Qanary – A Methodology for Vocabulary-driven Open Question Answering Systems

Andreas Both, Dennis Diefenbach, Kuldeep Singh, Saedeeh Shekarpour, Didier Cherix, Christoph Lange

Qanary – A Methodology for Vocabulary-driven Open Question Answering Systems

Andreas Both[1], Dennis Diefenbach[2], Kuldeep Singh[3], Saedeeh Shekarpour[4], Didier Cherix[5], and Christoph Lange[34]

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

Abstract. It is very challenging to access the knowledge expressed within (big) data sets. Question answering (QA) aims at making sense out of data via a simple-to-use interface. However, QA systems are very complex and earlier approaches are mostly singular and monolithic implementations for QA in specific domains. Therefore, it is cumbersome and inefficient to design and implement new or improved approaches, in particular as many components are not reusable.
Hence, there is a strong need for enabling best-of-breed QA systems, where the best performing components are combined, aiming at the best quality achievable in the given domain. Taking into account the high variety of functionality that might be of use within a QA system and therefore reused in new QA systems, we provide an approach driven by a core QA vocabulary that is aligned to existing, powerful ontologies provided by domain-specific communities. We achieve this by a methodology for binding existing vocabularies to our core QA vocabulary without re-creating the information provided by external components.
We thus provide a practical approach for rapidly establishing new (domain-specific) QA systems, while the core QA vocabulary is re-usable across multiple domains. To the best of our knowledge, this is the first approach to open QA systems that is agnostic to implementation details and that inherently follows the linked data principles.

Keywords: Semantic Web, Software Reusability, Question Answering, Semantic Search, Ontologies, Annotation Model

1 Introduction

Data volume and variety is growing enormously on the Web. To make sense out of this large amount of data available, researchers have developed a number of domain-specific monolithic question answering systems (e.g., [11, 6, 5, 23]). These QA systems perform well in their specific domain, but find limitation in their reusability for further research due to specific focus on implementation details. Hence, creating new question answering systems is cumbersome and inefficient; functionality needs to be re-implemented

and the few available integrable services each follow different integration strategies or use different vocabularies. Hence, an ecosystem of components used in QA systems could not be established up to now. However, component-oriented approaches have provided high values in other research fields (like service-oriented architectures or cloud computing) while increasing efficiency. This is achieved by establishing exchangeability and isolation in conjunction with interoperability and reusability. Increased efficiency of both creating new question answering systems as well as establishing new reusable services would be the major driver for a vital and accelerated development of the QA community in academics and industry.

However, currently the integration of components is not easily possible because the semantics of their required parameters as well as of the returned data are either different or undefined. Components of question answering systems are typically implemented in different programming languages and expose interfaces using different exchange languages (e.g., XML, JSON-LD, RDF, CSV). A framework for developing question answering systems should not be bound to a specific programming language as it is done in [14]. Although this reduces the initial effort for implementing the framework, it reduces the reusability and exchangeability of components. Additionally, it is not realistic to expect that a single standard protocol will be established that subsumes all achievements made by domain-specific communities. Hence, establishing just one (static) vocabulary will not fulfill the demands for an open architecture. However, a standard interaction level is needed to ensure that components can be considered as isolated actors within a question answering system while aiming at interoperability. Additionally this will enable the benchmarking of components as well as aggregations of components ultimately leading to best-of-bread domain-specific but generalized question answering systems which increases the overall efficiency [8]. Furthermore it will be possible to apply quality increasing approaches such as ensemble learning [7] with manageable effort.

Therefore, we aim at a methodology for open question answering systems with the following attributes (requirements): *interoperability*, i.e., an abstraction layer for communication needs to be established, *exchangeability and reusability*, i.e., a component within a question answering system might be exchanged by another one with the same purpose, *flexible granularity*, i.e., the approach needs to be agnostic the processing steps implemented by a question answering system, *isolation*, i.e., each component within a QA system is decoupled from any other component in the QA system.

In this paper we describe a methodology for developing question answering systems driven by the knowledge available for describing the question and related concepts. The knowledge is represented in RDF, which ensures a self-describing message format that can be extended, as well as validated and reasoned upon using off-the-shelf software. Additionally, using RDF provides the advantage of retrieving or updating knowledge about the question directly via SPARQL. In previous work, we have already established a QA system vocabulary `qa` [19]. `qa` is a core vocabulary that represents a standardized view on concepts that existing QA systems have in common, on top of an annotation framework. The main focus of this paper is to establish a methodology for integrating external components into a QA system. We will eliminate the need to (re)write adapters for sending pieces of information to the component (service call) or custom

interpreters for the retrieved information (result). To this end, our methodology binds information provided by (external) services to the QA systems, driven by the qa vocabulary. Because of the central role of the qa vocabulary, we call our methodology Qanary: **Q**uestion **an**swering vocabul**ary**. The approach is enabled for question representations beyond text (e.g., audio input or unstructured data mixed with linked data) and open for novel ideas on how to express the knowledge about questions in question answering systems. Using this approach, the integration of existing components is possible; additionally one can take advantage of the powerful vocabularies already implemented for representing knowledge (e.g., DBpedia Ontology[6], YAGO[7]) or representing the analytics results of data (e.g., NLP Interchange Format [9], Ontology for Media Resources[8]). Hence, for the first time an RDF-based methodology for establishing question answering systems is available that is agnostic to the used ontologies, available services, addressed domains and programming languages.

