relevant text snippets in a given context. The full text can be displayed by user request and automatically scroll down to that snippet. Our approach is based on the conversion of PDF papers into machine readable files that are indexed by a search engine. Index entries are reported as PDF annotations that are used to control the display. This workflow has been tested on the IVOA standard corpus as a proof of concept. It has then been applied to the XMM-Newton user guides for our catalog interface. Finally, it has been adapted to find resources within portals exposing a lot of various data collections.

1 Introduction

The idea for this project comes from our experience with the XMM-Newton [1] database interface which we are operating in Strasbourg. XMM-Newton is an X-ray space observatory orbiting since 2000 and working in pointed observation mode. Observations are requested by guest observers who get proprietary access to the data over one year. After this period, the data are made public. Every year or so, all data are compiled in a catalog [2] merging all source detections and linking them with other scientific products. The whole catalog is accessible through different interfaces, among those is the XCatDB [3] deployed in Strasbourg by the XMM-Newton Science Survey Consortium. In addition to source parameters, XCatDB users can take advantage of accessing technical data such as filter responses, energy band definitions and so on and so forth. This kind of information is spread out over various documents such as the XMM user's documentation or Web pages. Although these documents are publicly available, they would be more valuable if they were automatically connected with a database user interface.

2 Providing Users with a Rich Contextual Help

Scientific data products provided with the XMM-Newton catalog do not contain any significant information about the instruments or the data reduction process. This information is available in separate

⋆ e-mail: laurent.michel@astro.unistra.fr ORCID: 0000-0001-5702-0019

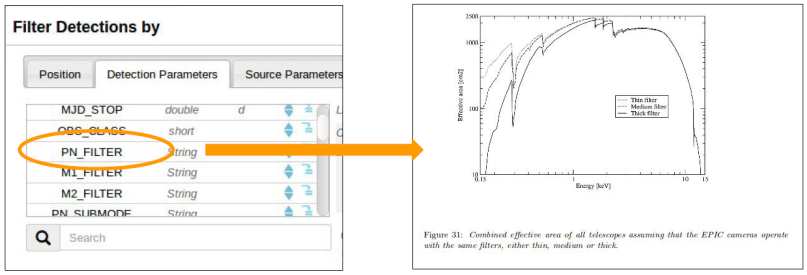

Figure 1. Implementation concept: the left scrolling list is a partial screen shot of the actual XCatDB interface; it contains all catalog columns available for filtering data. The search engine is invoked after some user action on a list item, to look for information related to that keyword within the documentation.

technical documents, Web pages or scientific papers. Our purpose is to make this ancillary data available on demand from the XCatDB interface. For instance, a user selecting sources detected with a given filter might be interested in knowing more about the response curve of that filter (fig. 1).

The most basic way to proceed would be to put links to those documents somewhere. However this wouldn't help the user get information focused on what he/she is doing. A more advanced possibility would be to extract by-hand information of interest and to feed tooltips [1]. This is an interesting solution, but with one major drawback: compiling by-hand such a data corpus is a big and expert job which is to big for our team. The solution we are exploring here is based on the use of a search engine, à la Google, running on a limited text corpus. The user could trigger search queries from different locations on the database user interface. Queries would be automatically setup according to the current users action. What would be returned is a selection of relevant text fragments displayed in their original context, while keeping access to the whole document. Thus, the user could start browsing the text corpus from a section related to what he/she is doing. It is important to note that unlike Web search engines we are not looking for documents but for fragments of them.

In other terms, we are considering merging in a single interface both data and documentation interfaces.

3 Component Breakdown

This project, achieved in the frame of a student internship, is a pathfinder project. It is split into independent modules, listed below, and intended to be put together after validation:

- Gathering a corpus of relevant texts
- Building an indexer enable to process the whole corpus
- Building a rendering tool providing entry points in the corpus from the text fragments found by the search engine
- Connecting these modules to the database interface.

This last step has not been achieved during this work.

[1] A small "hover box" with information about the item being hovered over (Wikipedia)

4 Text Corpus Creation

This is a critical point. All interesting entities must be described in various aspects (definition, calibration description, ...), but the documents must be close enough to the main topic in order not to pollute the results with irrelevant data. On the other hand it must be rich enough to provide non-trivial information. We started with indexing different XMM-Newton user guides (PDF), the catalog Web pages (HTML) and the reference paper [2] (PDF). The indexation process being rather short, it remains easy to modify the corpus and to rebuild the database.

