Question answering over Knowledge Bases

Dennis Diefenbach

N°d'ordre NNT : 2018LYSES017

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
Université Jean Monnet

Ecole Doctorale N° ED488
Sciences, Ingénierie, Santé

Spécialité / discipline de doctorat :
Informatique

Soutenue publiquement/à huis clos le 22/05/2018, par :
Dennis Diefenbach

Question Answering over Knowledge Bases

Devant le jury composé de :

Andreas Both, Head of Architecture, Web Technologies and IT Research, DATEV, Rapporteur
Brigitte Grau, Professor,Ecole nationale supérieure d'Informatique pour l'industrie et l'entreprise, Rapporteure
Vanessa Lopez, Research Engineer, Ireland Research Lab of IBM, Rapporteure
Gerhard Weikum , Professor and Research Director, Max Plank Institute in Saarbrücken, Examinateur
Elena Cabrio, Assistant Professor, University of Nice Sophia Antipolis, Examinatrice
Maret Pierre, Professor, Université Jean Monnet, Saint-Étienne, Directeur de thèse
Singh Kamal, Maître de conferences, Université Jean Monnet, Saint-Étienne, Co-Directeur de thèse





École Doctorale ED488 Sciences, Ingénierie, Santé

Question Answering over Knowledge Bases

Thesis prepared at **Université Jean Monnet de Saint-Étienne**
to obtain the degree of :

Docteur de l'Université de Lyon
Spécialité : **Informatique**

by

Dennis Diefenbach

Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d'Optique Graduate School,
Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France.

Jury members :

Pierre Maret	*Professor, Université Jean Monnet, Saint-Étienne*	Directeur
Kamal Singh	*Maître de conferences, Université Jean Monnet, Saint-Étienne*	Co-supervisor
Gerhard Weikum	*Professor and Research Director, Max Plank Institute in Saarbrücken*	Reviewer
Elena Cabrio	*Assistant Professor, University of Nice Sophia Antipolis*	Reviewer
Andreas Both	*Head of Architecture, Web Technologies and IT Research, DATEV*	Examiner
Brigitte Grau	*Professor, Ecole nationale supérieure d'Informatique pour l'industrie et l'entreprise*	Examiner
Vanessa Lopez	*Research Engineer, Ireland Research Lab of IBM*	Examiner

26th March 2018

Dennis Diefenbach
Question Answering over Knowledge-Bases
Reviewers: Gerhard Weikum and Elena Cabrio
Examiner : Andreas Both, Brigitte Grau and Vanessa Lopez
Supervisors: Pierre Maret, Kamal Singh

Université Jean Monnet
Laboratoire Hubert Curien
Connected Intelligence
Data Intelligence
18 Rue du Professeur Benoît Lauras
42023 Saint-Étienne

Contents

1 Introduction

1.1 Overview

Question Answering (QA) is a current research topic in computer science, with the aim to develop a system for automatically answering questions posed by users in natural language. For example if a person asks: “Give me the stock exchanges in New York.", the QA system should be able to answer automatically: “New York Stock Exchange, NASDAQ, OTC Markets Group and NYSE MKT.".

There are mainly three ways to find an answer to a question. The first is searching in free text (like in office documents, web-pages and books). In this case, given a question, the QA system should be able to identify, in a collection of documents, the exact sentence containing the answer. The second is to rely on some online services, like services for weather, current time and stock prices. In this case, the information has to be found by asking one of the underlying web-services. Finally, there is the possibility to search some Knowledge Bases (KB). Let’s first see what a KB is. A KB consists in a set of triples which encode information. For example the information that “Saint-Étienne is in the department of the Loire" can be stored as the triple (Saint-Étienne, department, Loire). KBs can contain millions of such triples.

Doing QA over a KB means that, given a question, one wants to find the answer in the KB. For example imagine a person asks: “In which department is Saint-Étienne?". In this case, we want to retrieve exactly the information contained in the triple above. This will contain the desired answer.

To summarize, the main problem of QA over KBs is to create an algorithm that can automatically search a set of triples and find the answer to a question. This is the type of QA that is tackled in this thesis.

Despite this main problem, there are many related problems. For example: how to present the answer to the question in a user-friendly way, how can an interaction be established between the user and the QA system and how to present the answer such that the user trusts the QA system.

QA over KBs is an exciting topic because it touches many areas in computer science and because it has a strong impact on our everyday life.

1.2 Motivation

In this section, I want to briefly describe why Question Answering (QA), and in particular Question Answering (QA) over Knowledge Bases (KB) is an important and interesting research topic. The argumentation follows mainly two lines. The first wants to show that in our everyday life we constantly try to answer questions. The second line, is that many of the answers can be found in existing Knowledge Bases that are part of the Semantic Web.

1.2.1 Question Answering in daily life

"Every day we search constantly to answer questions". One may think that this statement is an exaggeration, but I want to show that at a closer look it is not. Imagine you wake up in the morning. You may ask yourself: "What is the temperature outside?", "Will it be raining today?" or "Is there still milk in the fridge?". When going to work you may think: "Where is a nearby bakery?", "When is my first appointment?", "How long does it take until I'm in the office?", "What do they serve today in the canteen?", "Will John be in the office today?". While listening to some radio station you may think: "What is the name of the song in the Radio?", "What are other songs of the same artist?", "Where does this band come from?". Finally when you are receiving an e-mail from a researcher you may want to know: "What did he publish?", "What is his/her affiliation?", "How many papers did he/she publish?", "What is his/her website?".
These are only a few examples, but one can extend them to a lot of different situations like: being at an airport, eating in a restaurant, visiting a museum, searching for new shoes and so on.
These example show that the impact of QA systems is and will be huge.

1.2.2 The Semantic Web

Many of the questions of the previous section can be answered using information contained in Knowledge Bases published in the Semantic Web, a part of the World Wide Web. The Semantic Web consists of a set of standards, proposed by the World Wide Web Consortium (W3C), that are used to store and query KB.
As said in the section 1.1 the information in a KB is stored in triples. Let us return to our example phrase "Saint-Étienne is in the department of the Loire". In DBpedia[1] (a well known KB) this information is stored as one triple:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
```

[1] http://wiki.dbpedia.org

```
dbr:Saint-Étienne dbo:department dbr:Loire_(department) .
```

where “dbr:Saint-Étienne” is an abbreviation for “http://dbpedia.org/resource/Saint-Étienne”. Note that “http://dbpedia.org/resource/Saint-Étienne” is a web link, i.e. there exists a corresponding web page. The above data-model to store the information is called RDF. In this data-model, if a triple (s, p, o) is given, then s is called a subject, p is called a predicate and o is called the object. From a mathematical perspective, and also intuitively, one considers a KB as a graph. Each triple (s, p, o) corresponds to the edge of the graph. The edge starts from the vertex s and goes to the vertex o. Moreover, the edge has a label p. In this sense, a KB is a graph which has as vertices the subjects and the objects of the triples. More specifically a KB can be seen as a directed labeled graph. The W3C document containing the definition of RDF can be found at `https://www.w3.org/TR/rdf11-concepts/`. While there are also different ways to encode a Knowledge Base, RDF is the standard format.

The Semantic Web also contains a standard query language for RDF. It is called SPARQL. It allows to retrieve and manipulate data stored in RDF. For example, if one wants to retrieve the object $dbr : Loire\_(department)$, of the above triple, one could generate the following SPARQL query:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?x WHERE {
   dbr:Saint-Étienne dbo:department ?x
}
```

From another perspective the query returns the answer to the question: “In which department is Saint-Étienne?". This allows also to reformulate the problem of QA over KBs. One can say that the basic problem of QA over KBs is to convert a natural language question into a SPARQL query that is able to retrieve the answer to the asked question. The W3C document defining SPARQL can be found at `https://www.w3.org/TR/sparql11-query/`. RDF is published in the Web mainly in two ways. The first is by publishing dumps, i.e., files containing many triples in one of the existing RDF serializations. The set of publicly available RDF dumps in the Web are referred as the Linked Open Data (or simply LOD) cloud. A picture depicting it can be found in Figure 1.1. In February 2017, there were 2973 RDF datasets about music, publication, geography and many more containing 149 billion triples[2].

The second way to publish RDF data, is to embed RDF triples in HTML pages. For example a hotel can embed some information into it’s web-page such as it’s address, the number of stars, links to images and the telephone number. The RDF information will not be seen when

[2] `http://stats.lod2.eu`

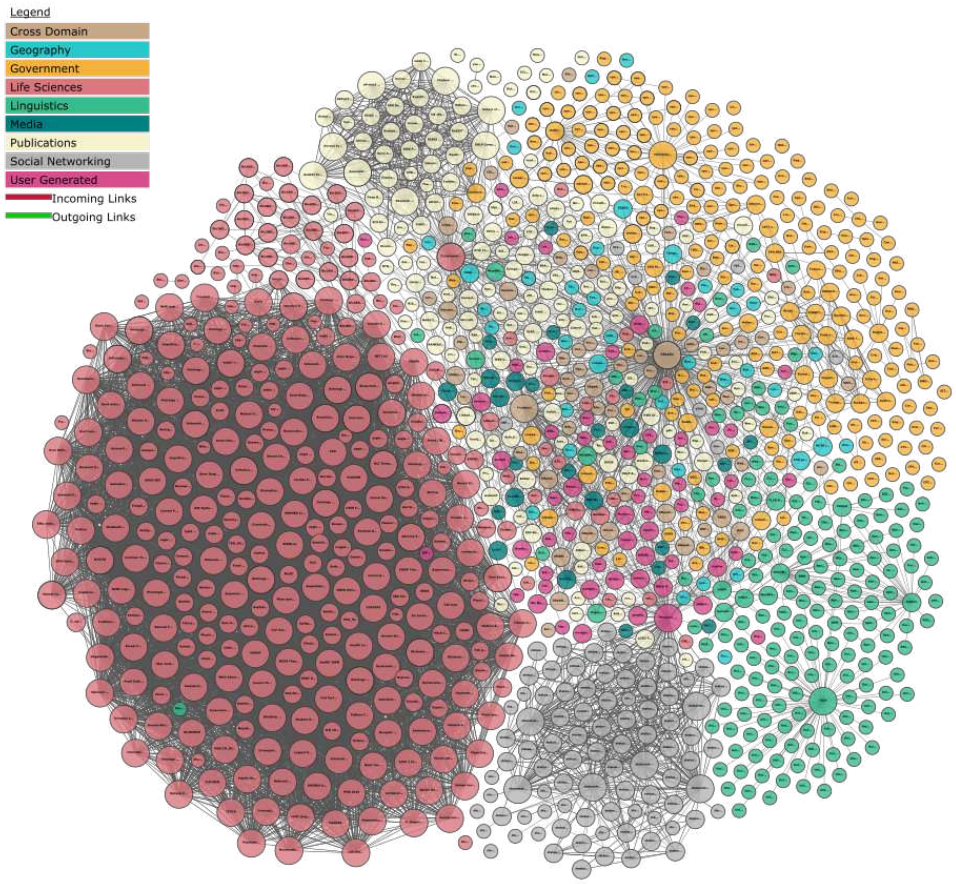

Figure 1.1: Datasets in the Linked Open Data (LOD) Cloud from `http://lod-cloud.net` (to explore this figure more in detail visit the link).

the HTML page is rendered, but can be crawled and has the nice property of being machine readable (this is not the case for the HTML tags that are rendered). Web Data Commons is a crawl of this embedded RDF data and can be found at `http://webdatacommons.org`. The crawl of November 2017 contains 38 billion triples.

One main goal of Question Answering over KBs is to take advantage of this amount of information. On the other side, QA over KBs is also seen as the main tool to make the data in the Semantic Web accessible to end-users. The information in the LOD cloud is written in RDF, which is clearly not a user-friendly format. Also extracting information from it using SPARQL is clearly impossible for end-users. The ability to automatically generate a SPARQL query, from a natural language question, is seen as the way to bridge the gap between end-users and Semantic Web data.

Figure 1.2: Answer of Google Search for "mayor of Saint-Étienne". Note that the answer was found in Wikipedia. The text snippet where the answer was found is shown. In this case, the answer is retrieved from free-text. Note also that the answer is wrong. (result from 15th January 2018)

1.2.3 Existing solutions in Industry

Industry is already presenting solutions to answer some of the questions presented in section 1.2.1. The aim is to bring the users as close as possible to the answer. There is a real race going on between the big players in this market. The most visible solutions are *Ok Google*[3], *Apple Siri*[4], *Alexa*[5], *WolframAlpha* [6] and *Microsoft Cortana*[7]. All these digital assistants are exposed to users via multiple channels like smart-phones, web-sites, chat-boots and smart speakers.

These digital assistants are a combination of QA systems over free text, web-services and Knowledge Bases. See figure 1.2, 1.3 and 1.4 to see how Google Search combines this 3 types of QA approaches.

Note that in all the above cases, it is not clear which algorithms are used. The companies do not share details about them.

1.3 Research questions and challenges

In this section, I want to describe the research questions that, from my point of view, are most important in the field of Question Answering over Knowledge Bases. These research questions arose during the analysis of the state-of-the-art done in [49] which corresponds to Chapter 2 and during subsequent research. Some of them are briefly mentioned in Table 2.9.

[3] https://support.google.com/websearch/answer/2940021?hl=en
[4] http://www.apple.com/ios/siri/
[5] https://developer.amazon.com/alexa
[6] http://www.wolframalpha.com
[7] http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana

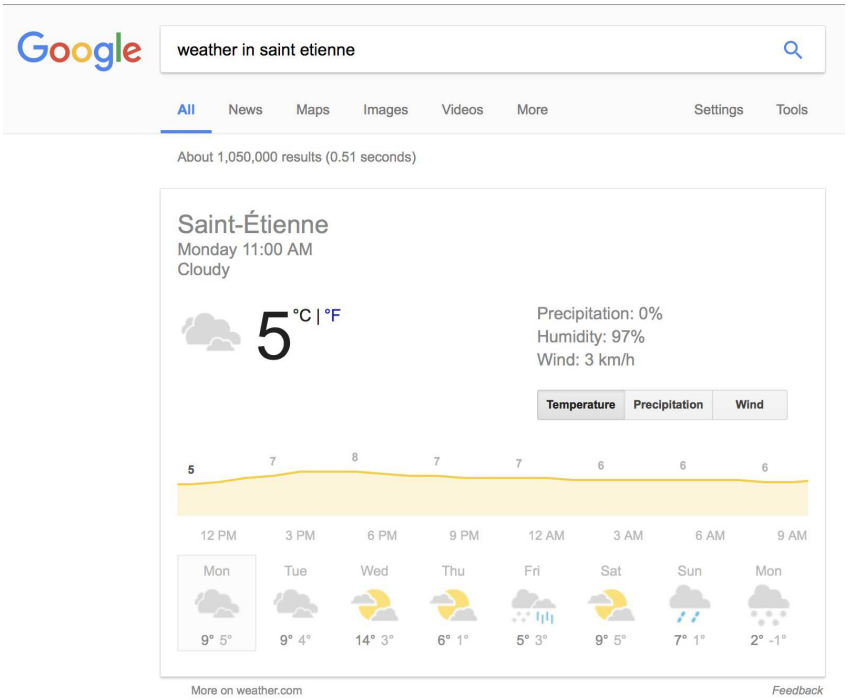

Figure 1.3: Answer of Google Search for "weather in Saint-Étienne". Note that the answer comes from weather.com. In this case the answer is retrieved via a web-service. (result from 15th January 2018)

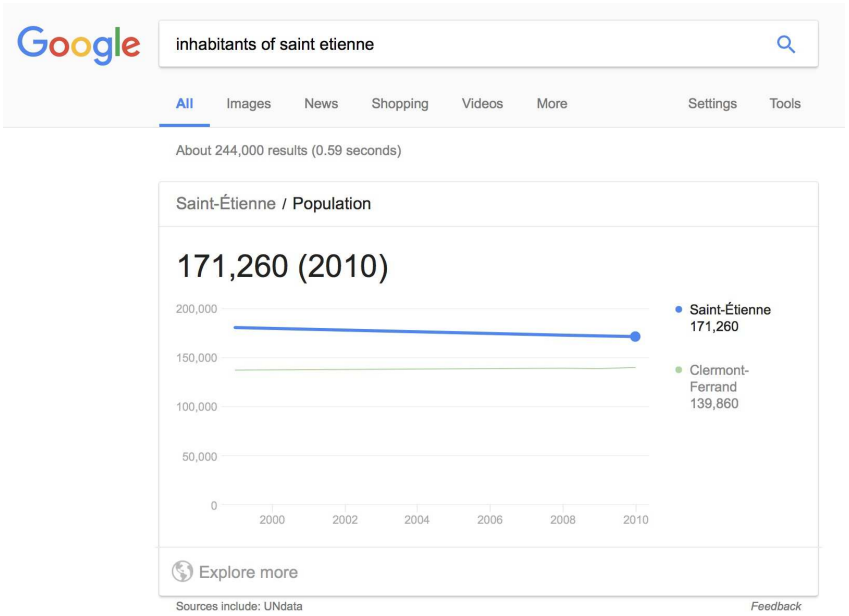

Figure 1.4: Answer of Google Search for "inhabitants of Saint-Étienne". This answer was found in the Google Knowledge Graph, a proprietary KB. (result from 15th January 2018)

1.3.1 R1 Multilinguality

Multilinguality in QA over KB can have two different meanings. In the first case, the KB consists of facts expressed in one language (for example in English) and is accessed by natural language questions in another language (for example French). In the second case, the knowledge in the KB, and the language of the questions, are the same (for example both French or both German) and multiple KBs exist in different languages. Here, multilinguality expresses the ability of a QA system to work for multiple languages (i.e. for French and German).
The first case clearly needs translations, either of the question to the language used in the KB, or of the facts of the KB to the corresponding language of the question. So the problem is somehow moved to a translation problem. In the second case the knowledge and the language are the same. So in this case no translation is needed, but the algorithm must be adapted to the corresponding language.
Multilinguality is a real challenge for QA over KBs. As can be seen in Table 3.3, Table 2.7 and Table 3.5, which summarizes the QA systems evaluated over QALD, WebQuestions and SimpleQuestions (the three most popular benchmarks), respectively, more than 90% of the proposed approaches work only for English. The important point is that, a QA approach over English, generally cannot be ported easily to a QA approach over another language. There are mainly two reasons for that. The first is that many QA systems over-rely on rules over features of natural language processing tools for Named Entity Recognition (NER), POS-tagging or dependency parsing. All these rules are strongly dependent on the language and therefore are also very difficult to port from one language to another. The second reason is that even though some approaches are language independent, by design, but they need a lot of training data in the corresponding language. Creating this training data is expensive and this limits the multilinguality.
Considering the difficulty to make a multilingual QA system this represents a difficult research question.

1.3.2 R2 Portability

Portability in QA over KB is a completely ignored challenge. I'm not aware of any work clearly stating this problem and the term is not used in literature. With portability, I refer to the problem of adapting a QA system to a new KB. Note that nearly all existing works, except **??**, were evaluated only on one KB (mostly DBpedia or Freebase).
Similar to multilinguality, portability is also a real challenge. Strangely, while multilinguality is recognized as a challenge by the research community, portability as an issue is not recognized. There is often a "non-written" assumption that, if one can do a good QA system over a KB, then it will also work on other KBs. This assumption is wrong. The problems are similar to the ones for multilinguality.

- First, many QA systems depend on some tools that are adapted to one KB. Most importantly these are Entity Linking (EL) tools. One could argue that, when changing the KB, one can simply use a EL tool for that KB. There are two possible problems here. While there are many EL tools for popular KB like DBpedia and Freebase, there are generally no EL tools for other KBs. Second, one can think that one can port the algorithm of the EL tool used. But many times the algorithm cannot be ported. For example DBpedia Spotlight, a well known EL tool with good performance, is using the free text of Wikipedia to perform EL. This means that the algorithm is not portable, for example, to KBs of scientific publications.

- The second problem that prevents to port a QA system to a new KB is again the training data. It is already expensive to create training data for KBs, like DBpedia and Wikidata, which contain general knowledge. Imagine creating training data for KBs about scientific publications or medicine, where you need domain experts!

For the above reasons, portability is a difficult challenge. Moreover, and most importantly, it is the key feature of a QA system to query the data of the Semantic Web which consists of thousands of KBs. Therefore, portability represents an important research question.

1.3.3 R3 Scalability

Scalability refers to the problem of answering a question in a time that is acceptable by the user. This problem mainly depends on the size of the data to query. Note that a KB like Wikidata contains more than 2 billion triples, which corresponds to roughly 250 Gb. On the other side, a user nowadays expects response times of around 1 second. Since a QA system is running on a server, in addition to the time required to compute the answer, it is also necessary to add the overhead of the network requests, and the retrieval and rendering of the information related to the answer. This shows that scalability becomes very easily a problem.
Tables 3.3, 2.7 and 3.5 summarize most of the existing proposed QA solutions. They also indicate the mean performance per query in terms of execution time, for the QA approaches (if indicated in the corresponding publication). This clearly depends on the underlying hardware, but independently from that, it provides a hint on the scalability of the proposed approach. Note that for most of the systems no performance in terms of execution time is indicated. One of the main reasons is that these QA systems do not exist as one pipeline, but often authors write sub-results of sub-pipelines into files and arrive step-wise at the final answer. For the few systems that provided the performance in terms of execution time, this varies, depending on the question, from a mean of 1 second for gAnswer [188] to more than 100 seconds for some questions in the case of SINA [132]. Most of the systems report a run-time of around 10 seconds. This is also an indicator that scalability is an issue.

1.3.4 R4 Robustness

Robustness refers to the ability of a QA system to reply also to questions that are not well-formulated. The variety of questions that users ask varies considerably. Users can ask full natural language questions or keyword questions, but also questions that are syntactically incorrect or that contain typos. This problem is not well addressed in research. One main problem is due to the benchmarks used by the research community. For example, the QALD benchmark offers only questions that are well formulated and do not contain syntactical errors. So while many researchers apply NLP tools to solve the proposed questions, it is questionable if such approaches would work when they are used for answering questions formulated by real users. Questions in WebQuestions and SimpleQuestions are not always syntactically correct, but still are quite well formulated.
For this reason it is unclear how existing systems behave when put in contact with real users.

1.3.5 R5 Query multiple datasets

As described in Section 1.2.2 the Semantic Web contains many different datasets about different domains. For example Wikidata contains information about general knowledge and MusicBrainz contains information about music. Moreover, some of the information in the different datasets is overlapping. For example the albums of the Green Days are contained both in Wikidata and MusicBrainz, but albums about a much smaller bands are contained only in MusicBrainz. For an end-user it is not clear which dataset contains which information. Even experts are not aware if a certain information is available in a certain dataset. For example if one searches information about Deskadena, a very small local music group in Italy, it is not clear where to find it and if it is available at all. It could be that a fan created the corresponding Wikidata entry and it could also be that someone created the entry in MusicBrainz. In any case, it is important to note that it is not easy to decide in which dataset to search for a given specific information. This shows that this task should not be left to the end-user but should be carried out automatically.
There are very few works in this direction. The existing work is described in Section 2.11. There are mainly two approaches. The first is to search all existing KBs and to combine the knowledge on the fly (i.e. by interlinking the resources on the fly). The second approach is to assume that the knowledge is already interlinked and to consider multiple KBs as one big KB. The existing works focus on very small KBs, since scalability in both the above cases becomes an issue.

1.3.6 R6 Answer presentation

Given a question the main task of QA over KBs is to automatically generate from a natural language question a SPARQL query and to retrieve the corresponding answer. This is also

the main focus of the research community. But for the end-user this process alone does not suffice. The result set of a SPARQL query is a JSON object containing a list of URIs. This is certainly not something presentable to an end-user. How to present this information to the user and how to enrich it with additional information, is a problem that is poorly addressed in research. The question here is, how many additional resources like text, images, external links can be presented to the user? Depending on the information displayed, does the user believe in the generated answer? Are there information that can be displayed to the user so that he/she is able to discover more about the dataset? All these questions are essential for making the QA system usable. Even though one can construct an algorithm that can solve the main task, i.e., translating natural language question to SPARQL queries, but if these issues are not addressed, the usage will still be limited.
While some QA systems offered a basic user interface that touches these issues, I'm not aware of any publication discussing these issues.

1.3.7 R7 User interaction

One of the consequence of the lack of research in the design of user interfaces for QA systems over KBs, is the lack of studies on the interaction between users and QA systems. There are mainly two cases where the interaction between a QA system and a user is important. The first is when the question posed by a user is ambiguous. For example, if a users asks for "Who is Quicksilver?" he/she could search for "Quicksilver (American professional wrestler from San Diego, California)" or also for "Quicksilver (comic book character)". The question is ambiguous in this sense. In these cases, it is important to have a mechanism which allows the QA system to inform the users that it is aware of different possibilities, and allows the user to choose between the different possibilities.
A second important interaction between the user and the QA system is the moment in which the answer is presented. Does the user believe in the given answer? Is he/she able to understand if the QA system misunderstood it? If the QA system is confident about the answer, it is important that the user believes it. Moreover, the user should be able to detect if the QA system misunderstood the question so that he/she can reformulate it, or interact differently to change the interpretation.

1.3.8 R8 Re-usability and composability of QA components

In the last 7 years, many different QA approaches were presented. One can find more than 40 approaches that were evaluated on the three most popular benchmarks: QALD, WebQuestions and SimpleQuestions. Moreover, a detailed analysis shows that each approach is not completely new, but many approaches share many common strategies. The easiest way to see this is to compare the Tables 2.3, 2.4, 2.5, 2.6. They show the different techniques used across the QA systems evaluated over QALD. Clearly many techniques are reused across many QA systems. Naturally, from this observation, an idea arises to encapsulate

these techniques in components and to make QA systems by composing them. In this way a new researcher can reuse the existing approaches, while concentrating only on a sub-part of the problem, and relying on existing work for the other parts.
There are multiple questions here. Is it possible to encapsulate these functionalities in a modular way, so that they can be combined with each other? Is it possible that a researcher is able to create a QA system, through a collaborative effort, by reusing existing work? How will such an infrastructure look like?
The advantages would be very big. Research groups could build together QA systems and advances on one component would immediately show their effect on the whole pipeline. The result would be that the entire field will advance much faster.

This concludes the list of some of the most pressing research questions in the area of QA over KBs.

1.4 Contributions and Discussion

In this section, I discuss what are the contributions of this thesis to the research field of QA over KBs. For each research challenge, I will point to the corresponding contribution provided in this thesis.

1.4.1 R1 R2 R4 R5 Multilinguality, Portability, Robustness, Query multiple datasets

The challenges about multilinguality, portability, robustness and querying multiple datasets, at first look, seem very different. The main contribution of this thesis is to present a novel approach for translating a natural language question into a SPARQL query, it is a significant contribution to tackle these challenges. I want first to sketch the approach and then analyze its consequences.
The approach is based on the idea that many questions can be understood by ignoring the syntax of the question, and just the semantics of the words suffice. I want to illustrate the idea using an example. Imagine the QA system receives the question: "Give me actors born in Strasbourg". A common approach is to first analyze the phrase syntactically and find out that Strasbourg is a named entity, born is a predicate and use this information to create the corresponding SPARQL query. I follow a completely different approach. I first consider to which possible concepts the consequent words in the questions can refer to. For example, I take into consideration all meaning that the word "actor" can have according to the underlying KB, i.e., it can refer to a "2003 film by Conor McPherson", a "programming language called actor" and to the concept of actor as it is intended in the question. The word

"born" can correspond to the intended meaning in the question, i.e., "the birth place" but could also be a "family name", a "district of the city Brüggen" and many other things. The question is that which is the right meaning among all of these? To make this decision, I look into the KB and I search for the most probable meaning. I interpret this as the identified concepts that are nearest in the graph. Based on the distance of different concepts in the graph, I construct all possible SPARQL queries. If I apply this strategy to the above question, using as a KB DBpedia, one gets over 200 possible SPARQL queries. We call each of these queries an interpretation of the question. Translated in natural language, some are just "Give me information about the district of Brüggen", "Give me information about the 2003 film by Conor McPherson." others are "Give me actors related to Strasbourg." and also the searched interpretation of "Give me actors born in Strasbourg.". Note that one will probably not find any connection in the KB between the "2003 film by Conor McPherson" and the "district of Brüggen" so that these concepts are probably not the ones intended. Once the interpretations are generated they are ranked and, if everything goes right, we find the requested SPARQL query.

While, at the first look, the approach may look quite complex and not so promising, it turns out that it works well and it also has a considerable number of advantages over traditional methods.

The first advantage is multilinguality. The same arguments above also work for the question in German: "Gib mir Schauspieler die in Straßbourg geboren sind.". The reason is that the semantics of a question is language independent, while the syntax is not. The limitation is clear that it is not possible to capture the difference between two questions if the system relies only on the syntax. I would like to give some examples. This approach cannot catch the difference between "Who are the children of Barack Obama?" and "Barack Obama is the child of who?". Also the difference between "What is a continent?" and "Give me the continents." or even between the keyword questions "continent" and "continents" is not caught. But many other questions, even quite complex ones can be solved. Moreover, it is possible to adapt the proposed approach to catch these syntactical variation. But these adaptations will not be multilingual. Note that also superlatives and comparatives are very language dependent and are for that reason not automatically considered.

The second advantage is portability. It is achieved thanks to two main properties. The first is the need for very few to no training data. This means that one does not need to construct big training datasets, which in turn can be very expensive. The second is that the approach is quite flexible regarding on how the knowledge is encoded into the KB. For example the information that "the capital of Europe is Brussels" can be encoded as the triples:

```
(Brussels,capital,Europe)
            OR
(Europe, capital, Brussels)
            OR
(Europe, type, capital of Europe)
            OR
```

```
(Brussels, type, capital)
(Brussels, located in, Europe)
```

Independent of how the information is encoded, the approach is able to adapt to different ways.
The third advantage is robustness. If the user asks "Give me actors born in Strasbourg", "actors born in Strasbourg", "actors Strasbourg born in", or "born actor Strasbourg" the approach can always identify the intended meaning. In particular this has a side benefit that this approach also works for keyword questions as well as syntactically wrong questions. However, note that, here the robustness is not achieved in the case some words are misspelled.
The fourth advantage is that the approach also works, without any modifications, for multiple KBs.
I implemented the approach in a QA system called WDAqua-core1. Currently, it can query 4 different KBs (DBpedia, Wikidata, MusicBrainz and DBLP) and 5 different languages (english, french, german, italian and spanish). A demo can be found under:

`www.wdaqua.eu/qa`.

More details are given in Chapter 3 which corresponds to the publication [55].

1.4.2 R3 Scalability

Addressing scalability was one of the main motivation behind implementing the approach described in the previous section. As specified in Section 1.3.3, scalability is an issue in itself for QA over KBs. But one of the main drawback, of the approach described above, is that a large solution space has to be explored. To perform this in acceptable time, an efficient index is necessary. Given 2 RDF resources, this index should allow to compute the distance between them in a KB, as fast as possible. I explored numerous ways to create such an index. One promising idea was explored in [54]. The idea was to pre-compute the distance between the resources. While the response times were good, the memory footprint of such an approach is just too big. This leads to the idea of computing the numbers on the fly by traversing the RDF graph. While an initial implementation showed that this way was possible, it still required to have the data in memory, which again is not scalable. Also the use of a graph database like neo4j (`https://neo4j.com`) could not solve the problem. The final solution was to use HDT (Header, Dictionary, Triples). HDT was designed to be an exchange format for RDF data. A not so well-known property is that HDT stores the data as a compressed adjacency list. This is an ideal structure for breadth search operations. This is the key ingredient for solving the scalability issue, both in terms of time and memory.

1.4.3 R6 Answer presentation

The work on how and which information could be presented to end-users of QA systems leads to two results. A front-end called Trill and a generalized framework for entity summarization called SummaServer.
The Semantic Web contains many information that can be useful for the user. These include images, abstracts from wikipedia, geo-coordinates and links to social networks and other related resources. All these resources are used by the front-end called Trill, when an answer is presented. A screen-shot of Trill can be found in Figure 1.5.
Although some KBs like Wikidata and DBpedia are quite rich with information which can be shown to end-users, this is not generally the case. For example, DBLP offers few such information. One possibility in such cases is to exploit the KB itself and show to the user some of the facts about the answer entity. This problem is referred as entity summarization. One contribution of this thesis is to extend an existing framework for entity summarization called SummaServer**??**, to be easily portable to any RDF based KB. The corresponding result is also integrated into Trill.
More information about Trill can be found in Section 4.1, which corresponds to the publication [46]. More information about the SummaServer can be found in Section 4.2 which corresponds to publication [56].

