Marie Skłodowska-Curie Actions (MSCA)
Innovative Training Networks (ITN)
H2020-MSCA-ITN-2014

WDAqua
"Answering Questions Using Web Data"
ETN, Grant no. 642795

http://wdaqua.informatik.uni-bonn.de

Dissemination level	Public
Type of document	Report
Contractual date of delivery	M16
Actual date of delivery	2016-05-02
Deliverable number	D4.1
Deliverable name	OpenQA specification and reference implementation
Deliverable leader	Fraunhofer IAIS
Work package	WP4
Tasks	T4.1
Coordinator (name, affiliation, contact)	Christoph Lange (FRAUNHOFER; christoph.lange-bever@iais.fraunhofer.de)
Other contributors	See page 2
EC project officer	Matias Pandolfo

Web & Data Science Techniques for Question Answering: A State of the Art

Kuldeep Singh[1], Christoph Lange[1,2], Dennis Diefenbach[3], and Andreas Both[4]

[1] FRAUNHOFER , {kuldeep.singh,christoph.lange-bever}@iais.fraunhofer.de
[2]UBO , langec@cs.uni-bonn.de
[3]UJMS, dennis.diefenbach@univ-st-etienne.fr
[4]Mercateo AG, andreas.both@mercateo.com

May 2016

Contents

1 Executive Summary

The "OpenQA specification and reference implementation" document describes the first step towards the development of the Open Question Answering (OpenQA) architecture to be used by the components developed in the WDAqua project. The OpenQA architecture aims towards a generic question answering architecture, which can be reusable, extensible and scalable. For this we follow a component oriented approach where each component can be integrated in the question answering (QA) pipeline independently from other components. We present how developers can integrate their components into the pipeline and the foundational data model of exchanging information between QA system components. This data model is based on our newly developed vocabulary, referred to as the `qa` ontology.[1] The `qa` vocabulary is designed to be extensible based on the demands of the developers, and covers all the requirements to develop open question answering systems. In the most recent reference implementation of the OpenQA architecture, we have integrated several components in the QA pipeline by way of `qa`-aware wrappers for state-of-the-art components. These are: Stanford NER, DBpedia Spotlight Spotter and FOX for named entity recognition (NER), DBpedia Spotlight Disambiguator and AGDISTIS for named entity disambiguation (NED), as well as the integrated solutions Alchemy API and Lucene Linker. Overall, these all address the step of Entity Linking (EL). The subsequent step of relation extraction is covered by a component wrapping PATTY, and the step of query construction and query execution by a component wrapping SINA. The latter two components have been implemented for an earlier version of the OpenQA architecture, but we outline how they could be ported to the latest version.

In this document, we describe the complete methodology, approach, and steps required to integrate an independent component within QA pipeline. The `qa` vocabulary is published at `https://w3id.org/wdaqua/qanary#`; its implementation can be found in the public source code repository of the WDAqua project at `https://github.com/WDAqua/QAOntology/`. The reference implementation of the components and the implementation of the pipeline can be found at `https://github.com/WDAqua/Qanary/`.

The next section provides an introduction for our work. Sec. 3 presents the existing problem and difficulties to build a generic QA architecture and our idea to solve it. In Sec. 4, we present our actual solution of the problem identified in the previous section. Thereafter, we present our methodology (Sec. 5) to integrate a new component within the QA pipeline. This section describes all the steps developers need to follow to integrate their components within the pipeline. In Sec. 6 we describe all the independent components which are now integrated in the QA pipeline. In Sec. 7, we conclude the report.

2 Introduction

Question Answering (QA) is a multi-disciplinary field, which bridges artificial intelligence, information retrieval, and knowledge engineering. The QA community has paid considerable attention to improving QA systems by taking into account the Web of Data as a source of background knowledge and of answers. It is important to note that most of the QA systems available so far are monolithic and focused on implementation details. Hence, their reusability and extensibility is very limited, which holds the community back from efficiently extending the existing research results. Although significant performance for special use cases is achieved by some QA systems, a shortage was observed in all of them. For example, either their focus is more on implementation details, rather than on a generic architecture which can be extensible, or finds limitations in reusability due to a tightly coupled implementation. Hence, there is a need of a generalized approach to bring the advancement of QA under a single umbrella.