5 Text Corpus Indexation

5.1 The Search Engine

A text corpus limited to a few tens of documents does not require high computing performance neither for the index storage nor for the query processing. Assuming we do not need specific features, we choose Elasticsearch [4], one of the most popular NOSQL database engine. The installation is easy and the tool comes with a convenient Web interface helpful to check both data ingestion and indexation. It also offers a powerful query language. Basically, Elasticsearch ingests text fragments formatted in JSON. Queries return those messages, still on JSON, matching at best the query. Elasticsearch is schema-free; there is no constraint on the JSON keys present in the ingested messages. Indexes are created for all keys of all ingested messages. Specific types (numerical, text, enumeration, ...) can however be attached to specific keys.

5.2 The Indexation Process

The text is first split in to short fragments. The fragments are then located within the original text and tagged with their natures (title, section, caption ...). The annotated fragments are packed in JSON files ready to be ingested by Elasticsearch (Fig 2). The search results must remain independent from the format of the searched documents. That means that the structure of the fragments must be the same for all text formats. Although the indexation process is enabled to ingest the whole corpus, it implements specific fragmentation routines for all supported text formats.

5.2.1 PDF Fragmentation

PDF is a printer format that maintains the same layout (paging, text location, font, colors, ...) for any device (screen, mobile, printer ...) but ignores the text structure. In fact, extracting text elements from a PDF document is akin to a data mining task more than a parsing task. We were using Grobid [5] to achieve this. Grobid is a machine learning tool enable to convert PDF documents into TEI files [6]. Grobid algorithm relies on knowledge bases built from large document samples. This tool is at the heart of the Science Miner project [7]. We had to patch the Grobid code to extract data of interest before the TEI generation. Our Grobid version returns a list of text fragments with their natures (section, title, ...) and their exact locations (page number and coordinate within the page). These data are packed in JSON messages ready for Elasticsearch.

We have to keep in mind that the text analysis with Grobid is not 100% reliable as any other machine learning tool.

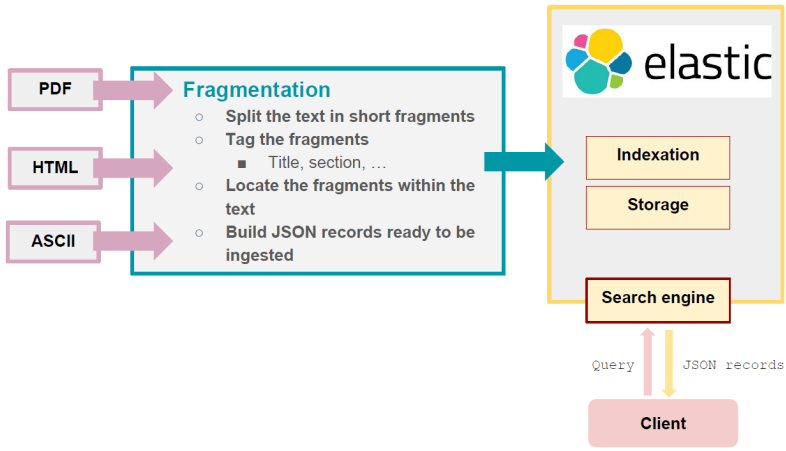

Figure 2. Indexation workflow: fragmentation tasks are specific for each text format but the indexation process is the same for all text fragments.

5.2.2 HTML Fragmentation

Unlike PDF, HTML documents are split into several independent pages spread out over different Web sites. We index one Web site at a time. A so called Web site is a set of HTML pages referenced by a master (home) page and confined in one domain. It is a sub-part of the corpus and each page is a document. Our indexer works on a local copy of the site downloaded with `wget` and analysed with `Craw4J` [8]. Using a local copy prevent issues with dynamically generated content. In addition, this allows the annotation of the content, for instance marking the fragment positions. The nature of the fragments is easy to retrieve with the HTML tags, but their locations are more difficult to get since the layout is very flexible and strongly dependent on the display device. The indexer extracts HTML fragments from the documents with an arbitrary length. These fragments are stripped from HTML tags and indexed by Elasticsearch. At the same time an empty tag with an incremental identifier is inserted into the HTML file. The fragment is then located by the text URL suffixed with `#Identifier`. As this feature is not working properly yet, it is replaced with a search facility similar to those implemented by all browsers.

6 Rendering Tool

Before connecting the search engine to the database interface, we built a Web application to validate the queries and to test different layouts for displaying searched results (Fig 3). Searched keywords are typed by the user on the top left search bar. A few buttons are available to setup both query and display. The fragments returned by Elasticsearch are listed below the search bar either ranked by relevance or grouped by documents. On the top of each fragment is an anchor running the display, on the right panel, of the whole text scrolled down to the fragment location. PDF files are displayed by the Mozilla Javascript library PDF.js [9]. The Javascript code has been patched to open the text at the requested position and to highlight all searched keywords. HTML files are displayed in a simple frame. The fragment location is achieved by a simple search based on JQuery selectors. This is a workaround while the tagging at indexation time does not allow the proper location of fragments.