1.4.4 R7 User interaction

This thesis contains two contributions on user interactions in the context of QA systems over KB.
The first contribution is a general mechanism for user interaction with a QA system. This mechanism was implemented for two scenarios. The first is what we call the "Did you mean" functionality. This mechanism enables a user to interact with a QA system in the case the question is ambiguous. A concrete example is given in figure 1.6. The second is similar, but it targets expert users. An example is given in figure 1.7. A more detailed description is given in Section 4.4 which corresponds to the publication [47].
The second contribution, is a method to generate an interpretation of a given SPARQL query that a user can easily understand. The corresponding tool is called SPARQLToUser. Why is this important for the user-interaction? Imagine a user asks: "Who is the mayor of Saint Chamond?". If the QA systems replies "Jean Rouger" then the user has only the choice to either believe the answer or not. But the QA system generated a SPARQL query which corresponds to how the QA system understood the question. SPARQLToUser would generate the following interpretation from the SPARQL query:

head of government / Saintes (commune in Charente-Maritime, France)

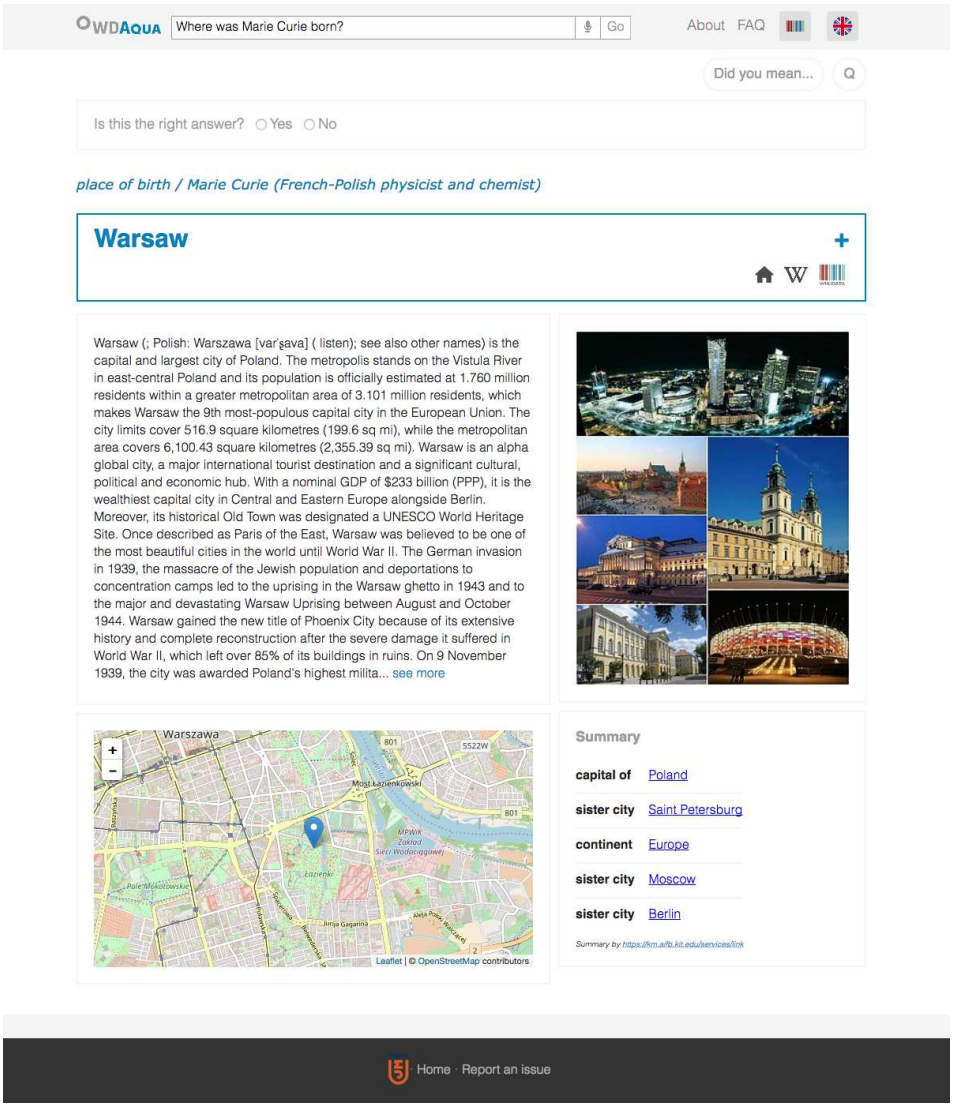

Figure 1.5: Screen-shot of the front-end Trill for the question "Where was Marie Curie born?"

From above, it is clear that the QA system did not understand "Saint-Chamond", but rather understood "Saintes (commune in Charente-Maritime, France)". So the user is able to decide that the answer is wrong with a high probability. A detailed description of SPARQLToUser and a corresponding user study is described in section 4.3, which corresponds to publication [32].

1.4.5 R8 Reusability and composability of QA components

The research on reusability and composability led to the Qanary architecture. It consists of three building blocks.

The first is the Qanary vocabulary. It consists of an RDF vocabulary that can be used to express information that QA components exchange between each other. For example, a QA component may want to communicate the information, to another component, that a

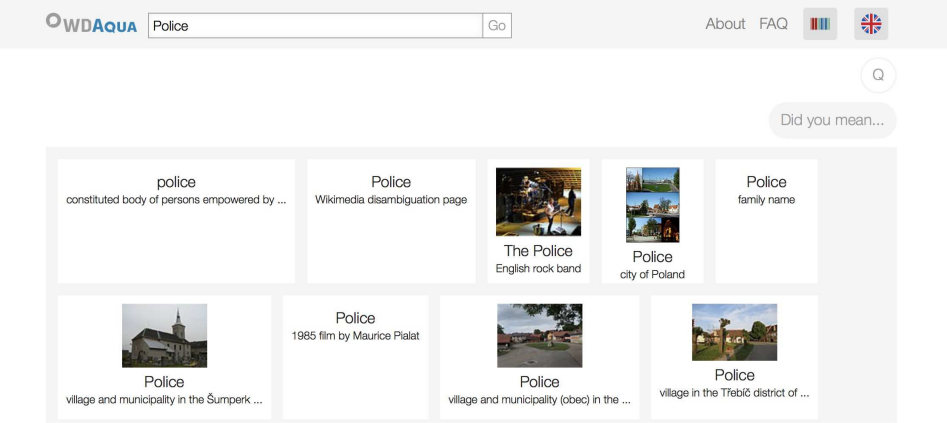

Figure 1.6: This screen-shot shows the "Did you mean" functionality. If a user just asks for "police" the QA system will return the police intended as "constituted body of persons empowered by the state to enforce the law". But also other meaning could have been meant by the user. For example there is a polish city called "Police". The "Did you mean" functionality presents such alternative meanings to the user. After clicking on them the answer is recomputed.

segment of text in the question is referring to a named entity, or that a segment of text was linked to an entity in a KB. The Qanary vocabulary can be seen as an effort to standardize the information flow between QA components using an RDF vocabulary. The Qanary Vocabulary is described in Section 5.1 and corresponds to the publication [133].
The second block is the Qanary methodology. The Qanary methodology describes how QA components can interact with each other using the Qanary Vocabulary. The idea is the following. All information about a new question are stored in a named graph N of a triple-store T using the Qanary vocabulary. Each component has the following structure:

- The component retrieves all needed information from a named graph N, in order to use it for computations (like the question itself, previously identified segments of text, previously identified entities).
- The component performs it's actual task.
- The component writes all information, which it has computed, back to N using the Qanary Vocabulary.

Since all components are speaking the same language, namely the Qanary vocabulary, they can be integrated with each other, and components with the same functionality can be exchanged. The Qanary methodology is described in Section 5.2 which corresponds to the publication [22].
The third block is the Qanary Ecosystem. It consists of a set of components and services that are implemented using the Qanary methodology.
Each component is implemented as a web-service which is registered to a central component.

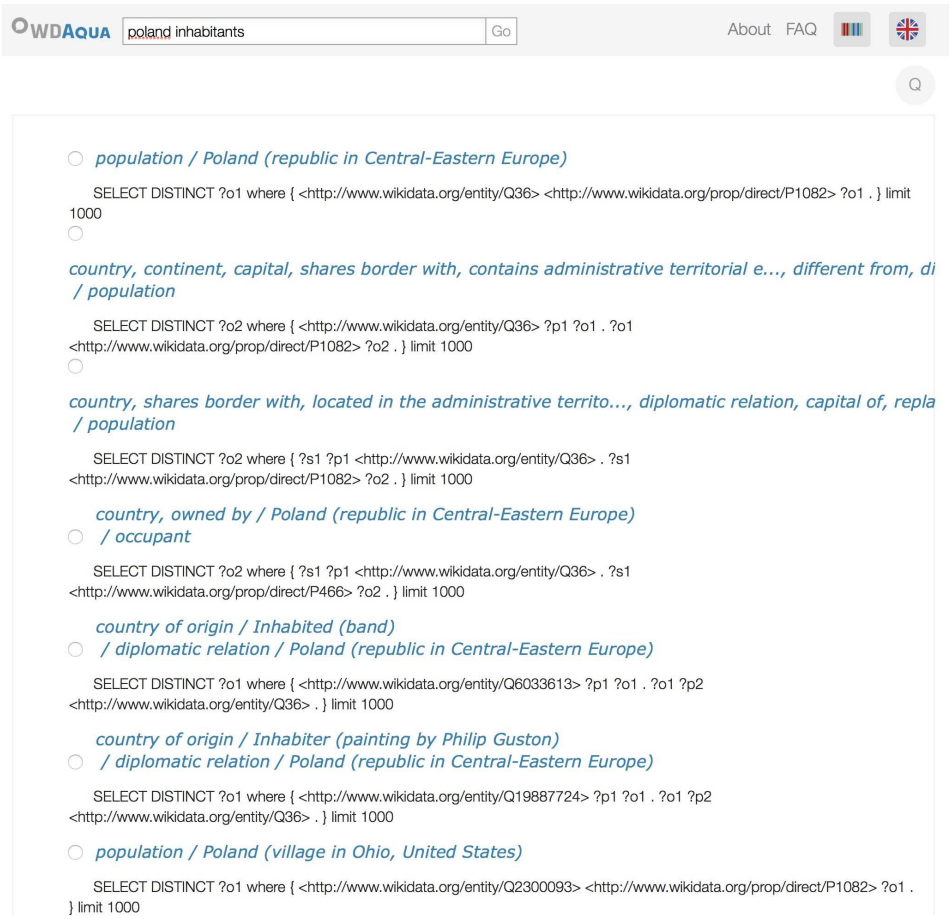

Figure 1.7: This screen-shot shows an interface that allows interaction between a user and a QA system. The user asked "poland inhabitants". The interface shows the SPARQL queries generated by the QA system together with the interpretations generated by SPARQLToUser. By clicking on one of the queries the answer set will be computed and shown to the user.

In this way, the research community is always aware about the components that are offered by other researchers and that can be used to compose a QA pipeline. Moreover, since the components are web-services, it is possible to combine services that are distributed around the web. This enables to really construct QA systems that are distributed over the Web. The Qanary ecosystem is presented in Section 5.3. The corresponding publication [50] introduced 7 components: 3 Named Entity Recognition tools, 2 Named Entity Disambiguation tools and 2 Entity Linking tools. More components were added to the Qanary ecosystem in [135]. It now supports 29 components.

The Qanary Ecosystem also introduces a set of services that are related to QA systems. These include, bench-marking with the Gerbil-for-QA [162] platform and the user interface Trill.

From my point of view, the Qanary architecture revealed a series of problems. The first is the assumption that the research community already offers a series of components that can be used to construct QA systems. This is not the case. Though there are many components

for EL, there are few components for relation linking and class linking. Also the components for query construction are very limited. The main reason is that even if a large number of QA systems were built in the last years, there are very few examples of QA systems that are open source or are running as web-services. In this sense, from my point of view, most of the research is lost since it is then either difficult or impossible to reproduce the results. Moreover, most of the presented components work only over DBpedia and for English. A second problem is the assumption that it is easy to model the information flow between QA components. Even if the Qanary vocabulary provides a first standardization in this direction, it is far from being complete. A third problem is related to scalability. The triple store that is assumed in the Qanary methodology is a big bottleneck from a performance perspective. The overhead is too big considering that it is only at the level of the infrastructure. Finally, [135] showed that even after considerable efforts in finding and integrating and combining QA components, the results on QALD-6 are disappointing. The QA pipeline with the best results, achieves one of the worst results on benchmark, and is far from the results of the best performing systems.

1.5 Organization of the thesis

In this section, I will outline the organization of this thesis. This is a PhD thesis in the form of a collection of publications. It consists of the following chapters:

- Chapter 2 describes the state-of-the-art in the area of QA over KBs. It corresponds to the publication "Core Techniques of Question Answering Systems over Knowledge Bases: a Survey" published in "Knowledge and Information systems" [49].

- Chapter 3 describes a new approach for QA over KBs. The corresponding publication is under submission at the Semantic Web Journal [55].

- Chapter 4 describes contribution on the User Interface. It consists of four publications: "Trill: A reusable Front-End for QA systems" [46] published at "ESWC Poster and Demo", "PageRank and Generic Entity Summarization for RDF Knowledge Bases" [56] published at ESWC 2018, "SPARQLtoUser: Did the question answering system understand me?" [32] published in "ISWC NLIWoD3 Workshop" and "Introducing Feedback in Qanary: How Users can interact with QA systems" [47] published in "ESWC 2016 Poster and Demo" track.

- Chapter 5 contains contributions on an architecture of reusable components for QA systems. It consists of three publications: "Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems" [133] published in "ICSC 2016", "Qanary – A Methodology for Vocabulary-driven Open Question Answering Systems" [22] published in "ESWC 2016" and "The Qanary Ecosystem:

getting new insights by composing Question Answering pipelines" [50] published in "ICWE 2017".

1.6 Conclusion and Future Work

This thesis presents significant contributions in the domain of QA systems over KBs.
I have motivated the importance of QA over KBs and analysed the contributions of this thesis with respect to the state-of-the-art. To summarize, the main contributions are:

- a new approach that is multilingual, scalable, easily portable to new KBs and that supports both full natural language question and keyword questions,
- the design of a User Interface that exposes the information about an answer in a user-friendly way,
- the study of some possible interactions between a QA system and an end-user
- the development of an architecture for reusable components which can be distributed over the web and can be combined and exchanged so that the research community can potentially develop QA systems with a community effort.

I integrated all contributions in a demo system that is publicly available under:

`www.wdaqua.eu/qa`

The list of publications, datasets and repositories that resulted from the work of this thesis are listed at page 199.
I consider this work far from finished and this research topic still offers many challenges and open problems. I want to briefly list and discuss them.

- Misspelling and auto-completion: an open problem is how to deal, in a scalable way, with situation in which the end-user does not spell single words correctly. Moreover, I believe that it would be helpful to the user if the QA system is able to use the knowledge in the underlying KB to give hints to the user during the formulation of the question. Note that it would be interesting not to guide the user in a very rigid way, as some proposed approaches did.
- Scalability to the whole Semantic Web is still a challenge. There are two problems. On one side there is the challenge of constructing an infrastructure and parallelizing the algorithm properly. On the other side there is also the problem to correctly interpret the underlying knowledge. I believe that some RDF standardization is needed for

this. Examples are properties that indicate images that can be exposed to users, disambiguated labels and stricter definitions in the use of the property "rdfs:label".

- The interlinking between big KBs like DBpedia, Wikidata and OpenStreetMap should be used in an extensive manner to provide qualitatively better results to the user.

- The QA technology can be used to construct chat-bots. While the technology is very similar, the main difference is to keep track of the context between one question and the other.

- QA over reefied-statements is addressed by some QA systems evaluated over WebQuestions, but it is poorly addressed in other contexts. This problem should be tackled in a general manner.

- In Knowledge Bases like Wikipedia there are information like the provenance of the information. It would be interesting to study how this can be presented to the user and how it can be used to compute a trust scores of the corresponding information.

- Speech recognition systems that are adapted and specialized to a given KB do not exist. Such systems would be very important to be able to do QA systems over KBs that can take speech input.

Question Answering over Knowledge Bases is and will remain an exciting research topic for years to come!

2 Core Techniques of Question Answering Systems over Knowledge Bases: a Survey[1]

2.1 Abstract

The Semantic Web contains an enormous amount of information in the form of knowledge bases (KB). To make this information available, many question answering (QA) systems over KBs were created in the last years. Building a QA system over KBs is difficult because there are many different challenges to be solved. In order to address these challenges, QA systems generally combine techniques from natural language processing, information retrieval, machine learning and Semantic Web. The aim of this survey is to give an overview of the techniques used in current QA systems over KBs. We present the techniques used by the QA systems which were evaluated on a popular series of benchmarks: Question Answering over Linked Data (QALD). Techniques that solve the same task are first grouped together and then described. The advantages and disadvantages are discussed for each technique. This allows a direct comparison of similar techniques. Additionally, we point to techniques that are used over WebQuestions and SimpleQuestions, which are two other popular benchmarks for QA systems.
Keywords: Question Answering; QALD; WebQuestions; SimpleQuestions; Survey; Semantic Web; Knowledge base

2.2 Introduction

Question answering (QA) is a very old research field in computer science. The first QA systems were developed to access data over databases in the late sixties and early seventies. More recent works designed QA systems over free text, i.e., finding the part of text that contains the answer of a question from a set of documents. These are referred as information retrieval (IR) based QA systems. In the last two decades, thanks to the development of the Semantic Web, a lot of new structured data has become available on the web in the form of knowledge bases (KBs). Nowadays, there are KBs about media, publications, geography,

[1] Corresponding publications is: Dennis Diefenbach, Vanessa Lopez, Kamal Singh and Pierre Maret, *Core Techniques of Question Answering Systems over Knowledge Bases: a Survey*, Knowledge and Information systems, 25 Sep 2017 [49]

life-science and more[2]. QA over KBs has emerged to make this information available to the end user. For a more detailed historical overview see [103]. The publication of new KBs accelerated thanks to the publication by the W3C of the de facto standards RDF[3] and SPARQL[4]. RDF is a format for publishing new KBs. SPARQL is a powerful query language for RDF, but it requires expert knowledge. The idea behind a QA system over KBs is to find in a KB the information requested by the user using natural language. This is generally addressed by translating a natural question to a SPARQL query that can be used to retrieve the desired information. In the following, when we speak about QA systems, we refer to QA systems over KBs. Due to the large number of QA systems nowadays, an in-depth analysis of all of them would require extensive work. Instead, we restrict to QA systems participating in a specific benchmark. This will allow for a comparison of the overall performance of the QA systems and gives at least a hint of how good the single techniques used in QA systems are. We concentrate here on the "Question Answering over Linked Data" (QALD) benchmark which is in line with the principles of the Semantic Web: it focuses on RDF KBs, it proposes KBs in different domains assuming that a QA system should be able to query any KB, it proposes large open-domain KB like DBpedia such that scalability becomes an issue, it offers questions where the information should be extracted from multiple interlinked KBs, it contains questions with multiple relations and operators such as comparatives and superlatives and it proposes multilingual questions assuming that the users can be of different nationalities. Moreover, we collected a list of QA systems evaluated over WebQuestions and SimpleQuestions, two popular benchmarks, and we point to approaches used by these QA systems and not by the ones evaluated over QALD.

Differently from other surveys in the field of QA over KBs, we focus on the techniques behind existing QA systems. We identified five tasks in the QA process and describe how QA systems solve them. This way each QA system participating in QALD is decomposed and compared. The tasks are question analysis, phrase mapping, disambiguation, query construction and querying distributed knowledge. Our main aim is to describe, classify and compare all techniques used by QA systems participating in QALD. This provides a good overview on how QA systems over KBs work.

The paper is organized as follows: in section 2.3, we position our survey with respect to past surveys on QA systems. In section 2.4, we describe three popular benchmarks in QA namely: WebQuestions, SimpleQuestions and QALD. In section 2.5, we describe how the compared QA systems were selected. In section 2.6, we give an overview over the five tasks. In section 2.7, we describe the techniques used for the question analysis task. In section 2.8, we discuss the techniques used for the phrase mapping task and, in section 2.9, the techniques for the disambiguation task are provided. In section 2.10, we describe how the different QA systems construct a query. Finally in section 2.11, we describe what changes should be made if the required information is distributed across different KBs. In section 2.12 and 2.13, we compare the analysed systems to the ones evaluated over WebQuestions

[2] http://lod-cloud.net

[3] http://www.w3.org/TR/rdf11-mt/

[4] http://www.w3.org/TR/sparql11-query/

and SimpleQuestions. In section 2.14, we give an overview of the evolution of the challenges in the field of QA. Based on the analysis of this publication, in section 3.8, we point to future developments in the domain of QA.

2.3 Related Work

There have been various surveys on QA Systems in the field of IR driven by the Text Retrieval Conference (TREC), the Cross Language Evaluation Forum (CLEF) and NII Test Collections for IR systems (NTCIR) campaigns, see [93] [4] [62].

In [4] [62], IR based QA systems are analysed based on three core components: (1) question analysis and classification, (2) information retrieval to extract the relevant documents with the answer and (3) answer extraction and ranking, which combine techniques from natural language processing (NLP), information retrieval (IR) and information extraction (IE), respectively [4]. Based on the overall approach in [62], QA systems are categorised into: Linguistic, Statistical and Pattern Matching.
In these surveys, only a few domain-specific QA approaches over structured KBs are briefly described. In the open domain QA systems analysed, KBs and taxonomies are used, however they are not used to find the answer, but rather are used as features in the QA process, where answers are extracted from text. For example, they are used to support the question type classification and predict the expected answer type or to semantically enrich the parse-trees to facilitate the question-answering matching, often combined with ML approaches (Support Vector Machine, Nearest Neighbors, Bayesian classifiers, Decision Tree, etc.) or approaches that learn text patterns from text.

Differently from these surveys, we deep dive into the techniques applied by QA systems to extract answers from KBs. In particular, we focused on answering open-domain queries over Linked Data that have been proposed since the introduction of the QALD in 2011. We will see that the QALD benchmark introduces different challenges for QA systems over KBs (detailed in section 2.14), for which a wide variety of novel or existent techniques and their combination have been applied. Moreover, we point to some techniques used by QA systems evaluated over WebQuestions and SimpleQuestions.
The existing QALD reports [101][28][158][157][159], surveys on the challenges faced by open domain QA systems over KBs [103][73] and the latest survey on the topic [87] gives short overviews for each QA system. They do not allow for a detailed comparison of the techniques used in each of the stages of the QA process. In this survey, the advantages and disadvantages of each technique applied by each of the 32 QA systems participating in any of the QALD campaigns is described.

Challange	Task	Dataset	Questions	Language
QALD-1	1	DBpedia 3.6	50 train / 50 test	en
	2	MusicBrainz	50 train / 50 test	en
QALD-2	1	DBpedia 3.7	100 train / 100 test	en
	2	MusicBrainz	100 train / 100 test	en
QALD-3	1	DBpedia 3.8	100 train / 99 test	en, de, es, it, fr, nl
	2	MusicBrainz	100 train / 99 test	en
QALD-4	1	DBpedia 3.9	200 train / 50 test	en, de, es, it, fr, nl, ro
	2	SIDER, Diseasome, Drugbank	25 train / 50 test	en
	3	DBpedia 3.9 with abstracts	25 train / 10 test	en
QALD-5	1	DBpedia 2014	300 train / 50 test	en, de, es, it, fr, nl, ro
	2	DBpedia 2014 with abstracts	50 train / 10 test	en
QALD-6	1	DBpedia 2015	350 train / 100 test	en, de, es, it, fr, nl, ro, fa
	2	DBpedia 2015 with abstracts	50 train / 25 test	en
	3	LinkedSpending	100 train / 50 test	en

Table 2.1: Overview of the QALD challenges launched so far.

2.4 Benchmarks for QA

Benchmarks for QA systems over KBs are a new development. In 2011, there was still no established benchmark that was used to compare different QA systems as described in [103]. Therefore, QA systems were tested on different KBs using small scale scenarios with ad-hoc tasks.

In the last number of years, different benchmarks for QA systems over KBs arised. The three most popular ones are WebQuestions [15], SimpleQuestion [21] and QALD[5] [101, 28, 158, 157, 159]. In fact QALD is not one benchmark but a series of evaluation campaigns for QA systems over KBs. Six challenges have been proposed up till now. An overview is given in Table 2.1. Each year one of the tasks included QA over DBpedia. In the following, we concentrate on these results. Moreover, to address the case where the requested information is distributed across many KBs, we consider QALD-4 Task 2. It offers 3 interlinked KBs, 25 training questions and 25 test questions that can be answered only by searching and combining information distributed across the 3 KBs. We compare these three benchmarks based on the dataset, the questions and the evaluation metrics.

2.4.1 Datasets

WebQuestions and SimpleQuestions contain questions that can be answered using Freebase[79]. The QALD challenges always included DBpedia[6] as an RDF dataset besides some others. The main difference between DBpedia and Freebase is how the information is structured. Freebase contains binary and also non-binary relationships. Non-binary relationships are stored using one type of reification that uses "mediator nodes". An example

[5] http://www.sc.cit-ec.uni-bielefeld.de/qald/

[6] http://www.dbpedia.org/

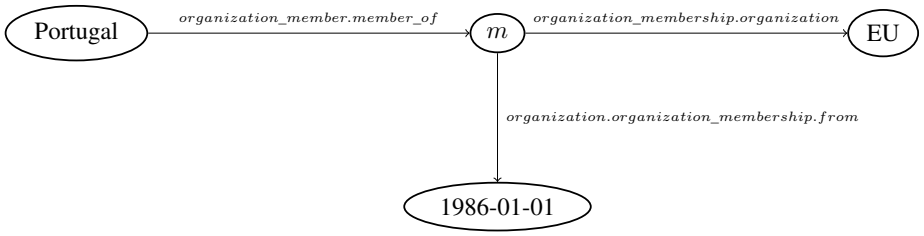

Figure 2.1: Freebase also contains non-binary relationships like "Portugal is a member of the European Union since 1986.". This information is stored in Freebase using one type of reification. It uses "mediator nodes" (in this example m). Note that in the dump, for each relationship, its inverse is also stored. This is not the case with DBpedia.

is given in Figure 2.1. This is not the case in DBpedia where only binary relations are stored. In particular temporal information is not present. Some of the works over WebQuestions like [21] and [89] converted non-binary relationships to binary relationships by dropping the "mediator node". [21] noted that this way still 86% of the questions in WebQuestions can be answered. Moreover, for each relationship, Freebase also stores its inverse, this is not the case for DBpedia since they are considered redundant. Regarding the size, the DBpedia endpoint contains nearly 400 million triples while the last Freebase dump contains 1.9 billion triples. Note that due to reification and due to the materialization of the inverse of each property, this is roughly the same amount of information. Moreover note that WebQuestions and SimpleQuestions are often tackeled by using only a subset of Freebase. This is denoted as FB2M and FB5M containing only 2 and 5 million entities, respectively.

2.4.2 Questions

WebQuestions contains 5810 questions which follow a very similar pattern due to the crowd-sourcing method used to construct the dataset. Around 97% of them can be answered using only one reified statement with potentially some constraints like type constraints or temporal constraints[7]. SimpleQuestion contains 108.442 questions that can be answered using one binary-relation.

The questions of QALD are prepared each year by the organizers of the challenge. They can be generally answered using up to three binary-relations and often need modifiers like ORDER BY and COUNT. Moreover, some of the questions are out of scope. The training set is rather small with about 50 to 250 questions. This results in few opportunities for supervised learning. Finally, while the SimpleQuestion and the QALD datasets are annotated with a SPARQL query for each question, the WebQuestions dataset is only annotated with the labels of the answers. There exists a dataset containing SPARQL queries for each of the question in WebQuestions, which is called WebQuestionsSP [180].

[7] https://www.microsoft.com/en-us/download/details.aspx?id=52763

2.4.3 Evaluation

For evaluation, three parameters are used: precision, recall, and F-measure. They are slightly the same across all benchmarks. Precision indicates how many of the answers are correct. For a question q it is computed as:

$$\text{Precision(q)} = \frac{\text{number of correct system answers for q}}{\text{number of system answers for q}}.$$

The second parameter is recall. For each question there is an expected set of correct answers, these are called the gold standard answers. Recall indicates how many of the returned answers are in the gold standard. It is computed as:

$$\text{Recall(q)} = \frac{\text{number of correct system answers for q}}{\text{number of gold standard answers for q}}.$$

We explain these parameters using the following example. Assume a user asks the question: "Which are the three primary colors?". The gold standard answers would be "green", "red" and "blue". If a system returns "green" and "blue", as the answers, then the precision is 1, since all answers are correct but the recall is only 2/3 since the answer "red" is missing.
The global precision and recall of a system can be computed into two ways, they are denoted as the micro and macro precision and recall, respectively. The micro precision and recall of a system are computed by taking the average of the precision and recall over all answered questions, i.e. non answered questions are not taken into account. The macro precision and recall are computed by taking the average of the precision and recall over all questions. The (micro or macro) F-measure is the weighted average between the (micro or macro) precision and recall and it is computed as follows:

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Note that the F-measure is near to zero if either precision or recall are near to zero and is equal to one only if both precision and recall are one. In this survey we report macro precision, recall and F-measure since these are the official metrics used by the benchmarks. Another parameter which is important for the user, is runtime. This is generally indicated as the average time taken by the QA system for answering the questions.

2.5 Selection of the QA systems

We briefly describe how the QA systems compared in this paper were selected. We consider the QA systems that either directly participated in the QALD challenges or that were

evaluated afterwards reusing the same evaluation set-up. To identify the latter systems, we searched in Google Scholar for all publications mentioning or citing the publications of the QALD challenges [101][28][158][157][159]. From among these, we took the publications referring to QA systems.
We excluded systems that were only able to answer questions formulated in a controlled natural language namely squall2sparql [69] and CANaLI [110]. Moreover, we exclude systems that propose a graphical user interface to guide users through the construction of a query. These include Sparklis [68] and Scalewelis [90].
Note that over the last years other types of QA systems have also emerged. Between these there is hybrid QA, i.e. QA using both free text and KBs. This problem was tackled by the QA systems HAWK [163] and YodaQA [12]. Another type of QA system is QA over statistical data expressed using the RDF Data Cube vocabulary. This problem was tackled by CubeQA [86] and QA^3 [7]. Since there are only a few works in these directions, we direct the readers directly to them.
The QA systems APAQ, AskDBpedia, KWGAnswer, MHE, NbFramework, PersianQA, RO_FII and UIQA that participated in QALD were excluded since no corresponding publication could be found.
Table 3.3 contains a list of the considered QA system with the scores they achieved on the QALD test sets.

2.6 The question answering process

To classify the techniques used in QA systems we divide the QA process into five tasks: question analysis, phrase mapping, disambiguation, query construction and querying distributed knowledge (see Figure 2.2).
In the following, we briefly describe the first four steps using a simple example. Assume the user inputs the question:

"What is the population of Europe?"

and that DBpedia is the queried KB.
In the question analysis task, we include all techniques that use purely syntactic features to extract information about a given question. These include, for example, determining the type of the question, it is a "What" question, identifying named entities, like "Europe", identifying relations and entities (subjects and objects), like the relation "is the population of", and their dependencies, like that there is a dependency between "is the population of" and "Europe".
In the phrase mapping task, one starts with a phrase s and tries to identify resources that with high probability correspond to s. For example, for the phrase "Europe" possible resources in

QA system	Lang	Total	Precision	Recall	F-measure	Runtime	Reference
QALD-1							
FREyA [38]	en	50	0.54	0.46	0.50	36 s	[101]
PowerAqua [100]	en	50	0.48	0.44	0.46	20 s	[101]
TBSL [156]∗	en	50	0.41	0.42	0.42	-	[156]
Treo [72]∗	en	50	-	-	-	-	-
QALD-2							
SemSeK [3]	en	100	0.35	0.38	0.37	-	[101]
BELA [166]∗	en	100	0.19	0.22	0.21	-	[166]
QAKiS [27]	en	100	0.14	0.13	0.13	-	[101]
QALD-3							
gAnswer [188]∗	en	100	0.40	0.40	0.40	≈ 1 s	[188]
RTV [76]	en	99	0.32	0.34	0.33	-	[28]
Intui2 [59]	en	99	0.32	0.32	0.32	-	[28]
SINA [132]∗	en	100	0.32	0.32	0.32	≈ 10-20 s	[132]
DEANNA [172]∗	en	100	0.21	0.21	0.21	≈ 1-50 s	[188]
SWIP [124]	en	99	0.16	0.17	0.17	-	[28]
Zhu et al. [187]∗	en	99	0.38	0.42	0.38	-	[187]
QALD-4							
Xser [170]	en	50	0.72	0.71	0.72	-	[158]
gAnswer [188]	en	50	0.37	0.37	0.37	0.973 s	[158]
CASIA [84]	en	50	0.32	0.40	0.36	-	[158]
Intui3 [58]	en	50	0.23	0.25	0.24	-	[158]
ISOFT [121]	en	50	0.21	0.26	0.23	-	[158]
Hakimov et al. [81]∗	en	50	0.52	0.13	0.21	-	[81]
QALD-4 Task 2							
GFMed [108]	en	25	0.99	1.0	0.99	-	[158]
SINA [132]∗†	en	25	0.95	0.90	0.92	≈ 4-120 s	[132]
POMELO [82]	en	25	0.87	0.82	0.85	≈ 2 s	[158]
Zhang et al. [186]∗	en	25	0.89	0.88	0.88	-	[186]
TR Discover [137]∗	en	25	0.34	0.80	0.48	-	[137]
QALD-5							
Xser [170]	en	50	0.74	0.72	0.73	-	[157]
QAnswer [130]	en	50	0.46	0.35	0.40	-	[157]
SemGraphQA [13]	en	50	0.31	0.32	0.31	-	[157]
YodaQA [12]	en	50	0.28	0.25	0.26	-	[157]
QALD-6							
UTQA [122]	en	100	0.82	0.69	0.75	-	[159]
UTQA [122]	es	100	0.76	0.62	0.68	-	[159]
UTQA [122]	fs	100	0.70	0.61	0.65	-	[159]
SemGraphQA [13]	en	100	0.70	0.25	0.37	-	[159]

† Evaluated on the training set

Table 2.2: This table summarizes the results obtained by the QA systems evaluated over QALD. We included the QALD tasks querying DBpedia and QALD-4 task 2 which addresses the problem of QA over interlinked KBs. We indicated with "∗" the systems that did not participate directly in the challenges, but were evaluated on the same benchmark afterwards. We indicate the average running time of a query for the systems where we found it. Even if the runtime evaluations were executed on different hardware, it still helps to give an idea about the scalability.

