Frankenstein: A Platform Enabling Reuse of Question Answering Components

Chapter *in* Lecture Notes in Computer Science · June 2018

DOI: 10.1007/978-3-319-93417-4_40

CITATION
1

READS
52

4 authors:

Kuldeep Singh
Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS
20 PUBLICATIONS **112** CITATIONS
SEE PROFILE

Andreas Both
DATEV eG
59 PUBLICATIONS **546** CITATIONS
SEE PROFILE

Arun Sethupat
University of Minnesota Twin Cities
6 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Saeedeh Shekarpour
University of Bonn
45 PUBLICATIONS **338** CITATIONS
SEE PROFILE

Frankenstein: A Platform Enabling Reuse of Question Answering Components

Kuldeep Singh[1(✉)], Andreas Both[2], Arun Sethupat[3], and Saeedeh Shekarpour[4]

[1] Fraunhofer IAIS, Sankt Augustin, Germany
kuldeep.singh@iais.fraunhofer.de
[2] DATEV eG, Nuremberg, Germany
contact@andreasboth.de
[3] University of Minnesota, Minneapolis, USA
sethu021@umn.edu
[4] University of Dayton, Dayton, USA
sshekarpour1@udayton.edu

Abstract. Recently remarkable trials of the question answering (QA) community yielded in developing core components accomplishing QA tasks. However, implementing a QA system still was costly. While aiming at providing an efficient way for the collaborative development of QA systems, the Frankenstein framework was developed that allows dynamic composition of question answering pipelines based on the input question. In this paper, we are providing a full range of reusable components as independent modules of Frankenstein populating the ecosystem leading to the option of creating many different components and QA systems. Just by using the components described here, 380 different QA systems can be created offering the QA community many new insights. Additionally, we are providing resources which support the performance analyses of QA tasks, QA components, and complete QA systems. Hence, Frankenstein is dedicated to improving the efficiency of the research process w.r.t. QA.

Keywords: Question answering · Reusability · Integration
Annotation model · Evaluation · Pipeline

Resource Type: *Software Framework*
Repository: https://github.com/WDAqua/Frankenstein
License: *GNU General Public License v3.0*

1 Introduction

Our research vision lies in *building up an infrastructure in which the state-of-the-art question answering (QA) core components can be easily integrated, run, and evaluated*. This vision was triggered by the fact that so far QA community

A. Gangemi et al. (Eds.): ESWC 2018, LNCS 10843, pp. 624–638, 2018.
https://doi.org/10.1007/978-3-319-93417-4_40

has released a considerable body of research as well as valuable running components accomplishing various QA tasks. To achieve our goal vision, we recently published foundations [3,22,25] being essential to solve the interoperability, integration and reusability issues. Initially, we published the `qa` vocabulary [22] as a flexible and extensible data model for annotating outputs of QA components. Thereafter, we proposed Qanary [3], a methodology for integrating components of QA systems which (i) utilises `qa` vocabulary for annotation, (ii) is independent of programming languages, (iii) is agnostic to domains and datasets, (iv) integrates components on different granularity levels. Lately, we published the Frankenstein [25], which is concerned with (1) a prediction mechanism to predict the performance of a component given a question and a required task; (2) an approach for composing performance-optimised pipelines[1] by integrating the most accurate components for the current QA tasks (i.e. the user's question). Frankenstein uses Qanary methodology to integrate state-of-the-art QA components within its architecture. However, we disregarded implementation details, reusability, configuration details, integration advantages of Frankenstein in [25], while this paper introduces Frankenstein as an application/platform addressing (a) how to build a new QA pipeline using 29 integrated components (b) how each component can be reused independently (c) how to evaluate the questions/texts. Hence, we *disassemble* the Frankenstein implementation to present a large set of reusable components from the QA community which can be run, evaluated and compared using the additional tools Frankenstein is offering. In other words, by decoupling Frankenstein architecture, the overall architecture becomes collection of 29 components as reusable resources, which can be either used to build QA pipeline or text analysis. We introduce major modules of Frankenstein which not only enable detecting optimum pipelines but also enable us to easily *run*, *evaluate* and *compare* any configured QA system. Frankenstein integrates 29 QA components by developing an individual wrapper for each component. Thus, end-user does not need to get involved in configuration and implementation details of components. In fact, these components can be directly reused to build QA systems. Consequently, just by using the QA components described in this paper 380 reasonable QA pipelines can be created with little effort. Hence, many new insights w.r.t. the performance of QA might be derived using these components and pipelines which also providing support for analytics as well as adopting additional components.

