QAESTRO Framework – Semantic Composition of QA Pipelines

Kuldeep Singh[1,2], Ioanna Lytra[1,2], Kunwar Abhinav Aditya[2] Maria-Esther Vidal[1,2]

[1] Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), Germany
[2] Institute for Applied Computer Science, University of Bonn, Germany
kuldeep.singh@iais.fraunhofer.de
{lytra,vidal}@cs.uni-bonn.de
s6kuadit@uni-bonn.de

Abstract. Many question answering systems and related components have been developed in recent years. Since question answering involves several tasks and subtasks, common in many systems, existing components can be combined in various ways to build the tailored question answering pipelines. QAestro framework provides the tools to semantically describe question answering components and automatically generate possible pipelines given developer requirements. We demonstrate the functionality of QAestro framework for building the question answering pipelines including different tasks and components. Attendees will be able to semantically describe question answering pipelines and integrate them into existing frameworks. A video of the demonstration is available online[3].

Keywords: Question Answering, Software Reusability, SAT solver

1 Introduction

Question answering (QA) systems allow users to extract useful information from linked open data sets such as DBpedia. At an abstract level, QA systems perform tasks, for example, question analysis, query generation, and answer generation as part of their QA pipeline [3]. However, a QA system developer implements these tasks either dedicating separate QA components for it or combining few tasks together as one component [3]. Hence, components performing one QA task in a QA system can be reusable in other QA systems. The Qanary ecosystem [2] supports the reusability of such QA components. However, there is no systematic way to describe existing QA components – either standalone or parts of other QA systems – based on their functionality, i.e., the task they perform. Therefore, with the increasing number of QA components, identifying all viable combinations of components when creating new QA pipelines requires a manual and time-consuming search in the large combinatorial space of solutions.

We demonstrate QAESTRO, a framework able to deal with the QA pipeline composition problem by casting it to the query rewriting problem [1] and leveraging state-of-the-art SAT solvers [3]. QAESTRO helps QA developers to semantically describe QA components and developer requirements based on these semantic descriptions; a controlled vocabulary is utilized to model QA tasks and exploited in the description of the QA components. Attendees will be able to semantically compose QA pipelines and reuse them in frameworks like QANARY, OKBQA, or QALL-ME [3]. Moreover, they will observe how QA component descriptions can be exploited to enhance reusability.

[3] https://www.youtube.com/watch?v=9lhamebx7JM&feature=youtu.be

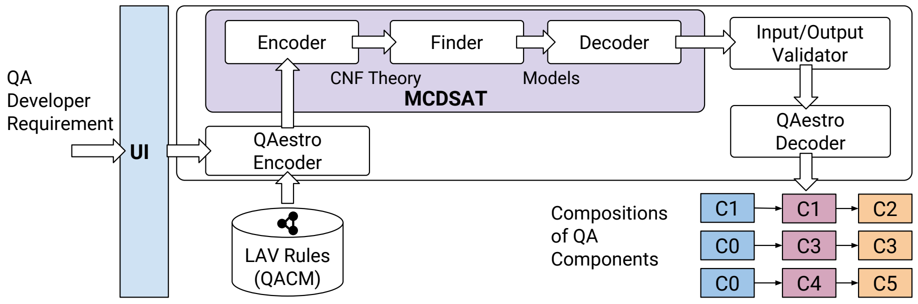


Fig. 1: The **QAESTRO Architecture.** QAESTRO receives as input a QA developer requirement and a set of rules describing QA components, and produces all the valid compositions that implement this requirement.