Figure 2.2: Tasks in the QA process

DBpedia are: dbr:Europe _(band)[8] (that refers to a Band called Europe), dbr: Europe (that refers to Europe as a continent) and dbr:Europe _(dinghy) (a particular type of boat).
In the disambiguation task, we include techniques that are used to determine which of the resources identified during the phrase mapping task are the right ones. In the above example, "Europe" cannot refer to a band or a boat since it does not make sense to speak about their population. Therefore, these two resources can be excluded.
The query construction task describes how to construct a SPARQL query that can be send to an endpoint. This part also covers the construction of queries that require special operators such as comparatives and superlatives. In the example above, the generated SPARQL query would be:

```
Select ?p where {
    dbr:Europe dbp:populationTotal ?p
}
```

In the distributed knowledge task, we include techniques that are used in the case when the information to answer the question must be retrieved from several KBs. Note that QA systems generally do not follow such a strict division in five tasks. However, based on the current state of the art (including systems not reviewed here because they were not evaluated for QALD), we have found that QA systems can typically be decomposed in this way. Note that the subdivision in tasks does not aim to present a general architecture for QA systems, but to classify and compare QA systems. Moreover, in general there is no preferred order to address the tasks, i.e. Xser [170] first performs disambiguation and then constructs the query while SEMPRE [15] proceeds the other way around.
In section 10, the tasks are mapped to the challenges identified in the state of the art for QA systems. The following five sections, describe the techniques that are used by QA systems to solve the different tasks. We focus on techniques that can be adapted to any KB, i.e. techniques that do not use some DBpedia/Wikipedia specific features.

2.7 Question analysis

In this step, the question of the user is analyzed based on purely syntactic features. QA systems use syntactic features to deduce, for example, the right segmentation of the question, determine which phrase corresponds to an instance (subject or object), property or class and the dependency between the different phrases. Note that some systems like Xser [170]

[8] @prefix dbr: <http://dbpedia.org/resource/>

WRB	VBD	DT	NNP	NNP	VBN	.
When	was	the	European	Union	founded	?

Figure 2.3: POS tags returned by the Stanford POS tagger `http://nlp.stanford.edu/software/tagger.shtml` for the question "When was the European Union founded?". For example DT stands for determiner, NNP for proper singular noun.

decide already in this step what part of the question corresponds to an instance, relation and class. Other systems like PowerAqua [100] use this step to find how the terms in the question relate to each other and make an hypothesis about the correspondence. The right one can be selected afterwards. We classify techniques for question analysis in techniques for recognizing named entities, techniques for segmenting the question using Part-of-speech (POS) tagging and techniques to identify dependencies using parsers.

2.7.1 Recognizing named entities

An important task in QA is to segment the question, i.e. identify contiguous spans of tokens that refer to a resource. As an example consider the question: "Who directed One Day in Europe?". The span of tokens "One Day in Europe" refers to a film. QA systems use different strategies to solve this problem.

Named Entity Recognition (NER) tools from NLP
One approach is to use NER tools used in natural language processing. Unfortunately, NER generally are adapted to a specific domain. It was observed in [84] that the Stanford NER tool[9] could recognize only 51.5% of the named entities in the QALD-3 training set.

N-gram strategy
A common strategy is to try to map n-grams (groups of n words) in the question to entities in the underlying KB. If at least one resource is found, then the corresponding n-gram is considered as a possible named entity (NE). This is used for example in SINA [132] and CASIA [84]. This has the advantage that each NE in the KB can be recognized. The disadvantage is that many possible candidates will appear and that the disambiguation will become computationally expensive.

Entity Linking (EL) Tools
Some engines use other tools to recognize NE. These are DBpedia Spotlight [34] and AIDA [182]. These tools do not only recognize NE, but also find the corresponding resources in the underlying KB, i.e. they perform entity linking. As a result, they perform many steps at once: identifying contiguous span of tokens that refer to some entity, find possible resources that can correspond to it and disambiguate between them.

[9] `http://nlp.stanford.edu/software/CRF-NER.shtml`

none	V-B	C-B	none	none	E-B	E-I	R-B	.
By	which	countries	was	the	European	Union	founded	?

Figure 2.4: Question annotated using the CoNLL IOB format. E-B indicates that the entity is beginning and E-I that it is continuing (this way phrases with more words are labeled).

2.7.2 Segmenting the question using POS tagging

POS tags are used mainly to identify which phrases correspond to instances (subjects or objects), to properties, to classes and which phrases do not contain relevant information. An example of a POS tagged question is given in Figure 2.3. Often nouns refer to instances and classes (like "European") and verbs to properties (like "founded" above). This does not always hold. For example in the question "Who is the director of Star Wars?" the noun "director" may refer to a property (e.g., `dbo:director`).
The general strategy using POS tags is to identify some reliable POS tags expressions to recognize entities, relations and classes. These expressions can then be easily identified using regular expressions over the POS tags.

Handmade rules
Some engines rely on hand written regular expressions. This is the case for PowerAqua ([100][102]), Treo ([72]) and DEANNA ([172]). PowerAqua uses regular expressions, based on the GATE NLP tool ([31]), to identify the question type and to group the identified expressions into triples of phrases. To do this PowerAqua relies on an extensive list of question templates, i.e., depending on the type of the question and its structure, the phrases are mapped to triples.

Learning rules
Instead of writing handmade rules, it was proposed to use machine learning algorithms to detect them [170]. The idea is to create a training set where questions are tagged with entities (E), relations (R), classes (C), variables (V) and "none" tags using the CoNLL IOB format (inside-outside-beginning) as in Figure 2.4. Once a training corpus is constructed it is used to build a phrase tagger. The features used by the phrase tagger built in Xser ([170]) are: POS-tags, NER-tags and the words of the question itself. This way one can construct a tagger that is able to identify entities, relations, classes and variables in a given question and learn automatically rules to do that starting from the training data. A similar approach was followed also by UTQA ([122]). The disadvantage is that a training corpus must be created. The advantage is that no handmade rules have to be found.

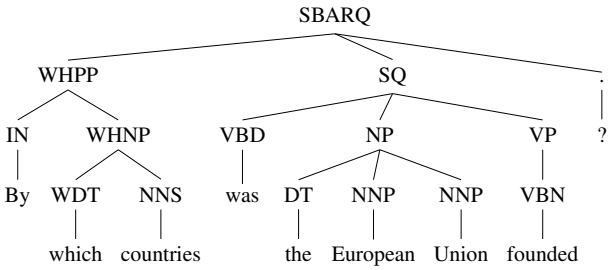

Figure 2.5: Parsing tree of the question "By which countries was the European Union founded?" returned by the Stanford Parser (`http://nlp.stanford.edu/software/lex-parser.shtml`). At the bottom of the tree are the words in the question and the corresponding POS tags. The tags above denote phrasal categories like noun phrase (NP).

2.7.3 Identifying dependencies using parsers

POS tagging information does not allow to identify the relation between the different chunks in a question. This is the reason for using parsers.
A parser is based on a formal grammar. A formal grammar consists of symbols (words) and production rules that are used to combine the symbols. Given a sentence, a parser computes the combination of production rules that generate the sentence according to the underlying grammar. Examples will be given in the next section. Note that there are different types of formal grammars. The systems participating to the QALD challenges use parsers based on different types of grammars. We present them below and show their advantages.

Parsers based on phrase structure grammars

Some parsers rely on phrase structure grammars. The idea of phrase structure grammars is to break down a sentence into its constituent parts. An example is given in figure 2.5.

These types of trees are used similarly to POS tags, i.e. one tries to find some graph patterns that map to instances, properties or classes with high confidence. Differently from POS tags, one can deduce the dependencies between entities, relations and classes. Parsers of such type are used in the QA systems Intui2 ([59]), Intui3 ([58]) and Freya ([38]).

Parsers based on dependency grammars

The idea behind dependency grammars is that the words in a sentence depend on each other, i.e. a word "A" depends on a word "B". "B" is called the head (or governor) and "A" is called the dependent. Moreover, parsers also generally indicate the type of relation between "A" and "B".

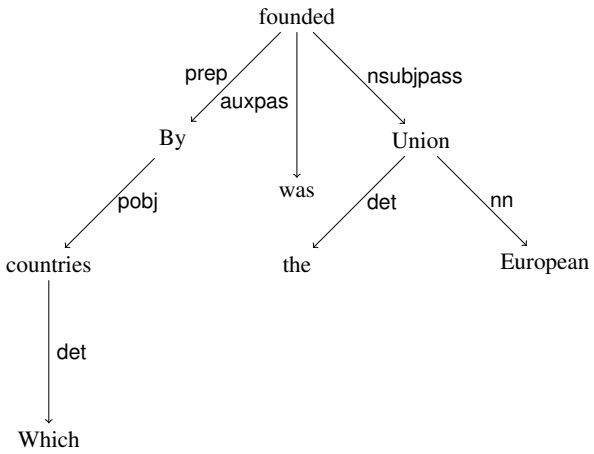

Figure 2.6: Result of the Stanford dependency parser for the questions "By which countries was the European Union founded?".

Standford dependencies, Universal dependencies

Figure 2.6 shows the result of the Stanford dependency parser for the questions "By which countries was the European Union founded?".

For example, the tree indicates that "founded" is the head of "By", or that "By" is the dependent of "founded". The Stanford parser also returns the type of the dependency, in the example above "prep" indicates that "by" is a preposition of "founded". The advantages of using dependency trees can be seen by looking at the example above. In the original question, the words in the relational expression "founded by" are not subsequent which makes them difficult to extract. This is not the case in the dependency tree where "by" is connected to "founded". This is the main reason why dependency representations are used for relation extraction. Moreover, the parsing tree contains grammatical relations like nsubj (nominal subject), nsubjpass (nominal passive subject) and others which can be used to identify the relations between different phrases in the question.

A method to extract relations is to search the dependency tree for the biggest connected subtree that can be mapped to a property. In the example above this would be the subtree consisting of the nodes "founded" and "by". The arguments associated with a relation are then extracted searching around the subtree. This is for example used by gAnswer [188]. Another strategy is to first identify named entities in the dependency tree and choose the shortest path between them as a relation. In the example above these are the entities "country" and "European Union". This is used in the pattern library PATTY [116].

Phrase dependencies and DAGs

While the Stanford dependency parser considers dependencies between words, the system Xser [170] considers dependencies on a phrase level, namely between phrases referring to

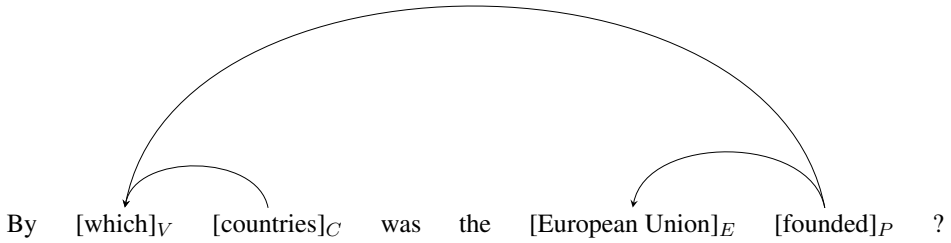

Figure 2.7: Parse tree for the questions "By which countries was the European Union founded?" using the Stanford dependency parser

entities, relations, classes and variables as in 2.7.2. The following dependency relation is considered: relations are the head of the two corresponding arguments and classes are the head of the one corresponding argument (arguments are either entities, variables or other relations and classes). As an example, look at the dependency graph of the question "By which countries was the European Union founded?" in figure 2.7. The phrase "founded" is the head of the phrase "which" and the entity "European Union". The class "country" is the head of the variable "which". Note that this is not a tree, but a direct acyclic graph (DAG). To compute the DAG, Xser uses a SHIFT-REDUCED parser that is trained on a manually annotated dataset.
The parser uses as features the POS tags of the words, the type of the phrase and the phrase itself. The advantage is that the parser learns automatically which is the relation between the phrases. The disadvantage is that one needs a manually annotated corpus.

2.7.4 Summary and research challenges for question analysis

Table 2.3 shows which strategy is used by the different engines for question analysis. We put a question mark if it is not clear from the description of the publication which technique is used. The research challenges that have been identified are the following: the identification of question types, the multilinguality and the identification of aggregation, comparison and negation operators. Section 2.14 will give a transversal view on research challenges through the surveys where they are discussed.

2.8 Phrase mapping

This step starts with a phrase (one or more words) *s* and tries to find, in the underlying KB, a set of resources which correspond to *s* with high probability. Note that *s* could correspond to an instance, a property, or a class. As an example, the phrase *EU* could correspond to the DBpedia resources `dbr:European_Union` but also to `dbr:University_of_Edinburgh` or `dbr:Execution_unit` (a part of a CPU). In the following, we want to provide an overview of the techniques used by QA systems to deal with these problems.

	NER	NE n-gram startegy	EL tools	POS hand-made	POS learned	Parser structural grammar	Dependency parser	Phrase dependencies and DAG	Reference
BELA				x					[166]
CASIA		x			x		x		[84]
DEANNA		x		x			x		[172]
FREyA						x			[38]
gAnswer							x		[188]
GFMed									[108]
Hakimov et al.								x	[81]
Intui2						x			[59]
Intui3	x			x					[58]
ISOFT			x	x			x		[121]
POMELO		x							[82]
PowerAqua				x					[100]
QAKiS	x	x							[27]
QAnswer		x					x		[130]
RTV	?						x		[76]
SemGraphQA		x					x		[13]
SemSeK		x				x	x		[3]
SINA		x							[132]
SWIP	?						x		[124]
TBSL				x					[156]
TR Discover									[137]
Treo				x			x		[72]
UTQA					x				[122]
Xser	?				x			x	[170]
Zhang et al.		x							[186]
Zhu et al.						x			[187]

Table 2.3: Each line of this table corresponds to a QA system. The columns indicate which of the techniques presented in the question analysis task (left table) and in the phrase mapping task (right table) is used. The columns refer to the subsections of section 2.7 and 2.8. We put a question mark if it is not clear from the publication which technique was used.

2.8.1 Knowledge base labels

RDF Schema introduces the property `rdfs:label`[10] to provide a human-readable version of a resource's name. Moreover, multilingual labels are supported using language tagging. For example, the following triples appear in DBpedia:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>;
PREFIX dbr: <http://dbpedia.org/resource/>;
dbr:European_Union rdfs:label "European Union"@en .
dbr:European_Union rdfs:label "Europäische Union"@de .
dbr:European_Union rdfs:label "Union européenne"@fr .
```

saying that the human-readable version of `dbr:European_Union` in french is `Union européenne`. By using the RDF Schema convention the resources corresponding to a phrase s can be found by searching all resources r in the KB whose label $label(r)$ is equal to or contains s. This strategy is used by all QA systems.
To answer a question in reasonable time some type of index is needed to search through all the labels. There are two common choices. The first is to use a triple-store build-in index like in Virtuoso[11]. This is for example used by SINA [132]. The second is to use Lucene. Engines that use Lucene are PowerAqua [100] and SemSeK [3]. In both cases one can search very fast through the labels. The advantage of a build-in triple-store index is that it is kept synchronized with the corresponding KB.
Several problems can arise that can be grouped in two categories. The first category contains problems that arise because the phrase s, as a string, is similar to $label(r)$ but not a strict subset. For example, s could be misspelled, the order of the words in s and $label(r)$ could be different or singular plural variations (like city/cites).
The second category contains problems that arise because s and $label(r)$ are completely different as strings but similar from a semantic point of view. This can happen for example when the questions contains abbreviations like *EU* for `dbr:European_Union`, nicknames like *Mutti* for `dbr:Angela_Merkel` or relational phrases like *is married to* corresponding to the DBpedia property `dbo:spouse`. All these problems arise because the vocabulary used in the question is different from the vocabulary used in the KB. The term "lexical gap" is used to refer to these problems.

2.8.2 Dealing with string similarity

One possibility to deal with misspelling is to measure the distance between the phrase s and the labels of the different resources of the KB. There are a lot of different distance metrics that can be used. The most common one are the Levenshtein distance and the Jaccard distance. Lucene for example offers fuzzy searches which return words similar to the given

[10] @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
[11] `https://github.com/openlink/virtuoso-opensource`

one based on the Levenshtein Distance or another edit distance. Moreover Lucene offers also stemming which allows to map "cities" to "city" since the stem is in both cases "citi".

2.8.3 Dealing with semantic similarity

In this section we want to consider the phrase mapping problem when s and $label(r)$ have only a semantic relation.

Databases with lexicalizations

One frequent approach is to use a lexical database. Examples for such databases are WordNet and Wiktionary. The strategy is to first expand s by synonyms $s_1, ..., s_n$ found in the lexical database. Then $s_1, ..., s_n$ are used to retrieve the resources instead of s. For example, WordNet returns for *EU* the synonyms *European Union, European Community, EC, European Economic Community, EEC, Common Market, Europe* which are then used to retrieve the corresponding resources in the KB. More precisely WordNet groups words in sets of roughly synonymous words called synsets. For example, the word *EU* is contained in two synsets. The first was described above and refers to *"an international organization of European countries formed after World War II to reduce trade barriers and increase cooperation among its members"*. The second synset contains *europium, Eu, atomic number 63* and refers to *"a bivalent and trivalent metallic element of the rare earth group"*.
WordNet is used for example in TBSL [156], PowerAqua [100] and SemSek [3]. The main advantage is that more mappings can be resolved improving recall. The disadvantage is that the expansion of s increases the number of possible corresponding resources. This can affect precision and has the side effect that the disambiguation process becomes computationally more heavy. Moreover lexical databases are often domain-independent and cannot help for domain-specific mappings.

Redirects

Another way to collect new labels of a resource is to follow, if present, the owl:sameAs links. The labels in the connected KB can be used then in the same way as in the original KB. Moreover, the anchor texts of links, mapping to an KB resource, can also be used. The labels generated by the anchor texts of the Wikipedia links are contained in DBpedia lexicalizations[12].

[12] http://wiki.dbpedia.org/lexicalizations

A database with relational lexcalizations (PATTY)

PATTY [116] is a Database with relational lexicalizations. The main objective is to collect natural language expressions and group them if they refer to the same relation. These groups are called "pattern synsets" in analogy of Wordnet. Moreover, the pattern synsets are organized in a taxonomy like synsets in WordNet. An example of such a pattern synset is *(is album, [[num]] album by, was album, [[det]] album with, [[adj]] album by)* where *[[num]]* corresponds to a cardinal number, *[[det]]* corresponds to a determiner and *[[adj]]* corresponds to an adjective (for this reason they are called "patterns"). Examples where different phrases express the same relation are: "Beatles*'s album* Revolver ...." , "Revolver is a *1966 album by* the Beatles" , "Revolver is *an album with* the Beatles", and "Revolver is the *seventh album by* the Beatles".
PATTY can be used as standard database with lexicalizations as in 3.6.1 and shares their advantages and disadvantages. PATTY is for example used by Xser [170].

Finding mappings using extracted knowledge

There are different tools that extract binary relations expressed in natural language from text corpora. An example of such a binary relation is *("Angela Merkel", "is married to", "Joachim Sauer")*. Note that it is a relation expressed in natural language so the subject, object and predicate are not mapped to any KB. Tools that can extract this type of triples are for example ReVerb [64], TEXTRUNNER [177], WOE [169] and PATTY [116]. An approach that use this data to find natural language representations of properties in a KB was described in [15]. It assumes that the subject and object of the binary relation are mapped to instances in an underlying KB. First the relational phrase is normalized to an expression rel. In a second step the classes of the subject and object are added obtaining $rel[class1, class2]$, i.e. in the example above *married[Person, Person]*. Then all entity pairs that are connected in the text by the relation rel and that matches the classes are computed. Denote this set as $Sup(rel[class1, class2])$. Moreover, for all properties in the KB the set of connected entity pairs are computed. For a property p we denote this set as $Sup(p)$. The relation $rel[class1, class2]$ is then aligned to the property p if the domain and range of both agree and $Sup(rel[class1, class2]) \cap Sup(p) \neq \varnothing$. This technique was used by CASIA [84]. The main disadvantage here is that the data can be very noisy. A similar approach is described in gAnswer [188].

Finding mappings using large texts

There are two methods that use large texts to find natural language representation for phrases. The first is the core of the BOA framework [75] and M-Atoll [165]. Both take as an input a

property *p* contained in a KB and try to extract natural language expressions for *p* from a large text corpus. To do that both extract from the KB the subject-object pairs *(x,y)* that are connected by the property p. Then the text corpus is scanned and all sentences are retrieved that contain both $label(x)$ and $label(y)$). At the end the segments of text between $label(x)$ and $label(y)$, or $label(y)$ and $label(x)$ are extracted. The idea is that these text segments are a natural language representation of the property p. These text segments are stored together with the range and domain of *p* and ranked such that the top ranked patterns are a more likely natural language representation of *p*. Using this data, a relational phrase r can be mapped to a property p by searching a similar text fragment in the BOA framework. This technique is for example used in TBSL [156]. Moreover Qakis [27] uses the pattern library WikiFrameworks [107] that was constructed in a very similar way. Also QAnswer [130] uses this approach, but the relational phrase is extracted using dependency trees. The big advantage of this approach is that the extracted relational phrases are directly mapped to the corresponding properties. The relational expressions found are tightly connected to the KB, differently to PATTY where the relations are more KB-independent.

Another approach that uses large texts is based on distributional semantics. The idea behind distributional semantics is that if two words are similar then they appear in the same context. An n-dimensional vector v_i is associated with each word w_i. The vectors are created such that words that appear in the same context have similar vectors. From among the QA systems participating in QALD, two tools that are based on distributional semantics are: word2vec[13] and Explicit Semantic Analysis (ESA)[14]. In both cases the vectors that are associated with the words have the property that the cosine similarity of a given pair of words is small if the words are similar. In the case of word2vec, the experimental results showed that the closest vectors to the vector of France *vec(France)* are the vectors *vec(Spain), vec(Belgium), vec(Netherlands), vec(Italy)*. Moreover, the vector *vec(queen)* is very near to the vector *vec(king)-vec(man)+vec(woman)*. In this sense, the semantics of the words are reflected into their associated vectors. The general strategy in the phrase mapping task is the following. Let us assume that there is a phrase *s* and a set of possible candidates $\{x_1, ..., x_n\}$ which can be instances, relations or classes. Then the vector representation v_0 of *s* and $v_1, ..., v_n$ of the lexicalizations of $x_1, ..., x_n$ are retrieved. Since the similarity of the words is reflected in the similarity of the vectors, the best candidates from $\{x_1, ..., x_n\}$ are the ones whose vectors are more similar to v_0. For example if the right semantics are captured then the vector $vec(spouse)$ should be similar to the vector of $vec(married)$. The main advantage is that this technique helps to close the lexical gap. However, the disadvantages are that it can introduce noise and that it is generally a quite expensive operation. For this reason the possible candidate set is generally not the entire set of instances, relations and classes, but only a subset of them. CASIA [84] for example uses this technique only for classes and uses word2vec as the tool. Treo uses a strategy such that it can assume that a phrase *s* corresponds to a property or class of a particular instance. In this case the candidate set contains only 10-100 elements. Here ESA is used as the tool.

[13] https://code.google.com/p/word2vec/

[14] http://code.google.com/p/dkpro-similarity-asl/

2.8.4 Wikipedia specific approaches

Some engines use other tools for the phrase mapping task namely: DBpedia lookup[15] and the Wikipedia Miner Tool[16]. The first is for example used by gAnswer [188] the second by Xser [170] and Zhu et al. [187].

2.8.5 Summary and research challenges for the phrase mapping task

Table 2.4 gives an overview showing which technique is used by which engine for phrase mapping. The important point in this step is that one has to find a balance between selecting as few candidate resources as possible to improve precision and time performance, and select enough candidates so that the relevant one is also selected to improve recall. The research challenges identified in the phrase mapping step are: filling the lexical/vocabulary gap and multilinguality. The latter applies if the vocabulary in the user query and the KB vocabulary are expressed (lexicalized) in different languages. See section 2.14 for a transversal view on the research challenges.

2.9 Disambiguation

Two ambiguity problems can arise. The first is that from the question analysis step the segmentation and the dependencies between the segments are ambiguous. For example, in the question "Give me all european countries." the segmentation can group or not the expression "european countries" leading to two possibilities. The second is that the phrase mapping step returns multiple possible resources for one phrase. In the example above "european" could map to different meanings of the word "Europe". This section explains how question answering systems deal with these ambiguities.

2.9.1 Local Disambiguation

Due to ambiguities, QA systems generate many possible interpretations. To rank them mainly two features are used. The first is the (string or semantic) similarity of the label of the resource and the corresponding phrase. The second is a type consistency check between the properties and their arguments. The first feature is used to rank the possible interpretations, the second to exclude some. These features are "local" in a sense that only the consistency between the two resources that are directly related is checked. The advantage is that "local" disambiguation is very easy and fast. Moreover, it is often very powerful. A main disadvantage is that actual KBs often do not contain domain and range information of

[15] https://github.com/dbpedia/lookup
[16] http://wikipedia-miner.cms.waikato.ac.nz

	Knowledge base labels	String similarity	Lucene index or similar	WordNet/Wiktionary	Redirects	PATTY	Using extracted knowledge	BOA or similar	Distributional Semantics	Wikipedia specific approaches	Reference
BELA	x	x	x	x	x				x		[166]
CASIA	x				x		x		x		[84]
DEANNA	x										[172]
FREyA	x	x		x							[38]
gAnswer	x				x		x			x	[188]
GFMed	x				x						[108]
Hakimov et al.	x							x			[81]
Intui2	x										[59]
Intui3	x			x	x					x	[58]
ISOFT	x		x			x			x		[121]
POMELO	x										[82]
PowerAqua	x		x	x	x						[100]
QAKiS	x							x			[27]
QAnswer	x	x	x	x	x			x			[130]
RTV	x		x						x		[76]
SemGraphQA	x		x	x	x						[13]
SemSeK	x		x	x	x				x		[3]
SINA	x										[132]
SWIP	x	x									[124]
TBSL	x	x	x	x				x			[156]
TR Discover	x										[137]
Treo	x		x						x		[72]
UTQA	x	x			x				x		[122]
Xser	x					x				x	[170]
Zhang et al.	x	x									[186]
Zhu et al.	x								x	x	[187]

Table 2.4: Each line of this table corresponds to a QA system. The columns indicate which of the techniques presented in the phrase mapping task is used. The columns refer to the subsections of section 2.8.

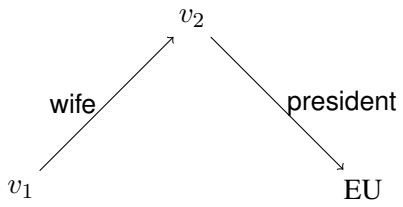

Figure 2.8: A graph G which is semantically equivalent to the question "Who is the wife of the president of the EU?". The ambiguity of "EU" in the phrase carries over to an ambiguity of the corresponding vertex of G.

a property so that the type consistency cannot be done. Consider for example the question "Who is the director of The Lord of the Rings?". In this case "The Lord of the Rings" is clearly referring to the film and not to the book. If the property corresponding to director has no domain/range information then this strategy would not allow to decide if "The Lord Of the Rings" is a book or a film. The interpretation as a book can only be excluded later by querying the KB. This form of disambiguation is used by all systems.

2.9.2 Graph search

The QA systems gAnswer [188], PowerAqua [100], SemSek [3] and Treo [72] use the graph structure of the KB to resolve the ambiguity although they follow two different strategies. We explain them using the following question as an example: "Who is the wife of the president of the EU?".

The QA system gAnswer [188] assumes that in the question analysis step the intention of a question q can be translated into a graph G. See Figure 2.8 for the question above. This contains a node for each variable, entity and class in the question, and an edge for each relation. Moreover, it is semantically equivalent to q. The ambiguity that arises in the phrase mapping step carries over to an ambiguity of vertices and edges of G. The idea of gAnswer is to disambiguate the vertices and edges of G by searching into the KB a subgraph isomorphic to G such that the corresponding vertices correspond to the segments of q with high probability. This is achieved by assigning a score to each possible match, which is proportional to the distance between the labels of the resources and the segments of the question. The top-k matches are retrieved. In a similar fashion, PowerAqua explores the candidate properties, however it uses an iterative approach to balance precision and recall, selecting first the most likely mappings and interpretations based on the question analysis, and re-iterating until an answer is found or all the solution space has been analysed. It assigns a score to each of the queries based on the score of the selected matches and if the matches are directly related or not (semantic distance).

The QA systems SemSek and Treo solve the ambiguity from another perspective. In this case, only the instances identified in the question analysis step are expanded in the phrase mapping step leading to ambiguity. The relational phrases are not expanded. For the concrete example "EU" would be identified and some candidate instances would be generated. The relational phrases "president of" and "wife of" are not expanded. Instead a graph search is started from

the instances and all the properties attached to them are compared to the relational phrases detected in the question. In this case, all properties attached to the different interpretations of "EU" are compared. If no relation fits then the expanded instance is excluded.
Note that while the second approach has higher recall since all attached properties are explored, though, the first has probably higher precision.
The approach used in SemSek, Treo and gAnswer assume that one can deduce all relational phrases from the question. However, they can also be implicit. PowerAqua, is able to find a subgraph to translate the user query, even if not all entities in the query are matched. For all of them, the performance will decrease if there is too much ambiguity, i.e. if there are too many candidate matches to analyse.

2.9.3 Hidden Markov Model (HMM)

Hidden Markov Models (HMM) are used by SINA [132] and RTV [76] to address the ambiguity that arises in the phrase mapping phase. The idea behind this strategy is sketched using an example. Assume the question is: "By which countries was the EU founded?". In a Hidden Markov Model (HMM) one has two stochastic processes $(X_t)_{t\in\mathbb{N}}$ and $(Y_t)_{t\in\mathbb{N}}$ where only the last one is observed. The possible values of the random variables X_t are called hidden states whereas the possible values of the random variables Y_t are called observed states. For the example above, the set of observed states is *{"countries", "EU", "founded"}*, i.e. the set of segments of the question that have an associated resource. The set of hidden states is *{dbo:Country, dbr:Euro, dbr:European_Union, dbr:Europium, dbp: founded, dbp:establishedEvent}*, i.e. the set of possible resources associated with the segments.
In a Hidden Markov Model the following dependency between the random variables are assumed:

- $P(X_t = x_t|X_0 = x_0; ...; X_{t-1} = x_{t-1}; Y_0 = y_0; ...; Y_t = y_t) = P(X_t = x_t|X_{t-1} = x_{t-1})$, i.e. the value of a variable X_t only depends from the value of X_{t-1} which means that $(X_t)_{t\in\mathbb{N}}$ is a Markov Chain;

- $P(Y_t = y_t|X_0 = x_0; ...; X_t = x_t; Y_0 = y_0; ...; Y_{t-1} = y_{t-1}) = P(Y_t = y_t|X_t = x_t)$, i.e. that the value of Y_t depends only from X_t.

If one indicates the conditional dependencies with an arrow one gets the diagram in figure 2.9.

In the context of disambiguation this means that the appearance of a resource at time t depends only on the appearance of another resource at time $t-1$ and that the segments appear with some probability given a resource. The disambiguation process is reduced to the case of finding the most likely sequence of hidden states (the resources) given the sequence of observed states (the segments). This is a standard problem that can be solved by the

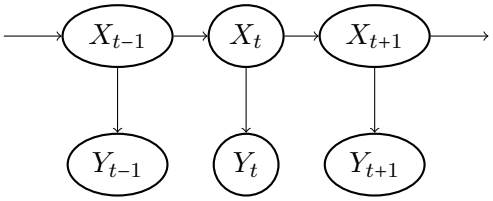

Figure 2.9: Conditional dependencies in a Hidden Markow Model

Viterbi Algorithm.
To complete the modeling one needs to indicate three more parameters:

- the initial probability, i.e. $P(X_0 = x)$ for $x \in X$;
- the transition probability, i.e. $P(X_t = x_1 | X_{t-1} = x_2)$ for $x_1, x_2 \in X$;
- the emission probability, i.e. $P(Y_t = y | X_t = x)$ for $x \in X$ and $y \in Y$.

These can be bootstrapped differently. In SINA the emission probability is set according to a string similarity measure between the label of the resource and the segment. In RTV the emission probabilities are estimated using word embeddings. In SINA the initial probabilities and the transition probabilities are estimated based on the distance of the resources in the KB and their popularity. Retrieving the distance between all resources is computationally expensive making this approach slow. In RTV the initial and transition probabilities are set to be uniform for all resources making the estimation fast but more inaccurate.
An advantage of this technique is that it is not necessary to know the dependency between the different resources but only a set of possible resources.

2.9.4 Integer Linear Program (ILP)

In the QA system DEANNA [172] it was proposed to set up an Integer Linear Program (ILP), which is an optimisation tool, to solve the disambiguation task. This technique addresses the ambiguity of the phrase mapping phase and some ambiguity that arises during the segmentation.
The ILP uses decision variables to model the problem, of finding the best answer, as an optimisation problem. In particular, it uses boolean variables to indicate if a segment of the question is chosen or not, if a resource corresponding to a segment is chosen or whether a segment corresponds to a property or an instance. The constraints include conditions such that the chosen segments do not overlap, such that if a segment is chosen then one corresponding resource must be chosen and so on. The optimization function includes three terms. The first increases if the label of a resource is similar to the corresponding segment. The second increases if two selected resources often occur in the same context. The third tries to maximize the number of selected segments. Note that the second term makes the

disambiguation process work. Thus, after solving the ILP the optimal values obtained point to the optimal answer, which is returned as the answer to the question.
The main disadvantage is that some dependencies between the segments have to be computed in the question analysis phase.

2.9.5 Markov Logic Network

The question answering system CASIA [84] uses a Markov Logic Network (MLN) for the disambiguation task. MLN is used to learn a model for choosing the right segmentation, for mapping phrases to resources, and grouping resources into a graph. The idea is to define some constraints using first-order logic formulas. MLN allows to consider some of them as hard constraints that must be fulfilled and others as soft constraints, i.e. if they are not satisfied a penalty is applied. In this case both the ambiguities that arise in the question analysis and phrase mapping stage are resolved.
Examples of hard constraints are: if a phrase of the question is chosen then one of the corresponding resources must be chosen, or that the chosen phrases cannot overlap. Examples for soft constraints are: if a phrase has a particular POS tag then it is mapped to a relation, the resource whose label is most similar to the corresponding phrase in the question must be chosen. The hard constraints of a MLN have a similar behavior as the constraints in a ILP while the soft constraints allow more flexibility. The penalty for the soft constraints are learned in a training phase.
The advantage of a MLN is that they allow more flexibility than an ILP in choosing the constraints. However, a training phase is needed.

2.9.6 Structured perceptron

The engine Xser [170] uses a structured perceptron to solve the disambiguation task. The idea is to consider features during disambiguation such as: the similarity of a phrase and the corresponding resource, the popularity of a label for a resource, the compatibility of the range and domain of a property with the types of the arguments and the number of phrases in the question that are in the same domain. In a training phase for each of the features f a weight w is computed such that the expected configuration z fulfills:

$$z = argmax_{y \in Y(x)} w \cdot f(x, y)$$

where x is the question and $Y(x)$ is the set of possible configurations of the resources and dependency relations. The configuration which maximizes the expression above is chosen. In this approach, the ambiguity that arises in the phrase mapping phase is resolved. However, a training phase is needed.

2.9.7 User feedback

There exists situations in which the QA engine cannot do the disambiguation automatically. This can happen because the disambiguation technique used by the engine does not suffice or because the question is really ambiguous. Therefore, some systems ask the user to resolve the ambiguity by choosing between some proposed resources. A system that relies heavily on user feedback for the disambiguation is Freya [37].

2.9.8 Summary and research challenges for disambiguation

Table 2.5 gives an overview of the different techniques used by the QA systems for disambiguation. Note that the systems GFMed, POMELO, TRDiscover, Zhang et al. were evaluated on the QALD-4 task 2 and there the disambiguation problem does not really arise since the three inter-linked KBs are too small. Beyond the disambiguation techniques, the research challenges in this domain are the lexical ambiguity of the matches, the use of vague prepositions or verbs (have/be) instead of explicit relationships (for example: movies of Pedro Almodovar) that can convey different intepretations (in the example: directed, produced, starring, etc.). These challenges are listed in section 2.14 where they are discussed through the previous surveys on KB-based QA systems.

2.10 Query construction

In this section, we describe how each QA system construct a SPARQL query. A problem arises during the query construction, the so called "semantic gap". Assume for example that a user asks the question: "Which countries are in the European Union?". One would probably assume that in the KB there are triples like:

```
dbr:Greece dbp:member dbr:European_Union .
dbr:France dbp:member dbr:European_Union .
```

But this is not the case, in DBpedia the requested information is encoded as:

```
dbr:Greece dct:subject dbc:Member_states_of_the_European_Union .
dbr:France dct:subject dbc:Member_states_of_the_European_Union .
```

So instead of a property "dbp:member" DBpedia uses the class "dbc:Member_states_of_ the_European _Union" to encode the information. The "semantic gap" refers to the problem that the KB encodes an information differently from what one could deduce from the question. This shows that in general it is impossible to deduce the form of the SPARQL query knowing only the question. Therefore, we classify the approaches for the query

	Local disambiguation	Graph search	HMM	LIP	MLN	Structured perceptron	User feedback	Reference
BELA	x							[166]
CASIA	x				x			[84]
DEANNA	x			x				[172]
FREyA	x						x	[38]
gAnswer	x	x						[188]
GFMed	x							[108]
Hakimov et al.	x							[81]
Intui2	x							[59]
Intui3	x							[58]
ISOFT	x							[121]
POMELO	x							[82]
PowerAqua	x	x						[100]
QAKiS	x							[27]
QAnswer	x							[130]
RTV	x		x					[76]
SemGraphQA	x							[13]
SemSeK	x	x						[3]
SINA	x		x					[132]
SWIP	x						x	[124]
TBSL	x							[156]
Treo	x	x						[72]
TR Discover	x							[137]
UTQA	?							[122]
Xser	x					x		[170]
Zhang et al.	x							[186]
Zhu et al.	x							[187]

Table 2.5: Each line of this table corresponds to a QA system. The columns indicate which of the techniques presented in the disambiguation task are used. The columns refer to the subsections of section 2.9. We put a question mark if it is not clear from the publication which technique was used.

construction based on how the SPARQL query form is deduced. We distinguish between approaches where the SPARQL query form is based on templates, approaches where it is deduced from the question analysis phase, where it is deduced using machine learning techniques or where it is deduced using only semantic information. The last subsection describes the approach of SemSek and Treo that do not generate a SPARQL query.

2.10.1 Query construction using templates

Some engines use templates to generate the SPARQL query, i.e. a set of predefined queries with some slots that have to be filled. The system QAKiS [27] restricts to select queries with only one triple. The system ISOFT [121] uses a small set of templates to generate SPARQL queries: these include ASK queries over one triple, some simple SELECT queries with one or two triples and templates that use a COUNT, ORDER BY or FILTER expression containing only one triple. Also, PowerAqua [100] assumes that the input question can be reduced to one or two linguistic triples (not more than two predicates) following a set of templates, then each linguistic triple is semantically matched into one or more KB triples that can be combined into a SELECT query. After some graph-based disambiguation the SPARQL query is constructed. The disadvantage is that not all questions can be treated using templates.

2.10.2 Query construction guided by information from the question analysis

Most of the systems start with the information obtained in the question analysis part and deduce from it the form of the SPARQL query.
Freya and Intui3 [58] start from the segmentation of the question. In the phrase mapping phase some segments have an associated resource. These resources are then combined into triples respecting the order of the segments in the question. If necessary some additional variables between the identified segments are added, for example if one relation is followed by another relation.
DEANNA [171] selects some triples phrase candidates in the question analysis phase using regular expressions over POS tags. These are mapped to resources in the phrase mapping phase. In the disambiguation phase the best phrase candidates and the best phrase mappings are chosen using a ILP. This returns a set of triples that is then used to construct a SELECT query.
The QA systems gAnswer, QAnswer, RTV and SemGraphQA start with a dependency tree. In gAnswer [188] first the relations and the associated arguments are deduced. A graph G is constructed which has an edge for each relation and a vertex for each argument. The graph G reflects the structure of the final SPARQL query. The relations and arguments are mapped to possible resources. The right resources are identified using the sub-isomorphism strategy described in section 2.9.2. Then the SPARQL query is constructed.

QAnswer [130] first scans the dependency tree to find subgraphs of tokens that correspond to some resources. This way many graphs with associate resources are created. Then a local disambiguation is performed. The top ranked graph is chosen and the SPARQL query is constructed from this graph.
RTV [76] uses the dependency graph to construct an ordered list of alternating properties and non properties. The corresponding resources are searched and disambiguated using a HMM. From this sequence the SPARQL query is generated.
Xser [170] uses three different machine learning algorithms. The first and the second are claimed to be KB independent. The first is used to determine the segments of the question corresponding to variables, properties, instances, and classes. The second is used to find the dependencies between the phrases as described in section 2.7.3. The third is used in the disambiguation phase, which is described in section 2.9.6 and is KB dependent. Since the first two algorithms are claimed to be KB independent this approach also constructs the form of the SPARQL by analyzing the question.
All these approaches share the same disadvantage. All of them make the implicit assumption that it is possible to deduce the structure of the SPARQL query from the structure of the question without knowing how the knowledge is encoded into the KB.

2.10.3 Query construction using Semantic Parsing

Semantic parsers are a particular type of parsers that couple syntactic rules to semantic composition. After parsing the question one gets a semantic interpretation of it.
From among the QA systems evaluated over QALD, different grammars for semantic parsers were used: GF grammars used by GFMed [108], feature-based context-free grammar (FCFG) used by TR Discover [137], Combinatory Categorial Grammar (CCG) used by Hakimov et al. [81] and lexical tree-adjoint grammars (LTAG) used by TBSL [156] and BELA [166]. We give a brief example using the CCG grammar. Consider the phrase "Barack Obama is married to Michelle Obama". To parse the sentence the following grammar rules are needed:

Lexical item	Syntactic category	Semantic representation
Barack Obama	NP	`dbr:Barack_Obama`
is	$(S\backslash NP)/(S\backslash NP)$	λ`f.`λ`x.f(x)`
married to	$(S\backslash NP)/NP$	λ`y.`λ`x.dbo:spouse(x,y)`
Michelle Obama	NP	`dbr:Michelle_Obama`

The first column indicates the phrases to which the rules are associated. The main syntactic categories are NP standing for noun phrase and S standing for sentence and combinations of them. The syntactic category $(S\backslash NP)/NP$ for example indicates that it can be combined with a noun phrase (NP) on the left and on the right to get a sentence S. Applying these rules, the sentence can be parsed from a syntactic point of view. Coupled to the syntactic rules is a semantic representation. Without going into details this is expressed

using lambda calculus. For example, the phrase *married to* semantically is a binary function which takes two arguments. Since it is the passive form of the property `dbo:spouse` the arguments are inverted. The semantic representation of *Michelle Obama* is just a constant which in DBpedia is `dbr:Michelle Obama`. The application of the above syntactic rule between *married to* and *Michelle Obama* results in the semantic representation $\lambda x.dbo : spouse(x, dbr : Michelle\_Obama)$, i.e. x is the spouse of *Michelle Obama*. This way the whole sentence can be parsed leading to the semantic representation $dbo : spouse(dbr : Barack\_Obama, dbr : Michelle\_Obama)$, i.e. *Barack Obama*'s spouse is *Michelle Obama*. This way the sentence is completely understood from a semantic point of view as well. The semantic representation can be translated then to a SPARQL query.
The main advantage of this approach is that one can directly get a semantic representation of a question. This also includes the superlative and comparative variations of the question. A disadvantage is that the questions have to be well-formulated, i.e. they are not robust with respect to malformed questions. The main disadvantage is that one needs to have for each lexical item a corresponding semantic representation. To generate the semantic representations, Hakimov et al. [81] adapted the algorithm of Zettlemoyer & Collins [184] that generates the semantic representation from a learning corpus of pairs of questions and the corresponding semantic representation. The main problem encountered by Hakimov et al. [81] is that many lexical items do not appear in the training corpus leading to low recall. To alleviate this problem, TBSL [156] generates candidate semantic representations of unknown lexical items based on their POS tags. In the example above, if the lexical item *married to* is unknown then possible interpretations are generated such as two binary functions with the arguments x and y exchanged. Since there is no knowledge about the fact that *married to* has to be mapped to the property $dbo : spouse$, thus some templates are generated, i.e. the parser is able to parse the question, but a binary property corresponding to *married to* has still to be found.

2.10.4 Query construction using machine learning

CASIA [84] uses a machine learning approach for the whole QA process. The question analysis phase is used to segment the question and to extract features like the position of a phrase, the POS tag of a phrase, the type of dependency in the dependency tree and some other. In the phrase mapping phase resources are associated with the segments and new features are extracted: the type of the resources and a score for the similarity between the segment and the resource. In the disambiguation phase the extracted features are used in a MLN (as described in 2.9.5) to find the most probable relation between the segments and to find the most probable mapping. The detected relations are then used to generate the SPARQL query. The disambiguation phase must be retrained for each new KB.

2.10.5 Query construction relaying on semantic information

The QA system SINA [132] was developed to deal primarily with keyword queries, i.e. instead of inserting the question "What is the capital of Belgium?" the user can also insert the keywords "capital Belgium". This implies that the relation between the different resources is not explicit like in a natural language question. In this case the dependencies between the resources have to be derived using the KB.
SINA first segments the question and finds associated resources. These are disambiguated using a HMM. Once the resources are determined, SINA constructs the query in the following way. For each instance or class, a vertex is created. For each property, an edge is created. The edges are used to connect the vertices if the types of the range and domain allow it. If not, then either one or two new vertices are created that correspond to variables. Note that the combinatorics could allow more than one graph. In this case, they are all considered because it is not clear which one reflects the user's intention. Moreover, at the end of the process, it is possible that one gets unconnected subgraphs. In this case, for each pair of vertices in two fixed subgraphs the set of possible properties that can connect them is computed and taken into account. All the possible graphs are translated to a SPARQL query and executed. The QA system POMELO proceeds in a similar way. The QA system developed by Zhang et al. also does not rely on syntactic features to construct the query. The query is generated using an ILP. First, those resources are identified which can be referred to by the question. These are then combined using some hand made rules into triple patterns. Between all possible triple patterns some are selected using an ILP.
The advantage of this strategy is that the graph is constructed starting from the underlying KB and not using the syntax of the question. The disadvantages are that this process is computationally complex and that the syntax of the question is not respected. For example, the systems will not see any difference between the questions "Who is the mother of Angela Merkel?" and "Angela Merkel is the mother of who?".

2.10.6 Approach not using SPARQL

Treo [72] and SemSek [3] do not generate a SPARQL query, but instead navigate through the KB by dereferencing resources. Consider the following example: "In which city was the leader of the European Union born?". SemSek first identifies a "central term" in the question, in this case "European Union". Using the dependency tree and starting from the "central term" an ordered list of potential terms corresponding to resources is generated. In the example above the list would be: "European Union", "leader", "born", "city". Then candidate resources for the first term (for example dbr:European_Union) are searched and the corresponding URI http://dbpedia.org/resource /European_Union is dereferenced to search all corresponding properties. These are compared with the second term in the list. The object is considered if one of the property is similar (as a string or semantically) to the second term in the list (like dbp:leaderName or dbp:leaderTitle). This generates two new

exploring directions in the algorithm. Otherwise the actual direction is not further analyzed. The systems continue like this and search the right answer in the graph.

2.10.7 Summary and research challenges for the query construction task

Table 2.6 gives an overview of the different techniques used by the QA systems for the query constructing. In this domain, the current research challenges that are identified are the interpretation of adjective modifiers and superlatives; the implementation of aggregation, comparison and negation operators; the resolution of syntactic and scope ambiguities; the non-compositionality, and the semantic tractability. Note that all challenges identified in previous surveys are discussed in section 2.14.

2.11 Querying distributed knowledge

In the previous sections, we discussed the techniques for answering questions over a single KB. Nevertheless, one pertinent question would be: what changes in the case of multiple KBs? Only few QA systems tackled this problem. We found in the literature that the problem can be classified into two groups. The first assumes that the KBs are disjoint graphs. The second assumes that the KBs are interlinked, i.e. resources that refer to the same entity are identified through the KBs using owl:sameAs links (two identified resources are called aligned).

2.11.1 Considering unconnected KBs

In this setting the KBs are not interlinked and in particular different KBs can refer to the same entity using different URIs. This scenario was tackeled by PowerAqua [100] and Zhang et al [186]. Assume for example that for the question "Which rivers flow in European countries?" a part of the information can be found into two different KBs. One containing information like "(?river, flow, ?country)" and the second one "(?country, type, European)" (i.e in general one can say that there are triple-patterns matching different KBs). These cannot be executed as a SPARQL query because the URIs for countries in the first and second KBs are different and not linked. Therefore, the results are retrieved independently from both KBs and merged by comparing the labels of the URIs. Then either a SPARQL query is generated that contains the aligned URIs (i.e. uri:a owl:sameAs uri:b) or the result is computed without a SPARQL query.

	Using templates	Using info. from the QA	Using Semantic Parsing	Using machine learning	Semantic information	Not generating SPARQL	Reference
BELA			x				[166]
CASIA				x			[84]
DEANNA		x					[172]
FREyA		x					[38]
gAnswer		x					[188]
GFMed			x				[108]
Hakimov et al.			x				[81]
Intui2		x					[59]
Intui3		x					[58]
ISOFT	x						[121]
POMELO					x		[82]
PowerAqua	x						[100]
QAKiS	x						[27]
QAnswer		x					[130]
RTV		x					[76]
SemGraphQA		x					[13]
SemSeK						x	[3]
SINA					x		[132]
SWIP	x						[124]
TBSL			x				[156]
Treo						x	[72]
TR Discover			x				[137]
UTQA	?						[122]
Xser		x					[170]
Zhang et al.					x		[186]
Zhu et al.						x	[187]

Table 2.6: Each line of this table corresponds to a QA system. The columns indicate which of the techniques presented in the query construction task are used. The columns refer to the subsections of section 2.10.

	Unconnected KBs	Interlinked KBs	Reference
GFMed		x	[108]
POMELO		x	[82]
PowerAqua	x		[100]
SINA		x	[132]
TR Discover		x	[137]
Zhang et al.	x		[186]

Figure 2.10: Each line of this table corresponds to a QA system and indicates which of the techniques presented in the distributed task it uses. The columns refer to the subsections of section 2.11.

2.11.2 Considering interlinked KBs

The task of querying interlinked KBs was proposed at QALD-4. It was tackled by GFMed [108], SINA [132], POMELO [82], TR Discover [137] and Zhang et al [186]. In the case where the KBs are interlinked there is no particular technique used. In fact one can see the interlinked KBs as one big KB. The identification with `owl:sameAs` links must be considered during query construction. Note that, in such a scenario, scalability can easily become a problem. This is not the case for the QALD-4 task since only three small KBs are interlinked.

2.12 QA systems evaluated over WebQuestions

In this section, we describe QA systems evaluated over WebQuestions. Table 2.7 contains a list of QA systems evaluated over WebQuestions. To identify these systems, we searched in Google Scholar for all publications citing the publication [15] which introduced the benchmark. In the following, we focus on techniques used by QA systems evaluated over WebQuestions that were not used by QA systems evaluated over QALD. Due to space limitations, a detailed analysis as for QALD is not possible.

Many of the techniques presented in section 2.7 about question analysis are also used by QA systems evaluated over WebQuestions. For example, [126] makes an extensive use of dependecy trees, while SEMPRE [15] and PARASEMPRE [20] use POS tags as features. In addition to the techniques presented in section 2.7 some works make use of neural networks. These have been successfully used in many NLP areas. Different architectures have been explored. Convolutional neural networks and recurrent neural networks where used to identify or focus on particular parts of the questions (like the one referring to the type, the relation or the entity contained in a question). Convolutional neural networks are for

QA systems	Precision	Recall	F-measure	Reference
Yao et al. (2014)	0.480	0.337	0.354	[176]
SEMPRE	0.413	0.480	0.357	[15]
Bao et al. (2014)	-	-	0.375	[10]
Bordes et al. (2014)	-	-	0.392	[20]
PARASEMPRE	0.466	0.405	0.399	[17]
Clarke et al. (2015)	-	-	0.401	[30]
Dong et al. (2015)	-	-	0.401	[60]
Yang et al. (2014)	-	-	0.413	[173]
GraphPharser	-	-	0.413	[125]
Bordes et al. (2015)	-	-	0.422	[21]
Zhang et al. (2016)	-	-	0.426	[185]
Yao (2015)	0.545	0.526	0.443	[175]
Yang et al. (2015)	-	-	0.449	[174]
Aqqu	0.604	0.498	0.494	[11]
AgendaIL	-	-	0.497	[16]
Wang et al. (2014)	0.525	0.447	0.453	[167]
Reddy et al. (2016)	0.611	0.490	0.503	[126]
Abujabal et al. (2017)	-	-	510	[2]
Yavuz et al. (2016)	-	-	0.516	[178]
STAGG	0.607	0.528	0.525	[179]
Ture et al. (2016)	-	-	0.522	[153]
Jain (2016)	0.649	0.552	0.557	[89]

Table 2.7: This table summarizes the QA systems evaluated over WebQuestions. It contains publications describing a QA system evaluated over WebQuestions that cited [15] according to google scholar.

example used in [60] while recurrent neural networks are used in [185] and [153].
Many QA system use similar approaches to the one presented in section 2.8 about the pharse mapping task. For example the strategy presented in section 2.8.3 is used by SEMPRE [15], [174] and GraphPharser [125]. One additional dataset that was used to close the lexical gap was the PARALEX corpus [65]. It contains 18 million pairs of questions from wikianswers.com which where tagged as having the same meaning by users. This dataset can be used to learn different ways to express the same relation. It was used for example by PARASEMPRE [20] and [21]. Moreover differently from QALD many works use the datasets and strategies presented in section 2.8 to create embeddings (using neural networks) for relations and entities. This is for example the case for [21], [173], [178], STAGG [179] and [153].
The disambiguation problem discussed in section 2.9 is approached with similar strategies both in QALD and WebQuestions. To disambiguate the entities, many QA systems evaluated over WebQuestions use EL tools for Freebase. To disambiguate between different generated queries, mainly machine learning algorithms are used like structured perceptrons, [126], and learning to rank used in Aqqu [11]. The features are mainly the same.
The query construction problem discussed in section 2.10 is simpler than for QALD. Many QA systems restrict to simple triple patterns like [21], [153] while most of them restrict to simple or reified triple patterns like it is done in Aqqu [11] or [126]. Some QA systems follow a strategy that is similar to the one presented in section 2.10.6. However, one strategy is not used over QALD, it is generally referred to "information retrieval approach". Consider for example the question: "When did Avatar release in UK?". First, the main entity of the question is searched. In the example, the entity is "Avatar". Then all entities connected to it via a simple or a reified statement are considered as a possible answer. This includes for example the date "17-12-2009" but also the director of Avatar, the characters that played in the film and many others. In this manner, the problem is reduced to a classification problem, i.e., determining if an entity is an answer or not. This is done, for example, by [176], [20], [21], [60] and [185].
The task of querying distributed knowledge is not addressed in the WebQuestions benchmark.
In summary, one can say that over WebQuestions more attention was put on closing the lexical gap while in QALD more effort was done in the query construction process. This is mainly due to the type of questions contained in the benchmarks.

2.13 QA systems evaluated over SimpleQuestions

In this section, we describe QA systems evaluated over SimpleQuestions. Table 3.5 contains a list of QA systems evaluated over SimpleQuestions. To identify these systems, we searched in Google Scholar for all publications citing the publication [21] which introduced the benchmark.
All systems evaluated over this dataset use neural networks architectures. In fact this

QA systems	F-measure	Reference
Bordes et al. (2015)	0.627	[21]
Yin et al. (2016)	0.683	[181]
Dai et al. (2016)*	0.626	[33]
Golub and He (2016)	0.709	[78]
Lukovnikov et al. (2017)	0.712	[104]

Table 2.8: This table summarizes the QA systems evaluated over SimpleQuestions. It contains publications describing a QA system evaluated over SimpleQuestions that cited [21] according to google scholar. Every system was evaluated over FB2M except the one marked with $(*)$ which was evaluated over FB5M.

benchmark was created to experiment with these machine learning technique since it needs a lot of training data to perform well. The different QA Systems evaluated over SimpleQuestions follow a similar strategy. Remember that every question in this benchmark can be solved by one single triple, i.e. only a subject and a predicate has to be found. The possible triples are found in the following way. First using an n-gram strategy (see section 2.7.1) candidate entities are identified. Then by looking in the KB all triples containing these entities as a subject are identified. The problem is reduced to choose between the list of triples the right one. This last step is done using different neural network architectures. Let's see more in detail how the different phases of the QA process are tackled.

The question analysis task is generally not treated as a separate task but some parts of the networks are dedicated to it. [78] and [181] use the so called attention mechanisms. This means that the network learns to focus on the parts of the questions that refer to a subject or to a predicate.

For the phrase mapping task the QA systems relay on the labels provided by the KB as described in subsection 2.8.1. Moreover the question is encoded either at the character level like in [78] or, using word embeddings like in [33] or, in both ways like in [104]. Word embedding is used to bridge the lexical gap following the idea presented in subsection 2.8.3. The character encoding is used to overcome the out-of-vocabulary problem, i.e. the number of possible words appearing in a question is too big to include them all in the vocabulary.

The disambiguation problem is handled as a ranking problem. Here the different neural network architectures are used to compute a score between a pair of a question, and a tuple of a subject and a predicate. The pair with the highest score is token.

The query construction problem is very simple since only SPARQL queries with one triple pattern must be generated. The problem of querying multiple KBs is not addressed.

The main problem tackled in this benchmark is the disambiguation problem.

Challenge	Survey, Years covered	Short description	Task
Identify Question types	[29], 2004-07, [101], 2005-12	Includes wh-questions, requests (*give me*), nominal or definitions, topicalised entities, how questions with adjective or quantifications (*how big/many*)	Question Analysis
Lexical gap, Vocabulary gap	[103] 2004-11, [101], 2005-12, [73], 2004-11, [74], 2011-12, [87], 2011-15	Query and databases may not be expressed using the same vocabulary (synonymy) or at the same level of abstraction. It requires to bridge the gap between the vocabulary in the user query and the KB vocabulary	Mapping
Multilingual QA	[28] QALD-3, [158] QALD-4, [87], 2011-15	Mediates between a user expressing an information need in her own language and the semantic data	Question Analysis, Mapping
Light expression Vagueness Lexical ambiguity	[29], 2004-07, [103], 2004-11, [101], 2005-12, [74], 2004-11, [87], 2011-15	Queries with words that can be interpreted through different ontological entities or semantically weak constructions. Relations that are expressed implicitly with the use of verbs such as *be/have* or light prepositions that can convey different meanings	Disambiguation
Semantic gap Conceptual complexity	[101], 2005-12, [73], 2004-11, [74], 2011-12	Queries that are not necessarily structured in the knowledge base in the same way than in the question.	Query construction
Spatial and temporal prepositions	[29], 2004-07, [101], 2005-12, [87], 2011-15	Requires to capture the domain-independent meaning of spatial (*in, next, thorough*) and temporal (*after, during*) prepositions	-
Adjective modifiers and superlatives	[29], 2004-07, [103], 2004-11, [101], 2005-12, [87], 2011-15	Superlative modifiers and attribute selectors (*how+adj*) require mapping each adjective to a KB predicate (e.g.,*area/ population* for *smallest*), as well as keeping the polarity for superlatives (order by ASC/DESC)	Disambiguation, Query construction
Aggregations, comparison and negation operators	[29], 2004-07, [103], 2004-11, [101], 2005-12, [87], 2011-15	Aggregation operators are those calculating a min, max, sum, an average or a count over a number of individuals fulfilling a certain property. Comparison operators compare numbers to a given order. The challenge is to realize a quantifier through logical operators	Question Analysis, Query Construction
Syntactic and Scope ambiguities	[29], 2004-07	Syntactic ambiguity regarding the constituent that prepositional phrases or relative clauses can attach to (to the last or to a non-precedent constituent in a sentence), or when multiple scope quantifiers are present in a query (*most, all, each, etc.*)	Query construction
Distributed QA Entity reconciliation	[103], 2004-11, [73], 2004-11, [158], QALD-4, [87], 2011-15	Combining facts across sources requires matching at schema level as well as entity level (find semantically equivalent dataset entities given a query entity) to join partial results or translations	Distributed Knowledge
Non-compositionality	[29], 2004-07	Parts of a question that do not correspond to any logical form and need to be ignored (e.g., *largest cities in the world* if *world* is not explicitly model)	Query construction
Semantic tractability	[73], 2004-11	To answer queries not supported by explicit dataset statements (e.g., inferring an entity *x* is an *actress* because of the statement *x starred y movie*)	Mapping, Disambiguation, Query construction
Out of scope	[29], 2004-07, [103], 2004-11	A system should inform about the fact that the question is out of scope of the KB (vs. out of the capabilities of the system)	Across all
Portability	[29], 2004-07, [103], 2004-11	The level of effort require to port the system to other sources (e.g., handcrafted lexicon, training)	Across all
Scalability	[103], 2004 - 2011	Required both in terms of KB size and their number while keeping real time performance	Across all
Hybrid QA, Semantic and textual gap	[103], 2004-11, [158] QALD-4	Combining both structured and unstructured information into one answer	-

Table 2.9: Challenges in the state of the art for QA systems over KBs

2.14 Evolution of research challenges of Question Answering over KBs

We look at the research challenges described in the literature since the beginning of open domain KB-based QA, through different published surveys and QALD evaluation reports. Table 2.9 lists each challenge, together with the reference(s) to the survey where it is discussed and the years covered, i.e., the range of publication years for the KB-based QA systems cited in the given survey, excluding other kind of QA systems that share some of the challenges (such as Natural Language Interfaces for Databases). We group similar challenges together and, if it applies, map them across the five tasks in the QA process. As such we associate the tasks, in which we cluster the techniques analysed in this paper, with the challenges they try to tackle.

The first survey by Cimiano and Minock [29] published in 2010 presents a quantitative analysis based on the input (user) questions as well as the outputs (formal queries) from a Geographical dataset, often used to evaluate the first domain-specific KB-based systems. The following challenges are identified: question types; language light; lexical ambiguities; syntactic ambiguities; scope ambiguities; spatial and temporal prepositions; adjective modifiers and superlatives, aggregation, comparison and negations; non-compositionality; out of scope; and variability of linguistic forms (i.e., questions matching to the same logical query - not included in Table 2.9).

Some of these first identified challenges, like capturing the meaning of spatial and temporal prepositions, are not yet tackled by the kind of QA systems surveyed here. Cimiano and Minock also argued that research on deep linguistic and semantic approaches, such as: paraphrase generation, discourse processing and guidance; could improve the performance of these systems by guiding users to generate a non-ambiguous form of the question (i.e., in line with the capabilities of the system), as well to process questions not in isolation, but in context with previously asked questions.

For these early systems, portability was one of the most challenging issues, although the customisation effort was rarely quantified or compared [29]. Lopez et al. 2011 survey [103] discusses the challenges that arised when moving from a classic KB system to an open domain QA system for the Web of data, in terms of portability, scalability, mapping and disambiguation, distributed query (fusion and ranking of facts across sources) and bridging the gap between the semantic data and unstructured textual information. Lexical ambiguities (noun homonymy or polysemy) are less of a problem when querying an unambiguous knowledge representation in a restricted domain. However, for open domain QA, a major challenge is to disambiguate the correct interpretation while scaling up to large and heterogeneous sources. Filtering and ranking techniques are often used to prune the large space of candidate solutions and obtain real time performance [103].

Freitas et al. 2012 [73] categorized the challenges on querying heterogeneous datasets into query expressivity, usability, vocabulary-level semantic matching, entity reconciliation and tractability. As stated in [73] differently from IR, entity centric QA approaches aim towards

more sophisticated semantic matching techniques to target queries with high expressivity (able to operate over the data, including superlatives, aggregators, etc.), without assuming that the users are aware of the internal representations (high usability through intuitive query interfaces). For Freitas et al. 2015 [74] the process of mapping and semantic interpretation of schema agnostic queries involves coping with conceptual complexity, term ambiguity, vocabulary gap (synonymy) and vagueness / indeterminacy (where words or propositions fail to map the exact meaning intended by the transmitter). Freitas et al. [74] proposed entropy measures to quantify the semantic complexity of mapping queries to database elements, showing that a large number of possible interpretations for words/prepositions had a negative impact in the F-measure reported for systems participating in QALD-1 and QALD-2.

Most of the challenges specified by Cimiano and Minock [29] for the first domain specific QA systems remain valid for the open domain questions presented in the QALD benchmarks. Based on these challenges and the results from QALD-1 and QALD-2 campaigns, Lopez et al. 2013 [101] analyzed the characteristic problems involved in the task of mapping NL queries to formal queries, such as: (1) the limitations to bridge the lexical gap (between the vocabulary in the user query and the KB vocabulary) using only string metrics and generic dictionaries; (2) the difficulties arising on interpreting words through different entities because of the openness and heterogeneity of the sources (including duplicated or overlapping URIs and complex conjunctive KB terms from YAGO categories); and (3) complex queries with information needs that can only be expressed using aggregation (requiring counting), comparison (requiring filters), superlatives (requiring sorting results), temporal reasoning (e.g., on the same day, etc.), or a combination of them.

As well as the challenges reported in Lopez et al. 2013 [101] for the first two campaigns, QALD-3 [28] introduced the multilinguality challenge and QALD-4 [158] introduced two new challenges as part of two new tasks. The first task, distributed QA, introduces questions for which answers are distributed across a collection of interconnected datasets in the biomedical domain. The second task, Hybrid QA, evaluates approaches that can process both structured and unstructured information, considering that lots of information is still available only in textual form, e.g., in the form of abstracts.

Lastly, the latest survey on challenges for semantic QA systems [87] classifies these into seven challenges: the lexical gap, ambiguity, multilingualism, complex queries, distributed knowledge, temporal and spatial queries and templates. The templates challenge refer to the systems that use templates to capture the structure of (complex) queries with more than one basic graph pattern.

2.15 Conclusions

This analysis of QA systems shows that most of the systems have many common techniques. For example, a lot of systems use similar techniques in the question analysis and phrase mapping task. Moreover, it shows that, QA systems generally concentrate on improving only some techniques, whereas, they leave aside some others. For example, a lot of systems

only use local disambiguation techniques, others only use basic techniques for the phrase mapping task.

It is nearly impossible to compare existing techniques individually. There are many reasons for that. From the performance of a monolithic system it is impossible to deduce the contribution of a single part. It is also not sufficient to just compare techniques that solve the same task against each other since a QA system is a combination of many components, and a balance between precision and recall must be found considering all steps. A component with high recall and low precision can be good or bad depending on the next steps. So a comparison can only be made by comparing the combinations of different techniques in a whole pipeline.

One solution to address these problems is to develop future QA systems using a modular approach. This will allow the community to contribute to new plug-ins in order to either replace existing approaches or create new ones. The aim is that a QA system should not always be build from scratch such that research can focus more on single tasks. As a side effect, it will become easier to compare techniques which solve the same task. These plug-ins should satisfy the constraints of the Semantic Web. They should be KB independent, scalable, possibly multilingual and require as few efforts as possible to set up a QA system over a new dataset. We are aware of four frameworks that attempt to provide a reusable architecture for QA systems. QALL-ME [67], openQA [109], OKBQA[17] and QANARY [22, 51]. Such integration would also allow to build and reuse services around QA systems like speech recognition modules and reusable front-ends [46]. This way the research community is enabled to tackle new research directions like developing speech recognition systems specifically designed to answer questions over KBs and study the interaction of QA systems with the user.

The comparison between QALD and WebQuestions shows that these benchmarks are quite similar. Despite that, both benchmarks are addressed by two quite isolated communities. That can be seen by the fact that QA systems are either evaluated on QALD or on WebQuestions and not on both. We hope that in future these communities will meet so that both can learn from each other. One possible meeting point is the Wikidata KB. There are several reasons for that. The Freebase KB is not updated anymore and all related services were shut down. The Wikidata dump contains both a reified and a non-reified version of the information so that moving from DBpedia and Freebase to Wikidata is simple. Moreover there are more and more well-maintained services around Wikidata that can be used to develop new QA systems.

At the same time, the fact that most QA systems are evaluated only over one KB, shows that more work is needed in creating QA systems that are truly KB independent. This would also allow to get new insights of how the quality of the KB affects the quality of the QA system. These point should be tackled more by the QA community.

Finally a poorly addressed point is the interaction of QA systems with users. We are not aware of works that study the usability of open domain KB-based QA systems and there

[17] http://www.okbqa.org

are a few works applying QA over KBs in real scenarios [137, 98]. The interaction with the user can also be a good opportunity to improve QA systems over time and to collect training data which is becoming more and more important to improve the performance and understanding of these systems.
Research in QA over KBs is and remains a hot and interesting topic!

3 Towards a Question Answering System over the Semantic Web[1]

3.1 Abstract

Thanks to the development of the Semantic Web, a lot of new structured data has become available on the Web in the form of knowledge bases (KBs). Making this valuable data accessible and usable for end-users is one of the main goals of Question Answering (QA) over KBs. Most current QA systems query one KB, in one language (namely English). These approaches are not designed to be easily adaptable to new KBs and languages.
We first introduce a new approach for translating natural language questions to SPARQL queries. It is able to query several KBs simultaneously, in different languages, and can easily be ported to other KBs and languages. In our evaluation, the impact of our approach is proven using 5 different well-known and large KBs: Wikidata, DBpedia, MusicBrainz, DBLP and Freebase as well as 5 different languages namely English, German, French, Italian and Spanish.
Second, we show how we integrated our approach, to make it easily accessible by the research community and by end-users.
To summarize, we provided a conceptional solution for multilingual, KB-agnostic Question Answering over the Semantic Web. The provided first approximation validates this concept.

3.2 Introduction

Question Answering (QA) is an old research field in computer science that started in the sixties [103]. In the Semantic Web, a lot of new structured data has become available in the form of knowledge bases (KBs). Nowadays, there are KBs about media, publications, geography, life-science and more[2]. The core purpose of a QA system over KBs is to retrieve the desired information from one or many KBs, using natural language questions. This is generally addressed by translating a natural language question to a SPARQL query. Current research does not address the challenge of multilingual, KB-agnostic QA for both full and keyword questions (Table 3.1).

[1] Corresponding publications is: Dennis Diefenbach, Andreas Both, Kamal Singh and Pierre Maret, *Towards a Question Answering System over the Semantic Web*, under review at the Semantic Web Journal [55]
[2] `http://lod-cloud.net`

QA system	Lang	KBs	Type
gAnswer [188] (QALD-3 Winner)	en	DBpedia	full
Xser [170] (QALD-4 & 5 Winner)	en	DBpedia	full
UTQA [122]	en, es, fs	DBpedia	full
Jain [89] (WebQuestions Winner)	en	Freebase	full
Lukovnikov [104] (SimpleQuestions Winner)	en	Freebase	full
Ask Platypus (`https://askplatyp.us`)	en	Wikidata	full
WDAqua-core1	en, fr, de, it, es	Wikidata, DBpedia, Freebase, DBLP, MusicBrainz	full & key

Table 3.1: Selection of QA systems evaluated over the most popular benchmarks. We indicated their capabilities with respect to multilingual questions, different KBs and different typologies of questions (full = "well-formulated natural language questions", key = "keyword questions").

There are multiple reasons for that. Many listed QA approaches rely on language-specific tools (NLP tools), *e.g.* SemGraphQA [13], gAnswer [188] and Xser [170]. Therefore, it is difficult or impossible to port them to a language-agnostic system. Additionally, many approaches make particular assumptions on how the knowledge is modelled in a given KB (generally referred to as "structural gap" [49]). This is the case of AskNow [61] and DEANNA [172].

There are also approaches which are difficult to port to new languages or KBs because they need a lot of training data which is difficult and expensive to create. This is for example the case of Bordes et al. [21]. Finally there are approaches where it was not proven that they scale well. This is for example the case of SINA [132].

In this paper, we present an algorithm that addresses all of the above drawbacks and that can compete, in terms of F-measure, with many existing approaches.

This publication is organized as follows. In section 3.3, we present related works. In section 3.4 and 3.5, we describe the algorithm providing the foundations of our approach. In section 3.6, we provide the results of our evaluation over different benchmarks. In section 3.7, we show how we implemented our algorithm as a service so that it is easily accessible to the research community, and how we extended a series of existing services so that our approach can be directly used by end-users. We conclude with section 3.8.

3.3 Related work

In the context of QA, a large number of systems have been developed in the last years. For a complete overview, we refer to [49]. Most of them were evaluated on one of the following three popular benchmarks: WebQuestions [15], SimpleQuestions [21] and QALD[3].
WebQuestions contains 5810 questions that can be answered by one reefied statement, SimpleQuestions contains 108442 questions that can be answered using a single, binary-relation, while the QALD challenge versions, contain between 100 and 450 questions, which also includes more complex questions than the previous ones, and are therefore, compared to the other, small datasets.
The high number of questions of WebQuestions and SimpleQuestions led to many supervised-learning approaches for QA. Especially deep learning approaches became very popular in the recent years like Bordes et al. [21] and Zhang et al. [185]. The main drawback of these approaches is the training data itself. Creating a new training dataset for a new language or a new KB might be very expensive. For example, Berant et al. [15], report that they spent several thousands of dollars for the creation of WebQuestions using Amazon Mechanical Turk. The problem of adapting these approaches to new dataset and languages can also be seen by the fact that all these systems work only for English questions over Freebase.
A list of the QA systems that were evaluated with QALD-3, QALD-4, QALD-5, QALD-6 can be found in Table 3.3. According to [49] less than 10% of the approaches were applied to more than one language and 5% to more than one KB. The reason is the heavy use of NLP tools or NL features like in Xser [170], gAnswer [188] or QuerioDali [98].
The problem of QA in English over MusicBrainz[4] was proposed in QALD-1, in the year 2011. Two QA systems tackled this problem. Since then the MusicBrainz KB[5] completely changed. We are not aware of any QA system over DBLP[6].
In summary, most QA systems work only in English and over one KB. Multilinguality is poorly addressed while portability is generally not addressed at all.
The fact that QA systems often reuse existing techniques and need several services to be exposed to the end-user, leads to the idea of developing QA systems in a modular way. At least four frameworks tried to achieve this goal: QALL-ME [67], openQA [109], the Open Knowledge Base and Question-Answering (OKBQA) challenge[7] and Qanary [133, 22, 50]. We integrated our system as a Qanary QA component called WDAqua-core1. We choose Qanary for two reasons. First, it offers a series of off-the-shelf services related to QA systems and second, it allows to freely configure a QA system based on existing QA components.

[3] http://www.sc.cit-ec.uni-bielefeld.de/qald/
[4] https://musicbrainz.org
[5] https://github.com/LinkedBrainz/MusicBrainz-R2RML
[6] http://dblp.uni-trier.de
[7] http://www.okbqa.org/

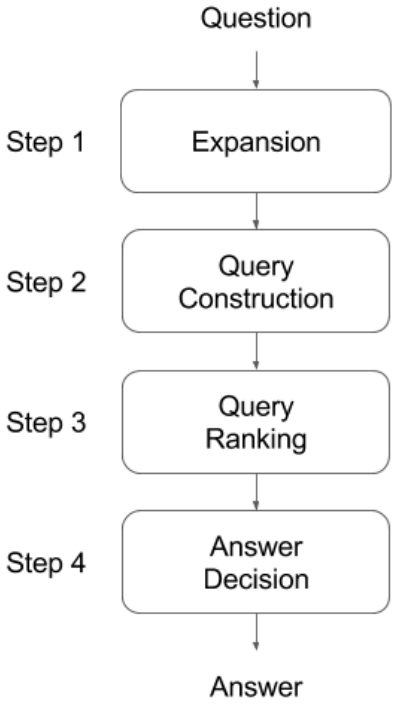

Figure 3.1: Conceptual overview of the approach

3.4 Approach for QA over Knowledge Bases

In this section, we present our multilingual, KB-agnostic approach for QA. It is based on the observation that many questions can be understood from the semantics of the words in the question while the syntax of the question has less importance. For example, consider the question "Give me actors born in Berlin". This question can be reformulated in many ways like "In Berlin were born which actors?" or as a keyword question "Berlin, actors, born in". In this case by knowing the semantics of the words "Berlin", "actors", "born", we are able to deduce the intention of the user. This holds for many questions, i.e., they can be correctly interpreted without considering the syntax as the semantics of the words is sufficient for them. Taking advantage of this observation is the main idea of our approach. The KB encodes the semantics of the words and it can tell what is the most probable interpretation of the question (w.r.t. the knowledge model described by the KB).

Our approach is decomposed in 4 steps: question expansion, query construction, query ranking and response decision. A conceptual overview is given in Figure 3.1. In the following, the processing steps are described. As a running example, we consider the question "Give me philosophers born in Saint-Etienne". For the sake of simplicity, we use DBpedia as KB to answer the question. However, it is important to recognize that no assumptions either about the language or the KB are made. Hence, even the processing of the running example is language- and KB-agnostic.

n	start	end	n-gram	resource
1	2	3	philosophers	dbrc:Philosophes
2	2	3	philosophers	dbr:Philosophes
3	2	3	philosophers	dbo:Philosopher
4	2	3	philosophers	dbrc:Philosophers
5	2	3	philosophers	dbr:Philosopher
6	2	3	philosophers	dbr:Philosophy
7	2	3	philosophers	dbo:philosophicalSchool
8	3	4	born	dbr:Born,_Netherlands
9	3	4	born	dbr:Born_(crater)
10	3	4	born	dbr:Born_auf_dem_Dar?
11	3	4	born	dbr:Born,_Saxony-Anhalt
⋮				
42	3	4	born	dbp:bornAs
43	3	4	born	dbo:birthDate
44	3	4	born	dbo:birthName
45	3	4	born	dbp:bornDay
46	3	4	born	dbp:bornYear
47	3	4	born	dbp:bornDate
48	3	5	born in	dbp:bornIn
49	3	5	born in	dbo:birthPlace
50	3	5	born in	dbo:hometown
52	5	6	saint	dbr:SAINT_(software)
53	5	6	saint	dbr:Saint
54	5	6	saint	dbr:Boxers_and_Saints
55	5	6	saint	dbr:Utah_Saints
56	5	6	saint	dbr:Saints,_Luton
57	5	6	saint	dbr:Baba_Brooks
58	5	6	saint	dbr:Battle_of_the_Saintes
59	5	6	saint	dbr:New_York_Saints
⋮				
106	5	6	saint	dbp:saintPatron
107	5	6	saint	dbp:saintsDraft
108	5	6	saint	dbp:saintsSince
109	5	6	saint	dbo:patronSaint
110	5	6	saint	dbp:saintsCollege
111	5	6	saint	dbp:patronSaintOf
112	5	6	saint	dbp:patronSaint(s)
113	5	6	saint	dbp:patronSaint'sDay
114	5	7	saint etienne	dbr:Saint_Etienne_(band)
115	5	7	saint etienne	dbr:Saint_Etienne
116	5	7	saint etienne	dbr:Saint-Étienne
117	6	7	etienne	dbr:Étienne

Table 3.2: Expansion step for the question "Give me philosophers born in Saint Étienne". The first column enumerates the candidates that were found. Here, 117 possible entities, properties and classes were found from the question. The second, third and fourth columns indicate the position of the n-gram in the question and the n-gram itself. The last column is for the associated IRI. Note that many possible meanings are considered: line 9 says that "born" may refer to a crater, line 52 that "saint" may refer to a software and line 114 that "Saint Étienne" may refer to a band.

3.4.1 Expansion

Following a recent survey [49], we call a lexicalization, a name of an entity, a property or a class. For example, “first man on the moon” and “Neil Armstrong” are both lexicalizations of `dbr:Neil_Armstrong`. In this step, we want to identify all entities, properties and classes, which the question could refer to. To achieve this, we use the following rules:

- All IRIs are searched whose lexicalization (up to stemming) is an n-gram N (up to stemming) in the question.
- If an n-gram N is a stop word (like “is”, “are”, “of”, “give”, ...), then we exclude the IRIs associated to it. This is due to the observation that the semantics are important to understand a question and the fact that stop words do not carry a lot of semantics. Moreover, by removing the stop words the time needed in the next step is decreased.

An example is given in Table 3.2. The stop words and the lexicalizations used for the different languages and KBs are described in section 3.6.1. In this part, we used a well-known Lucene Index[8] technology which allows fast retrieval, while providing a small disk and memory footprint.

3.4.2 Query construction

In this step, we construct a set of queries that represent possible interpretations of the given question within the given KB. Therefore, we heavily utilize the semantics encoded into the particular KB. We start with a set R of IRIs from the previous step. The goal is to construct all possible queries containing the IRIs in R which give a non-empty result-set. Let V be the set of variables. Based on the complexity of the questions in current benchmarks, we restrict our approach to queries satisfying 4 patterns:

```
SELECT / ASK var
 WHERE { s1 s2 s3 .  }
```

```
SELECT / ASK var
WHERE { s1 s2 s3 .
  s4 s5 s6 .  }
```

with $s1, \ldots, s6 \in R \cup V$ and $var \in \{s1, \ldots, s6\} \cap V$, i.e. all queries containing

[8] https://lucene.apache.org

one or two triple patterns that can be created starting from the IRIs in R. Moreover, for entity linking, we add the following two patterns:

```
SELECT ?x
WHERE { VALUES  ?x {iri} .  }
```

```
SELECT ?x
WHERE { VALUES  ?x {iri} .
  iri ?p iri1 .  }
```

`with iri, iri1` $\in$ `R`, i.e. all queries returning directly one of the IRIs in `R` with possibly one additional triple.
Note that these last queries just give back directly an entity and should be generated for a question like: "What is Apple Company?" or "Who is Marie Curie?". An example of generated queries is given in Figure 3.2.
The main challenge is the efficient construction of these SPARQL queries. The main idea is to perform in the KB graph a breadth-first search of depth 2 starting from every IRI in R. While exploring the KB for all IRIs $r_j \in R$ (where $r_j \neq r_i$) the distance d_{r_i,r_j} between two resources is stored. These numbers are used when constructing the queries shown above. For a detailed algorithm of the query construction phase, please see section 3.5. Concluding, in this section, we computed a set of possible SPARQL queries (candidates). They are driven by the lexicalizations computed in section 3.4.1 and represent the possible intentions expressed by the question of the user.

3.4.3 Ranking

Now the computed candidates need to be ordered by their probability of answering the question correctly. Hence, we rank them based on the following features:

- Number of the words in the question which are covered by the query. For example, the first query in Figure 3.2 is covering two words ("Saint" and "born").
- The edit distance of the label of the resource and the word it is associated to. For example, the edit distance between the label of `dbp:bornYear` (which is "born year") and the word "born" is 5.
- The sum of the relevance of the resources, (e.g., the number of inlinks and the number of outlinks of a resource). This is a knowledge base independent choice, but it is also possible to use a specific score for a KB (like page-rank).
- The number of variables in the query.

```
• SELECT DISTINCT ?y WHERE {
    dbr:Saint_(song) ?p ?x .
    ?x  dbo:hometown  ?y . }

• SELECT ?x {
  VALUES ?x { dbr:Saint_Etienne_(band) } }

• SELECT DISTINCT ?y WHERE {
    ?x dbo:birthPlace dbr:Saint-Etienne .
    ?x dbo:birthDate  ?y . }

• SELECT DISTINCT ?y WHERE {
    ?x ?p dbr:Philosophy  .
    ?x dbo:birthDate  ?y . }
```

Figure 3.2: Some of the 395 queries constructed for the question "Give me philosophers born in Saint Etienne.". Note that all queries could be semantically related to the question. The second one is returning Saint-Etienne as a band, the third one the birth date of people born in Saint-Etienne and the forth one the birth date of persons related to philosophy.

- The number of triples in the query.

If no training data is available, then we rank the queries using a linear combination of the above 5 features, where the weights are determined manually. Otherwise we assume a training dataset of questions together with the corresponding answers set, which can be used to calculate the F-measure for each of the SPARQL query candidates. As a ranking objective, we want to order the SPARQL query candidates in descending order with respect to the F-measure. In our exemplary implementation we rank the queries using RankLib[9] with Coordinate Ascent [113]. At test time the learned model is used to rank the queries, the top-ranked query is executed against a SPARQL endpoint, and the result is computed. An example is given in Figure 4.2.4. Note that, we do not use syntactic features. However, it is possible to use them to further improve the ranking.

3.4.4 Answer Decision

The computations in the previous section lead to a list of ranked SPARQL queries candidates representing our possible interpretations of the user's intentions. Although the quality of this processing step is high (as shown in several experiments), an additional confidence score is computed. We construct a model based on logistic regression. We use a training set consisting of SPARQL queries and the labels true or false. True indicates if the F-score of the SPARQL query is bigger than a threshold θ_1 or false otherwise. Once the model is trained, it can compute a confidence score $p_Q \in [0, 1]$ for a query Q. In our exemplary

[9] https://sourceforge.net/p/lemur/wiki/RankLib/

1.
```
SELECT DISTINCT ?x WHERE {
  ?x dbp:birthPlace dbr:Saint-Etienne .
  ?x rdf:type dbo:Philosopher . }
```

2.
```
SELECT DISTINCT ?y WHERE {
  ?x  dbo:birthPlace dbr:Saint-Etienne .
  ?x  dbo:philosophicalSchool ?y . }
```

3.
```
SELECT DISTINCT ?x WHERE {
  ?x  dbp:birthPlace dbr:Saint-Etienne . }
```

4.
```
SELECT DISTINCT ?x WHERE {
  ?x  dbo:hometown dbr:Saint-Etienne . }
```

Figure 3.3: The top 4 generated queries for the question "Give me philosophers born in Saint Étienne.". (1) is the query that best matches the question; (2) gives philosophical schools of people born in Saint-Étienne; (3)(4) give people born in Saint-Étienne or that live in Saint-Étienne. The order can be seen as a decreasing approximation to what was asked.

implementation we assume a correctly ordered list of SPARQL query candidates computed in section 3.4.3. Hence, it only needs to be checked whether $p_{Q_1} \geq \theta_2$ is true for the first ranked query Q_1 of the SPARQL query candidates, or otherwise it is assumed that the whole candidate list is not reflecting the user's intention. Hence, we refuse to answer the question. We answer the question if it is above a threshold θ_2 otherwise we do not answer it. Note that p_Q can be interpreted as the confidence that the QA system has in the generated SPARQL query Q, i.e. in the generated answer.

3.4.5 Multiple KBs

Note that the approach can also be extended, as it is, to multiple KBs. In the query expansion step, one has just to take in consideration the labels of all KBs. In the query construction step, one can consider multiple KBs as one graph having multiple unconnected components. The query ranking and answer decision step are literally the same.

3.4.6 Discussion

Overall, we follow a combinatorial approach with efficient pruning, that relies on the semantics encoded in the underlying KB.
In the following, we want to emphasize the advantages of this approach using some examples.

- **Joint disambiguation of entities and relations:** For example, for interpreting the question "How many inhabitants has Paris?" between the hundreds of different meanings of "Paris" and "inhabitants" the top ranked queries contain the resources called "Paris" which are cities, and the property indicating the population, because only these make sense semantically.

- **Portability to different KBs:** One problem in QA over KBs is the semantic gap, i.e. the difference between how we think that the knowledge is encoded in the KB and how it actually is. For example, in our approach, for the question "What is the capital of France?", we generate the query

```
SELECT ?x WHERE {
  dbr:France dbp:capital ?x .
}
```

  which probably most users would have expected, but also the query

```
SELECT ?x {
  VALUES ?x {dbr:List_of_capitals_of_France}
}
```

  which refers to an overview article in Wikipedia about the capitals of France and that most of the users would probably not expect. This important feature allows to port the approach to different KBs while it is independent of how the knowledge is encoded.

- **Ability to bridge over implicit relations:** We are able to bridge over implicit relations. For example, given "Give me German mathematicians" the following query is computed:

```
SELECT DISTINCT ?x WHERE {
  ?x  ?p1 dbr:Mathematician .
  ?x  ?p2 dbr:Germany .
}
```

  Here `?p1` is `dbo:field`, `dbo:occupation`, `dbo:profession` and `?p2` is `dbo:nationality`, `dbo:birthPlace`, `dbo:deathPlace`, `dbo:residence`. Note that all these properties could be intended for the given question.

- **Easy to port to new languages:** The only parts where the language is relevant are the stop word removal and stemming. Since these are very easy to adapt to new languages, one can port the approach easily to other languages.

- **Permanent system refinement:** It is possible to improve the system over time. The system generates multiple queries. This fact can be used to easily create new training

dataset as it is shown in [53]. Using these datasets one can refine the ranker to perform better on the asked questions.

- **System robust to malformed questions and keyword questions:** There are no NLP tools used in the approach which makes it very robust to malformed questions. For this reason, keyword questions are also supported.

A disadvantage of our exemplary implementation is that the identification of relations relies on a dictionary. Note that, non-dictionary based methods follow one of the following strategies. Either they try to learn ways to express the relation from big training corpora (like in [21]), s.t. the problem is shifted to create suitable training sets. Or text corpora are used to either extract lexicalizations for properties (like in [15]) or learn word-embeddings (like in [80]). Hence, possible improvements might be applied to this task in the future.

3.5 Fast candidate generation

In this section, we explain how the SPARQL queries described in section 3.4.2 can be constructed efficiently.

Let R be a set of resources. We consider the KB as a directed labeled graph G:

Definition 1. *(Graph) A directed labeled graph is an ordered pair $G = (V, E, f)$, such that:*

- *V is a non-empty set, called the vertex set;*
- *E is a set, called edge set, such that $E \subset \{(v, w) : v, w \in V\}$, i.e. a subset of the pairs of V;*
- *For a set L called labeled set, f is a function $f : E \to L$, i.e. a function that assigns to each edge a label $p \in L$. We indicate an edge with label p as $e = (v, p, w)$.*

To compute the pairwise distance in G between every resource in R, we do a breadth-first search from every resource in R in an undirected way (i.e. we traverse the graph in both directions).
We define a distance function d as follows. Assume we start from a vertex r and find the following two edges $e_1 = (r, p_1, r_1)$, $e_2 = (r_1, p_2, r_2)$. We say that $d_{r,p_1} = 1$, $d_{r,r_1} = 2$, $d_{r,p_2} = 3$ and so on. When an edge is traversed in the opposite direction, we add a minus sign. For example, given the edges $e_1 = (r, p_1, r_1)$ and $e_2 = (r_2, p_2, r_1)$, we say $d_{r,p_2} = -3$. For a vertex or edge r, and a variable x we artificially set $d_{r,x}$ to be any possible integer number. Moreover, we set $d_{x,y} = d_{y,x}$ for any x, y. The algorithm to compute these numbers can be found in Algorithm 1.

Data: Graph $G = (V, E, f)$ and a set R of edges and labels
Result: The pairwise distance between elements in R

for $r \in R \cap V$ **do**
  for e_1=$(r,p_1,r_1)$$\in E$ **do**
    if $p_1 \in R$ **then** $d_{r,p_1} = 1$; **if** $r_1 \in R$ **then** $d_{r,l_1} = 2$
    for $(e_2 = (r_1, p_2, r_2) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = 3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = 4$;
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = 2$: **if** $p_1, r_2 \in R$ **then** $d_{p_1,r_2} = 3$
    end
    for $(e_2 = (r_2, p_2, r_1) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = -3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = -4$
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = -2$; **if** $p_1, p_2 \in R$ **then** $d_{p_1,2_2} = -3$
    end
  end
  for e_1=$(r_1,p_1,r)$$\in E$ **do**
    if $p_1 \in R$ **then** $d_{r,p_1} = -1$; **if** $r_1 \in R$ **then** $d_{r,l_1} = -2$
    for $(e_2 = (r_1, p_2, r_2) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = 3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = 4$
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = 2$; **if** $p_1, r_2 \in R$ **then** $d_{p_1,r_2} = 3$
    end
    for $(e_2 = (r_2, p_2, r_1) \in E)$ **do**
      if $p_2 \in R$ **then** $d_{r,p_2} = 3$; **if** $r_2 \in R$ **then** $d_{r,2_2} = 4$
      if $p_1, p_2 \in R$ **then** $d_{p_1,p_2} = 2$; **if** $p_1, p_2 \in R$ **then** $d_{p_1,2_2} = 3$
    end
  end
end

Algorithm 1: Algorithm to compute the pairwise distance between every resource in a set R appearing in a KB.

Data: Graph $G = (V, E, f)$ and a set R of vertices and edges, and their pairwise distance d
Result: All connected triple patterns in G from a set R of vertices and edges with maximal K triple patterns

$L = \varnothing$ #list of triple patterns
$V_{s,o} = \varnothing$ #set of variables in subject, object position
$V_p = \varnothing$ #set of variables in predicate position
k=0
Function *generate (L,k)*
  for $s_1 \in (R \cap V) \cup V_{s,o} \cup \{x_{k,1}\}$ **do**
    for $s_2 \in (R \cap P) \cup V_p \cup \{x_{k,2}\}$ **do**
      for $s_3 \in (R \cap V) \cup V_{s,o} \cup \{x_{k,3}\}$ **do**
        if $k = 0 \wedge d_{s_2,s_3} = -1 \wedge d_{s_1,s_2} = 1 \wedge d_{s_1,s_3} = 2$ **then** $L \leftarrow L \cup \{(s_1, s_2, s_3)\}$
        for $T \in L^{(k)}$ **do**
          $b_1 = true; b_2 = true; b_3 = true; b_4 = true;$
          for $(t_1, t_2, t_3) \in T$ **do**
            if not $(s_1 = t_1 \wedge d_{t_1,s_2} = 2 \wedge d_{t_1,s_3} = 3 \wedge d_{t_2,s_2} = -2 \wedge d_{t_2,s_3} = -3 \wedge d_{t_3,s_2} = -3 \wedge d_{t_3,s_3} = -4)$ **then** $b_1 = false$
            if not $(s_1 = t_3 \wedge d_{t_1,s_2} = 3 \wedge d_{t_1,s_3} = 4 \wedge d_{t_2,s_2} = 2 \wedge d_{t_2,s_3} = 3 \wedge d_{t_3,s_2} = 1 \wedge d_{t_3,s_3} = 2)$ **then** $b_2 = false$
            if not $(s_3 = t_1 \wedge d_{t_1,s_2} = -1 \wedge d_{t_1,s_3} = -4 \wedge d_{t_2,s_2} = -2 \wedge d_{t_2,s_3} = -3 \wedge d_{t_3,s_2} = -1 \wedge d_{t_3,s_3} = -2)$ **then** $b_3 = false$
            if not $(s_3 = t_3 \wedge d_{t_1,s_2} = -3 \wedge d_{t_1,s_3} = 2 \wedge d_{t_2,s_2} = -2 \wedge d_{t_2,s_3} = -1 \wedge d_{t_3,s_2} = -1)$ **then** $b_4 = false$
          end
          if $b_1 = true \vee b_2 = true \vee b_3 = true \vee b_4 = true$ **then**
            $L \leftarrow L \cup (T \cup (s_1, s_2, s_3));$
            $V_{s,o} \leftarrow V_{s,o} \cup \{s_1, s_3\};$
            $V_p \leftarrow V \cup \{s_2\})$
          end
          if *(k!=K)* **then**
            return *generate(L,k+1)*
          end
        end
      end
    end
  end

Algorithm 2: Recursive algorithm to create all connected triple patterns from a set R of resources with maximal K triple patterns. L contains the triple patterns created recursively and $L^{(k)}$ indicates the triple patterns with exactly k triples. Note that the "if not" conditions very often are not fulfilled. This guarantees the speed of the process.

The algorithm of our exemplary implementation simply traverses the graph starting from the nodes in R in a breadth-first search manner and keeps track of the distances as defined above. The breadth-first search is done by using HDT [66] as an indexing structure[10]. Note that HDT was originally developed as an exchange format for RDF files that is queryable. A rarely mentioned feature of HDT is that it is perfectly suitable for performing breadth-first search operations over RDF data. In HDT, the RDF graph is stored as an adjacency list which is an ideal data structure for breadth-first search operations. This is not the case for traditional triple-stores. The use of HDT at this point is key for two reasons, (1) the performance of the breadth-first search operations, and (2) the low footprint of the index in terms of disk and memory space. Roughly, a 100 GB RDF dump can be compressed to a HDT file of a size of approx. 10 GB [66].
Based on the numbers above, we now want to construct all triple patterns with K triples and one projection variable recursively. Given a triple pattern T, we only want to build connected triple-pattern while adding triples to T. This can be done recursively using the algorithm described in Algorithm 2. Note that thanks to the numbers collected during the breadth-first search operations, this can be performed very fast. Once the triple patterns are constructed, one can choose any of the variables, which are in subject or object position, as a projection variable.

3.6 Evaluation

To validate the approach w.r.t. multilinguality, portability and robustness, we evaluated our approach using multiple benchmarks for QA that appeared in the last years. The different benchmarks are not comparable and they focus on different aspects of QA. For example SimpleQuestions focuses on questions that can be solved by one simple triple-pattern, while LC-QuAD focuses on more complex questions. Moreover, the QALD questions address different challenges including multilinguality and the use of keyword questions. Unlike previous works, we do not focus on one benchmark, but we analyze the behaviour of our approach under different scenarios.
We tested our approach on 5 different datasets namely Wikidata[11], DBpedia[12], MusicBrainz[13], DBLP[14] and Freebase[15]. Moreover, we evaluated our approach on five different languages namely: English, German, French, Italian and Spanish. First, we describe how we selected stop words and collected lexicalizations for the different languages and KBs, then we describe and discuss our results.

[10] https://www.w3.org/Submission/2011/03/
[11] https://www.wikidata.org/
[12] http://dbpedia.org
[13] https://musicbrainz.org
[14] http://dblp.uni-trier.de
[15] https://developers.google.com/freebase/

3.6.1 Stop Words and lexicalizations

As stop words, we use the lists, for the different languages, provided by Lucene, together with some words which are very frequent in questions like "what", "which", "give".
Depending on the KB, we followed different strategies to collect lexicalizations. Since Wikidata has a rich number of lexicalizations, we simply took all lexicalizations associated to a resource through `rdfs:label`[16], `skos:prefLabel`[17] and `skos:altLabel`. For DBpedia, we only used the English DBpedia, where first all lexicalizations associated to a resource through the `rdfs:label` property were collected. Secondly, we followed the disambiguation and redirect links to get additional ones and took also into account available demonyms `dbo:demonym` (i.e. to `dbr:Europe` we associate also the lexicalization "European"). Thirdly, by following the inter-language links, we associated the labels from the other languages to the resources. DBpedia properties are poorly covered with lexicalizations, especially when compared to Wikidata. For example, the property `dbo:birthPlace` has only one lexicalization namely "birth place", while the corresponding property over Wikidata `P19` has 10 English lexicalizations like "birthplace", "born in", "location born", "birth city". In our exemplary implementation two strategies were implemented. First, while aiming at a QA system for the Semantic Web we also can take into account interlinkings between properties of distinguished KBs, s.t. lexicalizations are merged from all KBs currently considered. There, the `owl:sameAs` links from DBpedia relations to Wikidata are used and every lexicalization present in Wikidata is associated to the corresponding DBpedia relation. Secondly, the DBpedia abstracts are used to find more lexicalizations for the relations. To find new lexicalizations of a property p we follow the strategy proposed by [75]. We extracted from the KB the subject-object pairs *(x,y)* that are connected by p. Then the abstracts are scanned and all sentences are retrieved which contain both $label(x)$ and $label(y)$. At the end, the segments of text between $label(x)$ and $label(y)$, or $label(y)$ and $label(x)$ are extracted. We rank the extracted text segments and we choose the most frequent ones. This was done only for English.
For MusicBrainz we used the lexicalizations attached to `purl:title`[18], `foaf:name`[19], `skos:altLabel` and `rdfs:label`. For DBLP only the one attached to `rdfs:label`. Note, MusicBrainz and DBLP contain only few properties. We aligned them manually with Wikidata and moved the lexicalizations from one KB to the other. The mappings can be found under `http://goo.gl/ujbwFW` and `http://goo.gl/ftzegZ` respectively. This took in total 1 hour of manual work.
For Freebase, we considered the lexicalizations attached to `rdfs:label`. We also followed the few available links to Wikidata. Finally, we took the 20 most prominent properties in the training set of the SimpleQuestions benchmark and looked at the lexicalizations of them in the first 100 questions of SimpleQuestions. We extracted manually the lexicalizations

[16] `rdfs: http://www.w3.org/2000/01/rdf-schema#`
[17] `skos: http://www.w3.org/2004/02/skos/core#`
[18] purl: http://purl.org/dc/elements/1.1/
[19] foaf: http://xmlns.com/foaf/

for them. This took 1 hour of manual work. We did not use the other (75810 training and 10845 validation) questions.

3.6.2 Experiments

To show the performance of the approach on different scenarios, we benchmarked it using the following benchmarks.

Benchmarks

QALD: We evaluated our approach using the QALD benchmarks. These benchmarks are good to see the performance on multiple-languages and over both full-natural language questions and keyword questions. We followed the metrics of the original benchmarks. Note that the metrics changed in QALD-7. The results are given in Table 3.3 together with state-of-the-art systems. To find these, we used Google Scholar to select all publications about QA systems that cited one of the QALD challenge publications. Note that, in the past, QA systems were evaluated only on one or two of the QALD benchmarks. We provide, for the first time, an estimation of the differences between the benchmark series. Over English, we outperformed 90% of the proposed approaches. We do not beat Xser [170] and UTQA [122]. Note that these systems required additional training data than the one provided in the benchmark, which required a significant cost in terms of manual effort. Moreover, the robustness of these systems over keyword questions is probably not guaranteed. We cannot prove this claim because for these systems neither the source code nor a web-service is available.
Due to the manual effort required to do an error analysis for all benchmarks and the limited space, we restricted to the QALD-6 benchmark. The error sources are the following. 40% are due to lexical gap (e.g. for "Who played Gus Fring in Breaking Bad?" the property `dbo:portrayer` is expected), 28% come from wrong ranking, 12% are due to the missing support of superlatives and comparatives in our implementation (e.g. "Which Indian company has the most employees?"), 9% from the need of complex queries with unions or filters (e.g. the question "Give me a list of all critically endangered birds." requires a filter on `dbo:conservationStatus` equal "CR"), 6% come from out of scope questions (i.e. question that should not be answered), 2% from too ambiguous questions (e.g. "Who developed Slack?" is expected to refer to a "cloud-based team collaboration tool" while we interpret it as "linux distribution"). One can see that keyword queries always perform worst as compared to full natural language queries. The reason is that the formulation of the keyword queries does not allow to decide if the query is an ASK query or if a COUNT is needed (e.g. "Did Elvis Presley have children?" is formulated as "Elvis Presley, children"). This means that we automatically get these questions wrong.

To show the performance over Wikidata, we consider the QALD-7 task 4 training dataset. This originally provided only English questions. The QALD-7 task 4 training dataset reuses questions over DBpedia from previous challenges where translations in other languages were available. We moved these translations to the dataset. The results can be seen in Table 3.4. Except for English, keyword questions are easier than full natural language questions. The reason is the formulation of the questions. For keyword questions the lexical gap is smaller. For example, the keyword question corresponding to the question "Qui écrivit Harry Potter?" is "écrivain, Harry Potter". Stemming does not suffice to map "écrivit" to "écrivain", lemmatization would be needed. This problem is much smaller for English, where the effect described over DBpedia dominates. We can see that the best performing language is English, while the worst performing language is Italian. This is mostly related to the poorer number of lexicalizations for Italian. Note that the performance of the QA approach over Wikidata correlates with the number of lexicalizations for resources and properties for the different languages as described in [91]. This indicates that the quality of the data, in different languages, directly affects the performance of the QA system. Hence, we can derive that our results will probably improve while the data quality is increased. Finally we outperform the presented QA system over this benchmark.

SimpleQuestions: SimpleQuestions contains 108442 questions that can be solved using one triple pattern. We trained our system using the first 100 questions in the training set. The results of our system, together with the state-of-the-art systems are presented in Table 3.5. For this evaluation, we restricted the generated queries with one triple-pattern. The system performance is 14% below the state-of-the-art. Note that we achieve this result by considering only 100 of the 75810 questions in the training set, and investing 1 hour of manual work for creating lexicalizations for properties manually. Concretely, instead of generating a training dataset with 80.000 questions, which can cost several thousands of euros, we invested 1 hour of manual work with the result of loosing (only) 14% in accuracy! Note that the SimpleQuestions dataset is highly skewed towards certain properties (it contains 1629 properties, the 20 most frequent properties cover nearly 50% of the questions). Therefore, it is not clear how the other QA systems behave with respect to properties not appearing in the training dataset and with respect to keyword questions. Moreover, it is not clear how to port the existing approaches to new languages and it is not possible to adapt them to more difficult questions. These points are solved using our approach. Hence, we provided here, for the first time, a quantitative analysis of the impact of big training data corpora on the quality of a QA system.

LC-QuAD & WDAquaCore0Questions: Recently, a series of new benchmarks have been published. LC-QuAD [152] is a benchmark containing 5000 English questions and it concentrates on complex questions. WDAquaCore0Questions [53] is a benchmark containing 689 questions over multiple languages and addressing mainly Wikidata, generated from the logs of a live running QA system. The questions are a mixture of real-world keyword and malformed questions. In Table 3.6, we present the first baselines for these benchmarks.

QA system	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-3								
WDAqua-core1	**en**	**full**	**100**	**0.64**	**0.42**	**0.51**	**1.01**	-
WDAqua-core1	**en**	**key**	**100**	**0.71**	**0.37**	**0.48**	**0.79**	-
WDAqua-core1	**de**	**key**	**100**	**0.79**	**0.31**	**0.45**	**0.22**	-
WDAqua-core1	**de**	**full**	**100**	**0.79**	**0.28**	**0.42**	**0.30**	-
WDAqua-core1	**fr**	**key**	**100**	**0.83**	**0.27**	**0.41**	**0.26**	-
gAnswer [188]∗	en	full	100	0.40	0.40	0.40	≈ 1 s	[188]
WDAqua-core1	**fr**	**full**	**100**	**0.70**	**0.26**	**0.38**	**0.37**	-
WDAqua-core1	**es**	**full**	**100**	**0.77**	**0.24**	**0.37**	**0.27**	-
WDAqua-core1	**it**	**full**	**100**	**0.79**	**0.23**	**0.36**	**0.30**	-
WDAqua-core1	**it**	**key**	**100**	**0.84**	**0.23**	**0.36**	**0.24**	-
WDAqua-core1	**es**	**key**	**100**	**0.80**	**0.23**	**0.36**	**0.23**	-
RTV [76]	en	full	99	0.32	0.34	0.33	-	[28]
Intui2 [59]	en	full	99	0.32	0.32	0.32	-	[28]
SINA [132]∗	en	full	100	0.32	0.32	0.32	≈ 10-20s	[132]
DEANNA [172]∗	en	full	100	0.21	0.21	0.21	≈ 1-50 s	[188]
SWIP [124]	en	full	99	0.16	0.17	0.17	-	[28]
Zhu et al. [187]∗	en	full	99	0.38	0.42	0.38	-	[187]
QALD-4								
Xser [170]	en	full	50	0.72	0.71	0.72	-	[158]
WDAqua-core1	**en**	**key**	**50**	**0.76**	**0.40**	**0.52**	**0.32s**	-
WDAqua-core1	**en**	**full**	**50**	**0.56**	**0.30**	**0.39**	**0.46s**	-
gAnswer [188]	en	full	50	0.37	0.37	0.37	0.973 s	[158]
CASIA [84]	en	full	50	0.32	0.40	0.36	-	[158]
WDAqua-core1	**de**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.04s**	-
WDAqua-core1	**fr**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.06s**	-
WDAqua-core1	**it**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.04s**	-
WDAqua-core1	**es**	**key**	**50**	**0.92**	**0.20**	**0.33**	**0.05s**	-
WDAqua-core1	**de**	**full**	**50**	**0.90**	**0.20**	**0.32**	**0.06s**	-
WDAqua-core1	**it**	**full**	**50**	**0.92**	**0.20**	**0.32**	**0.16s**	-
WDAqua-core1	**es**	**full**	**50**	**0.90**	**0.20**	**0.32**	**0.06s**	-
WDAqua-core1	**fr**	**full**	**50**	**0.86**	**0.18**	**0.29**	**0.09s**	-
Intui3 [58]	en	full	50	0.23	0.25	0.24	-	[158]
ISOFT [121]	en	full	50	0.21	0.26	0.23	-	[158]
Hakimov [81]∗	en	full	50	0.52	0.13	0.21	-	[81]
QALD-5								
Xser [170]	en	full	50	0.74	0.72	0.73	-	[157]
UTQA [122]	en	full	50	-	-	0.65	-	[122]
UTQA [122]	es	full	50	0.55	0.53	0.54	-	[122]
UTQA [122]	fs	full	50	0.53	0.51	0.52	-	[122]
WDAqua-core1	**en**	**full**	**50**	**0.56**	**0.41**	**0.47**	**0.62s**	-
WDAqua-core1	**en**	**key**	**50**	**0.60**	**0.27**	**0.37**	**0.50s**	-
AskNow[61]	en	full	50	0.32	0.34	0.33		[61]
QAnswer[130]	en	full	50	0.34	0.26	0.29	-	[157]
WDAqua-core1	**de**	**full**	**50**	**0.92**	**0.16**	**0.28**	**0.20s**	-
WDAqua-core1	**de**	**key**	**50**	**0.90**	**0.16**	**0.28**	**0.19s**	-
WDAqua-core1	**fr**	**full**	**50**	**0.90**	**0.16**	**0.28**	**0.19s**	-
WDAqua-core1	**fr**	**key**	**50**	**0.90**	**0.16**	**0.28**	**0.18s**	-
WDAqua-core1	**it**	**full**	**50**	**0.88**	**0.18**	**0.30**	**0.20s**	-
WDAqua-core1	**it**	**key**	**50**	**0.90**	**0.16**	**0.28**	**0.18s**	-
WDAqua-core1	**es**	**full**	**50**	**0.88**	**0.14**	**0.25**	**0.20s**	-
WDAqua-core1	**es**	**key**	**50**	**0.90**	**0.14**	**0.25**	**0.20s**	-
SemGraphQA[13]	en	full	50	0.19	0.20	0.20	-	[157]
YodaQA[12]	en	full	50	0.18	0.17	0.18	-	[157]
QuerioDali[98]	en	full	50	?	?	?	?	[98]

QA system	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-6								
WDAqua-core1	**en**	**full**	**100**	**0.55**	**0.34**	**0.42**	**1.28s**	-
WDAqua-core1	**de**	**full**	**100**	**0.73**	**0.29**	**0.41**	**0.41s**	-
WDAqua-core1	**de**	**key**	**100**	**0.85**	**0.27**	**0.41**	**0.30s**	-
WDAqua-core1	**en**	**key**	**100**	**0.51**	**0.30**	**0.37**	**1.00s**	-
SemGraphQA [13]	en	full	100	0.70	0.25	0.37	-	[159]
WDAqua-core1	**fr**	**key**	**100**	**0.78**	**0.23**	**0.36**	**0.34s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.57**	**0.22**	**0.32**	**0.46s**	-
WDAqua-core1	**es**	**full**	**100**	**0.69**	**0.19**	**0.30**	**0.45s**	-
WDAqua-core1	**es**	**key**	**100**	**0.83**	**0.18**	**0.30**	**0.35s**	-
WDAqua-core1	**it**	**key**	**100**	**0.75**	**0.17**	**0.28**	**0.34s**	-
AMUSE [80]	en	full	100	-	-	0.26	-	[80]
WDAqua-core1	**it**	**full**	**100**	**0.62**	**0.15**	**0.24**	**0.43s**	-
AMUSE [80]	es	full	100	-	-	0.20	-	[80]
AMUSE [80]	de	full	100	-	-	0.16	-	[80]
QALD-7								
WDAqua-core1	**en**	**full**	**100**	**0.25**	**0.28**	**0.25**	**1.24s**	-
WDAqua-core1	**fr**	**key**	**100**	**0.18**	**0.16**	**0.16**	**0.32s**	-
WDAqua-core1	**en**	**key**	**100**	**0.14**	**0.16**	**0.14**	**0.88s**	-
WDAqua-core1	**de**	**full**	**100**	**0.10**	**0.10**	**0.10**	**0.34s**	-
WDAqua-core1	**de**	**key**	**100**	**0.12**	**0.10**	**0.10**	**0.28s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.12**	**0.10**	**0.10**	**0.42s**	-
WDAqua-core1	**it**	**key**	**100**	**0.06**	**0.06**	**0.06**	**0.28s**	-
WDAqua-core1	**it**	**full**	**100**	**0.04**	**0.04**	**0.04**	**0.34s**	-

Table 3.3: This table (left column and upper right column) summarizes the results obtained by the QA systems evaluated with QALD-3 (over DBpedia 3.8), QALD-4 (over DBpedia 3.9), QALD-5 (over DBpedia 2014), QALD-6 (over DBpedia 2015-10), QALD-7 (2016-04). We indicated with "$\star$" the systems that did not participate directly in the challenges, but were evaluated on the same benchmark afterwards. We indicate the average running times of a query for the systems where we found them. Even if the runtime evaluations were executed on different hardware, it still helps to give an idea about the scalability.

QA System	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-7 task 4, training dataset								
WDAqua-core1	**en**	**full**	**100**	**0.37**	**0.39**	**0.37**	**1.68s**	-
WDAqua-core1	**en**	**key**	**100**	**0.35**	**0.38**	**0.35**	**0.80s**	-
WDAqua-core1	**es**	**key**	**100**	**0.31**	**0.32**	**0.31**	**0.45s**	-
Sorokin et al. [138]	en	full	100	-	-	0.29	-	[138]
WDAqua-core1	**de**	**key**	**100**	**0.27**	**0.28**	**0.27**	**1.13s**	-
WDAqua-core1	**fr**	**key**	**100**	**0.27**	**0.30**	**0.27**	**1.14s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.27**	**0.31**	**0.27**	**1.05s**	-
WDAqua-core1	**es**	**full**	**100**	**0.24**	**0.26**	**0.24**	**0.65s**	-
WDAqua-core1	**de**	**full**	**100**	**0.18**	**0.20**	**0.18**	**0.82s**	-
WDAqua-core1	**it**	**full**	**100**	**0.19**	**0.20**	**0.18**	**1.00s**	-
WDAqua-core1	**it**	**key**	**100**	**0.17**	**0.18**	**0.16**	**0.44s**	-

Table 3.4: The table shows the results of WDAqua-core1 over the QALD-7 task 4 training dataset. We used Wikidata (dated 2016-11-28).

QA System	Lang	Type	Total	Accuracy	Runtime	Ref
WDAqua-core1*	**en**	**full**	**21687**	**0.571**	**2.1 s**	**-**
Dai et al.*	en	full	21687	0.626	-	[33]
Bordes et al.	en	full	21687	0.627	-	[21]
Yin et al.	en	full	21687	0.683	-	[181]
Golub and He	en	full	21687	0.709	-	[78]
Lukovnikov et al.	en	full	21687	0.712	-	[104]

Table 3.5: This table summarizes the QA systems evaluated over SimpleQuestions. Every system was evaluated over FB2M except the ones marked with (∗) which were evaluated over FB5M.

Benchmark	Lang	Type	Total	P	R	F	Runtime
LC-QuAD	en	full	5000	0.59	0.38	0.46	1.5 s
WDAquaCore0Questions	mixed	mixed	689	0.79	0.46	0.59	1.3 s

Table 3.6: This table summarizes the results of WDAqua-core1 over some newly appeared benchmarks.

Multiple KBs: The only available benchmark that tackles multiple KBs was presented in QALD-4 task 2. The KBs are rather small and perfectly interlinked. This is not the case over the considered KBs. We therefore evaluated the ability to query multiple KBs differently. We run the questions of the QALD-6 benchmark, which was designed for DBpedia, both over DBpedia (only) and over DBpedia, Wikidata, MusicBrainz, DBLP and Freebase. Note that, while the original questions have a solution over DBpedia, a good answer could also be found over the other datasets. We therefore manually checked whether the answers that were found in other KBs are right (independently from which KB was chosen by the QA system to answer it). The results are presented in Table 3.7. WDAqua-core1 choose 53 times to answer a question over DBpedia, 39 over Wikidata and the other 8 times over a different KB. Note that we get better results when querying multiple KBs. Globally we get better recall and lower precision which is expected. While scalability is an issue, we are able to pick the right KB to find the answer!

Note: We did not tackle the WebQuestions benchmark for the following reasons. While it has been shown that WebQuestions can be addressed using non-reified versions of Freebase, this was not the original goal of the benchmark. More then 60% of the QA systems benchmarked over WebQuestions are tailored towards its reefication model. There are two important

Dataset	Lang	Type	Total	P	R	F	Runtime
DBpedia	en	full	100	0.55	0.34	0.42	1.37 s
All KBs supported	en	full	100	0.49	0.39	0.43	11.94s

Table 3.7: Comparison on QALD-6 when querying only DBpedia and multiple KBs at the same time.

points here. First, most KBs in the Semantic Web use binary statements. Secondly, in the Semantic Web community, many different reefication models have been developed as described in [77].

Setting

All experiments were performed on a virtual machine with 4 core of Intel Xeon E5-2667 v3 3.2GH, 16 GB of RAM and 500 GB of SSD disk. Note that the whole infrastructure was running on this machine, i.e. all indexes and the triple-stores needed to compute the answers (no external service was used). The original data dumps sum up to 336 GB. Note that across all benchmarks we can answer a question in less then 2 seconds except when all KBs are queried at the same time which shows that the algorithm should be parallelized for further optimization.

3.7 Provided Services for Multilingual and Multi-KB QA

We have presented an algorithm that can be easily ported to new KBs and that can query multiple KBs at the same time. In the evaluation section, we have shown that our approach is competitive while offering the advantage of being multilingual and robust to keyword questions. Moreover, we have shown that it runs on moderate hardware. In this section, we describe how we integrated the approach to an actual service and how we combine it to existing services so that it can be directly used by end-users.

First, we integrated WDAqua-core1 into Qanary [22, 50], a framework to integrate QA components. This way WDAqua-core1 can be accessed via RESTful interfaces for example to benchmark it via Gerbil for QA[162]. It also allows to combine it with services that are already integrated into Qanary like a speech recognition component based on Kaldi[20] and a language detection component based on [117]. Moreover, the integration into Qanary allows to reuse Trill [46], a reusable front-end for QA systems. A screenshot of Trill using in the back-end WDAqua-core1 can be found in Figure 4.1.
Secondly, we reused and extended Trill to make it easily portable to new KBs. While Trill originally was supporting only DBpedia and Wikidata, now it supports also MusicBrainz, DBLP and Freebase. We designed the extension so that it can be easily ported to new KBs. Enabling the support to a new KB is mainly reduced to writing an adapted SPARQL query for the new KB. Additionally, the extension allows to select multiple KBs at the same time.
Thirdly, we adapted some services that are used in Trill to be easily portable to new KBs. These include SPARQLToUser [32], a tool that generates a human readable version of a

[20] http://kaldi-asr.org

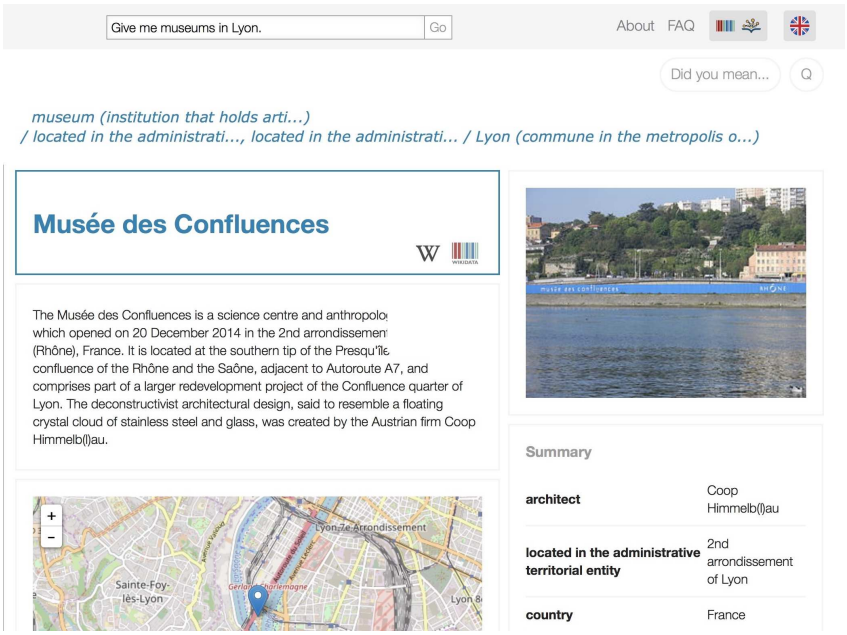

Figure 3.4: Screenshot of Trill, using in the back-end WDAqua-core1, for the question "Give me museums in Lyon.".

SPARQL query and LinkSUM [144] a service for entity summarization. All these tools now support the 5 mentioned KBs and the 5 mentioned languages.
A public online demo is available under `www.wdaqua.eu/qa`.

3.8 Conclusion and Future Work

In this paper, we introduced a novel concept for QA aimed at multilingual and KB-agnostic QA. Due to the described characteristics of our approach portability is ensured which is a significant advantage in comparison to previous approaches. We have shown the power of our approach in an extensive evaluation over multiple benchmarks. Hence, we clearly have shown our contributions w.r.t. qualitative (language, KBs) and quantitative improvements (outperforming many existing systems and querying multiple KBs) as well as the capability of our approach to scale for very large KBs like DBpedia.
We have applied our algorithm and adapted a set of existing services so that end-users can query, using multiple languages, multiple KBs at the same time, using an unified interface. Hence, we provided here a major step towards QA over the Semantic Web following our larger research agenda of providing QA over the LOD cloud.
In the future, we want to tackle the following points. First, we want to parallelize our approach, s.t. when querying multiple KBs acceptable response times will be achieved. Secondly, we want to query more and more KBs (hints to interesting KBs are welcome). Thirdly, from different lessons learned from querying multiple KBs, we want to give a set of recommendations for RDF datasets, s.t. they are fit for QA. And fourth, we want to extend our approach to also query reefied data. Fifth, we would like to extend the approach to be

able to answer questions including aggregates and functions. We believe that our work can further boost the expansion of the Semantic Web since we presented a solution that easily allows to consume RDF data directly by end-users requiring low hardware investments.

4 Trill: A Reusable and Easily Portable Front-End for Question Answering Systems over KB

4.1 Trill: A reusable Front-End for QA systems[1]

4.1.1 Abstract

The Semantic Web contains an enormous amount of information in the form of knowledge bases. To make this information available to end-users many question answering (QA) systems over knowledge bases were created in the last years. Their goal is to enable users to access large amounts of structured data in the Semantic Web by bridging the gap between natural language and formal query languages like SPARQL.
But automatically generating a SPARQL query from a user's question is not sufficient to bridge the gap between Semantic Web data and the end-users. The result of a SPARQL query consists of a list of URIs and/or literals, which is not a user-friendly presentation of the answer. Such a presentation includes the representation of the URI in the right language and additional information like images, maps, entity summaries and more.
We present Trill, the first reusable user-interface (UI) for QA systems over knowledge bases supporting text and audio input, able to present answers from DBpedia and Wikidata in 4 languages (English, French, German, and Italian). It is designed to be used together with Qanary, an infrastructure for composing QA pipelines. This front-end enables the QA community to show their results to end-users and enables the research community to explore new research directions like studying and designing user-interactions with QA systems.
Keywords: Question Answering Systems, Front-End, User Interaction, Answer Presentation

4.1.2 Introduction

Users' experience is an important factor for the success of a given application. Thus, the front-end of QA systems, which highly impacts the users' experience, is an important part of a QA system. On one side the research community has made a lot of efforts to increase the

[1] Corresponding publications is: Dennis Diefenbach, Shanzay Amjad, Andreas Both, Kamal Singh, Pierre Maret, *Trill: A reusable Front-End for QA systems*, ESWC 2016 Poster and Demo [46]

F-score over different benchmarks, i.e., the accuracy of the translation of a natural language question in a formal representation like a SPARQL query. On the other side not much attention has been paid to how the answer is presented and how users can interact with a QA system.
On a technical level one could say that a lot of attention has been paid to services in the back-end but little attention has been paid to the front-end. We hope to change this trend by presenting the first reusable front-end for QA systems over knowledge bases, i.e. a front-end that can be reused easily by any new QA system. It can currently be used for QA systems over DBpedia and Wikidata and supports 4 different languages.

4.1.3 Related work

In the last years a large number of QA systems were created by the research community. This is for example shown by the number of QA systems (more then 20 in the last 5 years) that were evaluated against the QALD benchmark [2]. Unfortunately only very few of them have a UI and even fewer are available as web services. Some exceptions that we are aware of are *SINA*[132], QAKiS[27] and *Platypus*[3]. Also industrial UI were created for well known systems like *Apple Siri*, *Microsoft Cortana*, *Google Search*, *Ok Google*, and *WolframAlpha*[4].
All the UIs mentioned above are tightly integrated with the corresponding QA systems. This is not the case for Trill. Trill can be used as a front-end for new QA systems. To achieve the modularity by making the front-end independent from the back-end of the QA system, we build Trill on top of a framework that provides a reusable architecture for QA systems. We are aware of four such architectures namely: QALL-ME [67], openQA [109], the Open KnowledgeBase and Question-Answering (OKBQA) challenge[5] and Qanary [133, 22, 51]. We choose to build Trill on top of the last mentioned framework, Qanary, since it allows the highest flexibility to integrate new QA components.

4.1.4 Description of Trill

In this section, we first describe the interaction of the front-end with the Qanary back-end. Then we describe which functionalities / presentational elements are implemented in the front-end. A screenshot showing most of the presentational elements is shown in Figure 4.1. A live demo can be found at `www.wdaqua.eu/qa`.
Trill builds on the APIs offered by Qanary. Qanary is a framework for composing QA

[2] `http://qald.sebastianwalter.org`
[3] `http://sina.aksw.org`, `http://live.ailao.eu`, `http://qakis.org/qakis2/`, `https://askplatyp.us/?`
[4] `http://www.apple.com/ios/siri`, `http://www.wolframalpha.com`, `http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana`, `www.google.com` `https://support.google.com/websearch/answer/2940021?hl=en`
[5] `http://www.okbqa.org/`

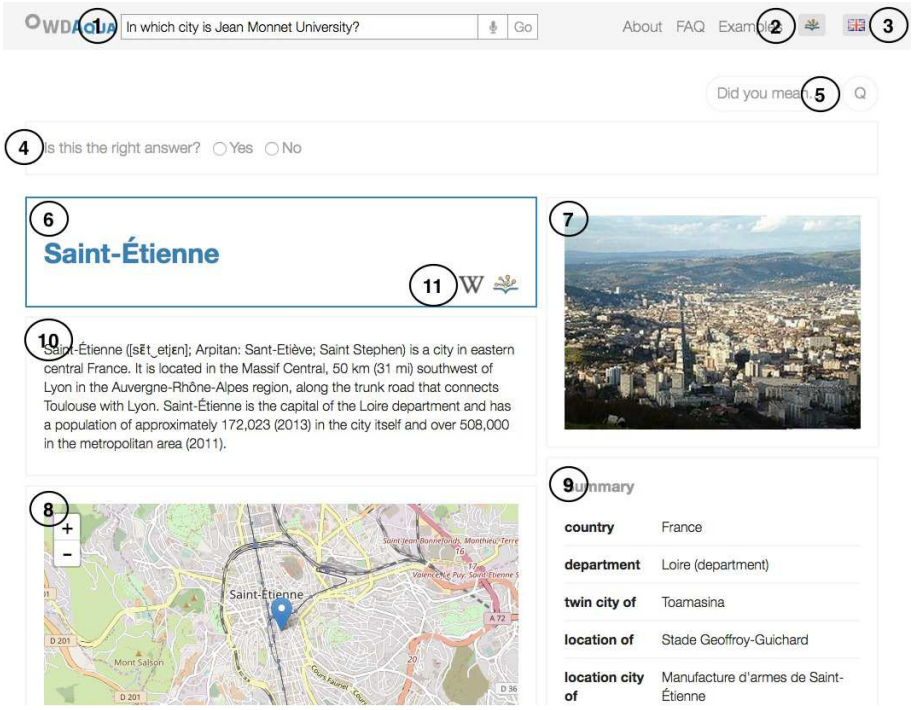

Figure 4.1: Screenshot of Trill for the question *In which city is Jean Monnet University?*. The numbers refer to the item list in Section 4.1.4

pipelines. A QA system can be seen as a pipeline, i.e., as a sequence of services that are called one after the other and finally are able (or not) to compute the answer for a given question. Qanary has APIs that allow to combine components integrated in Qanary starting from a text or an audio question. The human accessible versions are exposed for example under:

`www.wdaqua.eu/qanary/startquestionansweringwithtextquestion`,
`www.wdaqua.eu/qanary/startquestionansweringwithaudioquestion`

A call to these APIs require as parameters both the question and a list of Qanary components. A call will activate the mentioned components which will be executed as a pipeline in the defined order. Trill accesses the Qanary back-end using these APIs. In particular the demo running under `www.wdaqua.eu/qa` calls a Qanary component called WDAqua-core0 [52] created in the frame of the WDAqua project[6].
Now we give a list of all the functionalities and presentational elements that are available in Trill. The numbers in the item list correspond to the numbers in Figure 4.1.

1. **Text and audio input**: The front-end contains a search-bar component. This allows the user to input a text question using a text bar or a vocal question using a microphone[7].

[6] `http://www.wdaqua.eu`
[7] This feature is supported only for Chrome and Firefox due to dependencies on the MediaStream Recording API.

2. **Language selection**: The user has the possibility to change between 4 languages (English, French, German, and Italian) by selecting the flag of the corresponding country. The whole webpage will re-render using labels of the selected language and also the answers to a question will be adapted to the corresponding language.

3. **Knowledge-base selection**: The user has the option to choose an available knowledge base to query. Actually this is limited to DBpedia and Wikidata. Note that the answer presentation is coupled to the knowledge base since different properties are used to express the same information, i.e., the image can be retrieved in DBpedia using the property `dbo:thumbnail` while in Wikidata it is the property `wdt:P18`.

4. **Feedback functionality**: We implemented an easy-to-use feedback functionality where a user can choose if the answer is correct or not. This information is logged in the back-end and can be used to create a training set.

   Answer presentation We implemented several components to display information about the answer that can be useful for a user. Here is a list.

   5. **Generated query**: For expert users, we give the possibility to see the generated SPARQL query for the question.

   6. **Label** In the case the answer is a URI, we display its label in the corresponding language, i.e., `wd:Q90` will be displayed as *Paris* in English and *Parigi* in Italian. If the answer is a literal, we display it directly.

   7. **Image** One component retrieves the image (if it exists) of the answer and displays it.

   8. **Map** If available the geo-coordinates are used to display the position of the answer on a map. The map is showed using Leaflet[8] Javascript Library which retrieves maps from OpenStreetMap[9].

   9. **Top-k Properties** A summary of the entity is visualized using an external service called LinkSUM[145] provided by Andreas Thalhammer.

   10. **Text description from Wikipedia abstract**: Every entity in DBpedia is linked to the corresponding Wikipedia page and for many Wikidata entities this is also the case. Following these links we use the Wikipedia API to retrieve the introduction of the article.

[8] http://leafletjs.com
[9] http://openstreetmap.org/

	Audio input	Language sel.	KB sel.	Feedback func	Generated query	Entity label	Image	Map	Top-k properties	Text description	External links	Language support	KB support
SINA						x	x	x		x	x	en	dbpedia
QAKiS			x			x	x				x	en	dbpedia
Platypus	x					x	x	x		x	x	en	wikidata
Trill	x	x	x	x	x	x	x	x	x	x	x	en, de, fr, it	dbpedia, wikidata

Table 4.1: Table comparing the features of Trill with other open source UIs.

11. **External Links** If available, we present some external links to the user. This include links to DBpedia, Wikidata and Wikipedia.

12. **Ranking of list answers** In the case of a list of answers, we rank them in the case of DBpedia. For example, if a user asks about *italian lakes* the result will be a long list of answers. Without ranking, the first answer will probably be a small unknown lake in Italy. For this reason we reorder the results using the page rank scores presented in [149] that are available online[10]. Hence, if a user asks for *italian lakes*, he will get at a first position in the result list the entity *Lake Como* followed by *Lake Maggiore*.

The code can be found under `https://github.com/WDAqua/Trill`. Table 4.1 compares the features of other open source front-ends with Trill.

4.1.5 Conclusion

We have presented Trill, a reusable user-interface for QA systems over knowledge bases that can be used on top of Qanary. While many efforts were made to develop better QA technology in the back-end, little work was done in the front-end.
We hope that Trill can change this trend. Trill can be used for any future QA pipeline integrated into Qanary as an off-the-shelf front-end. Moreover, it can be used to deeply study the interactions of the end-users with QA systems. This includes: 1. collect end-user queries to create easily large and realistic benchmarks, 2. studies to analyze which questions users ask, 3. study how much context information should be presented to a user together with the answer, 4. create interfaces, like the disambiguation interfaces in [47], to allow users to interact with QA systems. These examples shows how advances on the front-end

[10] http://people.aifb.kit.edu/ath/#DBpedia_PageRank

can also be beneficial for classical QA research in the back-end.

4.2 PageRank and Generic Entity Summarization for RDF Knowledge Bases [11]

4.2.1 Abstract

Ranking and entity summarization are operations that are tightly connected and recurrent in many different domains. Possible application fields include information retrieval, question answering, named entity disambiguation, co-reference resolution, and natural language generation. Still, the use of these techniques is limited because there are few accessible resources. PageRank computations are resource-intensive and entity summarization is a complex research field in itself.

We present two generic and highly re-usable resources for RDF knowledge bases: a component for PageRank-based ranking and a component for entity summarization. The two components, namely PAGERANKRDF and SUMMASERVER, are provided in form of open source code along with example datasets and deployments. In addition, this work outlines the application of the components for PageRank-based RDF ranking and entity summarization in the question answering project WDAqua.

Keywords: RDF, ranking, PageRank, entity summarization, question answering, linked data

4.2.2 Introduction

PageRank scores and entity summaries are important tools in many applications that are relying on RDF data. We want to start with concrete examples in the question answering domain:

PageRank scores can be used as a feature to disambiguate between resources. Suppose a user asks just for “River". While there are many different meanings for “River" like a film or a village, the most probable one is the one referring to a natural watercourse. PageRank scores can be used in this context to rank the different meanings of “River" and to present to the user the most probable one. Another possible application of PageRank scores is the ranking of an answer set. Suppose a user asks “Which lakes are located in Italy?” Without any ranking, the resulting list could easily start with an unknown lake like “Lake Reschen”. This is probably not very relevant information for the user. By ranking the answer set properly, like in the information retrieval context, the usefulness of the answer for the user is increased.

[11] Corresponding publications is: Dennis Diefenbach, Andreas Thalhammer, *PageRank and Generic Entity Summarization for RDF Knowledge Bases*, 2018 ESWC [56]

Entity summaries can be presented to the user together with the answer to enrich the context. Moreover they can also be useful for increasing the discoverability of the dataset in the sense that the user can explore suggested properties of the answer. These examples show the importance of PageRank based ranking and entity summaries in the question answering domain.

On one side, PageRank-based ranking and entity summaries can be essential tools in many domains like information retrieval, named entity disambiguation [151], entity linking[147], co-reference resolution, and natural language generation. On the other side, PageRank computations are resource-intensive and entity summarization is a research field in its own. So, while there are potentially many application areas the lack of easy access to re-usable resources is limiting the use of these technologies.
We therefore present two highly re-usable resources for **1) PageRank [26] on RDF graphs** (PAGERANKRDF) that can be combined to a **2) generic framework for entity summarization** (SUMMASERVER). Both components are well documented and licensed under the MIT License (see Section 4.2.3). This enables extensibility and reusability without any types of restrictions. The framework has matured from earlier contributions [146, 148, 150] in the context of the WDAqua[12] research project with a focus on re-usable components for question answering [46, 52].

This paper is organized as follows: In Section 4.2.3 we provide an overview of the presented resources. In Section 4.2.4 we first analyze the performance of PAGERANKRDF with respect to scalability in time and memory which are the limiting resources during PageRank computations. Second we compare the PageRank scores when computed over the RDF graph and when computed over the corresponding link structure of Wikipedia. In Section 4.2.7 we describe the SUMMASERVER component. We also describe how to extend SUMMASERVER in order to generate summaries for new knowledge bases and its API. In Section 4.2.10 we describe how PAGERANKRDF and SUMMASERVER are used in a existing question answering system called WDAqua-core0. In Section 4.2.13 we compare this work to existing ones and we conclude with Section 4.2.14.

4.2.3 Resources

The main contribution of this work encompasses the following two resources.

R 1 A command line tool called PAGERANKRDF to compute PageRank scores over RDF graphs. The source code of PAGERANKRDF can be found at `https://github.com/WDAqua/PageRankRDF` with a complete documentation and usage instructions. It is released under the permissive MIT Licence. Moreover we deliver some

[12] WDAqua (Answering Questions using Web Data) – `http://wdaqua.eu/`

derived resources with the PageRank scores for some known datasets in the LOD cloud, namely:

R 1.1 DBLP[13], using a dump provided by Jörg Diederich of the 22.07.2017.

R 1.2 DBpedia [9][14], using the dump of latest release of English DBpedia[15]

R 1.3 Freebase [19][16], using the last Freebase dump before shutdown.

R 1.4 MusicBrainz[17], using the dump of December 2016 generated using MusicBrainz-R2RML (`https://github.com/LinkedBrainz/MusicBrainz-R2RML`).

R 1.5 Scigraph[18], using the current release of February 2017 (`http://scigraph.springernature.com/`).

R 1.6 Wikidata [164][19], using the dump from the 28 Septermber 2017.

The datasets are available at `https://figshare.com/projects/PageRank_scores_of_some_RDF_graphs/28119`.

R 2 An easily extensible framework for entity summarization called SUMMASERVER. It allows to generate entity summaries and currently supports the following knowledge bases: DBLP, Freebase, MusicBrainz, Scigraph, and Wikidata. Moreover it can be easily extended to support new knowledge bases. The source code of the SUMMASERVER can be accessed at `https://github.com/WDAqua/SummaServer`. It is released under the permissive MIT Licence. Moreover, we deliver a running service of the SummaServer. It can generate summaries for the above knowledge bases that can be accessed at the following service endpoints:

R 2.1 `https://wdaqua-summa-server.univ-st-etienne.fr/dblp/sum`

R 2.2 `https://wdaqua-summa-server.univ-st-etienne.fr/freebase/sum`

[13] `http://dblp.l3s.de/dblp++.php`
[14] `www.dbpedia.org`
[15] All files retrived by: wget -r -nc -nH –cut-dirs=1 -np -l1 -A '*ttl.bz2' -A '*.owl'-R '*unredirected*'–tries 2 http://downloads.dbpedia.org/2016-10/core-i18n/en/, i.e. all files published in the english DBpedia. We exclude the following files: nif_page_structure_en.ttl, raw_tables_en.ttl and page_links_en.ttl. The first two do not contain useful links, while, the last one contains the link structure of Wikipedia that was already used in previews works [148].
[16] `http://freebase.com`
[17] `https://musicbrainz.org`
[18] `http://scigraph.springernature.com/`
[19] `www.wikidata.org`

R 2.3 `https://wdaqua-summa-server.univ-st-etienne.fr/musicbrainz/sum`

R 2.4 `https://wdaqua-summa-server.univ-st-etienne.fr/scigraph/sum`

R 2.5 `https://wdaqua-summa-server.univ-st-etienne.fr/wikidata/sum`

As a side note: From a previous contribution [150] there already exists the summaClient JavaScript component. It is a client of the SUMMASERVER that can be easily embedded in web pages. It is also licensed under the MIT License and can be accessed at `https://github.com/athalhammer/summaClient`.

4.2.4 Computation of PageRank on RDF Graphs

In the following we describe Resource R 1, namely PAGERANKRDF, a command line tool for computing PageRank scores over RDF graphs. In particular we analyze it's scalability in terms of time and memory, which are the limiting resources for PageRank computation. Then we analyze the quality of PageRank scores of Wikidata comparing, the RDF based PageRank scores, with the PageRank scores computed using the links between the corresponding Wikipedia articles.

4.2.5 Runtime Comparison: Non-HDT version vs. HDT-version

Implementing the Pagerank algorithm is a fairly easy task. The main problem is to make it scalable in time and memory. We present two different ways to compute the PageRank scores over RDF graphs. Both implement the PageRank algorithm as presented by Brin and Page in [26]. The first implementation is a straight-forward implementation of the algorithm that takes as input an RDF dump in one of the current formats (like N-triples, Turtle) and computes the corresponding PageRank scores. The second implementation takes as input an RDF graph in HDT format [66]. HDT is a format for RDF that stores the graph in a very efficient way in terms of space. Generally, a factor $\times 10$ between the space consumption of the original RDF dump in one of the usual formats and the corresponing HDT dump is realistic. Moreover at the same time the RDF graph remains querieble, in the sense that triple patterns can be resolved in milliseconds. An HDT file contains three sections: the **H**eader (which simply contains some metadata), the **D**ictionray (which is a compressed mapping between URIs and integers) and the **T**riples (which are also compressed using the Dictionary and additional compression techniques). The second implementation is based on two observations. First, only the graph structure is important for the computation of the PageRank scores, i.e. the last section of the HDT file. Second, the dictionary section, i.e.

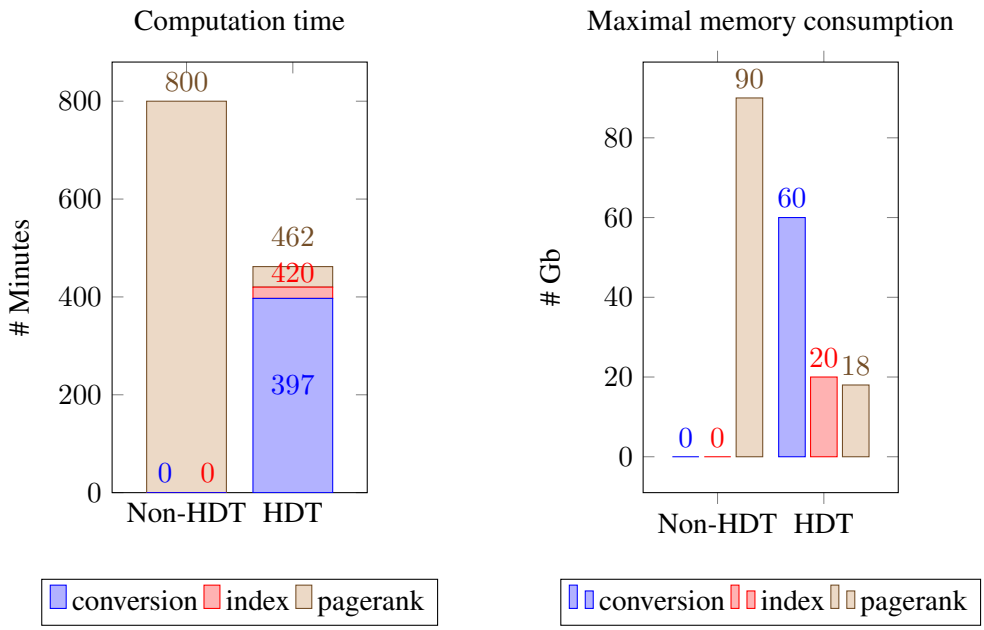

Figure 4.2: This figure shows the time consumption and maximal memory consumption for the computation of the PageRank scores for Wikidata. We choose the dump of the 28 September 2017 which has a size of 237 Gb and 2.2 billion triples. The left figures shows the time consumption of the two implementation. The Non-HDT version takes 13 hours. The HDT version takes 42 minutes when the HDT file is already computed and 8.8 hours when the HDT file has to be generated from a different serialization. The right figure shows the memory consumption for the two implementation. The first implementation needs 90 Gb of RAM while the second 18 Gb if the HDT file is already computed and 60 Gb otherwise.

the URIs, are occupying most of the space. The implementation basically computes the PageRank scores on the third section of the HDT file and uses the dictionary only at the end to assign the scores to the different URIs. This makes the second implementation much more time and memory efficient.

In Figure 4.2 the two implementations are compared by computing the PageRank scores for the Wikidata dump of the 28 September 2017 which has a size of 237 Gb and contains 2.2 billion triples. It shows that when starting from an HDT dump of the graph the time consumption is reduced by a factor of $\times 19$ and the memory consumption by a factor of $\times 5$. In particular this last point is important since it allows the computation of PageRank scores of bigger datasets on affordable hardware. Note that HDT dumps of online available datasets, can be found in the LOD laundromat [14][20] or under `http://www.rdfhdt.org/datasets/`. Moreover they can be easily created using the corresponding command line tools[21].

[20] http://lodlaundromat.org/

[21] http://www.rdfhdt.org/manual-of-the-c-hdt-library/

4.2.6 Input Comparison: RDF relations vs. Wikipedia links

Next to the standard parameters "damping factor" and "number of iterations", PageRank [26] computations naturally depend most strongly on the input graph. Thalhammer and Rettigner showed in their work "PageRank on Wikipedia: Towards General Importance Scores for Entities" [148] that link filtering and weighting can have a strong influence on the output of PageRank calculations. In the same work it was indicated that the output of PageRank computations on the extracted RDF version of Wikipedia (i.e., DBpedia) could correlate less with page-view-based rankings than PageRank computations on the Wikipedia link graph. However, the experiment was not performed and the following question is still open: "How do PageRank computations based on RDF relations compare to those based on Wikipedia links?" In order to answer this question, we start with the assumption that a higher ranking correlation (in our case Spearman's ρ and Kendall's τ)[22] to page-view-based rankings indicates a better ranking outcome.

The input data consists of three different ranking computations: PageRank on the Wikidata RDF graph (via PAGERANKRDF on a dump from September 28, 2017), PageRank on the Wikipedia link graph (computed with danker v0.1.0[23] on a Wikipedia dump from October 2, 2017 with option ALL[24]), and SubjectiveEye3D[25] by Paul Houle. The latter reflects the aggregated Wikipedia page view counts of the years 2008 to 2013 with different normalization factors (particularly considering the dimensions articles, language, and time). The datasets consist of different numbers of entities:

- Wikidata, PAGERANKRDF : 38 433 113 Q-IDs (total 80 646 048 resources)
- Wikidata, danker (Wikipedia links): 17 645 575 Q-IDs
- SubjectiveEye3D: 6 211 717 Q-IDs
- PAGERANKRDF ∩ danker ∩ SubjectiveEye3D: 4 253 903 Q-IDs

danker only includes entities from the Wikipedia namespace 0 (Main/Article), which particularly excludes "File" (namespace 6) and "Category" (namespace 14). Both types of entities are included in the SubjectiveEye3D dataset which, in consequence, reduces the number of

[22] Both correlation measures have a codomain of $[-1, 1]$ where -1 means fully anti-correlated and 1 means fully correlated. For computing Spearman's ρ we used the R `cor.test` function and for computing Kendall's τ we used the function `cor.fk` of the R package pcaPP `https://cran.r-project.org/web/packages/pcaPP/`.

[23] danker v0.1.0 – `https://github.com/athalhammer/danker/releases/tag/v0.1.0`

[24] The option ALL uses the different language editions in a voting style using "bag of links semantics": if 200 languages cover the link *USA* → *Barack Obama* it is given more importance than 15 languages that cover the link *USA* → *Donald Trump*. The PageRank algorithm supports multiple occurrences of the same link by default.

[25] SubjectiveEye3D – `https://github.com/paulhoule/telepath/wiki/SubjectiveEye3D`

Table 4.2: Spearman's ρ / Kendall's τ correlations of PageRank on RDF relations vs. Wikipedia links (via danker) and the comparison to SubjectiveEye3D.

	PAGERANKRDF	danker	SubjectiveEye3D
PAGERANKRDF	1.000 / 1.000	0.427 / 0.328	0.184 / 0.138
danker	0.427 / 0.328	1.000 / 1.000	0.400 / 0.276
SubjectiveEye3D	0.184 / 0.138	0.400 / 0.276	1.000 / 1.000

entities in the intersection set significantly. Another reduction factor were articles that have been deleted since 2013.

The result of the mutual correlation computations is outlined in Table 4.2. Both PageRank-based rankings have a positive correlation with the page-view-based ranking. The results show that danker correlates stronger with SubjectiveEye3D than PAGERANKRDF for both ranking correlation measures. However, danker is tailored to the Wikipedia and/or Wikidata setting while PAGERANKRDF generalizes for all existing RDF graphs. We therefore recommend danker for Wikipedia-related knowledge bases (i.e., DBpedia, Wikidata, and YAGO* [141]) and PAGERANKRDF for others.

The correlation between danker and SubjectiveEye3D is weaker than expected from the more positive results of [148]. In that work, the PageRank experiments are based on page link datasets of English Wikipedia. In contrast, the danker ALL option factors in page links from all Wikipedia language editions and therefore reduces bias towards English Wikipedia. One possibility for the lower correlation could be that SubjectiveEye3D maintains a rather strong bias towards English Wikipedia (despite the mentioned normalization steps).

4.2.7 Re-usable API for Serving Summaries of Entities

In this section we present Resource R2—namely the SUMMASERVER—a service implementation that serves summaries of entities contained in RDF graphs. We first recapitulate the SUMMA API design [150] which is implemented by SUMMASERVER. Then, we sketch how a typical entity summarization service can be implemented using the SUMMASERVER code base.

4.2.8 The SUMMA API

The SUMMA API [150] is composed of two main components:

- SUMMA Vocabulary.[26]
- RESTful interaction mechanism.

[26] Available at `http://purl.org/voc/summa`

This combination enables seamless integration with other Semantic Web components and a large degree of freedom with respect of the underlying entity summarization algorithm(s). When requesting a summary of an RDF entity only two parameters are mandatory:

entity the URI of the target resource (i.e., the resource to be summarized).

topK the number of triples the summary should contain.

The first interaction with the RESTful server is an HTTP `POST` request for creating a summary (see Listing 4.1). Note that the identifier of the summary in the summary request is a blank node (Turtle notation). The request basically says: "I would like the server to create a summary that complies with the given parameters." The server then responds with HTTP code 201 (`CREATED`). The `Location` header field denotes where we can find the newly created summary for future reference (i.e., to be accessed via `GET`): `https://wdaqua-summa-server.univ-st-etienne.fr/wikidata/sum?entity=http://www.wikidata.org/entity/Q42&topK=5&maxHops=1&language=en`.

Listing 4.1: Example `POST` request for creating a new summary.