The contribution of this paper is to release the Frankenstein modules containing two kinds of open-source resources namely (i) reusable components as well as (ii) component-wise runners and evaluators. These resources are briefly described in the following:

R 1 **Reusable QA Components:** We collected 29 QA components accomplishing various QA tasks, i.e. named entity identification/recognition (NER), Named Entity Disambiguation (NED), Relation Linking (RL), Class Linking (CL), and Query Builder (QB). Then, we implemented a wrapper for every

[1] Please be noted that a full QA pipeline is composed of all the necessary tasks to transform a user-supplied textual question into a formal query.

included component which enables these popular tools to be easily integrated and reused in the Frankenstein framework. Therefore, these components can be used for building modular question answering systems which might analyse text, provide knowledge extraction etc. Furthermore, the wrapper annotates the output of the components using the `qa` vocabulary to provide machine readability and homogeneity among outputs of all components.

R2 **Evaluators for components and benchmarks:** We have automatised the process of running and evaluating any component integrated within Frankenstein. Thus, it enables evaluating and comparing QA components for individual stages of QA pipeline. Consequently, it is possible to analyse the performance of each QA component as well as of the whole QA systems which lead to completely new insights on the performance of particular QA tasks. Hence, researchers are enabled to easily uncover quality flaws and improve the performance while aiming at existing or novel fields of applicability. The evaluator components are independent of the input benchmark, and it is configurable in easy steps based on the requirement of the user.

This work is substantially impactful for QA and NLP communities because (1) it facilitates comparison of NED, NER, CL, and QB components w.r.t. any given gold standard; (2) it can easily integrate new and upcoming components at any stage of QA pipeline to ensure scalability. Thus, by this platform, the research community will be empowered to an automatic approach which easily reuses the core components and facilitates running and comparing the performance of components over any given benchmarks.

The rest of this paper is organised as follows. Section 2 presents the importance and impact of this work for the research community. Section 3 lists all of the components integrated so far along with their characteristics. The major modules of Frankenstein are presented in Sect. 4. Section 5 presents our plan for availability and sustainability of resources. Section 6 reviews the state-of-the-art and we close with the conclusion and future work in Sect. 7.

2 Broader Impact

Impact on QA Community. Recently, QA community was supported by the modular approaches such as openQA [14], Qanary [3,23], OKBQA [12], QALL-ME [7], and Frankenstein [25] aiming at integrating and reusing the existing QA core components. Frankenstein is a smart solution on top of Qanary to the limitations observed in the prior approaches. For example, openQA expects Java implementation of the components which is not possible in most of the cases. Also, openQA and QALL-ME have configuration difficulties and its components are not directly reusable in other approaches. More importantly these frameworks do not support a dynamic pipeline methodology. Moreover, the distinguished features united within Frankenstein makes it scalable, user-friendly and fully automatic which are rare in the prior approaches. Apart from these general characteristics, Frankenstein resources make the researchers needless of

developing a QA full pipeline. In fact, researchers can focus on improving individual stages of QA pipelines while reusing R1 for other QA tasks to complete their pipeline.

For example, recent work on query builder component [33] has reused results of components for building and evaluating QA pipeline for its empirical study. In this way, QA researchers can focus on independent stages to make it more accurate and intelligent. Furthermore, using the automated process of evaluation (i.e. R2) within Frankenstein assists researchers to easily integrate their newly developed component and evaluate its performance against the-state-of-the-art components over any given benchmark.

Impact Beyond QA Community. Although the primary contribution of our work targets the QA community, other disciplines – particularly information extraction (IE) and Natural language processing (NLP) communities – are beneficial of Frankenstein because of the common tasks such as NED, RL, and CL. For example, 11 of NER components and 9 of NER components are integrated into Frankenstein and coupled with tools (i.e. R1 and R2). These components are also utilised in information retrieval and social media analytics for entity recognition and disambiguation on large textual corpora or a tweet corpus. Any given benchmark can be uploaded to R2 therefore possibly, a domain-specific evaluation of performance is published. Enabling these communities to reuse the existing components opens new perspectives for the future steps. For example, there is no meticulous study about the performance details as for where each component is well-performed or what is its pitfalls.

3 Reusable Components and Characteristics

Named Entity Recognition and Disambiguation Components. The aim of the named entity recognition (NER) task is to recognise the entities present in the question and the aim of named entity disambiguation is to link these spotted entities to its knowledge base mentions (e.g. for DBpedia [1]). For instance, in the example question "Who is the mayor of Berlin?", an ideal component performing NER task recognises `Berlin` as entity and components for NED task link it to its DBpedia mention `dbr:Berlin`[2]. The following NER and NED components are now available as reusable resources within Frankenstein.