For this, we follow a two step process. In the first step, we present a vocabulary called `qa`, which was originally proposed in [7]. In the second step, we present the Qanary methodology, which was originally presented in [1]. Qanary relies on linked data vocabularies and provides a fast track to integrating QA components into a light-weight, message-driven, component-oriented architecture. In this report, we describe this two-step process and its reference implementation.

3 Problem and Idea

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.

[1] We use the terms "ontology" and "vocabulary" as synonyms. The term "ontology" emphasizes the underlying logical formalisation, whereas the term "vocabulary" emphasizes its role as a language for mutual understanding of information exchanged. In the Semantic Web and Linked Data communities, the term "vocabulary" is typically used for lightweight ontologies, which are largely implemented in RDFS, possibly with a few OWL features.

- *exchangeability and reusability*, i.e., it should be possible to exchange a component within a question answering systemby another one serving the same purpose.
- *flexible granularity*, i.e., the approach needs to be agnostic w.r.t. the processing steps implemented by a question answering system.
- *isolation*, i.e., each component within a QA system should be decoupled from any other component in the QA system.

As QA components are heterogeneous in their implementation, we have identified that a consistent standard interaction level, i.e., a (self-describing) abstraction of the implementation is needed to promote interoperability. This abstraction provides an implementation independent communication mechanism on top of existing implementations. Hence, it promotes reusability of the QA components and systems by decoupling the components within a QA pipeline. Therefore, exchangeability and reusability are important requirements. Isolation has been identified as another requirement: each component should run independently of other components, i.e., it is enabled to be loosely coupled with QA systems. Flexible granularity of the components is required so they can be integrated at any step of the QA process, i.e., in contrast to other QA frameworks the granularity of our architecture is not pre-defined and therefore open for future (special or general) components. To the best of our knowledge, no existing QA system or framework meets these requirements. Taking all of the above issues into account motivated us to introduce a vocabulary that formalizes typical information that various components of a QA system need to exchange with other components. In our concrete implementation, we aim at meeting the requirements listed above by building on this vocabulary.

4 Solution

Our proposed vocabulary extends the Web Annotation Data Model[2] (WADM), which we describe in subsection 4.1. In subsection 4.2 we describe the `qa` vocabulary and in subsection 4.3 the methods to align the input and output components to it.

4.1 The Web Annotation Data Model

The WADM uses the `oa` vocabulary to express annotations. Annotations have 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 the Turtle serialization of RDF, is:

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

Fig. 1 depicts such an annotation. The `oa` vocabulary provides the concept of *selectors*, which provide access to specific parts of the annotated resource (here: the question). For a text question, this is typically done by introducing a new `oa:SpecificResource` and a `oa:TextPositionSelector`:

```
<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 two basic provenance properties: the creator using the `oa:annotatedBy` property and the time it was generated using the `oa:annotatedAt` property.

4.2 The qa vocabulary

We extended the WADM to express annotations which are specific and well-established within the QA scope. It is assumed that a question can be retrieved from a specific URI that we denote with `URIQuestion`.

`URIQuestion` is an instance of the class `qa:Question`. The question is annotated with two resources, `URIAnswer` and `URIDataset`, of types `qa:Answer` and `qa:Dataset` respectively. These resources are further annotated with information about the answer (such as the expected answer type, the expected answer format and the answer itself) and

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

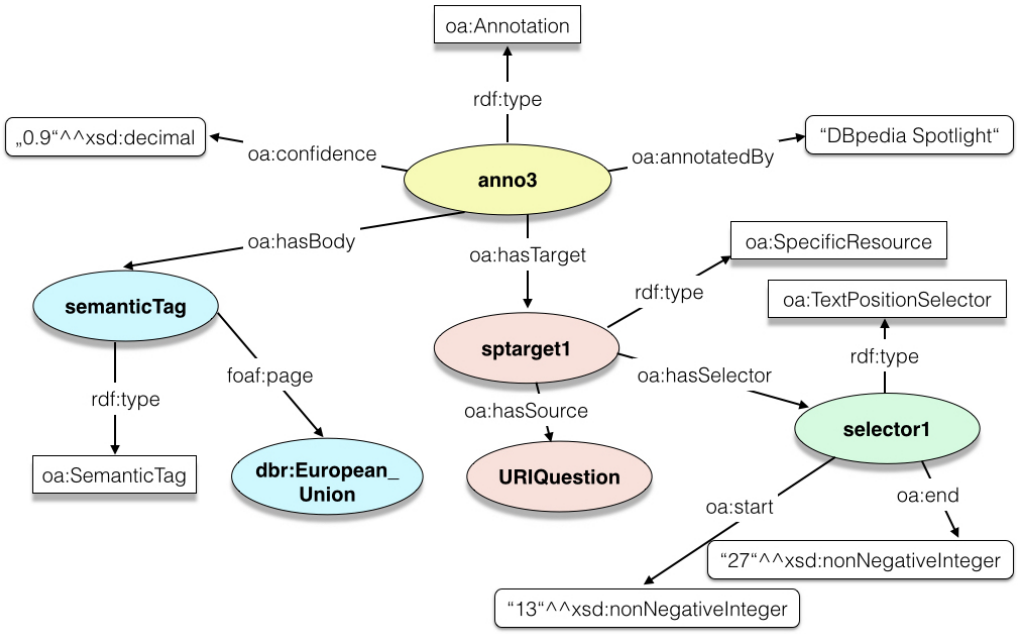

Figure 1: An annotation of the question “Where was the European Union founded?”. The part “European Union” is selected using a Specific Resource and a Selector. Moreover, a semantic tag is associated to it.

information about the dataset (like the URI of an endpoint expressing where the target dataset is available). Hence, the minimal structure of all concepts is uniform (provenance, time, and confidence are expressible via `oa:annotatedBy`, `oa:AnnotatedAt`, and `qa:score`), and the concepts can be extended to more precise annotation classes. The complete OWL implementation of the vocabulary can be found in the public source code repository of the WDAqua project at `https://github.com/WDAqua/QAOntology`. The stable identifier of the vocabulary's namespace is realised as a w3id URI and can be found at `https://w3id.org/wdaqua/qanary#`. Using this vocabulary, it is possible to express all information that QA components can generate about a question. The `qa` vocabulary is extensible, hence it is possible to add annotations in the future, as needed for expressing novel ideas which cannot be predicted currently. For example, with the help of suitable selectors, this vocabulary is sufficiently generic to also support audio questions and can be extended to annotate audio input.

4.3 Alignment of QA 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 ways of expressing information. However, here, we provide three typical solution patterns, which match standard use cases and expose the intended behaviour.

As running example, we consider an implemented exemplary question answering system with a pipeline of three components (EL=NEI+NED, relation detection, and query generation and processing). In the following section, the components are described briefly, including a possible implementation of the alignment of their custom vocabularies to `qa`.

4.4 NE Identification and Disambiguation via DBpedia Spotlight

DBpedia Spotlight [4] provides the annotation information via a JSON interface. An adapter was implemented, which translates the untyped properties returned by DBpedia Spotlight into RDF using the NLP Interchange Format (NIF) [3]. The core of NIF consists of a vocabulary that represents strings as RDF resources. A special URI design is used to pinpoint annotations to a part of a document. These URIs can then be used to attach arbitrary annotations to the respective character sequence. Based on these URIs, annotations can be interchanged between different NLP tools. On top of this service,

[3] `https://site.nlp2rdf.org/`