```
curl -d "[ a <http://purl.org/voc/summa/Summary> ; \
<http://purl.org/voc/summa/entity> \
<http://www.wikidata.org/entity/Q6414> ; \
<http://purl.org/voc/summa/topK> 5 ] ." \
-H "Content-type: text/turtle" \
-H "Accept: application/ld+json" \
https://wdaqua-summa-server.univ-st-etienne.fr/wikidata/sum
```

Different client applications can request summaries and interpret the returned content. For this, SUMMASERVER can parse and create output in all standard RDF serializations (in accordance to the provided `Content-type` and `Accept` header parameters). As a matter of fact, summaries do not necessarily need to be requested via `POST` requests but can also be directly accessed via `GET` (SUMMASERVER keeps the URL layout). However, the interaction mechanism could also return summaries identified by non-speaking URIs like `https://wdaqua-summa-server.univ-st-etienne.fr/wikidata/sum/xyz`. An example implementation of a client—the summaClient JavaScript component (`https://github.com/athalhammer/summaClient`)—can interact with any server that implements the SUMMA API layout (see for example Section 4.2.12).

For more details on SUMMA the reader is kindly referred to [150].

4.2.9 Implementation Guide

We briefly want to describe the idea used by the SUMMASERVER to generate summaries. Imagine one wants to generate the summary for an entity, like the Wikidata entity Q6414

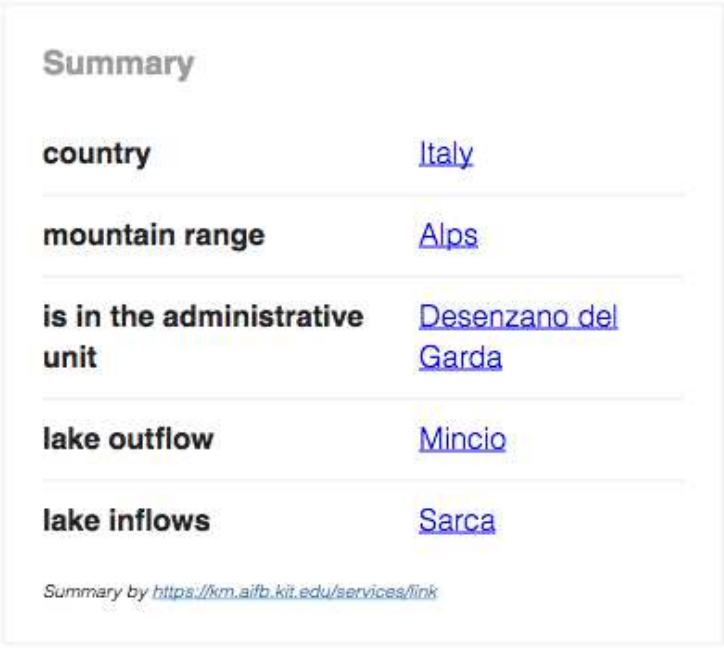

Figure 4.3: Example of a summary for “Lake Garda”.

corresponding to “Lake Garda", one of the biggest Italian lakes. The objective is to present to the user, between all facts that are known about this entity, the ones that best summarize it. An example of a summary for the "Lake Garda" is given in Figure 4.3. The idea presented in [146] generates the summary for a target entity X using the following straight-forward strategy. First the knowledge base is explored around X in a breadth-first traversal up to a certain depth (typically only 1, i.e., the next neighbours). For all reached entities the PageRank scores are considered and ranked in decreasing order. The entities corresponding to the first *topK* scores are shown in the summary. In the concrete example of “Lake Garda" the first 5 entities would be “Italy", “Alps", “Desenzano del Garda", “Mincio" and “Sarca". Note, during the breadth first search the knowledge base can be either traversed in a directed or in an undirected way.

In the following, we assume that the PageRank scores for all entities in the knowledge base where computed (for example using the command line tool in Section 4.2.4) and stored using the vRank vocabulary [129]. Moreover the PageRank scores are loaded in a SPARQL endpoint together with the original knowledge base.
Setting up the SUMMASERVER to generate summaries for entities in G reduces to: indicate the address of the SPARQL endpoint and writing three SPARQL queries. We want to describe the three queries using as a concrete example the Wikidata knowledge base.

Listing 4.2: QUERY 1: This query retrives for an ENTITY the corresponding label in the language LANG. For Wikidata the query is

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?l
WHERE {
```

```
  <ENTITY> rdfs:label ?l .
  FILTER regex(lang(?l), "LANG", "i") .
}
```

(note that this information must be given since there are multiple ways to express the label of an entity. For example in MusicBrainz it is indicated with properties like <http://xmlns.com/foaf/0.1/name> and <http://purl.org/dc/elements/1.1/title>)

Listing 4.3: QUERY 2: This query must retrieve the resources connected to the resource ENTITY, order them according to the PageRank score and take the first TOPK. Moreover it retrieves the labels of the founded resources in the language LANG.

```
PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vrank: <http://purl.org/voc/vrank#>
PREFIX wdd: <http://www.wikidata.org/prop/direct/>
SELECT DISTINCT ?o ?l ?pageRank
WHERE {
  <ENTITY> ?p ?o .
    FILTER (?p != rdf:type && ?p != wdd:P31
    && ?p != wdd:P735 && wdd:P21>
    && ?p != wdd:P972 && wdd:P421>
    && ?p != wdd:P1343 )
  ?o rdfs:label ?l .
  regex(lang(?l), "LANG", "i") .
  graph <http://wikidata.com/pageRank> {
    ?o vrank:pagerank ?pageRank .
  }
}
ORDER BY DESC (?pageRank) LIMIT TOPK
```

(note that we do not traverse the edges with some labels like rdf:type and wdd:P31).

Listing 4.4: QUERY 3: This query must retrieve given two resource, ENTITY and OBJECT, the label of the property between them in the language LANG. For Wikidata we use the following query:

```
PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vrank:<http://purl.org/voc/vrank#>
SELECT ?p ?l
WHERE {
<ENTITY> ?p <OBJECT> .
OPTIONAL {
   ?o <http://wikiba.se/ontology-beta#directClaim> ?p .
   ?o rdfs:label ?l .
   FILTER regex(lang(?l), "LANG", "i")
}}
ORDER BY asc(?p) LIMIT 1
```

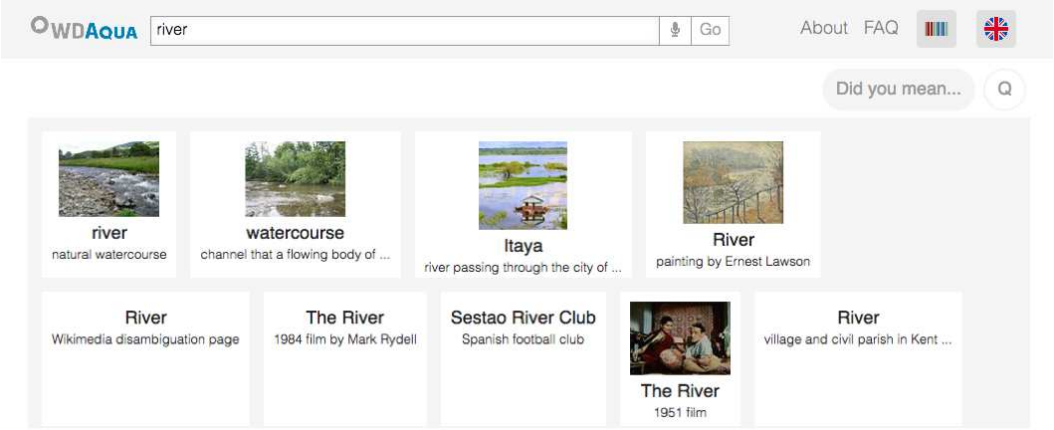

Figure 4.4: Screenshot of WDAqua-Core0 for the question “River". The “did you mean” functionality shows other possible meanings of “River" that the user could have intended. The ranking, from left to right, from top to bottom, is based on PageRank scores.

(note that in Wikidata the label of a direct property is not directly attached to it .)

We have implemented such queries for the following knowledge bases: Wikidata, DBpedia, DBLP, MusicBrainz, Freebase and the Scigraph. The implementations can be found at `https://github.com/WDAqua/SummaServer/tree/master/src/main/java/edu/kit/aifb/summarizer/implemented`. After indicating the endpoint and writing the three above queries the SUMMASERVER provides a summarization service for the corresponding knowledge base. For a more detailed instruction we refer to `https://github.com/WDAqua/SummaServer#extending-to-a-new-knowledge-base-kb`.

4.2.10 Use case: Question Answering

In this section we describe one concrete use case. We describe how the PageRank scores and the summarization service are used in a question answering system over knowledge bases. Concretely we describe the usage in WDAqua-Core0 [52].

4.2.11 PageRank for Question Answering

PageRank scores are used by WDAqua-core0 at two places. The first is for disambiguating entities. Suppose a user just asks for “River" and the question answering system uses Wikidata as an underlying knowledge bases. Multiple entities could be meant like `Q4022` (a natural watercourse), `Q2784912` (a village and civil parish in Kent) or `Q7337056)` (a studio album by Izzy Stradlin). The question is ambiguous but one still wants to present the most probable interpretation to the user. PageRanks are used here to identify the most probable intended interpretation by the user, i.e. the one with the highest PageRank between the possible candidates. A concrete usage is shown in Figure 4.4.

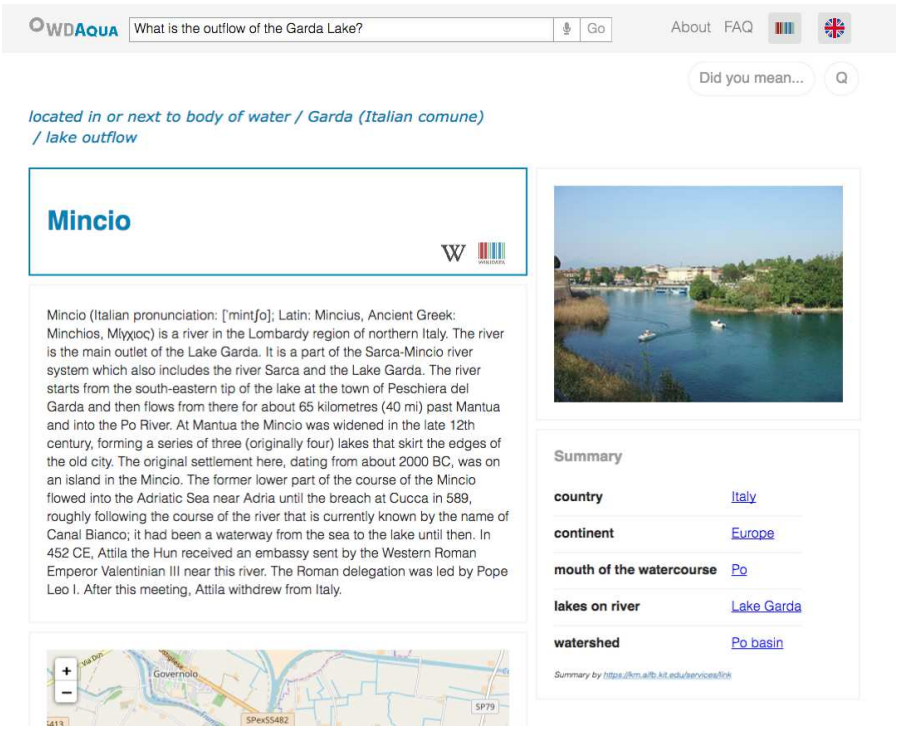

Figure 4.5: Screenshot of WDAqua-Core0 for the question “What is the outflow of Lake Garda?". The entity summary is on the right-bottom part. Note that the links are discoverable, i.e. by clicking on “Po” information of “Po” are displayed (in the same way if the user asked directly for “Po").

A second application of PageRank scores is once the answer set is already computed. Imagine the following scenario. A user asks “Give me lakes in Italy." There are hundreds of lakes in Italy and currently there are 499 in Wikidata. Returning just a list will not be very useful for the user. Since the order is random the first presented lakes will be some unknown lake like the “Lago di Posta Fibreno". Ranking the answers according to PageRank will give as top rank lakes the “Lago di Garda" or the “Lago di Como" which is probably more relevant information for the user.

The PageRank scores used in WDAqua-Core0 correspond to R 1.1, R 1.2, R 1.3, R 1.4, R 1.5, R 1.6 and are computed using the tool presented in Section 4.2.4.

4.2.12 Entity Summarization for Question Answering

Entity summaries are used in Trill [46], the front-end used by WDAqua-Core0. An example is given in Figure 4.5. The summarization service is used mainly for two reasons: First, to add context to the retrieved answer. An expected result of this is that the confidence of the user in the answer is increased. Second, to increase discoverability within the dataset, i.e., offering a number of facts related to the answer entity the user. The facts are browseable in the sense that the summary facts are clickable links that allow to easily explore other information in the graph that are connected to the original entities.

WDAqua-Core0 currently uses the summarization services offered by SUMMASERVER corresponding to R 2.1, R 2.2, R 2.3, R 2.4, R 2.5. As explained in Section 4.2.6 the PageRank scores computed over the linked structure of Wikipedia express better the page views of the user. Since in DBpedia every entity corresponds to a Wikipedia article, for DBpedia we use the PageRank scores computed over the linked structure of Wikipedia.
The client is integrated as a React component into Trill for easy re-usability.
A demo of WDAqua-Core0, that uses the above-described services, can be found at `www.wdaqua.eu/qa`.

4.2.13 Related Work

We touch on two fields in this work, namely ranking for RDF knowledge bases and entity summarization. For a good survey on ranking for RDF knowledge bases we refer the reader to Roa-Valverde and Sicilia [128]. Recent work on this topic includes Ngomo et al. [118] which gives an alternative to traditional PageRank computation. They show that, although PageRank has only a time complexity of $O(n+m)$ (in its most efficient implementation) it can get expensive to compute. Therefore, we presented an efficient implementation of PageRank that, when data is already provided in HDT format (as often already done; see LOD laundromat [14]), has a very high time and memory efficiency.[27] For a detailed overview on the field of entity summarization we kindly refer the reader to Section 2.2 of [143]. Recent work includes Pouriyeh et al. [123], where topic modeling is explored for entity summarization.

The presented work is intended to provide findable, accessible, interoperable, and re-usable (FAIR) baselines for ranking and entity summarization in RDF knowledge bases. It stands in the light of the FAIR guiding principles [168] that every modern researcher should try to adhere to. We build on the work that initially presented the SUMMA API [150]. Next to the service endpoints presented in this work, this API definition has been implemented by [144] with the show case of DBpedia and is online available (and therefore can be reused by researchers and practitioners). We encourage all researchers in the field also to publish their research prototypes along the FAIR guiding principles by adhering to the SUMMA API definition. To the best of our knowledge, DBpedia/Wikidata PageRank[28] [148] is currently the only public source for pre-computed datasets of knowledge bases that can easily be loaded into triple stores. PAGERANKRDF builds on this work and provides general, affordable PageRank computation for RDF knowledge bases. An initial implementation of PAGERANKRDF was used for experiments by Andreas Harth that are documented at `http://harth.org/andreas/2016/datenintelligenz/`.

[27] As [118] was published very recently, a comparison can only be provided in the final version of this work.
[28] DBpedia/Wikidata PageRank – `http://people.aifb.kit.edu/ath/`

4.2.14 Summary

We have presented two important and tightly connected resources: a command line tool for computing PageRank scores called PAGERANKRDF [R1], and a framework for computing entity summaries called SUMMASERVER [R2]. The code is open source and available under an open licence. We have proven that PAGERANKRDF can scale up to large datasets and we are publishing the computed scores for example knowledge bases. Moreover, we have described the SUMMASERVER and shown how it can be extended to new knowledge bases. Finally, we have shown how the existing resources are used in a concrete scenario, namely in an existing question answering system.
The presented resources will be maintained and used within the WDAqua ITN Project[29] [94]. Due to the popularity of the previously published PageRank scores [148] for DBpedia/Wikidata and the number of possible applications in different research areas, we believe that the presented resources are an important contribution for the Semantic Web community.