The next section motivates our work. Sec. 3 reviews related work. In Sec. 4 the problem is will be broken down to actual requirements. Thereafter, we present our approach (Sec. 5) followed by a methodology to align existing vocabularies to our qa vocabulary in Sec. 6. In Sec. 7, we present a case study where a QA system is created containing actual reusable and exchangeable components. Sec. 8 concludes, also describing future research tasks.

2 Motivation

QA systems can be classified by the domain of knowledge in which they answer questions, by supported types of demanded answer (factoid, boolean, list, set, etc.), types of input (keywords, natural language text, speech, videos, images, plus possibly temporal and spatial information), data sources (structured or unstructured), and based on traditional intrinsic software engineering challenges (scalability, openness, etc.) [13].

Closed domain QA systems target specific domains to answer a question, for example, medicine [1] or biology [2]. Limiting the scope to a specific domain or ontology makes ambiguity less likely and leads to a high accuracy of answers, but closed domain systems are difficult or costly to apply in a different domain. *Open domain* QA systems either rely on cross-domain structured knowledge bases or on unstructured corpora (e.g., news articles). DBpedia [3], and Google's non-public knowledge graph [20] are examples of semantically structured general-purpose *Knowledge Bases* used by open domain QA systems. Recent examples of such QA systems include PowerAqua [11], FREyA [6], QAKiS [5], and TBSL [23]. QuASE [21] is a corpus-based open domain QA system that mines answers directly from Web documents.

Each of these QA systems addresses a different subset of the space of all possible question types, input types and data sources. For example, PowerAqua finds limitation in linguistic coverage of the question, whereas TBSL overcomes this shortcoming and

[6] http://dbpedia.org/services-resources/ontology

[7] YAGO: A High Quality Knowledge Base; http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

[8] W3C Recommendation 09 February 2012, v1.0, http://www.w3.org/TR/mediaont-10/

provides better results in linguistic analysis [23]. It would thus be desirable to combine these functionalities of [23] and [12] into a new, more powerful system.

For example, the open source web service DBpedia Spotlight [15] analyzes texts leading to named entity identification (NEI) and disambiguation (NED), using the DBpedia ontology (cf., Subsection 3.2). AIDA [10] is a similar project, which uses the YAGO ontology (cf., Subsection 3.2). AGDISTIS [24] is an independent NED service, which, in contrast to DBpedia Spotlight and AIDA, can use any ontology, but does not provide an interface for NEI. The PATTY system [17] provides a list of textual patterns that can be used to express properties of the YAGO and DBpedia ontologies. As these components have different levels of granularity and as there is no standard message format, combining them is not easy and demands the introduction of a higher level concept and manual work.

3 Related Work

We have already reviewed the state of the art of QA systems in section 2. Work that is related to ours in a closer sense includes other frameworks that aim at providing an abstraction of QA systems, as well as other ontologies used by QA systems.

3.1 Abstract QA frameworks

The QALL-ME framework [8] is an attempt to provide a reusable architecture for multilingual, context aware QA. QALL-ME uses an ontology to model structured data of a specific domain at a time. However, it focuses on closed domain QA, and finds limitation to get extended for heterogeneous data sources and open domain QA systems.

openQA [14] on other hand is an extensible framework for answering questions using open domain knowledge. openQA has a pipelined architecture to incorporate multiple external QA systems such as SINA [18] and TBSL to answer questions. openQA requires all components of the pipeline to be implemented in Java. The OKBQA Hackathon [9], on the other hand, is a collaborative effort to develop knowledge bases and question answering systems that are generic and independent of programming languages.

3.2 Ontologies for question answering

Ontologies play an important role in question answering. First they can be used as a knowledge source to answer the questions. Prominent examples are the DBpedia Ontology and YAGO. DBpedia is a cross domain dataset of structured data extracted from Wikipedia articles (infoboxes, categories, etc.). The DBpedia Ontology is "a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia".[10]

The YAGO ontology unifies semantic knowledge extracted from Wikipedia with the taxonomy of WordNet. The YAGO Knowledge Base contains more than 10 million

[9] OKBQA Hackathon: http://2015.okbqa.org/development/documentation (last accessed: 2016-03-04)

[10] http://wiki.dbpedia.org/services-resources/ontology (last accessed: 2016-03-08)

entities and more than 120 million facts about these entities. YAGO links temporal and spatial dimensions to many of its facts and entities.