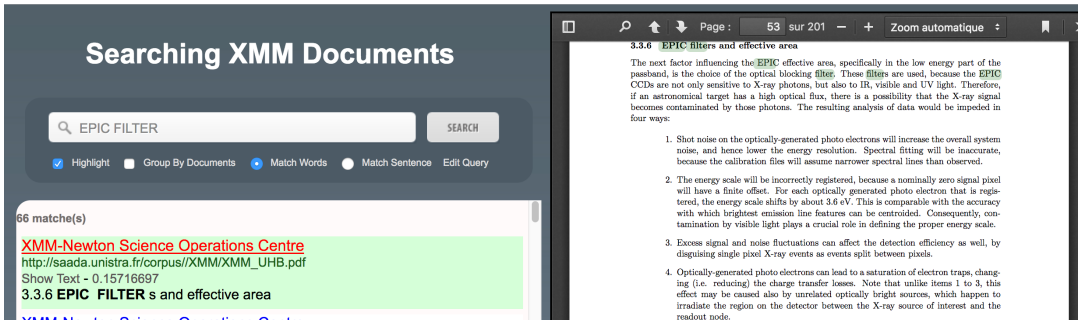

Figure 3. Rendering Tool developed to test the search engine.

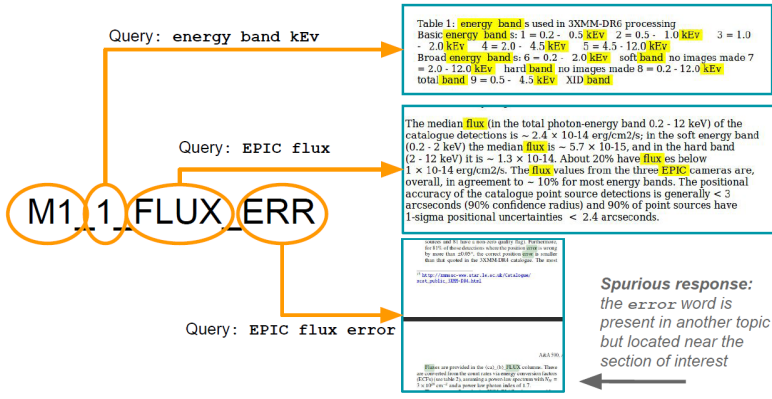

Figure 4. Compound keyword breakdown: We get relevant results by splitting compound keywords in individual components and then submitting their definitions to the search engine.

7 Database Integration

7.1 Rough Test

The initial scope of the project, connecting any meta data displayed on the screen with the search engine, has been limited to the 332 XMM catalog keywords. The reason for this restriction is that the catalog is the main resource exposed by the XCatDB. Catalog quantities are used to both filter scientific queries and to provide additional data to the searched results. We expect that users will be first interested in knowing more about these quantities. To get an idea of the system efficiency, we first submitted all column names to the search engine. The success rate is given in the table below.

Although not satisfactory, these results are not that bad. We skip the cases returning either confusing responses or just text fragments mentioning the keyword without further explanation. These cases are rare enough to be considered as noise.

Most of the successful keywords are related to one specific quantity (e.g., `DR3SRCID` is the identifier of the same source in the third release of the catalog) whereas most of the failing keywords (No responses) refer to compound quantities.

Good	None	Confusing	Just mentioned
77	240	6	9

Table 1. Search engine success score for the the catalogue keywords: Good) helpful response, None) no response at all, Confusing) irrelevant response, Just mentioned) the keyword is cited without further explanations

XMM is a complex observatory which runs, among others instruments, 3 cameras (MOS1, MOS2 and PN) working in 9 different energy bands (numbered from 1 to 9). A given quantity (e.g., a flux) can be used in 27 different keywords, or even 54 if we consider the errors attached to that quantity. The documentation describes all cameras, all energy bands and all measurements but it doesn't give a specific description for each set of camera/band/measurement. If we want to get information about a compound keyword, we have to first separate each component and to submit one of them to the search engine. The choice of that component depends on what the user is expecting. Furthermore, components cannot be directly submitted, they must be translated in more comprehensive expressions. Proceeding this way works for most of the keywords. Fig 4 shows an real example.