```
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) .
} } };
```

Figure 2: Aligning an identified NE to a new `qa` annotation using SPARQL

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 if `nif:beginIndex` and `nif:endIndex` exist.

This is expressed by the following rule of first order logic:

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

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:

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

Fig. 2 shows our implementation of this rule in SPARQL. After applying this rule, named entities and their identified resources are available within the *Knowledge Base (KB)*. The Knowledge Base is a repository central to the whole concrete QA pipeline, where all the information about the QA process is stored. This information may be data models, input questions, retrieved answers, and all the intermediate outputs generated by components while executing the pipeline.

4.5 Relation detection using the PATTY Relation Extractor

PATTY is a collection of semantically-typed relational patterns mined from large corpora. The patterns are organised into synonyms and subsumptions. PATTY [5] can be used to provide lexical representations of DBpedia properties and to extract relation present in the input question. 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. 2.

For example, the question "Where did Barack Obama graduate?" will now contain the following 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.

4.6 Query Construction and Query Execution via SINA

SINA [6] is a query construction and execution component that 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`, like 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␣.␣}".
```

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.

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

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.

5 Methodology for Vocabulary-Driven Integration of QA Components

In our methodology, 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[4]), the `qa` vocabulary is defined as explained in Section 4.2. 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. 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 `qa` vocabulary provides the information needed by the components in the implemented question answering system, i.e., a self-describing, consistent knowledge base is available, thus fulfilling the interoperability requirement. 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

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

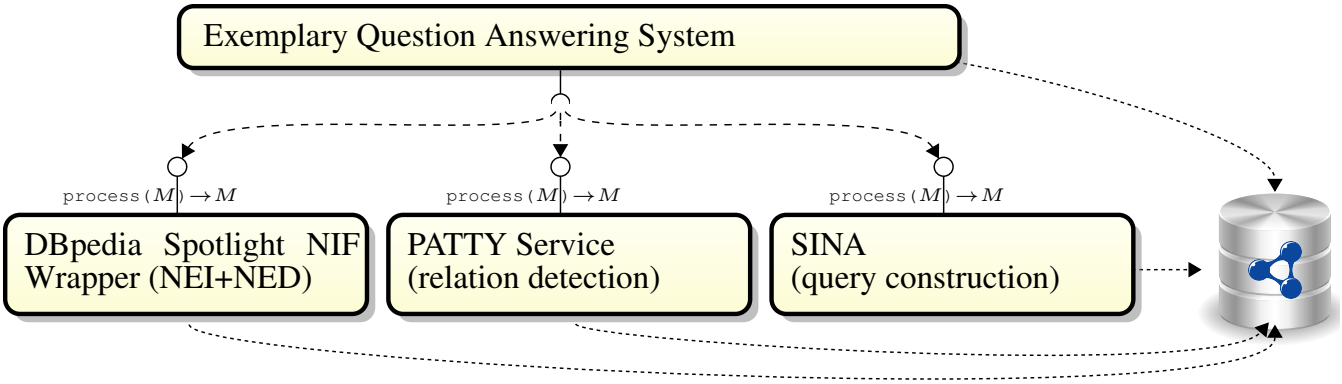

Figure 3: Architecture of the exemplary question answering system.

fact and the alignment of the component vocabularies fulfill exchangebility requirement, 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 results in a message-driven architecture. However, the methodology might also be implemented by different architectures.

Additionally the following core principles apply to the architecture:

- data-driven, where the data should be self-describing; therefore we use RDF
- programming language independent; therefore we encapsulate the components as RESTful services and provide access to the SPARQL endpoint
- self-containing components; therefore components register themself to the QA process component and need just a minimalistic message to be integrated
- reusability: besides the RESTful interfaces there is also the attribute that we have only loosely coupled components. Someone might implement a completely different framework on top of `qa`, but still all components will be reusable.

5.1 Approach

Here, we describe our approach of integrating independent components, namely DBpedia Spotlight, PATTY, and SINA using Qanary with minimal programming effort. Our aim here is to depict how we have integrated three totally independent components in the WDAqua reference architecture, according to the core principles explained above.