Ontologies can also be used to model the search process in a question answering system. For example, the research presented in [22] describes a search ontology that abstracts a user's question. One can model complex queries using this ontology without knowing a specific search engine's syntax for such queries. This approach provides a way to specify and reuse the search queries. However, the approach focuses on textual queries, i.e., it is not completely agnostic to possible question types (e.g., audio, image, ...). Search Ontology also does not cover other possibly useful properties, such as the dataset that should be used for identifying the answer.

However, so far no ontology has been developed that would provide a common abstraction to model the whole QA process.

4 Problem Statement, Requirements and Idea

Our work is motivated by existing QA systems not being sufficiently interoperable and their components not being sufficiently reusable, as pointed out in section 2. Related work on abstract frameworks and QA ontologies has not yet solved the interoperability problem fully, as explained in section 3. In this section, we provide a precise statement of the problem, from which we derive requirements for an abstraction, and finally present our idea for an ontology that can drive extensible QA infrastructures.

4.1 Problem statement

Question answering systems are complex w.r.t. the components needed for an adequate quality. Sophisticated QA systems need components for NEI, NED, semantic analysis of the question, query building, query execution, result analysis, etc. Integrating multiple such components into a QA system is inconvenient and inefficient, particularly considering the variety of input and output parameters with the same or similar semantics (e.g., different terms for referring to "the question", or "a range of text", or "an annotation with a linked data resource", or just plain string literals where actual resources are used). As no common vocabulary for communicating between components exists, the following situation is observable for components that need to be integrated: (1) a (new) vocabulary for input values is established, (2) a (new) vocabulary for the output values is established, (3) input or output values are represented without providing semantics (e.g., as plain text, or in JSON or XML with an ad hoc schema). Confronted with these scenarios, developers of QA systems have the responsibility to figure out the semantics of the components, which is time-consuming and error-prone. Hence, efficiently developing QA systems is desirable for the information retrieval community in industry and academics. We observed in Sections 2 and 3 that the number of reusable (components of) QA systems is negligible so far.

4.2 Requirements

From the previous problem statement and our observations, we derived the following requirements for a vital ecosystem of QA system's components:

Req. 1 (Interoperability)
Components of question answering systems are typically implemented in different programming languages and expose interfaces using different exchange languages (e.g., XML, JSON-LD, RDF, CSV). It is not realistic to expect that a single fixed standard protocol will be established that subsumes all achievements made by domain-specific communities. However, a consistent standard interaction level is needed. Therefore, we demand a (self-describing) abstraction of the implementation.

Req. 2 (Exchangeability and Reusability)
Different domains or scopes of application will require different components to be combined. Increasing the efficiency for developers in academia and industry requires a mechanism for making components reusable and enable a best-of-breed approach.

Req. 3 (Flexible Granularity)
It should be possible to integrate components for each small or big step of a QA pipeline. For example, components might provide string analytics leading to Named Entity Identification (NEI) (e.g., [15]), other components might target the Named Entity Disambiguation (NED) only (e.g., [24]) and additionally there might exist components providing just an integrated interface for NEI and NED in a processing step.

Req. 4 (Isolation)
Every component needs to be able to execute their specific step of the QA pipeline in isolation from other components. Hence, business, legal and other aspects of distributed ownership of data sources and systems can be addressed locally per component. This requirement targets the majority of the QAS platform, to enable benchmarking of components and the comparability of benchmarking results. If isolation of components is achieved, ensemble learning or similar approaches are enabled with manageable effort.

No existing question answering system or framework for such systems fulfills these requirements. However, we assume here that fulfilling these requirements will provide the basis for a vital ecosystem of question answering system components and therefore unexpectedly increased efficiency while building question answering systems.

4.3 Idea

In this paper we are following a two step process towards integrating different components and services within a QA system.

1. On top of a standard annotation framework, the Web Annotation Data Model (WADM[11]), the `qa` vocabulary is defined. This generalized vocabulary covers a common abstraction of the data models we consider to be of general interest for the QA community. It is extensible and already contains properties for provenance and confidence.
2. Vocabularies used by components for question answering systems for their input and output (e.g., NIF for textual data annotations, but also any custom vocabulary) are aligned with the `qa` vocabulary to achieve interoperability of components.

[11] W3C Working Draft 15 October 2015, `http://www.w3.org/TR/annotation-model`

Hence, a generalized representation of the messages exchanged by the components of a QA system is established, independently of how they have been implemented and how they natively represent questions and answers.

