Introducing Feedback in Qanary: How Users can interact with QA systems

Dennis Diefenbach, Niousha Hormozi, Shanzay Amjad, Andreas Both

Introducing Feedback in Qanary: How Users can interact with QA systems

Dennis Diefenbach[1], Niousha Hormozi[2], Shanzay Amjad[3], Andreas Both[4]

[1] Lab. Hubert Curien, Saint Etienne, France, dennis.diefenbach@univ-st-etienne.fr
[2] University of Athens, Athens, Greece, nhormozi@di.uoa.gr
[3] University of Ottawa, Ottawa, Canada, samja088@uottawa.ca
[4] DATEV eG, Germany, contact@andreasboth.de

Abstract. Providing a general and efficient Question Answering system over Knowledge Bases (KB) has been studied for years. Most of the works concentrated on the automatic translation of a natural language question into a formal query. However, few works address the problem on how users can interact with Question Answering systems during this translation process. We present a general mechanism that allows users to interact with Question Answering systems. It is built on top of Qanary, a framework for integrating Question Answering components. We show how the mechanism can be applied in a generalized way. In particular, we show how it can be used when the user asks ambiguous questions.

Keywords: User Interaction, Question Answering Systems, User Interface

1 Introduction

In recent years, there has been a fast growth in available data sets. One very important use case is finding relevant results for a user's requests, and by using different data sets. The field of Question Answering (QA) tackles this information retrieval use case by providing a (natural language) interface aiming at easy-to-use fact retrieval from large data sets in knowledge bases (KB). While following this path it could be observed that additional challenges are rising while the data sets are growing. For example, while having only recent geospatial information a term like "Germany" can be identified directly without ambiguity. However, after adding historic data several instances called "Germany" will be available, representing different entities (over time).

In the past years several question answering systems were published working on-top of linked data, cf., Question Answering over Linked Data (QALD) [8]. Many QA systems follow a single interaction approach, where the user asks a question and retrieves an answer. However, many questions cannot be understood because the context is only clear in the users' mind, or because the ambiguity of the considered data set is not known by the user. Hence, not any kind of question can be interpreted correctly following an ad hoc single-interaction approach. User interaction in QA systems (i.e., how users can influence a QA process) is not well explored. Only a few QA systems exists involving the user in the retrieval process, e.g., Querix [7], Freya [2], and Canalis [10].

In this paper we extend the Qanary methodology [1] a framework for integrating Question Answering components. Our main contribution is a generalized user feedback mechanism in Qanary.

2 Related Work

In the context of QA, a large number of systems have been developed in the last years. For example, more than twenty QA systems were evaluated against the QALD benchmark (cf., http://qald.sebastianwalter.org). However, only few of them address user interaction. Freya [2] uses syntactic parsing in combination with the KB-based look-up in order to interpret the question, and involves the user if necessary. The user's choices are used for training the system in order to improve its performance. Querix [7] is a domain-independent natural language interface (NLI) that uses clarification dialogs to query KBs. Querix is not "intelligent" by interpreting and understanding the input queries; it only employs a reduced set of NLP tools and consults the user when hitting its limitations. Canali [10] shows completions for a user's query in a drop-down menu appearing under the input window. The user has the option of clicking on any such completion, whereby its text is added to the input window.

Since QA systems often reuse existing techniques, the idea to develop QA systems in a modular way arise. Besides Qanary, three frameworks tried to achieve this goal: QALL-ME [6], openQA [9] and the Open Knowledge Base and Question-Answering (OKBQA) challenge (cf., http://www.okbqa.org/). To the best of our knowledge these frameworks do not address the problem of integrating user interaction.

3 A generalized Feedback Mechanism on-top of Qanary

3.1 The Qanary methodology

QA systems are generally made up by several components that are executed in a pipeline. Qanary offers a framework to easily integrate and reuse such components. Here, we briefly describe the Qanary methodology. Qanary components interact with each other by exchanging annotations (following the W3C WADM[5]) expressed in the *qa* vocabulary [11] and stored in a central triplestore T. For example, an annotation expressing that the sub-string between character 19 and 32 of the question "Who is the wife of Barack Obama?" refers to `dbr:Barack_Obama` is expressed as:

```
PREFIX qa: <https://w3id.org/wdaqua/qanary#>
PREFIX oa: <http://www.w3.org/ns/oa#>
<anno1>  a  qa:AnnotationOfInstance;
  oa:hasTarget [ a              oa:SpecificResource;
                 oa:hasSource   <URIQuestion>;
                 oa:hasSelector [ a oa:TextPositionSelector;
                          oa:start "9"^^xsd:nonNegativeInteger;
                          oa:end   "21"^^xsd:nonNegativeInteger ]];
  oa:hasBody     dbr:Barack_Obama .
  oa:annotatedBy <http://wdaqua.eu/component1> .
  oa:annotatedAt "2017-02-22T21:40:51+01:00" .
```