Qanary enriches a process-independent Knowledge Base (KB) in each step. The Knowledge Base is a centralised repository where all the information is stored as RDF. Unlike in a traditional QA pipeline, the output of the first component, DBpedia Spotlight, is not directly passed to the second component, PATTY, but is fed into a KB on the abstract level that is defined by the `qa` vocabulary and by aligning existing vocabularies to it. The second component needs particular input; it fetches the required input directly from the KB and pushes its output back to KB. The third component proceeds in a similar way and so on. Each component can access all the messages generated by the previous components stored in a triple store through SPARQL SELECT queries and can update that information using SPARQL UPDATE queries. We follow a three-step process to implement an exemplary QA system:

1. Information gathering: In general, every component has a particular need for information as input. To ensure free access to the required information, every QA component is enabled to execute SPARQL queries and can thus retrieve any knowledge about the question. As the `qa` vocabulary provides a normalized representation of the data, each component only has to know `qa` to access the data. For example, DBpedia Spotlight needs a text query as its input. It might fetch it from the question URI (`<URIQuestion> a qa:Question`), following linked data principles. Additional RDF information about the question can be retrieved by executing a SPARQL query, e.g., to fetch named entities already annotated within a textual question. To access existing components, we have implemented light-weight wrappers that send information to the particular component and recieve the output. The sample code[5] is shown below:

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

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


// Retrieve the question using an HTTP request
RESTClient myRestClient = new RESTClient();


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

2. Information retrieval: Each component performs actions on extracted information and produces some results. In the next step, the wrapper retrieves the computed information from the component. Before pushing it to the KB, it is stored in a temporary location and the defined bindings to the `qa` vocabulary are applied. When a new component needs to be added in the pipeline, and that needs question as input, it can fetch it from the Knowledge Base. As we have mentioned, that question is exposed under a URI `URIQuestion` in the local host like `<http://localhost:8080/Question>`. The developers can write a small SPARQL query to fetch the actual question from this URI. It does not mean that developers need to develop a Web Server for this; a SPARQL query will sufficiently satisfy the requirement.
3. Store results in the Knowledge Base: After the bindings have been applied to the retrieved information, the resulting information is pushed to the KB, i.e. a triple store.
Hence, following Qanary, all QA components are independent from each other and reusable. For example, if a new state-of-the-art named entity disambiguation (NED) method evolves, or new input types come into the picture, researchers just need to replace the NED (in our case study this is DBpedia Spotlight), following the three steps mentioned above, and the new component can be integrated easily into the QA system. Additionally, it becomes reusable for any other QA system following the Qanary methodology.

A possible extension of the described QA system might incorporate support for spoken questions. Hence, a component `C1` is required that translates an audio stream to a textual question, as required by DBpedia Spotlight. The `qa` vocabulary is extensible and already covers the requirements for audio streams. Audio inputs can be represented as byte streams, and this vocabulary provides support to annotate parts of input query, even if it is byte or text. Now the individual vocabulary of `C1` needs to be aligned to `qa`. To integrate `C1` into the QA system, a light-weight wrapper has to be implemented that fetches the required information and passes it to `C1`. The above mentioned three-step process will be followed and `C1` can be integrated easily and efficiently into the QA system.

We have implemented a concrete pipeline with the three components DBpedia Spotlight, PATTY and SINA using the first, now outdated version of the general pipeline framework; the implementation details can be found at `https://github.com/WDAqua/Pipeline`. In the following section, we explain the components that we have already implemented for the latest version of the pipeline, at `https://github.com/WDAqua/Qanary`. In any case, the general principles of ontology, architecture and methodology have remained the same, and up-to-date guidance will always be provided in the public source repositories of the WDAqua project at `https://github.com/WDAqua/`.

