QALD-Mini-Project

Lukas Blübaum, Nick Düsterhus, and Ralf Keller

University of Paderborn ,Warburger Str. 100, 33098 Paderborn, Germany
{lukasbl,nduester,rkeller}@uni-paderborn.de

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction (Ralf)

The World Wide Web is filled with information for everyone to explore. Most information on the Web is unstructured, what makes it hard to process for humans and computers. The Semantic Web is a approach to provide structured data in the Web that is easy to process by computers and can be processed to be easily understood by humans.

Most Semantic Web Databases use SPARQL as the language to query the database. That means that in order to search the semantic web the user has to learn a query language, that is not very easy to learn. This is not user friendly and can be improved.

The goal of our project is to provide a interface that takes a question formulated in natural language and answers it by querying DBPedia. The interface will be able to be used over the web via HTTP-POST requests. We aim for a F-measure of at least 0.1.

Our project uses a Template-based approach. That means we have defined templates of SPARQL-Queries that are modified at predefined locations based on the question asked. The project consists of three components: The *Question-Answering (QA) Engine*, the *Question-Processor* and the *SPARQLQueryBuilder*. We use the Library *qa.annotation* to find entities, propierties and classes, qa.commons to load and store QALD-datasets and GERBIL QA, a wrapper for web communication.

The *QA Engine* is responsible for providing the interface to users. It reads questions from the Webservice or a predefined dataset and passes the question to the Question Processor. Furthermore the QA Engine is responsible for outputting the answer, i.e sending a HTTP-Response to the user.

Relevant entities contained in the question have to be identified and the question has to be analyzed to determine the type of the question. This is done by the *Question Processor* component.

To get a answer a SPARQL-Query has to be build and executed on a endpoint. That is the responsibility of the *SPARQLQueryBuilder*. This component uses the processed information provided by the Question Processor, builds a SPARQL-Query by using predefined templates and executes the query on an endpoint provided by DBPedia.

2 Simplified Procedure

(Nick)

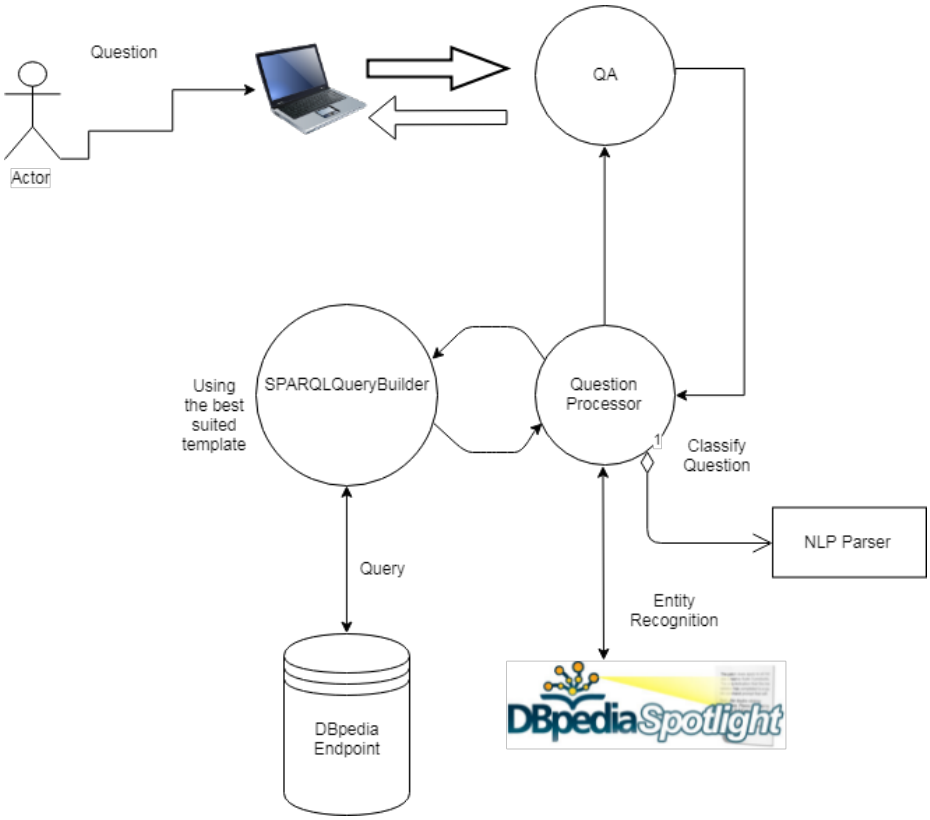

Fig. 1. Architecture.

3 QA System (Ralf)

The QA-System component is the component that reads questions from various datasources and relays them to other components, mainly to the *Question Processor*. It is also responsible for outputting the answer to asked questions.

There are three supported input-modes:

1. Ask a single question stored in a variable
2. Load datasets with multiple questions from a JSON-File

3. Run a queryable webservice

Option 1 is mainly used for debugging purposes and is not of further interest. The second option is used to run practice datasets provided by the research group. Our aim of a F-Score of at least 0.1 refers to these datasets. Option 3 is used to provide a webservice a user can use to get their questions answered.