[5] Web Annotation Data Model: https://www.w3.org/TR/annotation-model/

Note that each annotation also contains information about the components that created them (`oa:annotatedBy`) and the time when this happened (`oa:annotatedAt`). We consider a QA system with several components C_1,...,C_n. Running these components over a new question q would lead to the following workflow: (1) Qanary is called through an API saying that it should process question q using the components C_1, ...,C_n (this API can be found under: `http://www.wdaqua.eu/qanary/startquestionansweringwithtextquestion`). (2) Qanary generates in a triplestore T a new named graph G where q is stored (using the *qa* vocabulary). (3) The components are subsequently called, i.e., the address of the triplestore T and the named graph G is passed to C_i. Component C_i retrieves annotations from G and uses them to generate new knowledge about the question. This is written back to G in the form of new annotations. (4) When all the components have been called, the address of T and G are returned. This way full access to all knowledge generated during the process is possible.

The vocabulary that is used for the annotations is described in [11]. The Qanary methodology and their services are described in [1,4].

3.2 Collecting User Feedback within a Qanary process

A user cannot change the internal algorithm of a component, but only affect the behavior of a component by the annotations it uses as input. Assume that the user wants to interact with the process after component C_i. The generalized workflow would be as follows: (1) Components C_1,...,C_i are called (cf., Sec. 3.1). (2) All the generated knowledge is stored in G. The user or the application accesses G and retrieves the annotation needed and creates new ones. (3) The QA system is restarted from component C_{i+1}, i.e., a QA process executing C_{i+1},..., C_n is started using T and G.

To avoid conflicting annotations of the same type, we enforce that both the components and the user create annotation with a timestamp, and the components read only the annotations with the last timestamp. Note that during the QA process existing annotations are not deleted, s.t., G contains a full history of all the information generated at the different point in time by any component and by the user.

4 Use Cases

We created some user interface components that follow the approach described in Sec. 3.2. In the back-end we used WDAqua-core0 [5]. It consists of two components: C_1 a query generator that translate a question (in keywords or natural language) into SPARQL queries and C_2 a query executor. The interfaces are integrated in Trill [3], a reusable front-end for QA systems that can be used for QA pipelines integrated in Qanary. The code can be found under `https://github.com/WDAqua/Trill`. We first describe in detail one of the interface components we implemented. It can be used by users to resolve the ambiguity of a question. We then briefly describe the other interfaces.

Example: Removing Entity Ambiguity from a Qanary process One of the main problems in QA systems is to disambiguate between different meanings associated to a user's question. Given "What is the capital of Germany?", the sub-string "Germany" can refer to the actual "Federated Republic of Germany" but also to "East Germany" (former GDR) or "West Germany", which will lead to different answers.

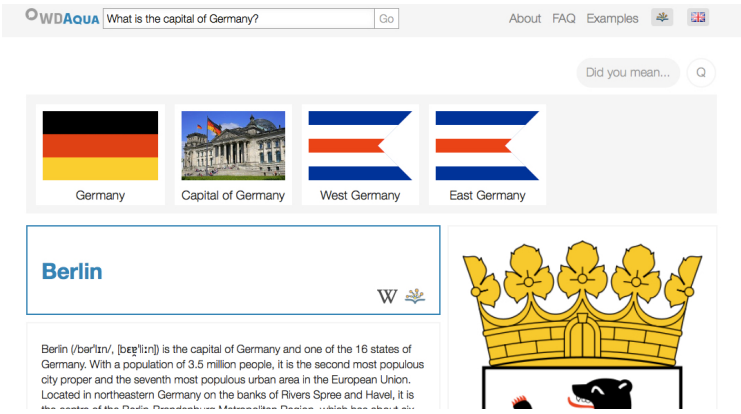

Fig. 1. Snapshot of the disambiguation interface for the question: "What is the capital of Germany?". By clicking on "Did you mean" several entities, the question might refereed to, are shown. These include the actual "Federal Republic of Germany" but also the "Capital of Germany" (as an entity), "West Germany", "East Germany", "Allied-Occupied Germany" and others. By clicking on the entity, the question is interpreted differently and a new answer is presented, e.g., if the user clicks on "West Germany", the answer "Bonn" is computed.