1. **Entity Classifier** uses rule base grammar to extract entities in a text [5]. Its REST endpoint is available for wider use for NER task.
2. **Stanford NLP Tool:** Stanford named entity recogniser is an open source tool that uses Gibbs sampling for information extraction to spot entities in a text [8].
3. The **Babelfy** is a multilingual, graph-based approach that uses random walks and the densest subgraph algorithm to identify and disambiguate entities

[2] `dbr` corresponds to http://dbpedia.org/resource/.

present in a text [16]. We have used public API[3] of Babelfy for NER and NED task as separate components.
4. **AGDISTIS** is a graph based disambiguation tool that couples the HITS algorithm with label expansion strategies and string similarity measures to disambiguate entities in a given text [30]. The code is publicly available[4].
5. **DBpedia Spotlight** is a web service[5] that uses vector-space representation of entities and using the cosine similarity, recognise and disambiguate the entities [15].
6. **Tag Me** matches terms in a given text with Wikipedia, i.e. links text to recognise named entities. Furthermore, it uses the in-link graph and the page dataset to disambiguate recognised entities to its Wikipedia URls [6]. Tag Me is open source, and its REST API endpoint[6] is available for further (re-)use.
7. **Other APIs:** Besides the open-source available components, there are many commercial APIs that also provides open access for the research community. We have used such APIs for NER and NED tasks. Aylien API[7] is one of such APIs that uses natural language processing and machine learning for text analysis. Its text analysis module also consists of spotting and disambiguation entities. TextRazor[8], Dandelion[9], Ontotext[10] [13], Ambiverse[11], and MeaningCloud[12] are other APIs that have been used by us for NER and NED tasks.

Relation Linking Components. Relation Linking (RL) task aims to disambiguate the natural language (NL) relations present in a question to its corresponding mention in a knowledge base (KB). Considering our example question "Who is the mayor of Berlin?" a relation linker component would correctly link the text "mayor of" to `dbo:leader`[13]. For Relation Linking (RL), we rely on following open source components:

1. **ReMatch** maps natural language relations to knowledge graph properties by using dependency parsing characteristics with adjustment rules [18]. It then carries out a match against knowledge base properties, enhanced with word lexicon Wordnet via a set of similarity measures. It is an open source tool, and the code is available for reuse as RESTful service[14].

[3] http://babelfy.org/guide.
[4] https://github.com/dice-group/AGDISTIS.
[5] https://github.com/dbpedia-spotlight/dbpedia-spotlight.
[6] https://services.d4science.org/web/TagMe/documentation.
[7] http://docs.aylien.com/docs/introduction.
[8] https://www.textrazor.com/docs/rest.
[9] https://dandelion.eu/docs/api/datatxt/nex/getting-started/.
[10] http://docs.s4.ontotext.com/display/S4docs/REST+APIs.
[11] https://developer.ambiverse.com/.
[12] https://www.meaningcloud.com/developer.
[13] i.e. http://dbpedia.org/ontology/leader.
[14] https://github.com/mulangonando/ReMatch.

2. **RelationMatcher:** This component [24] devise semantic-index based representation of PATTY [19] (a knowledge corpus of linguistic patterns and its associated properties in DBpedia) and a search mechanism over this index with the purpose of enhancing relation linking task. We call this component RelationMatcher in Frankenstein. The main idea of this component to (1) improve linguistic similarity with cosine similarity by converting PATTY into vector space, (2) address the problem of PATTY patterns not being uniform by introducing some penalty function.
3. **RelMatch:** The disambiguation module (DM) of OKBQA framework [12] provides disambiguation of entities, classes, and relations present in a natural language question [12]. This module is the combination of AGDISTIS and disambiguation module of AutoSPARQL project [27]. We name it as RelMatch. The DM module is an independent component in OKBQA framework and available for reuse[15]. We name this component as "OKBQA relation Disambiguator".
4. **RNLIWOD:** Natural Language Interfaces for the Web of Data ((NLIWOD) community group[16] provides reusable components for enhancing the performance of QA systems. We utilise one of its components to build similar relation linking component[17]. We call this component "RNLIWOD relation linker".
5. **Spot Property:** This component is the combination of RNLIWOD and OKBQA disambiguation module [12] for relation linking task. We call this component Spot Property.

Components for Class Linking. To correctly generate a SPARQL query for a NL query, it is necessary to also disambiguate classes against the ontology.[18] For example, considering the question "Which river flow through Seoul" the word "river" needs to be mapped to `dbo:River`[19]. In Frankenstein, we deployed two components for this task:

1. **NLIWOD CLS:** NLIWOD Class Identifier is one among the several other tools provided by NLIWOD community for reuse. The code for class identifier is available on GitHub (see footnote 17).
2. **OKBQA Class Identifier:** This component is part of OKBQA disambiguation module (see footnote 15). We reused it for specific task of class linking.