Thereafter, the vocabulary `qa` provides the information needed by the components in the implemented question answering system, i.e., a self-describing, consistent knowledge base is available – fulfilling Req. 1. Hence, any component can use the vocabulary for retrieving previously annotated information and to annotate additional information (computed by itself), i.e., each component is using this knowledge base as input and output. This fact and the alignment of the component vocabularies fulfills Req. 2, as each component can be exchanged by any other component serving the same purpose with little effort, and any component can be reused in a new question answering system. Following this process might result in a message-driven architecture (cf., Sec. 7) as it was introduced earlier for search-driven processes on hybrid federated data sources (cf., [4]). However, the methodology might be implemented by different architectures.

5 Approach

5.1 Web Annotation Framework

The Web Annotation Data Model (WADM), currently a W3C Working Draft, is a framework for expressing annotations. A WADM annotation has at least a target and a body. The target indicates the resource that is described, while the body indicates the description. The basic structure of an annotation, in Turtle syntax, looks as follows:

```
<anno> a            oa:Annotation ;
       oa:hasTarget <target>      ;
       oa:hasBody   <body>        .
```

Additionally the `oa` vocabulary provides the concept of selectors, which provide access to specific parts of the annotated resource (here: the question). Typically this is done by introducing a new `oa:SpecificResource`, which is annotated by the selector:

```
<mySpTarget> a              oa:SpecificResource ;
             oa:hasSource   <URIQuestion> ;
             oa:hasSelector <mySelector> .
<mySelector> a              oa:TextPositionSelector ;
             oa:start       "n"^^xsd:nonNegativeInteger ;
             oa:end         "m"^^xsd:nonNegativeInteger .
```

Moreover one can indicate for each annotation the creator using the `oa:annotatedBy` property and the time it was generated using the `oa:annotatedAt` property.

5.2 Vocabulary for Question Answering Systems

In [19] we introduced the vocabulary for the Qanary approach. Following the data model requirements of question answering systems, this vocabulary – abbreviated as `qa` – is used for exchanging messages between components in QA systems [19].

Qanary extends the WADM such that one can express typical intermediate results that appear in a QA process. It is assumed that the question can be retrieved from

a specific URI that we denote with `URIQuestion`. This is particularly important if the question is not a text, but an image, an audio file, a video or data structure containing several data types. `URIQuestion` is an instance of an annotation class called `qa:Question`. The question is annotated with two resources `URIAnswer` and `URIDataset` of types `qa:Answer` and `qa:Dataset` respectively. All of these new concepts are subclasses of `oa:Annotation`. Hence, the minimal structure of all concepts is uniform (provenance, service URL, and confidence are expressible via `qa:Creator`, `oa:annotatedBy`, and `qa:score`) and the concepts can be extended to more precise annotation classes.

These resources are further annotated with information about the answer (like the expected answer type, the expected answer format and the answer itself) and information about the dataset (like the URI of an endpoint expressing where the target data set is available). This model is extensible since each additional information that needs to be shared between components can be added as a further annotation to existing classes. For example, establishing an annotation of the question is possible by defining a new annotation class `qa:AnnotationOfQuestion` (using OWL Manchester Syntax):

```
Class: qa:AnnotationOfQuestion
EquivalentTo: oa:Annotation that oa:hasTarget some qa:Question
```

For additional information about the vocabulary we refer to [19].

5.3 Integration of (external) component interfaces

Following the Qanary approach, existing vocabularies should not be overturned. Instead, any information that is useful w.r.t. the task of question answering will have to be aligned to Qanary to be integrated on a logical level, while the domain-specific information remains available. Hence, we provide a standardized interface for interaction while preserving the richness of existing vocabularies driven by corresponding communities or experts. Existing vocabularies will be aligned to Qanary via axioms or rules. These alignment axioms or rules will typically have the expressiveness of first-order logic and might be implemented using OWL subclass/subproperty or class/property equivalence axioms as far as possible, using SPARQL *CONSTRUCT* or *INSERT* queries, or in the Distributed Ontology Language DOL, a language that enables heterogeneous combination of ontologies written in different languages and logics [16]. The application of these alignment axioms or rules by a reasoner or a rule engine will translate information from the Qanary knowledge base to the input representation understood by a QA component (if it is RDF-based), and it will translate the RDF output of a component to the Qanary vocabulary, such that it can be added to the knowledge base. Hence, after each processing step a consolidated representation of the available knowledge about the question is available.