To allow the user to choose between different options we use the workflow presented in Sec. 3.2. In the example implementation, C_1 produces 30 SPARQL query candidates which are ranked based on relevance and stored in T using the annotations `qa:AnnotationOfSparqlQueries`. E.g., for the given question the first 3 candidates are:

1. `SELECT ?x WHERE { dbr:Germany dbo:capital ?x . }`
2. `SELECT ?x { VALUES ?x { dbr:Capital_of_Germany } }`
3. `SELECT ?x WHERE { dbr:West_Germany dbp:capital ?x . }`

For the disambiguation interface we extract 30 queries from the endpoint, extract the resources from them (in the example `dbr:Germany`, `dbr:Capital_of_Germany`, `dbr:West_Germany`) and show them to the user. By clicking on one of the resources, the corresponding query is ranked first and the re-ranked queries are written to the triple-store using again the annotations `qa:AnnotationOfSparqlQueries`. C_2 is called which executes the first-ranked query. Finally the answer is shown. The example is implemented as a user interface component in Trill and shown in Fig. 1.

Additional Examples Similarly to the above interface we have also created a user interface component that allows a user to directly choose between SPARQL queries. This interface component can for example be used, by expert users, to construct a training data set to learn how to rank SPARQL queries. We created also two interfaces components for language and knowledge base selection. They are easy-to-use drop down menus. The user selects the knowledge base or the language. Each time a new question is sent to the back-end the corresponding annotations selecting the language or the knowledge base are generated. A demo with the different interface components can be found under `www.wdaqua.eu/qa`.

5 Conclusion and Future Work

In this paper we have showed a generalized feedback for QA processes. It is an approach on-top of the Qanary framework. Qanary provides an established paradigm for collecting knowledge about a given user question in a triplestore. Our extension provides a mechanism for the user to interact with any type of information that is stored in the triplestore. Hence, computing the correct answer through the feedback of the user is thereby enabled on various steps during the QA process. This leads to the novel option not only to create a QA system from the Qanary ecosystem, but also to establish an interactive QA process on-top of it. All these arrangements are dedicated to support the QA research community in improving QA processes.

In the future, we will extend the user interface components and collect training sets for the community to be used for improving the involved back-end components, the overall QA process, and particularly the quality of QA.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642795.

References

1. A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, and C. Lange. Qanary a methodology for vocabulary-driven open question answering systems. In *ESWC*, 2016.
2. D. Damljanovic, M. Agatonovic, and H. Cunningham. Freya: An interactive way of querying linked data using natural language. In *ESWC*, 2011.
3. D. Diefenbach, S. Amjad, A. Both, K. Singh, and P. Maret. Trill: A reusable Front-End for QA systems. In *ESWC P&D*, 2017.
4. D. Diefenbach, K. Singh, A. Both, D. Cherix, C. Lange, and S. Auer. The Qanary Ecosystem: getting new insights by composing Question Answering pipelines. In *ICWE*, 2017.
5. D. Diefenbach, K. Singh, and P. Maret. WDAqua-core0: A Question Answering Component for the Research Community. In *ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7)*, 2017.
6. Ó. Ferrández, Ch. Spurk, M. Kouylekov, I. Dornescu, S. Ferrández, M. Negri, R. Izquierdo, D. Tomás, C. Orasan, G. Neumann, B. Magnini, and J.L.V. González. The QALL-ME framework: A specifiable-domain multilingual Question Answering architecture.
7. E. Kaufmann, A. Bernstein, and R. Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *ISWC*, 2006.
8. V. Lopez, C. Unger, P. Cimiano, and E. Motta. Evaluating question answering over linked data. *Web Semantics Science Services And Agents On The World Wide Web*, 2013.
9. E. Marx, R. Usbeck, A. Ngonga Ngomo, K. Höffner, J. Lehmann, and S. Auer. Towards an open question answering architecture. In *SEMANTiCS*, 2014.
10. Giuseppe M Mazzeo and Carlo Zaniolo. Question answering on RDF KBs using controlled natural language and semantic autocompletion. *Semantic Web Journal (under review)*, 2016.
11. K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. Towards a message-driven vocabulary for promoting the interoperability of question answering systems. In *ICSC*, 2016.