Components for Query Builder. A query builder generates SPARQL queries using disambiguated entities, relations and classes which can serve as input from previous steps in QA pipeline. We have used two components for this task:

1. **NLIWOD QB:** Template based query builders are widely used in QA community for SPARQL query construction (e.g. HAWK [29], TBSL [27] etc).

[15] http://repository.okbqa.org/components/7.
[16] see also https://www.w3.org/community/nli/.
[17] https://github.com/dice-group/NLIWOD.
[18] https://www.w3.org/TR/rdf-schema/#ch_class.
[19] i.e. http://dbpedia.org/ontology/River.

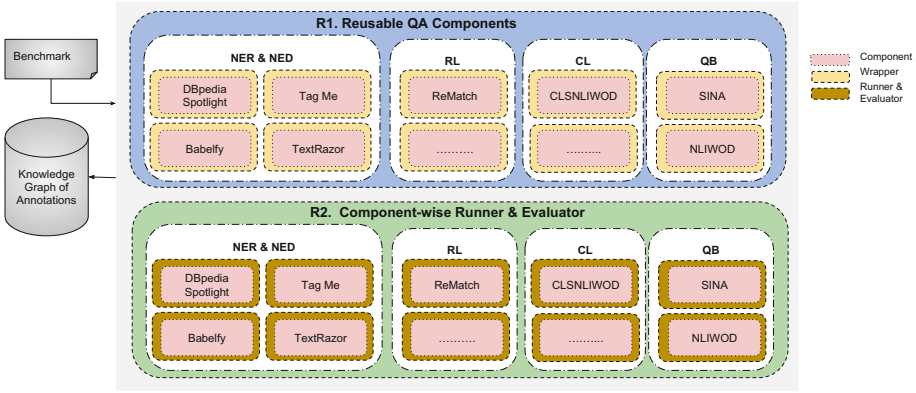

Fig. 1. Modules of Frankenstein (i) Reusable QA component wrappers and (ii) Evaluators.

We build a template based SPARQL query construction based on NLIWOD reusable resources (see footnote 17).

2. **SINA Query Builder:** SINA is a keyword and natural language query search engine that is based on Hidden Markov Models for choosing the correct dataset to query [21]. We decoupled the existing SINA implementation to use its Query builder as independent component.

The complete list of components, their expected input and output can be found in our public GitHub repository[20].

4 Approach for Building Reusable QA Components Within Frankenstein

Figure 1 represents the resource-wise (module-wise) architecture of Frankenstein. It is decoupled into two independent categories, (i) R1 which provides an individual wrapper for each component, and (ii) R2 which provides an individual runner & evaluator for every integrated component. In the following, these two sets of resources are described in more details.

4.1 Integration Approach and Its Challenges

Here, we present the integration approach and its associated challenges for integrating components accomplishing tasks of NER, NED, RL, CL, and QB using Qanary methodology applied in Frankenstein.

[20] The components are listed with file name Component_List.csv in https://github.com/WDAqua/Frankenstein.

Employing Qanary Methodology and qa Vocabulary. Qanary follows a micro service-based architecture where all components are accessible as RESTful services [3] to be possibly integrated into a Qanary QA process. A QA process within Qanary is a knowledge-driven process where input/output about *question*, *answer*, *annotations* generated in different steps of QA pipeline is conceptualized and annotated by the `qa` vocabulary. Each component integrated into a QA pipeline populates a local knowledge graph (typically its output is annotated by `qa` vocabulary) shared with other components within Qanary. This way establishes a standard communication between the components to be a foundation for exchangeability and composability.

In order to be able to annotate outputs generated by all the QA tasks, we had to extend the original version of `qa` vocabulary [22] by adding new concepts for RL, CL, and QB tasks, and reuse annotations of NER and NED s from [4]. E.g., to describe relations appeared in the natural language question, we introduce the annotation:

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
qa:AnnotationOfRelation  rdf:type         owl:Class ;
                         rdfs:subClassOf  qa:AnnotationOfQuestion .
```

For instance, for the given question `Who is the mayor of Berlin?`, the annotated output of RNLIWOD component for RL task is shown below[21]. There, the output i.e., http://dbpedia.org/ontology/leaderName is annotated by the `qa` vocabulary. However w.r.t. `qa` extension, we also introduce further annotations `qa:AnnotationOfClass` and `qa:AnnotationOfAnswerSPARQL` for CL and QB tasks. We reused `qa:AnnotationOfSpotInstance` and `qa:AnnotationOfInstance` for NER and NED tasks from [4].