6 Integrating and Benchmarking Entity Linking tools with Qanary

With the development of the Qanary methodology, the foundations for the QA ecosystem have now been implemented. The next logical step is to provide relevant, commonly used components as reusable resources. Most state-of-the-art QA systems require Entity Linking (EL) functionality, which can be decomposed into Named Entity Recognition (NER) and Named Entity Disambiguation (NED), at the beginning of their question analyses. Providing reusable resources, initially for EL, and subsequently also for further steps of the QA process, is consequently of great benefit for the community. Hence, we contribute two types of key resources for applying the Qanary methodology in particular and to empower the QA community in general:

- **Contribution Components**

  Qanary Components: We implemented component wrappers for well known components dedicated to the tasks of named entity identification/recognition (NEI/NER) and named entity disambiguation (NED), i.e. to the overall task of entity linking (EL), which is a fundamental step in the state-of-the-art QA pipelines. Hence, these components are populating the ecosystem established by Qanary and can now be easily reused for new QA systems, i.e., they

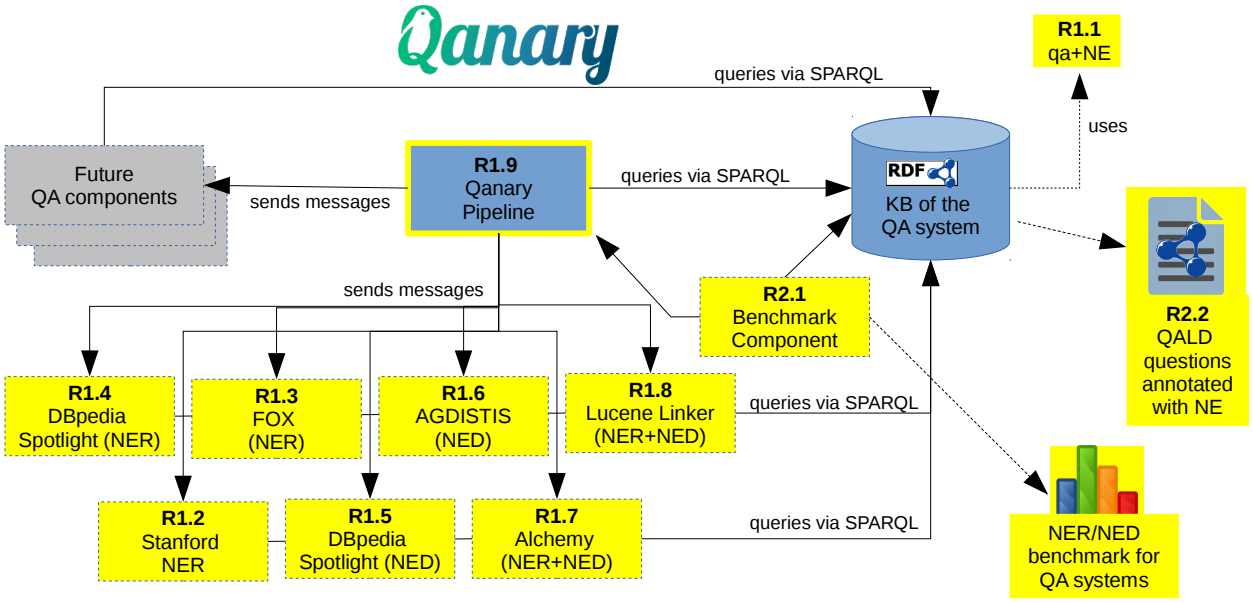

Figure 4: Overview about developed components and derived contributions

are integrable just by configuration. We implemented wrappers for the named entity spotter of *DBpedia Spotlight* [3], the *Stanford NER tool* [2] and the *Federated knOwledge extraction Framework* (FOX) [8] as well as the NED components *Agnostic Disambiguation of Named Entities Using Linked Open Data* (AGDISTIS) [9] and the named entity disambiguator of DBpedia Spotlight. In addition two combined approaches for NEI and NED are also provided as components: *IBM Alchemy*[6] and *Lucene Linker* – a component which we implemented following the idea of the QA system SINA. All these reusable resources can be found in the public WDAqua source repository at `https://github.com/WDAqua/Qanary`.

