Qanary – the Fast Track to Creating a Question Answering System with Linked Data Technology

Kuldeep Singh[1], Andreas Both[2], Dennis Diefenbach[3], Saedeeh Shekarpour[4], Didier Cherix[6], and Christoph Lange[15]

[1] Fraunhofer IAIS, Sankt Augustin, Germany, kuldeep.singh@iais.fraunhofer.de
[2] Mercateo AG, Germany, andreas.both@mercateo.com
[3] Laboratoire Hubert Curien, Saint-Etienne, France, dennis.diefenbach@univ-st-etienne.fr
[4] Knoesis Center, USA, saeedeh@knoesis.org
[5] University of Bonn, Germany, langec@cs.uni-bonn.de
[6] FLAVIA IT-Management GmbH, Germany, didier.cherix@gmail.com

Abstract. Question answering (QA) systems focus on making sense out of data via an easy-to-use interface. However, these systems are very complex and integrate a lot of technology tightly. Previously presented QA systems are mostly singular and monolithic implementations. Hence, their reusability is limited. In contrast, we follow the research agenda of establishing an ecosystem for components of QA systems, which will enable the QA community to elevate the reusability of such components and to intensify their research activities.
In this paper, we present a reference implementation of the Qanary methodology for creating QA systems. Qanary relies on linked data vocabularies and provides a fast track to integrating QA components into a light-weight, message-driven, component-oriented architecture.

Keywords: Software Reusability, Question Answering, Semantic Search, Ontology, Annotation Model

1 Motivation

The Web of Data is every day. Researchers have developed a variety of monolithic Question Answering (QA) systems (e.g., [3, 2]) to make sense out of the enormous amount of available web data. Although the field of QA is large and many state-of-the-art QA systems exist, researchers are facing difficulties to reuse them because of their focus on implementation details and for lack of a generic approach for designing QA systems. For example, PowerAqua [3] links information available across distributed semantic resources to answer queries whereas TBSL [8] presents an approach that parse the question to produce SPARQL template that depictis the internal structure of the question. However, TBSL provides better results regarding linguistic analysis of questions, whereas PowerAqua is limited w.r.t. linguistic coverage of questions. Combining the capabilities of both systems will provide better functionalities. However, these systems are monolithic and they cannot easily be combined, which reduces their applicability to new domains and the options for synergy effects.

In other research areas, such as service-oriented architectures or cloud computing, the vision of building an ecosystem of components within a dedicated field has already proven its significance for the rapid advancement of research. Therefore, establishing a methodology – on a conceptual and implementation level – is considered crucial for managing the challenges of question answering. The Qanary approach [1] provides such a methodology. Driven by linked data technology and particularly by vocabularies, it integrates the knowledge of QA components into an overall component-based QA system. However, the conceptual layer for QA systems leaves the implementation of the QA system open.

We have implemented the Qanary using a message-driven and light-weight architecture that provides a fast track for integrating QA components, and uses standard RDF technology. We present a reference implementation of a framework for QA systems, manifesting the abstract/conceptual layer of Qanary. The framework covers the main features of component-based systems, i.e., interoperability, exchangeability and reusability, flexible granularity, as well as isolation of components (cf. [1]). Therefore, a sophisticated framework level is achieved while hiding implementation details of the integrated components and establishing the `qa` vocabulary [7] as representation of the knowledge about the user's question and the search query derived from it.

Following our long-term research agenda, this framework provides a significant step towards a best-of-breed approach for integrating the most suitable QA components for the planned domain of application. As components integrated by this framework, we initiate hereby an ecosystem for QA components and promote the reusability of existing technology. Hence, efficiency for establishing new QA systems is increased while the effort for providing reusable components is reduced.

The next section covers our approach to create a QA system following the Qanary methodology. We also briefly introduce the `qa` vocabulary. Section 3 presents a methodology for vocabulary-driven integration of QA components. Section 4 concludes.