```
<tag:stardog:api:0.9278702836234858>
  a <http://www.wdaqua.eu/qa#AnnotationOfRelation>;
  oa:core/hasTarget _:bnode_5daa0259_7cbe_4e6c_8504_3f73463e0fd8_69;
  oa:hasBody <http://dbpedia.org/ontology/leaderName>;
  oa:annotatedBy <http://RNLIWOD.com>;
  oa:AnnotatedAt "2017-10-02T13:04:21.27"^^xsd:dateTime.
```

Alignment of QA Component Annotations. To ensure interoperability of a new component with existing ones, it has to express the semantics of its generated information using the `qa` vocabulary. We call this the *alignment* of the component to `qa`. There are at least three ways to align the knowledge of a component about the given question. (1) SPARQL queries: A component can execute SPARQL INSERTs in the knowledge base to generate new annotations expressed using the `qa` vocabulary, (2) OWL axioms: When a component already generates information in a specific vocabulary like the NIF vocabulary used by DBpedia Spotlight [15] OWL axioms might express the semantic relation to the specific vocabulary to the `qa` vocabulary (e.g. by defining `owl:sameAs` rules), (3) Distributed Ontology Language (DOL): It enables heterogeneous combinations

[21] Where `oa` is identified as http://www.w3.org/ns/openannotation/core/.

of ontologies written in different languages and logics [17]. We presented alignments of some existing components using the Qanary approach in [3] and reused similar alignments in this paper.

Wrapping Components and Challenges. During the development of Qanary wrappers in Frankenstein for different components, we encountered several challenges. The first challenge was to deal with interoperability issues among the components. For instance, a number of components were available as RESTful service, while a few ones [18,24] had the only open source code. Thus, we implemented a RESTful service on top of their source codes to make them easily reusable. The second challenge was associated with the heterogeneous output formats of the components. While some just provide output in JSON (e.g. [6,16]), some provide output in their own specific vocabulary (e.g. OntoText [13]). A more challenging case was decoupling SINA from its monolithic implementation required to change the complete package structure, dependencies, input format etc. of the original code to make it reusable.

4.2 Integrating Evaluation Module

Another set of valuable resources in Frankenstein is its evaluation modules. These modules have three configurations (1) benchmark creation, (2) pipeline configuration, and (3) evaluators. The configurations are briefly described below:

Creating Benchmarks for QA Tasks. We follow the methodology provided in [4,24] to create benchmarks for each individual stage of QA pipeline. For our running example `Who is the mayor of Berlin?` the corresponding SPARQL query in QALD-5[22] is:

```
PREFIX dbo: <htp:/dbpedia.org/ontology/>
PREFIX res: <htp:/dbpedia.org/resource/>
SELECT DISTINCT ?uri WHERE {
    res:Berlin dbo:leader ?uri.
}
```

For NED and RL tasks, our modules compare the detected named entities and relations by the components with the entities or relations mentioned in the corresponding SPARQL query (e.g. `res:Berlin` for NED and `dbo:leader` for RL). For class linking (CL) components, a similar approach is applied when questions contain class references. To assess the performance of QB components, we run the generated SPARQL query and the benchmark SPARQL query, then we compare the return answers. For our running question a query builder's performance is evaluated by For evaluating the complete pipeline, answers of the pipeline can be compared with the gold standard answers. In future, we plan to provide a simple configuration that directs the SPARQL results to GERBIL [31] which is an evaluation platform for complete QA processes.

[22] https://qald.sebastianwalter.org/index.php?x=home&q=5.

Pipeline Configuration and Runner. To ease the process of composing pipelines, we have automatised the whole process of configuring and running them using Bash scripts. Based on the required task, the users can choose the components, update the script and automatically run the pipeline in three different modes- (1) Frankenstein static, (2) Frankenstein dynamic, and (3) Frankenstein improved [25]. Below a sample Bash command is represented:

```
(a) ../serverUpdateAndRun.sh stanfordNER
(b) ../serverUpdateAndRun.sh Babefly AGDISTIS
```

The first command i.e., `(a)`, runs a single component i.e., `stanfordNER` and the second command simultaneously run the two components `Babefly` and `AGDISTIS`. These scripts are very useful when the user want to evaluate 1000 s of questions at bulk. However, Frankenstein is provided with an in-built UI from Qanary for executing pipeline with a single input question.

Pipeline Execution. We implemented an independent module called LC-Evaluator within Frankenstein for executing pipelines. This module is customised in an automatic way for evaluating every individual component of the pipeline. This module obtains questions from a text file (supports csv and txt formats). A user can run a single component or a pipeline containing multiple components. However, the pipeline executor automatically passes the questions sequentially to the associated components. Relying on Qanary methodology, the outputs of components (annotations) are stored in a knowledge graph (i.e., Stardog v4.1.3[23]). Then, the pipeline executor reads the annotations of a particular question from the triplestore and creates an independent file using the turtle format (TTL) for the given input question with the label "questionID_component.ttl". This process is efficient in case of a large number of questions or text. The user can upload the text file containing annotations of a question, then execute the LC-Evaluator component, and all the output is automatically generated in form of `.ttl` extension for each question.

Pipeline Evaluator. We developed individual benchmarks for each step of QA pipeline used in evaluation module. Currently, since LC-QuAD [26] and QALD-5 [28] are the most popular and largest state-of-the-art gold standards, thus we developed individual benchmarks out of them for each separate QA task. In the future, we plan to provide additional benchmark files (e.g. for other QALD series). For NED task and for full pipeline evaluation, we plan to integrate pipeline evaluator to GERBIL platform [32]. Using these benchmarks, users are enabled to evaluate the performance of their components for any step of QA pipeline w.r.t. other QA components in Frankenstein.

5 Availability and Sustainability

In this section, we describe the accessibility of resources and our plan for its sustainability. We have published the source code of all the reusable components

[23] https://www.stardog.com/.

integrated in Frankenstein at our public GitHub repository[24] under GPL 3.0[25]. The GitHub repository includes following files:

- R1 in the Qanary folder. Detailed instructions have been provided how to install and use these components integrated using Qanary.
- R2 and detailed instructions to use are present in Evaluation folder.
- A complete component list of 29 integrated components of R1 with its input, output, API restriction is provided.

Regarding the sustainability, the resources are currently maintained by WDAqua project[26]. Once project ends in December 2018, the repository will be transferred to AskNow[27], an initiative to bring all the question answering tools and techniques under a single repository.

6 Related Work

A large number of QA systems were developed in the last years. This can be inferred from the number of QA systems (>38 in the last 5 years) that were evaluated against QALD[28], a popular benchmark for QA systems. Unfortunately, most QA systems follow a monolithic approach, not only at an implementation level but also conceptually. Hence, there is limited reusability for further research. As a consequence creating new QA systems is cumbersome and inefficient. On the other hand, many QA systems reuse existing components. For example, there are services for named entity identification (NEI) and disambiguation (NED) like DBpedia Spotlight [15] and AIDA [11] that are already reused across several QA systems. We are aware of at least three frameworks attempting to provide a reusable architecture for QA systems besides Frankenstein. QALL-ME [7] provides a reusable architecture skeleton for building multilingual QA systems. The main disadvantages are that it proposes a fixed pipeline that cannot be changed. openQA [1] allows combining multiple QA systems. The main downside is that it does not offer modularisation of QA systems and requires the QA systems to be implemented in Java using the provided interfaces. OKBQA [12] is a recent and effective attempt to develop question answering systems with a collaborative effort. OKBQA has 24 components whereas Frankenstein has 29 components. The limitation of OKBQA is that it divides the components into four tasks, namely template generation, disambiguation, query generation, and answer generation and follow strict input/output format. However, in Frankenstein and its resources, the number of tasks is not fixed. When a new component performing a new task needs to be integrated into Frankenstein, it can be done just be extending `qa` vocabulary which is not the case in OKBQA. Frankenstein is built on top of Qanary ecosystem that also provides entity annotation benchmarking

[24] https://github.com/WDAqua/Frankenstein.
[25] https://www.gnu.org/licenses/gpl.html.
[26] http://wdaqua.eu/.