Each new annotation class (with a specific semantics) can be derived from the existing annotation classes. Additionally, the semantics might be strengthened by applying restrictions to `oa:hasBody` and `oa:hasTarget`.

```
PREFIX itsrdf: <http://www.w3.org/2005/11/its/rdf#>
PREFIX nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>
PREFIX qa:     <http://www.wdaqua.eu/qa#>
PREFIX oa:     <http://www.w3.org/ns/openannotation/core/>

INSERT {
    ?s a oa:TextPositionSelector .
    ?s oa:start ?begin .
    ?s oa:end ?end .
    ?x a qa:AnnotationOfNE .
    ?x oa:hasBody ?NE .
    ?x oa:hasTarget [  a    oa:SpecificResource;
                       oa:hasSource    <URIQuestion>;
                       oa:hasSelector  ?s  ] .
    ?x qa:score ?conf .
    ?x oa:annotatedBy 'DBpedia␣Spotlight␣wrapper' .
    ?x oa:annotatedAt ?time
} WHERE { SELECT ?x ?s ?NE ?begin ?end ?conf
          WHERE { graph <http://www.wdaqua.eu/qa#tmp> {
                          ?s itsrdf:taIdentRef ?NE .
                      ?s nif:beginIndex ?begin .
                      ?s nif:endIndex ?end .
                      ?s nif:confidence ?conf .
                      BIND (IRI(CONCAT(str(?s),'#',str(RAND()))) AS ?x) .
                      BIND(now() as ?time) .
} } };
```

Fig. 1: Aligning identified NE to a new `qa` annotation using SPARQL

6 Alignment of Component Vocabularies

Our goal in this section is to provide a methodology for binding the `qa` vocabulary to existing ones used by QA systems. Of course, it is not possible to provide a standard solution for bindings of all existing vocabularies due to the variety of expressing information. However, here we provide three typical solution patterns matching standard use cases and presenting the intended behavior.

As running example we consider an implemented exemplary question answering system with a pipeline of three components (NEI+NED, relation detection, and query generation and processing; cf., Sec. 7). In the following the components are described briefly and also a possible alignment implementation of the custom vocabulary to `qa`.

6.1 NE Identification and Disambiguation via DBpedia Spotlight

DBpedia Spotlight [15] provides the annotated information via a JSON interface. An adapter was implemented translating the untyped properties DBpedia Spotlight is returning into RDF using NIF. On top of this service we developed a reusable service that aligns the NIF concepts with the annotations of `qa`. First we need to align the implicit NIF selectors defining the identified named entities with the `oa:TextPositionSelector` while aligning the `oa:TextPositionSelector` with `nif:String` on a logical level iff `nif:beginIndex` and `nif:endIndex` exist. This is expressed by the fol-

lowing first-order rule:

$$\begin{aligned} &\text{rdf:type}(?s, \text{nif:String}) \wedge \text{nif:beginIndex}(?s, ?b) \wedge \text{nif:endIndex}(?s, ?e) \\ \Longrightarrow\ &(\exists ?x \bullet \text{rdf:type}(?x, \text{oa:TextPositionSelector}) \wedge \text{oa:start}(?x, ?b) \wedge \text{oa:end}(?x, ?e)) \end{aligned} \tag{1}$$

Additionally the identified resource of the named entity (`taIdentRef` of the vocabulary `itsrdf`) needs to be constructed as annotation. We encode this demanded behavior with the following rule:

$$\begin{aligned} &\text{itsrdf:taIdentRef}(?s, ?NE) \wedge \text{nif:confidence}(?s, ?conf)t \\ \Longrightarrow\ &\text{rdfs:subClassOf}(\text{qa:AnnotationOfEnitites}, \text{oa:AnnotationOfQuestion}) \ \wedge \\ &(\exists ?sp \bullet \text{rdfs:type}(?sp, \text{oa:SpecificResource}) \wedge \text{oa:hasSource}(?sp, <\text{URIQuestion}>) \wedge \\ &\text{oa:hasSelector}(?sp, ?s)) \ \wedge \ (\exists ?x \bullet \text{rdfs:type}(?x, \text{oa:AnnotationOfNE}) \wedge \\ &\text{oa:hasBody}(?x, ?NE) \wedge \text{oa:hasTarget}(?x, ?sp) \wedge \text{qa:score}(?x, ?conf)) \end{aligned} \tag{2}$$

Fig. 1 shows our SPARQL implementations of this rule. After applying this rule, named entities and their identified resources are available within the `qa` vocabulary.

6.2 Relation detection using PATTY lexicalization

PATTY [17] can be used to provide lexical representation of DBpedia properties. Here we created a service that uses the lexical representation of the properties to detect the relations in a question. The service adds annotations of type `qa:AnnotationOfEntity`. Consequently, the question is annotated by a selector and a URI pointing to a DBpedia resource comparable to the processing in Fig. 1.