7.2 Baselines for an Effective Integration

These rough tests show that compound keywords cannot be used to query the search engine. The actual queries must be inferred from the individual components. The list below summarizes such a preprocessing.

1. **Detecting** whether the keyword is a compound keyword. If not, it can be used without further processing
2. **Extracting** the components. This can be helped by the syntax rules used to build the keywords
3. **Translating** each component in one or more words is comprehensive for the search engine. For instance, the component 1, which refers to the energy band 1, is absolutely not discriminating for Elasticsearch, whereas the expression `energy band` will return the precise definition of all energy bands
4. **Asking** to the user with the component he/she wants to submit (e.g., camera M1, energy band, flux or flux error)
5. **Submitting** the translation of the selected component

Fig 5 illustrates such a workflow. This processing has not been implemented yet. We just had a quick look at ontologies which are the natural framework to handle quantities described by plain text and bound with semantic links. This is, however, ahead of the current work.

8 Conclusion

Our pathfinder project validates the concept of a service that allows access to documentation from a database interface through a search engine. However, it showed the necessity of having an interface between the database metadata and the search engine. This will be the last step before any deployment on a production system. In addition to the XMM catalog, we operated it on the 13,000 Vizier [10] catalogue descriptors and on the IVOA [11] standard corpus with interesting results. This shows that our system could be transposed in other contexts than the XMM catalogue.

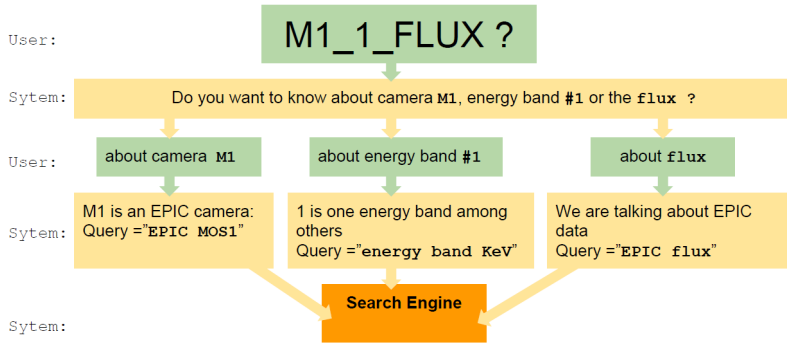

Figure 5. Getting information about compound keywords is not straightforward. The user must first say on which component they are interested. The component must be translated in comprehensive words which can be submitted to the search engine.

References

[1] ESA, *Esa science & technology: Xmm-newton*, [Online; accessed 19-July-2017], `http://sci.esa.int/xmm-newton/`

[2] S.R. Rosen, N.A. Webb, M.G. Watson, J. Ballet, D. Barret, V. Braito, F.J. Carrera, M.T. Ceballos, M. Coriat, R. Della Ceca et al., **590**, A1 (2016), `1504.07051`

[3] L. Michel, F. Grisé, C. Motch, A.N. Gomez-Moran, *The XCatDB, a Rich 3XMM Catalogue Interface*, in *Astronomical Data Analysis Software an Systems XXIV (ADASS XXIV)*, edited by A.R. Taylor, E. Rosolowsky (2015), Vol. 495 of *Astronomical Society of the Pacific Conference Series*, p. 173

[4] Elasticsearch, *Elasticsearch — Wikipedia, the free encyclopedia* (2017), [Online; accessed 19-July-2017], `https://en.wikipedia.org/wiki/Elasticsearch`

[5] Patrice Lopez and individual contributors, *kermitt2/grobid: A machine learning software for extracting information from scholarly documents*, [Online; accessed 19-July-2017], `https://github.com/kermitt2/grobid`

[6] Wikipedia, *Text encoding initiative — Wikipedia, the free encyclopedia* (2017), [Online; accessed 19-July-2017], `https://fr.wikipedia.org/wiki/Text_Encoding_Initiative`

[7] P. Lopez, *Semantic Scholar*, `https://www.semanticscholar.org/`, [Online; accessed 19-July-2017]

[8] Yasser Ganjisaffar and individual contributors, *yasserg/crawler4j: Open source web crawler for java*, [Online; accessed 19-July-2017], `https://github.com/yasserg/crawler4j`

[9] Mozilla and individual contributors, *Pdf.js*, [Online; accessed 19-July-2017], `https://mozilla.github.io/pdf.js/`

[10] CDS - Strasbourg, *http://vizier.u-strasbg.fr/*, [Online; accessed 19-July-2017], `http://vizier.u-strasbg.fr/`

[11] IVOA, *www.ivoa.net*, [Online; accessed 19-July-2017], `http://www.ivoa.net/`