- **Contribution Benchmark**

  Entity Linking Benchmark for QA: The Entity Linking (EL) task should first identity named entity candidates present in the input question, then a link to a knowledge resource should be provided to disambiguate the named entity (short: NE). We devised a benchmark (also part of `https://github.com/WDAqua/Qanary`) based on the well-known Question Answering over Linked Data (QALD) challenge[7], which enables comparative assessment of QA systems. Our contribution here has three aspects: First, we provide researchers with a tool for comparing NED and NER w.r.t. QA, thus enabling them to compare their components with the state-of-the-art just by implementing a Qanary wrapper around any novel functionality they may have implemented. Second, we provide the results of comparing existing tools, i.e., an expressive benchmark of the quality of entity linking components w.r.t. natural language questions (which are significantly shorter than typical text). Third, we compute a list of questions which are completely annotated w.r.t. the entity linking process. Hence, researchers working on a processing step of a QA system after the entity linking process (e.g. those WDAqua researchers who are working on query generation and execution, and on human-computer interaction) can reuse these annotations for creating an environment for convenient testing and continuous improvement.

These group of resources are depicted in the Figure 4 (resources highlighted in yellow). All the highlighted components are now integrated in OpenQA pipeline using the Qanary reference implementation. The complete reference implementation can be found at `https://github.com/WDAqua/Qanary`. This reference implementation is an extension of our exemplary question answering system described in Sec. 5. Please note that the latest reference implementation code does not currently include components for PATTY and SINA, but these components can be ported to the latest pipeline very easily by following the steps mentioned at `https://github.com/WDAqua/Qanary`. As a result, the QA community is empowered to easily reuse the entity linking functionality for QA systems (or other tools depending on named entities) and reuse a pro-

[6] `http://www.alchemyapi.com/`
[7] `http://greententacle.techfak.uni-bielefeld.de/cunger/qald`

found benchmark for QA systems both for the evaluation of new entity linking components and as input for components active in the subsequent processing steps of a QA system (e.g., relation detection or query computation).

7 Conclusion

Qanary establishes a methodology towards an Open Question Answering (OpenQA) architecture that is independent from the process implemented by concrete state-of-the-art QA systems. It is open for extension and ready for any new idea of how to solve QA tasks; many of such ideas are expected in the further course of the WDAqua project. Additionally our approach is built on top of formal logic to support reasoning and querying in a well-defined way and is independent from the actual implementation (the case study presented above has to be considered as just one possible implementation). When a new requirement evolves, or a new component needs to be included in the pipeline, this can be accomplished via a "fast track" with minimal programming effort. Following the Qanary methodology, we meet all the requirements for a vital ecosystem of QA system components that are actually reusable. At first, we have integrated three components in the reference architecture namely, SINA, PATTY and DBpedia Spotlight. Further, we have extended our implementation to integrate 7 NED and NER components (Stanford NER, DBpedia Spotligtht NED, FOX, DBpedia Spotlight NER, AGDISTIS, Lucene Linker, Alchemy API) and a benchmarking component to evaluate performance of EL task. Hence, the Qanary methodology described in this reference document constitutes the first logical step towards actual open QA systems.

References

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

[2] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, 2005.

[3] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. DBpedia spotlight: Shedding light on the web of documents. In *I-Semantics*, 2011.

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

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

[6] S. Shekarpour, E. Marx, A.-C.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.

[7] 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.

[8] R. Speck and A. Ngonga Ngomo. Ensemble learning for named entity recognition. In *ISWC*. 2014.

[9] R. Usbeck, A. Ngomo, M. Röder, D. Gerber, S. Coelho, S. Auer, and A. Both. AGDISTIS - graph-based disambiguation of named entities using linked data. In *ISWC*. Springer, 2014.