2 Approach

2.1 Requirements for Open Question Answering systems

We have identified four key requirements, namely, *interoperability*, *exchangeability and reusability*, *flexible granularity*, and *isolation* for open QA systems [1]. The QA components are heterogeneous in their implementation, therefore, we have identified that a consistent standard interaction level, i.e., a (self-describing) abstraction of the implementation is needed to promote interoperability. This abstraction will further promote reusability to enhance efficiency of the user to build a new QA system. Hence, exchangeability and reusability are important requirements. Isolation is another identified requirement where each component should run independently of other components, i.e., it is enabled to be loosely coupled with QA systems. Flexible granularity of the components is required so they can be integrated at any step of the QA process, i.e., in contrast to other QA frameworks the granularity is not pre-defined and therefore open for future (special or general) components. To the best of our knowledge, no existing QA system or framework meets these requirements. Therefore, in our concrete implementation of the Qanary methodology, we aim at meeting these requirements.

2.2 The qa Vocabulary

In [7], we presented a vocabulary for question answering (abbreviated as qa), for representing the knowledge about a question within a QA system. Following the Qanary methodology, the qa vocabulary[7] is used to represent transitional results during the QA process, i.e., each component increases the knowledge about the given question by creating or enriching instances of the concepts qa:Question, qa:Dataset or qa:Answer. The qa vocabulary provides the main concepts needed to express the information for annotating a question with knowledge that was computed during the QA process. Each time a component is executed, properties (or information) such as provenance of annotation, score of annotation, relation between annotations, etc., are annotated to the message to make it available for subsequent components in the QA process. Hence, after every step of the QA process, the knowledge base (short: KB) is enriched with additional information about the question.

2.3 Integration by Vocabulary Alignment

We consider the fact that Qanary should not overrule existing (domain-specific) vocabularies. Therefore, it is intended to align existing vocabularies to the qa vocabulary, s.t., the computed data is available in a normalized representation and can easily be reused by other component just by knowing the concepts of qa. This can be done by using axioms or rules. The OWL subclass/sub property or class/property equivalence might be used to implement alignment axioms or rules. A reasoner or a rule engine can be used to map information from the Qanary KB to the input representation understood by a QA component (if the latter is RDF-based). A reasoner further translates the RDF output of a QA component to the extended vocabulary for uniformity, then adds it to the KB. An alternative option is to use SPARQL CONSTRUCT or INSERT queries to translate the data computed by a component to a representation that is aligned with the qa vocabulary.

3 Methodology for Vocabulary-driven Integration of Question Answering Components

To illustrate the power of the Qanary methodology, we took three independent components – DBpedia Spotlight [4], PATTY [5], and SINA [6] – arranged in the same order in the pipeline to build an exemplary QA system (cf. [1]). Here, we describe our approach of integrating them using Qanary with minimal programming effort. Our aim here was not to develop an actual QA system or to answer some specific questions by depicting a QA process, but to support and evaluate our claim that it is possible to reuse existing QA components by creating a new abstraction level for interoperability.

Qanary enriches a process-independent KB in each step. Unlike in a traditional QA pipeline, the output of the first component, DBpedia Spotlight, is not directly passed to the second component, PATTY, but is fed into a KB via the abstract level defined by the qa vocabulary and by aligning existing vocabularies to it. The second component

[7] cf., https://github.com/WDAqua/QAOntology

needs particular input, and it fetches required input directly from the KB and pushes its output back to KB. The third component does the same. Each component can access all the messages generated by the previous components stored in a triple store through SPARQL SELECT queries and can update that information using SPARQL UPDATE queries. We follow a three-step process to implement an exemplary QA system:

1. Information gathering: In general, every component has a particular need for information as input. To ensure free access to the required information, every QA component is enabled to execute SPARQL queries and can thus retrieve any knowledge about the question. As the `qa` vocabulary provides a normalized representation of the data, each component only has to know `qa` to access the data. For example, DBpedia Spotlight needs a text query as its input. It might fetch it from the question URI (`<URIQuestion> a qa:Question`), following linked data principles. Additional RDF information about the question can be retrieved by executing a SPARQL query, e.g., to fetch named entities already annotated within a textual question. To access existing components, we have implemented light-weight wrappers that send information to the particular component to wrap around and perform its action. The sample code[8] is shown below:

```
// Execute a SPARQL query to retrieve the question URI
String sparqlQuery = "PREFIX qa: <http://www.wdaqua.eu/qa#>
                      SELECT ?questionURI FROM " + namedGraph + "
                      WHERE {?questionURI a qa:Question}";
QueryExecution qExe = QueryExecutionFactory.sparqlService(endpoint,
    QueryFactory.create(sparqlQuery));
ResultSet result = qExe.execSelect();
URL uriQuestion = result.next().getResource("questionURI");

// Retrieve the question using an HTTP request
RESTClient myRestClient = new RESTClient();
String question = rstclnt.getResults(uriQuestion.toString());

// Send the question to the DBpedia Spotlight (local, port 8099)
String serviceUrl = "http://localhost:8099/" +
    URLEncoder.encode(question, "UTF-8");
String serviceResult = myRestClient.getResults(serviceUrl);
```