For example, the question "Where did Barack Obama graduate?" will now contain the annotation:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
  <urn:uuid:a...> a oa:TextPositionSelector ;
          oa:start "24"^^xsd:nonNegativeInteger ;
          oa:end   "33"^^xsd:nonNegativeInteger ;
  <urn:uuid:b...> a qa:AnnotationOfEntity ;
          oa:hasBody dbo:almaMater ;
          oa:hasTarget [  a              oa:SpecificResource  ;
                          oa:hasSource   <URIQuestion> ;
                          oa:hasSelector <urn:uuid:a...> ] ;
          qa:score "23"^^xsd:decimal ;
          oa:annotatedBy <http://wdaqua.example/Patty> ;
          oa:annotatedAt "2015-12-19T00:00:00Z"^^xsd:dateTime .
```

In our use case the PATTY service just extends the given vocabulary. Hence, components within a QA system called after the PATTY service will not be forced to work with a second vocabulary. Additionally, the service might be replaced by any other component implementing the same purpose (Req. 2 and Req. 4 are fulfilled).

6.3 Query Construction and Query Execution via SINA

SINA [18] is an approach for semantic interpretation of user queries for question answering on interlinked data. It uses a Hidden Markov Model for disambiguating entities and resources. Hence, it might use the triples identifying entities while using the

annotation of type qa:AnnotationOfEntity, e.g., for "Where did Barack Obama graduate?" the entities http://dbpedia.org/resource/Barack_Obama and http://dbpedia.org/ontology/almaMater are present and can be used. The SPARQL query generated by SINA as output is a formal representation of a natural language query. We wrap SINA's output into RDF as follows:

```
PREFIX sparqlSpec: <http://www.w3.org/TR/sparql11-query/#>
<urn:uuid:...> sparqlSpec:select "SELECT␣*␣WHERE␣{
␣␣<http://dbpedia.org/resource/Barack_Obama>
␣␣<http://dbpedia.org/ontology/almaMater>␣?v0␣.␣}".
```

As this query, at the same time, implicitly defines a *result set*, which needs to be aligned with the qa:Answer concept and its annotations. We introduce a new annotation oa:SparqlQueryOfAnswer, which holds the SPARQL query as its body.

$$\begin{aligned} & \text{sparqlSpec:select}(?x, ?t) \wedge \text{rdf:type}(?t, \text{xsd:string}) \\ \Longrightarrow\ & \text{rdfs:subClassOf}(\text{oa:SparqlQueryOfAnswer}, \text{oa:AnnotationOfAnswer}) \wedge \\ & (\exists ?x \bullet \text{rdfs:type}(?x, \text{oa:SparqlQueryOfAnswer}) \wedge \text{oa:target}(?x, <\text{URIAnswer}>) \wedge \\ & \text{oa:body}(?x, \text{"SELECT} \ldots \text{"})) \end{aligned} \tag{3}$$

The implementation of this rule as a SPARQL INSERT query is straightforward and omitted due to space constraints. Thereafter, the knowledge base of the question contains an annotation holding the information which SPARQL query needs to be executed by a query executor component to obtain the (raw) answer.

6.4 Discussion

In this section we have shown how to align component-specific QA vocabularies. Following our Qanary approach each component's knowledge about the current question answering task will be aligned with the qa vocabulary. Hence, while using the information of the question answering system for each component there is no need of knowing other vocabularies than qa. However, the original information is still available and usable. In this way Req. 4 is fulfilled, and we achieving Req. 2 by being able to exchange every component.

Note that the choice of how to implement the alignments depends on the power of the triple store used. Hence, more elegant vocabulary alignments are possible but are not necessarily usable within the given system environment (e.g., an alternative alignment for Sec. 6.1, implemented as an OWL axiom, is given in the online appendix[12]).

Here our considerations finish after the creation of a *SELECT* query from an input question string. A later component should execute the query and retrieve the actual resources as result set. This result set will also be used to annotate URIAnswer to make the content available for later processing (e.g., HCI components).

7 Case Study

In this section we present a QA system that follows the idea presented in Sec. 4.3. Note that in this paper our aim was not to present a pipeline that performs better by quantitative criteria (e.g., F-measure) but to show that the alignment of isolated, exchangeable

[12] alternative alignment: https://goo.gl/hdsaq4

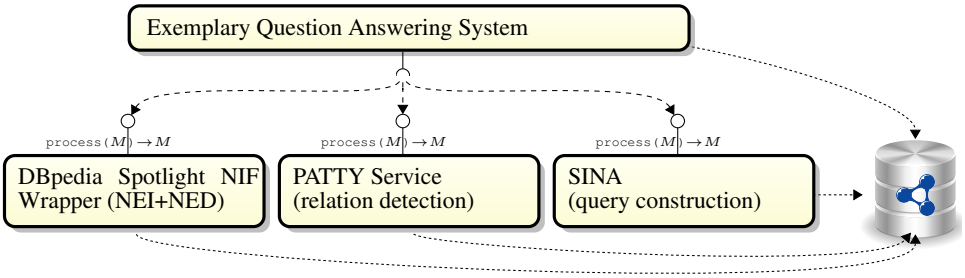

Fig. 2: Architecture of the exemplary question answering system.

components is possible in an architecture derived from the Qanary methodology. In this paper, we have extended the vocabulary proposed in [19] to align individual component vocabularies together to integrate them into a working QA architecture. Without such an alignment, these components cannot be integrated easily together because of their heterogeneity.

Our exemplary QA system consists of three components: DBpedia Spotlight for named entity identification and disambiguation, a service using the relational lexicalizations of PATTY for relation detection, and the query builder of SINA. All information about a question is stored in a named graph of a triple store using the QA vocabulary. As a triple store, we used Stardog[13].

The whole architecture is depicted in Fig. 2. Initially the question is exposed by a web server under some URI, which we denote by `URIQuestion`. Then a named graph reserved for the specific question is created. The WADM and the `qa` vocabularies are loaded into the named graph together with the predefined annotations over `URIQuestion` described in Sec. 5.2. Step by step each component receives a message M (cf., Fig. 2) containing the URI where the triple store can be accessed and the URI of the named graph reserved for the question and its annotations. Hence, each component has full access to all the messages generated by the previous components through SPARQL *SELECT* queries and can update that information using SPARQL *UPDATE* queries. This in particular allows each component to see what information is already available. Once a component terminates, a message is returned to the question answering system, containing the endpoint URI and the named graph URI (i.e., the service interface is defined as `process`$(M) \rightarrow M$). Thereafter, the retrieved URI of the triple store and the name of the named graph can be passed by the pipeline to the next component.

Now let us look into detail about the working of each component.

The first component wraps DBpedia Spotlight and is responsible for linking the entities of the question to DBpedia resources. First it retrieves the URI of the input question from the triple store and then downloads the question from that URI. It passes the question to the external service DBpedia Spotlight by using its REST interface. The DBpedia Spotlight service returns the linked entities. The raw output of DBpedia Spot-

[13] `http://stardog.com/`, community edition, version 4.0.2

light is transformed using the alignment from Subsection 6.1 to update the information in the triple store with the detected entities.

The second component retrieves the question from the URI and analyses of all parts of the question for which the knowledge base does not yet contain annotations. It finds the most suitable DBpedia relation corresponding to the question using the PATTY lexicalizations. These are then updated in the triple store (cf., Sec. 6.2).

The third component ignores the question and merely retrieves the resources with which the question was annotated directly from the triple store. The query generator of SINA is then used to construct a SPARQL query which is then ready for sending to the DBpedia endpoint.

We implemented the pipeline in Java but could have used any other language as well. The implementation of each component requires just a few lines of code (around 2–3 KB of source code); in addition, we had to implement wrappers for DBpedia Spotlight and PATTY (4–5 KB each) to adapt their input and output (e.g., to provide DBpedia Spotlight's output as NIF). Note that this has to be done just once for each component. The components can be reused for any new QAS following the Qanary approach.

Overall, it is important to note that the output of each component is not merely passed to the next component just like other typical pipeline architecture, but every time when an output is generated, the triple store is enriched with the knowledge of the output. Hence, it is a message-driven architecture built on-top of a self-describing blackboard-style knowledge base containing valid information of the question. Each component fetches the information that it needs from the triple store by itself.