2 The QAESTRO Architecture

QAESTRO is built on top of MCDSAT [3] which relies on state-of-the-art SAT solvers to efficiently enumerate compositions of QA components. QAESTRO allows a developer to build a QA pipeline on demand; it utilizes a controlled vocabulary which encodes the properties of generic QA tasks and is used to semantically describe QA components. Further, it enumerates the compositions of the QA components that implement a given QA developer requirement. Fig. 2 illustrates the architecture of QAESTRO. It accepts as input a QA developer requirement which is a conjunctive query over QA tasks which a developer wants to implement. Furthermore, all the mappings of existing QA components to QA tasks act as input to QAESTRO encoder that translates it into an instance of the Query Rewriting Problem (QRP) and passes it to MCDSAT. MCDSAT encodes the instance of QRP into a CNF theory in a way that encoding of this theory correspond to solutions of QRP. A SAT solver is used in MCDSAT to model all valid query re-writings that correspond to models of the CNF theory. The output of QAESTRO is the valid compositions of QA components based on the corresponding developer requirement. For the detailed description of the semantic descriptions of QA components, developer requirements, and an empirical evaluation of QAESTRO, the reader can refer to [3].

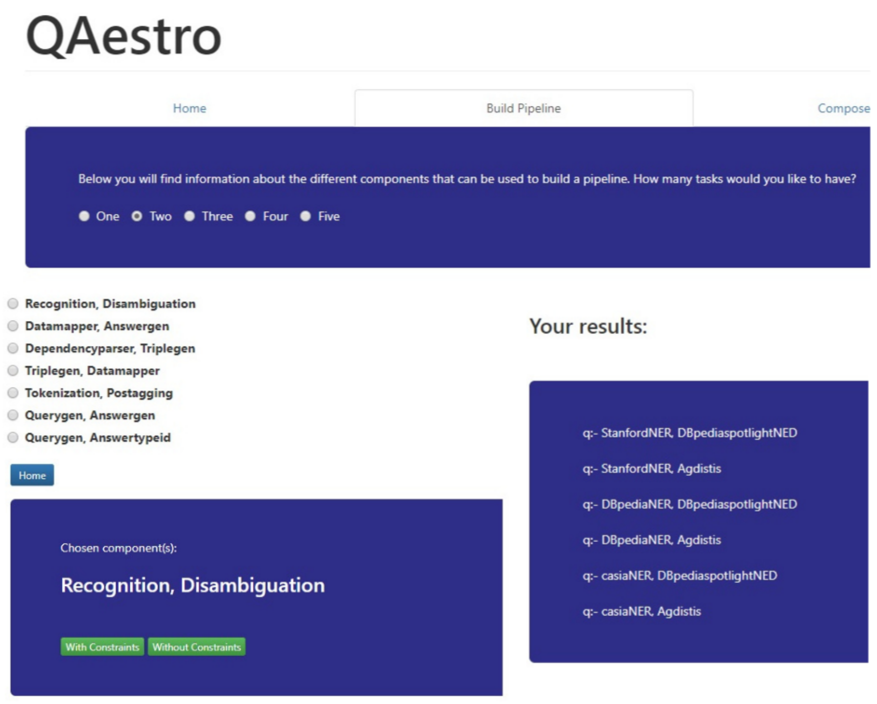
3 Demonstration of Use Case

We motivate our demonstration by considering the problem of semantic composition of a QA pipeline. Let us consider two semantically described QA components:

$AIDA(\$y, z) :- disambiguation(x, y, z, t), question(y), disEntity(z)$
$StanfordNER(\$y, x) :- recognition(y, x), question(y), entity(x)$

These rules state the following functionalities of AIDA and Stanford NER components: (i) AIDA[4] implements the QA task of disambiguation; a question is received

[4] https://gate.d5.mpi-inf.mpg.de/webaida/


QAestro

Home | Build Pipeline | Compose

Below you will find information about the different components that can be used to build a pipeline. How many tasks would you like to have?