[27] https://github.com/AskNowQA.
[28] https://qald.sebastianwalter.org.

and benchmark 6 QA components [4]. Combining efforts of Frankenstein, Qanary ecosystem and OKBQA will benefit the QA community, as we have reused many independently available QA components from OKBQA repository[29] and plan to provide Frankenstein components for OKBQA.

Besides the QA frameworks, evaluation frameworks like GERBIL [32] have emerged over the last years. GERBIL provides means to benchmark several QA systems on multiple datasets in a comparable and repeatable way fostering the open science methodology. Using GERBIL, many entity disambiguation components can be evaluated using different datasets. However, GERBIL does not provide support for other stages of QA pipeline besides entity annotation and linking. Very recently, this framework is further extended for benchmarking the complete QA pipelines [31]. We plan to integrate some of the reusable components of R2 along with the possibility to benchmark complete QA pipeline in GERBIL.

Besides the question answering community, reusability of components has been a long trend in software engineering [2,10,20]. For example, Rainbow is a popular framework that uses reusable infrastructure to support the self adaptation of software systems [9]. Apache UIMA[30] is another open source project that supports reusable framework, tools, annotators to build software systems for knowledge extraction and information analysis.

7 Conclusion and Future Work

In this paper, we decoupled the Frankenstein architecture and presented reusable resources as an extension to the Qanary. Frankenstein is dedicated to extending the Qanary ecosystem by the following contributions:

R1 It contributes a large set of new components to the ecosystem of reusable components initiated by the Qanary. Consequently, researchers and practitioners are now enabled to create a large set of different QA systems out-of-the-box due to the composability features inherited from the Qanary. We calculated that only by using the components directly provided by Frankenstein/Qanary 380 different ready-to-use QA pipelines can be created with small invest on time.

R2 It provides additional evaluators and intermediate data representations as well as corresponding tools which enables researchers and practitioners to evaluate the tasks within the current QA process. Using these new data sources it is possible to establish quality improving operations on particular QA components or the whole QA process while aiming at improving the QA process for particular fields of applicability. Our recent publication (cf., [25]) already proved possible impact but the additional potential for the QA community is significant.

[29] http://repository.okbqa.org/components/.
[30] https://uima.apache.org/.

Hence, by using the resources provided by Frankenstein the efficiency for building and evaluating QA systems increased significantly for academics and industry which might lead to a boost of new research results.

Frankenstein is contributing to our broader research agenda of offering the QA community an efficient way of applying their research to a research field which is driven by many different fields, consequently requiring a collaborative approach to achieve significant progress.

In the future, we will add even more resources to Frankenstein and its environment, particularly data from evaluations on-top of well-known benchmarks (e.g. the QALD series) leading to new insights on the strengths and flaws of the QA components available in the community as well as machine learning tools enabling optimisation of QA systems with less manual efforts.

Acknowledgements. Parts of this work received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 642795, project: Answering Questions using Web Data (WDAqua). We thank Maria Esther Vidal for her valuable inputs.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
2. Batory, D.S., O'Malley, S.W.: The design and implementation of hierarchical software systems with reusable components. ACM Trans. Softw. Eng. Methodol. **1**, 355–398 (1992)
3. Both, A., Diefenbach, D., Singh, K., Shekarpour, S., Cherix, D., Lange, C.: Qanary – a methodology for vocabulary-driven open question answering systems. In: Sack, H., Blomqvist, E., d'Aquin, M., Ghidini, C., Ponzetto, S.P., Lange, C. (eds.) ESWC 2016. LNCS, vol. 9678, pp. 625–641. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34129-3_38
4. Diefenbach, D., Singh, K., Both, A., Cherix, D., Lange, C., Auer, S.: The qanary ecosystem: getting new insights by composing question answering pipelines. In: Cabot, J., De Virgilio, R., Torlone, R. (eds.) ICWE 2017. LNCS, vol. 10360, pp. 171–189. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60131-1_10
5. Dojchinovski, M., Kliegr, T.: Entityclassifier.eu: real-time classification of entities in text with Wikipedia. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013. LNCS (LNAI), vol. 8190, pp. 654–658. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40994-3_48
6. Ferragina, P., Scaiella, U.: Fast and accurate annotation of short texts with Wikipedia pages. IEEE Softw. **29**(1), 70–75 (2012)