2. Information retrieval: Each component performs actions on extracted information and produces some results. In the next step, the wrapper retrieves the computed information from the component. Before pushing it to the KB, it is stored in a temporary location and the defined bindings to the `qa` vocabulary are applied.

3. Store results in triple store: After binding is applied on the retrieved information, the information is pushed to the KB, i.e. a triple store.

Hence, following Qanary all QA components are independent from each other and reusable. For example, if a new state-of-the-art named entity disambiguation (NED) method evolves, or new input types come into the picture, researchers just need to replace the NED (in our case study this is DBpedia Spotlight), following above mentioned three steps and the new component can easily be integrated in the QA system. Additionally, it becomes reusable for any other QA system following the Qanary methodology.

A possible extension of the described QA system might incorporate support for spoken questions. Hence, a component `C1` is required that translates an audio stream

[8] using Apache Jena: `https://jena.apache.org/`

to a textual question, which is required by DBpedia Spotlight. The `qa` vocabulary is extensible and already covers the requirements for audio streams. Now the individual vocabulary of `C1` needs to be aligned to `qa`. To integrate `C1` into the QA system, a light-weight wrapper has to be implemented that fetches the required information and passes it to `C1`. The above mentioned three-step process will be followed and `C1` can be integrated easily and efficiently into the QA system.

For details of the implementation of our exemplary QA system, please refer to our case study at `https://github.com/WDAqua/Pipeline`.

4 Conclusion

Qanary establishes a methodology independent from the process actually implemented by concrete QA systems. Hence, it is open for extension and ready for any new idea of how to solve QA tasks. Additionally our approach is built on top of formal logic to support reasoning and querying in a well-defined way and is independent from the actual implementation (the case study has to be considered as just one possible implementation). When a new requirement evolves, or a new component needs to be included in the pipeline, this can be accomplished via a "fast track" with minimal programming effort. Following the Qanary methodology, we meet all the requirements for a vital ecosystem of QA system components that are actually reusable. Hence, Qanary constitutes the first logical step towards actual open QA systems.

Acknowledgements Parts of this work received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 642795, project: Answering Questions using Web Data (WDAqua).

Bibliography

[1] Andreas Both, Dennis Diefenbach, Kuldeep Singh, Saedeeh Shekarpour, Didier Cherix, and Christoph Lange. Qanary – a methodology for vocabulary-driven open question answering systems. In *ESWC*, 2016. *to appear*.

[2] D. Damljanovic, M. Agatonovic, and H. Cunningham. Freya: An interactive way of querying linked data using natural language. In *ESWC Workshops*, 2011.

[3] V. Lopez, M. Fernández, E. Motta, and N. Stieler. PowerAqua: Supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3):249–265, 2011.

[4] P. N. Mendes, M. Jakob, A. García-Silva, and Ch. Bizer. DBpedia Spotlight: shedding light on the web of documents. In *I-SEMANTICS*, 2011.

[5] N. Nakashole, G. Weikum, and F. M. Suchanek. PATTY: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, 2012.

[6] S. Shekarpour, E. Marx, A.-C. Ngonga Ngomo, and S. Auer. SINA: Semantic interpretation of user queries for question answering on interlinked data. *Web Semantics: Science, Services and Agents on the WWW*, 30:39–51, 2015.

[7] K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. Towards a message-driven vocabulary for promoting the interoperability of question answering systems. In *10th IEEE Int. Conf. on Semantic Computing (ICSC)*, 2016.

[8] Ch. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and Ph. Cimiano. Template-based question answering over RDF data. In *WWW*, 2012.