The QA-Component consists of multiple subcomponents. One subcomponent is the *QA* class. This class is the main class of the program, the *main* Method is invoked on startup. It is responsible for checking which mode the program should run in. The mode is defined by a flag in the code that has to be set at compile time. It loads the question and releys them to the Question Processor, based on which mode is selected. If the program runs in *Dataset* mode, the QA class is also responsible for writing the results as a JSON-File to the local storage. To load the dataset, we use the *LoaderController* class provided by the research group.

The second component is the *QA-System* component. This class' only purpose is it to take a question and return a *Answer Container*. The Answer Container contains the answer of the question and the query used to get the result. The QA-System relays the question to the QuestionProcessor to get the Answer Container. The QA-System is mainly used by the webserver to determine the answer to a given question.

3.1 Webservice

- Uses Spring
- listens on /gerbil for HTTP-POST requests
- reads the question and relays it to the QA-System
- returns the result to the sender of the requests as JSON

4 Question Preprocessing

(Lukas)

5 Template Overview

(Lukas)

5.1 Example

5.2 Superlatives, Comparatives and Temporal Aggregators

(Ralf))(Lukas)

When the user enters a question like *What is the highest mountain in Germany?* the SPARQLQueryBuilder builds a query that fetches all mountains in Germany, sorts them descending by height and selects the first entry. This process is applicable for every other superlative and comparative.

The QueryBuilder has to know which superlatives and comparatives there are, which attributes they compare and if the result has to be descending or ascending. There is no online resource which provides this information, so we created an Enum called *Comparisons* that contains a selection of superlatives and comparatives in connection with the corresponding sort order (ascending for lowest, smallest, descending for highest, longest) and a reference to appropriate DBPedia attributes.

The *SPARQLQueryBuilder* iterates over all comparatives and superlatives and checks if the question contains any of them. If the Builder detects a comparative or superlative it selects the appropriate predefined Query-Template and modifies it according to the data saved in the enum. By using this approach we have a quick and uncomplicated way to handle comparatives and superlatives. The disadvantage of this approach is that we can only handle comparatives or superlatives that we defined on attributes we defined. If the user wants to compare an attribute we have not thought about this approach will not yield any result.

5.3 Example

6 Benchmarking and Evaluation

(Nick)

7 Summary

(Nick)

7.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

GERBIL Experiment

Experiment URI: http://gerbil-qa.aksw.org/gerbil/experiment?id=201806200001 and http://w3id.org/gerbil/qa/experiment?id=201806200001
Type: QA
Matching: Me - strong entity match

Annotator	Dataset	Language		Micro F1	Micro Precision	Micro Recall	Macro F1	Macro Precision	Macro Recall	Error Count	avg millis/doc	Macro F1 QALD	Timestamp	GERBIL version
test (uploaded)	QALD8 Test Multilingual	en		0,2857	0,5385	0,1944	0,2124	0,2154	0,2114	0	0,0244	0,33	2018-06-20 10:48:50	0.2.3
test (uploaded)	QALD8 Test Multilingual	en	Answer Type	1	1	1	1	1	1	0			2018-06-20 10:48:50	0.2.3
test (uploaded)	QALD8 Test Multilingual	en	C2KB	0,3949	0,4559	0,3483	0,3723	0,3854	0,376	0			2018-06-20 10:48:50	0.2.3
test (uploaded)	QALD8 Test Multilingual	en	P2KB	0,3133	0,3824	0,2653	0,2764	0,2967	0,2772	0			2018-06-20 10:48:50	0.2.3
test (uploaded)	QALD8 Test Multilingual	en	RE2KB	0,1928	0,2353	0,1633	0,1951	0,1911	0,2033	0			2018-06-20 10:48:50	0.2.3

Fig. 2. Gerbil experiment for the QALD8-Test set, with an F-measure of 0.33.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.
Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. **??**).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4], and a homepage [5]. Multiple citations are grouped [1–3], [1, 3–5].

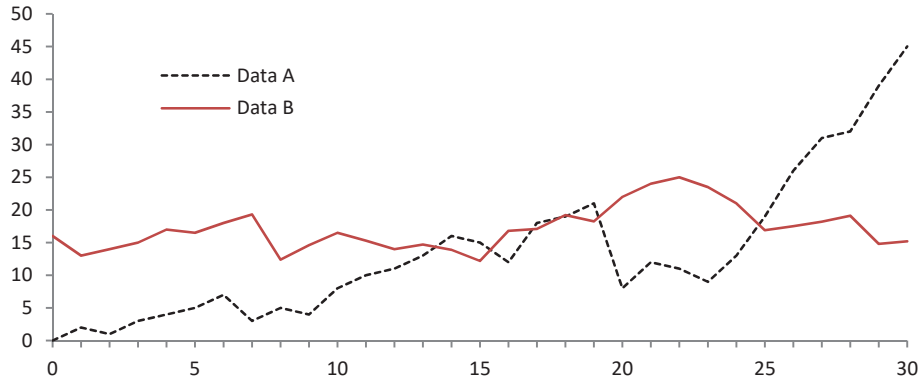

Fig. 3. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017