7. Ferrández, Ó., Spurk, C., Kouylekov, M., Dornescu, I., Ferrández, S., Negri, M., Izquierdo, R., Tomás, D., Orasan, C., Neumann, G., Magnini, B., González, J.L.V.: The QALL-ME framework: a specifiable-domain multilingual question answering architecture. J. Web Sem. **9**(2), 137–145 (2011)

8. Finkel, J.R., Grenager, T., Manning, C.D.: Incorporating non-local information into information extraction systems by Gibbs sampling. In: Proceedings of the Conference of 43rd Annual Meeting of the Association for Computational Linguistics, ACL 2005, 25–30 June 2005. University of Michigan, USA (2005)
9. Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**(10), 46–54 (2004)
10. Heineman, G.T., Councill, W.T.: Component-Based Software Engineering. Putting the Pieces Together, p. 5. Addison-Wesley, Boston (2001)
11. Ibrahim, Y., Amir Yosef, M., Weikum, G.: AIDA-social: entity linking on the social stream. In: Exploiting Semantic Annotations in Information Retrieval (2014)
12. Kim, J.-D., Unger, C., Ngonga Ngomo, A.-C., Freitas, A., Hahm, Y.-G., Kim, J., Nam, S., Choi, G.-H., Kim, J.-U., Usbeck, R., et al.: OKBQA framework for collaboration on developing natural language question answering systems (2017)
13. Kiryakov, A., Popov, B., Terziev, I., Manov, D., Ognyanoff, D.: Semantic annotation, indexing, and retrieval. J. Web Sem., 49–79 (2004)
14. Marx, E., Usbeck, R., Ngonga Ngomo, A.-C., Höffner, K., Lehmann, J., Auer, S.: Towards an open question answering architecture. In: Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, 4–5 September 2014, pp. 57–60. ACM (2014)
15. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: DBpedia spotlight: shedding light on the web of documents. In: I-SEMANTICS (2011)
16. Moro, A., Raganato, A., Navigli, R.: Entity linking meets word sense disambiguation: a unified approach. TACL **2**, 231–244 (2014)
17. Mossakowski, T., Kutz, O., Lange, C.: Three semantics for the core of the distributed ontology language. In: Formal Ontology in Information Systems (2012)
18. Mulang, I.O., Singh, K., Orlandi, F.: Matching natural language relations to knowledge graph properties for question answering. In: Semantics 2017 (2017)
19. Nakashole, N., Weikum, G., Suchanek, F.M.: PATTY: a taxonomy of relational patterns with semantic types. In: EMNLP-CoNLL (2012)
20. Neighbors, J.M.: The Draco approach to constructing software from reusable components. IEEE Trans. Softw. Eng. **5**, 564–574 (1984)
21. Shekarpour, S., Marx, E., Ngonga Ngomo, A.-C., Auer, S.: SINA: semantic interpretation of user queries for question answering on interlinked data. J. Web Sem. **30**, 39–51 (2015)
22. Singh, K., Both, A., Diefenbach, D., Shekarpour, S.: Towards a message-driven vocabulary for promoting the interoperability of question answering systems. In: ICSC 2016, CA, USA, pp. 386–389 (2016)
23. Singh, K., Both, A., Diefenbach, D., Shekarpour, S., Cherix, D., Lange, C.: Qanary - the fast track to creating a question answering system with linked data technology. In: ESWC 2016 Satellite Events, Crete, Greece, pp. 183–188 (2016)
24. Singh, K., Mulang, I.O., Lytra, I., Jaradeh, M.Y., Sakor, A., Vidal, M.-E., Lange, C., Auer, S.: Capturing knowledge in semantically-typed relational patterns to enhance relation linking. In: Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA (2017)
25. Singh, K., Radhakrishna, A.S., Both, A., Shekarpour, S., Lytra, I., Usbeck, R., Vyas, A., Khikmatullaev, A., Punjani, D., Lange, C., Vidal, M.E., Lehmann, J., Auer, S.: Why reinvent the wheel-let's build question answering systems together. In: The Web Conference (WWW 2018) (2018, to appear)

26. Trivedi, P., Maheshwari, G., Dubey, M., Lehmann, J.: LC-QuAD: a corpus for complex question answering over knowledge graphs. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 210–218. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_22
27. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: WWW (2012)
28. Unger, C., Forascu, C., López, V., Ngonga Ngomo, A.-C., Cabrio, E., Cimiano, P., Walter, S.: Question answering over linked data (QALD-5). In: Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum, Toulouse, France, 8–11 September 2015. CEUR-WS.org (2015)
29. Usbeck, R., Ngonga Ngomo, A.-C., Bühmann, L., Unger, C.: HAWK-hybrid question answering using linked data. In: Proceedings of The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, 31 May -4 June 2015 (2015)
30. Usbeck, R., Ngonga Ngomo, A.-C., Röder, M., Gerber, D., Coelho, S.A., Auer, S., Both, A.: AGDISTIS - graph-based disambiguation of named entities using linked data. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 457–471. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_29
31. Usbeck, R. Röder, M., Hoffmann, M., Conrads, F., Huthmann, J., Ngonga Ngomo, A.-C., Demmler, C., Unger, C.: Benchmarking question answering systems. Semant. Web J. (to be published)
32. Usbeck, R., Röder, M., Ngonga Ngomo, A.-C., Baron, C., Both, A., Brümmer, M., Ceccarelli, D., Cornolti, M., Cherix, D., Eickmann, B., Ferragina, P., Lemke, C., Moro, A., Navigli, R., Piccinno, F., Rizzo, G., Sack, H., Speck, R., Troncy, R., Waitelonis, J., Wesemann, L.: GERBIL: general entity annotator benchmarking framework. In: Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, 18–22 May 2015 (2015)
33. Zafar, H., Napolitano, G., Lehmann, J.: Formal query generation for question answering over knowledge bases. In: ESWC 2018 (2018, to Appear)