In conclusion, the case study clearly shows the power of the approach. The knowledge representation is valid and consistent using linked data technology. Moreover, each component is now isolated (cf., Req. 4), exchangeable and reusable (cf., Req. 2), as the exchanged messages follow the `qa` vocabulary (cf., Req. 1), which contains the available pieces of information about the question, and their provenance and confidence. The components are independent and lightweight, as the central triple store holds all knowledge and takes care of querying and reasoning. As Qanary does not prescribe an execution order or any other processing steps, Req. 3 is also fulfilled.

The case study is available as online appendix[14].

8 Conclusion and Future Work

We have presented an extensible, generalized architecture for question answering systems. The idea is driven by the observation that, while many question answering systems have been created in the last years, the number of reusable components among them are still negligible. Hence, the creation of new question answering systems is cumbersome and inefficient at the moment. Most of the created QA systems are monolithic in their implementation; neither the systems nor their components can be reused. To overcome this problem, our approach follows the linked data paradigm to establish a self-describing vocabulary for messages exchanged between the components of a QA system. Qanary – the question answering vocabulary – covers the requirements of open

[14] https://github.com/WDAqua/Pipeline

question answering systems and their integrated components. However, our goal is not to establish an independent solution. Instead, by using the methodology of annotations, Qanary is designed to enable the alignment with existing/external vocabularies, and it provides provenance and confidence properties as well.

On the one hand, developers of the components for question answering (e.g., question analyses, query builder, ...) can now easily use our standard vocabulary and also have descriptive access to the knowledge available for the question via SPARQL. Additionally, aligning the knowledge of such components with our vocabulary and enabling them for broader usage within question answering systems is now possible. Fulfilling the requirements (cf., Req. 1–4) this ultimately sets the foundation for rapidly establishing new QA systems. A main advantage of our approach are the reusable ontology alignments, increasing the efficiency and the exchangeability in an open QA system.

Our contribution to the community is a vocabulary and a methodology, which take into account the major problems while designing (complex) question answering systems. Via alignments, our vocabulary is extensible with well-known vocabularies while preserving standard information such as provenance. This enables best-of-breed QA approaches where each component can be exchanged according to considerations about quality, domains or fields of application. Additionally, meta approaches such as ensemble learning can be applied easily. Hence, the approach presented in this paper provides a clear advantage in comparison to earlier closed monolithic approaches. Eventually, for the first time the foundations for a vital ecosystem for components of question answering systems is on the horizon. The paper already provides some components and the alignment of their vocabulary to the Qanary vocabulary. In the future, we will integrate further available components by implementing wrappers for them and specifying ontology alignments. Checking the logical consistency of alignments is also a future issue. Additionally an extension for benchmarking and a corresponding framework is planned to be established.

Acknowledgments 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). We would like to thank the anonymous peer reviewers for their constructive feedback.

Bibliography

[1] A. B. Abacha and P. Zweigenbaum. Medical question answering: translating medical questions into SPARQL queries. In *ACM IHI*, 2012.

[2] S. J. Athenikos and H. Han. Biomedical question answering: A survey. *Computer Methods and Programs in Biomedicine*, 99(1):1–24, 2010.

[3] S. Auer, Ch. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A nucleus for a web of open data. In *ISWC + ASWC*, 2007.

[4] A. Both, A.-C. N. Ngomo, R. Usbeck, D. Lukovnikov, Ch. Lemke, and M. Speicher. A service-oriented search framework for full text, geospatial and semantic search. In *SEMANTiCS*, 2014.

[5] E. Cabrio, J. Cojan, A. P. Aprosio, B. Magnini, A. Lavelli, and F. Gandon. QAKiS: an open domain QA system based on relational patterns. In *Proc. of the ISWC 2012 Posters & Demonstrations Track*, 2012.

[6] D. Damljanovic, M. Agatonovic, and H. Cunningham. Freya: An interactive way of querying linked data using natural language. In *ESWC Workshops*, 2011.
[7] T. G. Dietterich. Ensemble learning. In *The Handbook of Brain Theory and Neural Networks*. 2002.
[8] 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. *J. Web Sem.*, 9(2):137–145, 2011.
[9] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer. Integrating NLP using linked data. In *ISWC*, 2013.
[10] Y. Ibrahim, M.A. Yosef, and G. Weikum. Aida-social: Entity linking on the social stream. In *Exploiting Semantic Annotations in Information Retrieval*, 2014.
[11] 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.
[12] V. Lopez, E. Motta, M. Sabou, and M. Fernandez. PowerAqua: A multi-ontology based question answering system–v1. OpenKnowledge Deliverable D8.4, 2007.
[13] V. Lopez, V. Uren, M. Sabou, and E. Motta. Is question answering fit for the semantic web? A survey. *Semantic Web*, 2(2):125–155, 2011.
[14] E. Marx, R. Usbeck, A.-C. N. Ngomo, K. Höffner, J. Lehmann, and S. Auer. Towards an open question answering architecture. In *SEMANTiCS*, 2014.
[15] P. N. Mendes, M. Jakob, A. García-Silva, and Ch. Bizer. DBpedia Spotlight: shedding light on the web of documents. In *I-SEMANTICS*, 2011.
[16] T. Mossakowski, O. Kutz, and C. Lange. Three semantics for the core of the Distributed Ontology Language. In *Formal Ontology in Information Systems*, 2012.
[17] N. Nakashole, G. Weikum, and F. M. Suchanek. PATTY: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, 2012.
[18] S. Shekarpour, E. Marx, A.-C.N. 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.
[19] K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. Towards a message-driven vocabulary for promoting the interoperability of question answering systems. In *Proc. of the 10th IEEE Int. Conf. on Semantic Computing (ICSC)*, 2016.
[20] A. Singhal. Introducing the knowledge graph: things, not strings. *Official Google Blog, May*, 2012.
[21] H. Sun, H. Ma, W.-T. Yih, C.-T. Tsai, J. Liu, and M.-W. Chang. Open domain question answering via semantic enrichment. In *WWW*, 2015.
[22] A. Uciteli, Ch. Goller, P. Burek, S. Siemoleit, B. Faria, H. Galanzina, T. Weiland, D. Drechsler-Hake, W. Bartussek, and H. Herre. Search Ontology, a new approach towards semantic search. In *Workshop on Future Search Engines*, 2014.
[23] Ch. Unger, L. Bühmann, J. Lehmann, A.-C.N. Ngomo, D. Gerber, and Ph. Cimiano. Template-based question answering over RDF data. In *WWW*, 2012.
[24] R. Usbeck, A.-C. Ngonga Ngomo, M. Röder, D. Gerber, S. A. Coelho, S. Auer, and A. Both. AGDISTIS – graph-based disambiguation of named entities using linked data. In *ISWC*, 2014.