One [✓ Two] Three Four Five

- Recognition, Disambiguation
- Datamapper, Answergen
- Dependencyparser, Triplegen
- Triplegen, Datamapper
- Tokenization, Postagging
- Querygen, Answergen
- Querygen, Answertypeid

Home

Chosen component(s):

Recognition, Disambiguation

With Constraints | Without Constraints

Your results:

q:- StanfordNER, DBpediaspotlightNED

q:- StanfordNER, Agdistis

q:- DBpediaNER, DBpediaspotlightNED

q:- DBpediaNER, Agdistis

q:- casiaNER, DBpediaspotlightNED

q:- casiaNER, Agdistis

Fig. 2: **QAESTRO Demo.** A QA developer wants to implement Named Entity Recognition (NER) and Named Entity Disambiguation (NED) tasks together with a constraint that the NED component accepts question and entity as input.

as input (marked with \$), and a disambiguated entity is produced as output; (ii) Stanford NER[5] implements the QA task of entity recognition; it takes a question as input and outputs spotted entities in the question. In these rules, $disEntity$, $recognition$, $disambiguation$ etc. are the controlled vocabulary terms in QAESTRO. Using similar rules, we have semantically described 51 QA components from 25 QA systems. Consider a QA system developer requirement to implement two QA tasks, for instance, NER and NED. Such requirements can be semantically described as:

$QADevReq :- recognition, disambiguation$

Using 51 QA components from 20 QA systems[6] and 30 different QA system developer requirements, we will demonstrate the following use cases:

– **QA developer requirement to implement one QA task.** Developers will be able to express requirements similar to the ones in the previous example and implement a pipeline of one QA task. QAESTRO allows a developer to choose which task she wants to implement and returns the list of components that are associated with the chosen task. Component semantic descriptions are presented as well.

[5] http://nlp.stanford.edu:8080/ner/

[6] http://wdaqua.eu/QAestro/qasystems/

- **QA developer requirement to implement two QA tasks.** We further demonstrate how QAESTRO can semantically compose the possible combination of components when a developer wants to implement two tasks (e.g., named entity recognition, named entity disambiguation together). This can be done with and without constraints. In case of "with constraint" option, the developer has a specific requirement, e.g. "only return the components which accept particular input in one of the tasks". For example, while implementing NER and NED tasks together, the developer wants to know the composition of QA pipeline in which the NED component accepts an entity as input from the NER task. In this case, QAESTRO will not return all the components implementing these two tasks, but only the QA components which produce an entity as output in the NER task and the QA components which accept this entity as input in the NED task. Hence, by demonstrating this use case, we illustrate the power of a SAT solver in QAESTRO to check whether the interpretation of developers' requests holds.
- **QA developer requirement to implement three, four, and five QA tasks.** In this use case, we demonstrate how QAESTRO returns valid compositions of QA components implementing three or more QA tasks based on developer requirements. For example, if a developer seeks to implement NER, NED, and query generation (i.e., the components which build a SPARQL query from disambiguated resources) together and see the possible compositions of QA components, QAESTRO will return all viable compositions. In case the developer has some constraints on a particular task (e.g., its input or output), QAESTRO respects this as well. Similarly, we demonstrate QAESTRO for four and five QA tasks.

All the use cases of QAESTRO are publicly available [7].

4 Conclusion

We demonstrate the QAESTRO framework that allows to semantically describe QA components, as well as developer requirements to compose a QA pipeline from reusable QA components. We illustrate the functionality of QAESTRO using 51 semantically described QA components and different developer requirements. Moreover, attendees will be able to specify QA pipelines and integrate them into existing QA frameworks.

Acknowledgement 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] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4), 2001.
[2] K. Singh, A. Both, D. Diefenbach, S. Shekarpour, D. Cherix, and C. Lange. Qanary - the fast track to creating a question answering system with linked data technology. In *ESWC 2016 Satellite Events*.
[3] K. Singh, I. Lytra, M.-E. Vidal, D. Punjani, H. Thakkar, C. Lange, and S. Auer. QAestro – semantic-based composition of question answering pipelines. In *DEXA 2017*.

[7] http://wdaqua.eu/QAestro/