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).

[29] http://wdaqua.eu

4.3 SPARQLtoUser: Did the question answering system understand me? [30]

4.3.1 Abstract

In Question Answering (QA) systems, user questions are automatically converted into SPARQL queries. We present SPARQLtoUser, a simple method to produce a user understandable version of a SPARQL query so that the users can check what the QA system understood. Compared to existing similar methods, SPARQLtoUser is multilingual and is not restricted to one single KB, but easily portable to any KBs. This paper presents our method to generate the representation of SPARQL queries, and compares this method to existing ones. Moreover, we present the results of our user study that show that users have difficulties to understand whether the responses of a QA system are correct or not, with or without SPARQL representations.
Keywords: Question answering system, User feedback, User interaction, SPARQLToUser, SPARQL2NL, Spartiqulator

4.3.2 Introduction

Question answering (QA) systems over Knowledge Bases (KB) are aiming to directly deliver an answer to a user's question. Achieving this with high precision is one important research area [49]. Unfortunately the error rates are still very high. For example on popular benchmarks like WebQuestions [15], SimpleQuestions [21] and QALD[31] [159] F-measures of respectively 60 %, 70 %, 50 % are realistic. Also the precision is similar meaning that QA systems often do not retrieve the right answer for the user. Since the questions in the above mentioned benchmarks are generally all answerable over the given data, the F-measures in real scenarios are even lower. It is therefore important that users are able to understand if the information provided by a QA system is the one they asked for, so that they can easily estimate if they can believe the answers or not. In the following, we present a tool called SPARQLtoUser, which given a SPARQL query, produces a representation of it that can be shown to end-users. We compare our method with existing ones, namely SPARQL2NL [119] and Spartiqulator [63]. Moreover, we show for the first time that users have difficulties to distinguish right answers from false ones.

[30] Corresponding publications is: Dennis Diefenbach, Youssef Dridi, Kamal Singh,, Pierre Maret, *SPARQLtoUser: Did the question answering system understand me?*, 2nd International Workshop on Benchmarking Linked Data and NLIWoD3: Natural Language Interfaces for the Web of Data co-located with 16th International Semantic Web Conference [32]

[31] `http://www.sc.cit-ec.uni-bielefeld.de/qald/`

4.3.3 Related work

QA over Knowledge Bases is a well-studied problem in computer science. For an overview of the published approaches we refer to [49].
The task of presenting a SPARQL query to an end-user was tackled by verbalizing the query, i.e. by reformulating the SPARQL query using natural language. There are mainly two works that address this task: SPARQL2NL [119] and Spartiqulation [63]. Both tools can be used to translate a SPARQL query to an English sentence. Both approaches were tested on DBpedia and MusicBrainz.
Two main applications in the QA domain are used to motivate the need of such tools. The first was pointed out by Ell et al. [63]: "The verbalization allows the user to observe a potential discrepancy between an intended question and the system-generated query.". The second was pointed out by Ngonga Ngomo et al. [119]. In this case, the idea is to verbalize all the SPARQL queries generated by a QA system and to let the user, if the answer is wrong, the possibility to choose a different SPARQL query by selecting the corresponding verbalization. We are not aware of any test of the efficacy of such approaches.

4.3.4 A simple representation of a SPARQL query for the user.

Users can ask questions in various languages, and they may target various KBs. Therefore, a generic method to build a representation of SPARQL queries (generated automatically from user questions) which is easily understandable by the users, should support multi-linguality and should be easily portable to different KBs. In this section, we present our method called SPARQLtoUser. We explain its origin and its mechanism.

The need for such a tool was derived from our QA webservice available under `www.wdaqua.eu/qa`. It uses in the backend a QA system called WDAquaCore0 [52] which is integrated into the Qanary Ecosystem [50] and is exposed to end-users through a reusable front-end called Trill [46]. The QA system is multilingual, and we are able to port it easily to different KBs.

We observed that in many situations users would not be able to understand what was the interpretation of their questions by the QA system and, even as a developer, it was difficult to understand it by looking at the SPARQL query. Since the QA system supports different languages, and since different KBs can be queried, we came to develop a generic solution which is valid for any new languages and for new domains (i.e. KBs). In particular, the multilingual constraint brought us to not to try to verbalize the query, but to find a more schematic representation. An example can be seen in Figure 4.6. The missing verbalization can be seen as an additional burden for the user. One objective of this work is to measure and compare this burden to other approaches.

To generate the representation, we keep the original order of the triple patterns of the query and we substitute the URIs with the label in the corresponding language, i.e., the triple patterns

$$\begin{matrix} s_{1,1} & s_{1,2} & s_{1,3} \\ \vdots & & \\ s_{k,1} & s_{k,1} & s_{k,1} \end{matrix}$$

(where s indicates a slot) are converted to

$$\begin{matrix} f(s_{1,1}) & slash(s_{1,1}) & g(s_{1,2}) & slash(s_{1,3}) & f(s_{1,3}) \\ \vdots & & & & \\ f(s_{k,1}) & slash(s_{k,1}) & g(s_{k,2}) & slash(s_{k,3}) & f(s_{k,3}) \end{matrix}$$

where

$$f(s_{i,j}) = \begin{cases} label(s_{i,j}), & \text{for } s_{i,j} \text{ a URI} \\ \text{""}, & \text{for } s_{i,j} \text{ a variable} \\ s_{i,j}, & \text{for } s_{i,j} \text{ a literal} \end{cases}$$

$$g(s_{i,j}) = \begin{cases} label(s_{i,j}), & \text{, for } s_{i,j} \text{ a URI} \\ \{label(s) | \ (s \text{ a uri such that if } s_{i,j} \text{ is substituted} & \\ \text{with } s \text{ the triple patterns has a solution in the KB} & \text{, for } s_{i,j} \text{ a variable} \\ ) \ \} & \end{cases}$$

$$slash(s_{i,j}) = \begin{cases} / & \text{, for } s_{i,j} \text{ a URI} \\ \text{""} & \text{, for } s_{i,j} \text{ a variable} \end{cases}$$

As a concrete example the query:

```
PREFIX wd : <http://www.wikidata.org/>
SELECT DISTINCT ?x
WHERE {
   wd:entity/Q79848 wd:prop/direct/P1082 ?x .
} limit 1000
```

is converted to:

```
Southampton (city in Hampshire, England) / population
```

since "Q79848" refers to "Southampton" and "P1082" to the "population".
We want to briefly describe the substitution of the predicates. If a predicate is an URI, we

just put its label. If it is a variable, we expand it using all the properties that potentially could be placed in the variable position. This is motivated by the following example. In QA often hidden relations are expressed in the question. For example for the question “films by Charlie Chaplin” the proposition “by” can have different meanings like "film editor, screenwriter, director, composer, producer, cast member". WDAquaCore0 therefore produces as an interpretation:

```
SELECT DISTINCT ?x
WHERE {
  ?x ?p1 <http://www.wikidata.org/entity/Q11424> .
  ?x ?p2 <http://www.wikidata.org/entity/Q882> .
}
limit 1000
```

where the properties are variables. SPARQLtoUser expands the variables ?p1 and ?p2 with the above described meanings. By showing this information, we want to show to the user what the QA system understood and give the users the lexicon to narrow down the search. Note that the label function can differ from the KB. Generally the strategy is to grab just the label, i.e. rdfs:label. But in some cases this does not identify the entity in an unambiguous way. For example in Wikidata the monument "Colosseum" and the band "Colosseum" have the same label. To identify the entity in a unique way, we add the description in parenthesis, i.e. "Colosseum (elliptical amphitheatre in Rome)" and "Colosseum (British band)". Depending on the KB a different property can be used to grab a verbalization of the resource and a property disambiguating it. Moreover for count queries, we surround the query with “How many” and for an ask query with “check”. We limited the support of SPARQLtoUser to the types of queries that WDAquaCore0 can produce, but one can easily think about extension for this approach for other types of queries and aggregators.
Note that the approach is easily portable to a new KB and that it is also multilingual in the sense that if the KB has not English labels the approach still can be applied.
Table 4.3 gives an overview of the main differences between SPARLQ2NL, Spartiqulation and SPARQLtoUser.

The code of SPARQLtoUser can be found at `https://github.com/WDAqua/SPARQL_to_User`. It is released under the MIT licence. The service is exposed as a webservice under the url `https://wdaqua-sparqltouser.univ-st-etienne.fr/sparqltouser`. A description of the API can be found under `https://github.com/WDAqua/SPARQL_to_User/blob/master/README.md`.

4.3.5 A user study to identify the efficacy of SPARQL representations for users.

As mentioned in the related works, one of the main motivations of SPARQL2NL and Spartiqulation was to allow users to better interact with QA systems. While both works

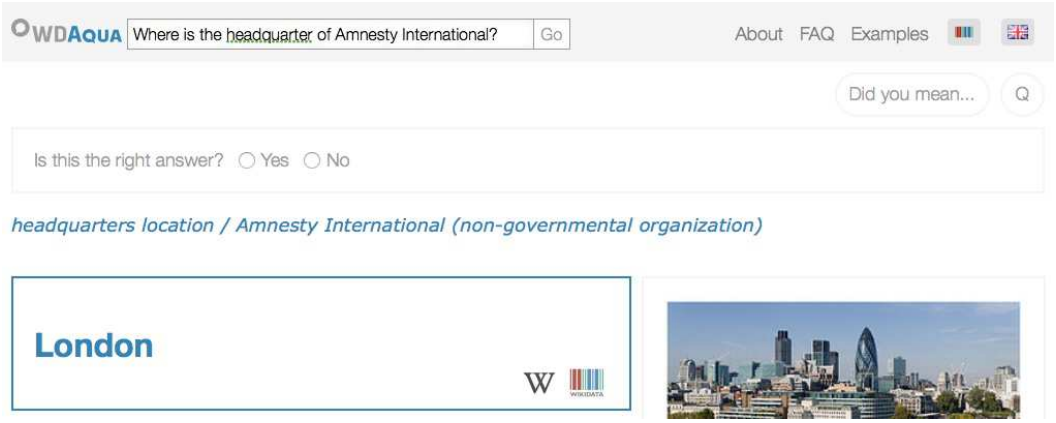

Figure 4.6: Snapshot of the Trill front-end. The interpretation generated by SPARQLtoUser is shown in blue over the answer "London").

Tool	SPARQL support	Datasets (tested on)	Languages	Portability
SPARQL2NL	part of SPARQL 1.1 syntax	DBpedia, MusicBrainz	English	no
Spartiqulator	part of SPARQL 1.1	DBpedia, MusicBrainz	English	yes
SPARQLToUser	triple patterns, ASK and SELECT queries, COUNT modifiers	DBpedia, MusicBrainz, Wikidata, Dblp	multilingual	yes

Table 4.3: Main features of SPARQLToUser, SPARQL2NL, Spartiqulator

performed a users study to test the quality of the verbalisation process, the impact of the SPARQL query representation was never analyzed in the context of QA.

We prepared a survey to test that up to which degree a SPARQL query representation helps the user to understand the interpretation of a user's question by a QA system. To test the effectiveness of a SPARQL representation, we conducted a controlled experiment using A/B testing. In this approach, two versions of the same user-interface, with only one different UI-element, are presented to two different groups of users. The users are not aware that two different UI-elements are being tested. During A/B testing the objective is to measure whether the difference in the UI has an impact on some measurable variable, for example, the click-through rate of a banner advertisement.

Our objective was to test the impact of the SPARQL representation on the ability of a user, to understand the interpretation of a question by a QA system. We, therefore, choose 10 questions and presented to the users the answers that a QA system could have produced. Out of the 10 questions 4 were answered correctly while 6 were answered wrongly. Since our objective is to find out if users are able to determine whether the given answer is correct or not, we ask the users if they believe that the given answer is correct or not. Our objective in the A/B testing is to maximize the number of times the user believes the answer is right, when the answer is actually right, plus the number of times the user believes the answer is wrong, when it is actually wrong. We created 3 different versions of the same UI. The first, denoted **A**, does not contain any representation of the generated SPARQL query, the second, denoted **B**, contains the representation produced by SPARQLtoUser, and the third,

denoted **C**, the one produced by SPARQL2NL. We did not test the interpretations given by Spartiqulation because the approach is similar to that of SPARQL2NL. An example of the 3 versions shown to the users can be seen in Figure 4.7. The list of all questions together with the answers can be found in Table 4.4. We tried to take questions that do not cover general knowledge so that it is highly probable that the users do not know the correct answer.

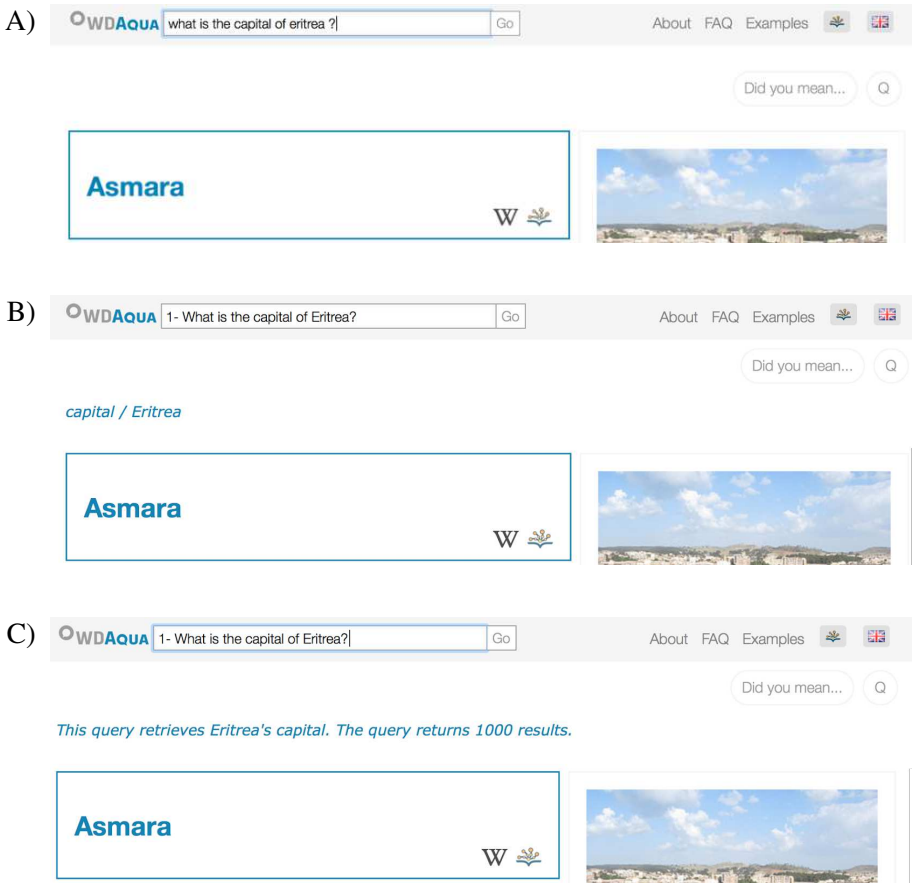

Figure 4.7: Three compared versions of the Trill front-end for the question "What is the capital of Eritrea?". A) with no SPARQL representation, B) with the representation of SPARQLtoUser, and C) with the representation of SAPRQL2NL.

The questions with incorrect answers generally present the problem of ambiguity. Sometimes the information provided with the answer, like the description or the top-k properties, allows to decide if the answer is correct or not. In other situations, the user can only choose to trust or not the given answer. The 3 forms send to the groups A, B, C can be found respectively under:

A) `https://goo.gl/forms/vCN7aZEaUAxWNMse2`

B) `https://goo.gl/forms/1fgauBGn5rb4JJ492`

C) `https://goo.gl/forms/WTibSJHjEBRgnaS62`.

N.	Question	Answer	correct	A	B	C
1	What is the capital of Eritrea?	Asmara	yes	12/4	13/3	14/2
2	What is the capital of Ghana?	Koumbi Saleh	no	14/2	12/4	14/2
3	Who is the president of Eritrea ?	Isaias Afwerki	yes	14/2	9/7	12/4
4	In which country was Claudia Cardinale born?	Italy	no	6/10	9/7	8/8
5	Where did Karl Marx die ?	Ladenburg	no	8/8	11/5	14/2
6	In which museum is the Davide of Michelangelo ?	Galleria Borghese	no	12/4	10/6	12/4
7	In which city was Michael Jackson born ?	Fairefax, Virginia	no	9/7	10/6	10/6
8	What is the population of Canada?	36286425	yes	9/7	11/5	9/7
9	Where did Mussolini die ?	Forlì	no	8/8	12/4	9/7
10	Who is the doctoral supervisor of Albert Einstein ?	Alfred Kleiner	yes	14/2	13/3	13/3
Total				106/54	110/50	115/45

Table 4.4: This table shows the results of the user study. 10 questions different questions where shown to 48 people. These where divided into 3 equally sized groups and received 3 different versions of a UI following an A/B testing methodology. The three groups are denoted A,B,C. The UI shown to group A did not contain any representation of the SPARQL query, group B got the representation of SPARQLtoUser and group C the representation of SPARQL2NL.

The summary of the experimental results can be found in Table 4.4. In total 3 groups of 16 people answered our survey, i.e. for each version of the UI we have a sample size of 160 (16 people for 10 questions).

The experiments lead to the following conclusions. In general, users have big difficulties to understand the interpretation given by a QA system to the questions asked, independently of the approach. Without a representation of the SPARQL query the error rate is 34 %. With the representation given by SPARQLtoUser the error rate is 31 % and with the one of SPARQL2NL it is 28 %. Unfortunately, there is no significant difference between the different approaches. Also the statistical power is not high enough to say that there is no significant difference, i.e. we can not state with high enough probability that it makes no different to put or not a SPARQL representation.
This is an unexpected result for us. We thought that the impact of the representation of the SPARQL query would be high enough to show a significant difference with a sample size of 160. The experiment shows that the impact is lower than what we expected.
We want to discuss some unexpected observations. For example question 3 got better results without any SPARQL representation. Question 3 was "Who is the president of Eritrea ?". The presented answer is correct and the description of the answer says that clearly: "Isaias Afwerki Tigrinya is the first President of Eritrea, a position he has held since its independence in 1993.". In this case the representation of the SPARQL query given to the user seems to distract the user, since many users answer the question correctly without any interpretation. For the question "In which country was Claudia Cardinale born?" we

assume that the QA system did a mistake when disambiguating the entity and returned the result for "Claudia Octavia" instead of "Claudia Cardinale". Both the representations of SPARQLtoUser and SPARQL2NL contain the string "Claudia Octavia" and not "Claudia Cardinale". Still many users believed that the answer is correct. Apparently, here the representation of the SPARQL query was ignored by many users.

4.3.6 Conclusion

Since current QA systems over KB still have low F-measures, it is important to let the users understand how the QA system interpreted their questions. We have presented a new representation of a SPARQL query for users. It is not based on verbalization, as done in previews works, but it is based on a simple rewriting of the query. The advantage is that this representation can be used with any language and it can be easily ported to new KBs.
We have for the first time conducted a user study to measure the impact of SPARQL representations on the understand-ability of users of the interpretation of a question by a QA system. We found that if an impact exists then it is much smaller than we believed. More research in this area with quantitative methodologies is needed. In the future, we want to repeat our experiment with a larger number of participants and with more variations of the UI. In particular, we want to create a version that puts more emphasis on the SPARQL representation and one version that clearly states "I interpreted your question as: (followed by the interpretation)". The interaction of the users with a QA system is a poorly addressed, but interesting field.

4.4 Introducing Feedback in Qanary: How Users can interact with QA systems [32]

4.4.1 Abstract

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

Keywords: User Interaction, Question Answering Systems, User Interface

4.4.2 Introduction

In recent years, there has been a fast growth in available data sets. One very important use case is finding relevant results for a user's requests, and by using different data sets. The field of Question Answering (QA) tackles this information retrieval use case by providing a (natural language) interface aiming at easy-to-use fact retrieval from large data sets in knowledge bases (KB). While following this path it could be observed that additional challenges are rising while the data sets are growing. For example, while having only recent geospatial information a term like "Germany" can be identified directly without ambiguity. However, after adding historic data several instances called "Germany" will be available, representing different entities (over time).
In the past years several question answering systems were published working on-top of linked data, cf., Question Answering over Linked Data (QALD) [99]. Many QA systems follow a single interaction approach, where the user asks a question and retrieves an answer. However, many questions cannot be understood because the context is only clear in the users' mind, or because the ambiguity of the considered data set is not known by the user. Hence, not any kind of question can be interpreted correctly following an ad hoc single-interaction approach. User interaction in QA systems (i.e., how users can influence a QA process) is not well explored. Only a few QA systems exists involving the user in the retrieval process, *e.g.* Querix [92], Freya [35], and Canalis [111].

[32] Corresponding publications is: Dennis Diefenbach, Niousha Hormozi, Shanzay Amjad, Andreas Both, *Introducing Feedback in Qanary: How Users can interact with QA systems*,ESWC 2015 Poster & Demo [47]

In this paper we extend the Qanary methodology [22] a framework for integrating Question Answering components. Our main contribution is a generalized user feedback mechanism in Qanary.

4.4.3 Related Work

In the context of QA, a large number of systems have been developed in the last years. For example, more than twenty QA systems were evaluated against the QALD benchmark (cf., http://qald.sebastianwalter.org). However, only few of them address user interaction. Freya [35] uses syntactic parsing in combination with the KB-based look-up in order to interpret the question, and involves the user if necessary. The user's choices are used for training the system in order to improve its performance. Querix [92] is a domain-independent natural language interface (NLI) that uses clarification dialogs to query KBs. Querix is not "intelligent" by interpreting and understanding the input queries; it only employs a reduced set of NLP tools and consults the user when hitting its limitations. Canali [111] shows completions for a user's query in a drop-down menu appearing under the input window. The user has the option of clicking on any such completion, whereby its text is added to the input window.

Since QA systems often reuse existing techniques, the idea to develop QA systems in a modular way arise. Besides Qanary, three frameworks tried to achieve this goal: QALL-ME [67], openQA [109] and the Open Knowledge Base and Question-Answering (OKBQA) challenge (cf., http://www.okbqa.org/). To the best of our knowledge these frameworks do not address the problem of integrating user interaction.

4.4.4 A generalized Feedback Mechanism on-top of Qanary

The Qanary methodology

QA systems are generally made up by several components that are executed in a pipeline. Qanary offers a framework to easily integrate and reuse such components. Here, we briefly describe the Qanary methodology. Qanary components interact with each other by exchanging annotations (following the W3C WADM[33]) expressed in the qa vocabulary [133] and stored in a central triplestore T. For example, an annotation expressing that the sub-string between character 19 and 32 of the question "Who is the wife of Barack Obama?" refers to `dbr:Barack_Obama` is expressed as:

```
PREFIX qa: <https://w3id.org/wdaqua/qanary#>
PREFIX oa: <http://www.w3.org/ns/oa#>
<anno1> a qa:AnnotationOfInstance;
  oa:hasTarget [ a oa:SpecificResource;
```

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

```
            oa:hasSource <URIQuestion>;
            oa:hasSelector [ a oa:TextPositionSelector;
                   oa:start "9"^^xsd:nonNegativeInteger;
                   oa:end "21"^^xsd:nonNegativeInteger ]];
oa:hasBody dbr:Barack_Obama .
oa:annotatedBy <http://wdaqua.eu/component1> .
oa:annotatedAt "2017-02-22T21:40:51+01:00" .
```

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

The vocabulary that is used for the annotations is described in [133]. The Qanary methodology and their services are described in [22, 50].

Collecting User Feedback within a Qanary process

A user cannot change the internal algorithm of a component, but only affect the behavior of a component by the annotations it uses as input. Assume that the user wants to interact with the process after component C_i. The generalized workflow would be as follows: (1) Components C_1,...,C_i are called (cf., Sec. 4.4.4). (2) All the generated knowledge is stored in G. The user or the application accesses G and retrieves the annotation needed and creates new ones. (3) The QA system is restarted from component C_{i+1}, i.e., a QA process executing C_{i+1},..., C_n is started using T and G. To avoid conflicting annotations of the same type, we enforce that both the components and the user create annotation with a timestamp, and the components read only the annotations with the last timestamp. Note that during the QA process existing annotations are not deleted, s.t., G contains a full history of all the information generated at the different point in time by any component and by the user.

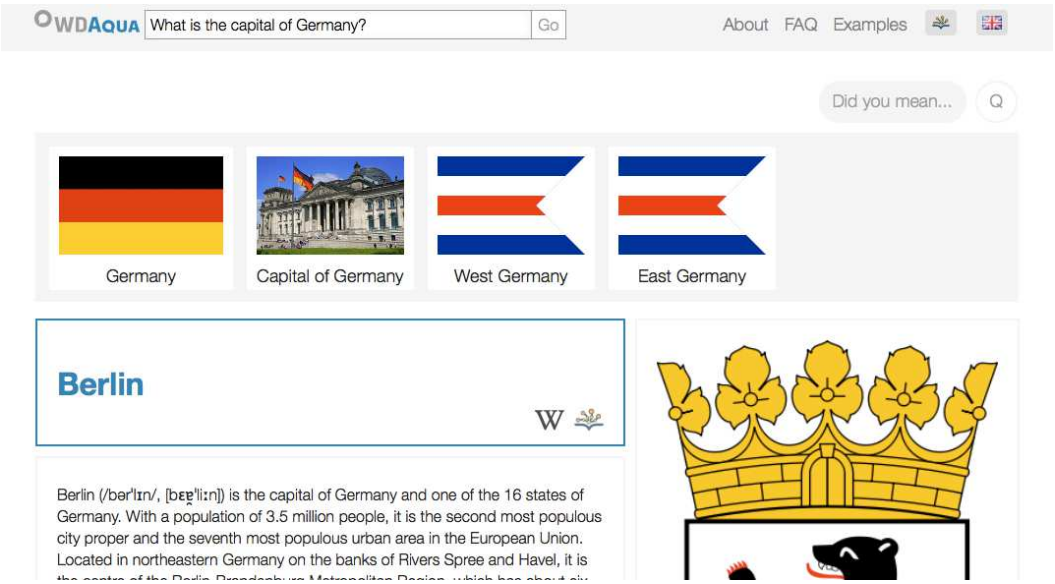

Figure 4.8: Snapshot of the disambiguation interface for the question: "What is the capital of Germany?". By clicking on "Did you mean" several entities, the question might refereed to, are shown. These include the actual "Federal Republic of Germany" but also the "Capital of Germany" (as an entity), "West Germany", "East Germany", "Allied-Occupied Germany" and others. By clicking on the entity, the question is interpreted differently and a new answer is presented, *e.g.* if the user clicks on "West Germany", the answer "Bonn" is computed.

4.4.5 Use Cases

We created some user interface components that follow the approach described in Sec. 4.4.4. In the back-end we used WDAqua-core0 [52]. It consists of two components: C_1 a query generator that translate a question (in keywords or natural language) into SPARQL queries and C_2 a query executor. The interfaces are integrated in Trill [46], a reusable front-end for QA systems that can be used for QA pipelines integrated in Qanary. The code can be found under `https://github.com/WDAqua/Trill`. We first describe in detail one of the interface components we implemented. It can be used by users to resolve the ambiguity of a question. We then briefly describe the other interfaces.

Example: Removing Entity Ambiguity from a Qanary process One of the main problems in QA systems is to disambiguate between different meanings associated to a user's question. Given "What is the capital of Germany?", the sub-string "Germany" can refer to the actual "Federated Republic of Germany" but also to "East Germany" (former GDR) or "West Germany", which will lead to different answers. To allow the user to choose between different options we use the workflow presented in Sec. 4.4.4. In the example implementation, C_1 produces 30 SPARQL query candidates which are ranked based on relevance and stored in T using the annotations `qa:AnnotationOfSparqlQueries`. E.g., for the given question the first 3 candidates are:

1. `SELECT ?x WHERE { dbr:Germany dbo:capital ?x . }`

2. `SELECT ?x { VALUES ?x { dbr:Capital_of_Germany } }`

3. `SELECT ?x WHERE { dbr:West_Germany dbp:capital ?x . }`

For the disambiguation interface we extract 30 queries from the endpoint, extract the resources from them (in the example `dbr:Germany`, `dbr:Capital_of_Germany`, `dbr:West_Germany`) and show them to the user. By clicking on one of the resources, the corresponding query is ranked first and the re-ranked queries are written to the triple-store using again the annotations `qa:AnnotationOfSparqlQueries`. C_2 is called which executes the first-ranked query. Finally the answer is shown. The example is implemented as a user interface component in Trill and shown in Fig. 4.8.

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

4.4.6 Conclusion and Future Work

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

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

5 Qanary: An architecture for Question Answering Systems

5.1 Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems [1]

5.1.1 Abstract

Question answering (QA) is one of the biggest challenges for making sense out of data. Web of Data has attracted the attention of question answering community and recently, a number of schema-aware question answering systems have been introduced. While research achievements are individually significant; yet, integrating different approaches is not possible due to lack of a systematic approach for conceptually describing QA systems. In this paper, we present a message-driven vocabulary built upon an abstract level. This vocabulary is concluded from conceptual views of different question answering systems. In this way, we are enabling researchers and industry to implement message-driven QA systems and to reuse and extend different approaches without the interoperability and extension concerns.
Keywords: Semantic Web, Software Reusability, Question Answering, Semantic Search, Ontologies, Annotation Model

5.1.2 Introduction

Web of Data is enormously growing (currently more than 84 billion triples[2]) as well as enterprise data. Still, taking advantage of this rapidly growing data is challenging. Therefore, automatic as well as intelligent approaches are needed to (1) make sense of the data, (2) make the data accessible, and (3) provide easy-to-use interfaces for querying data. Question answering (QA) is multi discipline, and it bridges artificial intelligence, information retrieval, and knowledge base. Recently, the question answering community paid considerable attention for adapting and improving QA systems by taking Web of Data into account. As

[1] Corresponding publications is: Kuldeep Singh, Andreas Both, Dennis Diefenbach and Saeedeh Shekarpour *Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems*,ICSC 2016 [133]

[2] observed on 14 October 2015 at `http://stats.lod2.eu/`

the result of these attempts, there is an emerging generation of QA systems, which are applied on Web of Data (*e.g.* [1, 96, 156]).

It is important to note that most of the available QA systems are more focused on implementation details and have limited reusability and extensibility in other QA approaches. Hence, considering the challenges of QA systems there is a need of a generalized approach for architecture or ontology of a QA system and semantic search to bring all state-of-the advancement of QA under a single umbrella. While many of these system achieved significant performance for special use cases, a shortage was observed in all of them. We figured out that the existing QA systems suffer from the following drawbacks: (1) potential of reusing its components is very weak, (2) extension of the components is problematic, and (3) interoperability between the employed components are not systematically defined. Therefore, there is a need for a descriptive approach that define a conceptual view of QA systems. This approach must cover all needs of current QA systems and be abstracted from implementation details. Moreover it must be open such that it can be used in future QA systems. This will allow interoperability, extensibility, and reusability of QA approaches and components of QA systems. In this paper, we introduce a generalized ontology which covers the need for interoperability of QA systems on a conceptual level. We initiate a step towards a message-driven interoperable approach that will be used to build systems which follow a philosophy of being actually open for extensions. Our approach collects and generalizes the necessitated requirements from the state-of-the-art of QA systems. We model the conceptual view of QA systems using and extending the Web Annotation Data Model. This model empowers us for designing a message-driven approach for QA systems. To the best of our knowledge, in this way we will establish for the first time a conceptual view of QA systems. The rest of this paper is structured as follows. Section 5.3.3 describes the diverse field of QA systems and its classification. It also covers current approaches for establishing an abstraction of QA systems and their limitations. Section 5.1.4 introduces dimensions of QA based on which many requirements to build open QA systems can be identified. Section 5.1.5 describes the existing problem and our proposed idea of an implementation independent compatibility level in detail. Section 5.1.6 details the requirement of message-driven QA systems which are derived from the state-of-the-art QA approaches. Section 5.1.7 illustrates our proposed ontology with a case study to address all the requirements. Conclusion and future work are presented in Section 5.1.8.

5.1.3 Related Work

In the recent past, different systems for QA have been developed. This section provides a brief overview on various QA system, their scope of applicability, and their different components. The QA systems can be distinguished based on scope of applicability and approaches. Some of them consider a specific domain to answer a query, they are known as *closed domain* QA systems. These QA systems are limited to a specific Knowledge Base

(KB), for example medicine [1]. However, when scope is limited to an explicit domain or ontology, there are less chances of ambiguity and high accuracy of answers. It is also difficult and costly to extend closed domain systems to a new domain or reusing it in implementing a new system.

To overcome the limitations of closed domain QA systems, researchers have shifted their focus to *open domain* QA systems. Freebase [18], DBpedia [8], and Google's knowledge graph [136] are few examples of open domain Knowledge Bases. Through KBs like Google's knowledge graph are not publically available. Open domain QA systems use publically available semantic information to answer questions. Other type of QA systems described in [142] extract answers from an unstructured corpus (*e.g.* news articles), or other various form of documents available over Web. They are known as corpus based QA systems. QuASE [142] is one of such corpus based QA system that mines answers directly from the Web.

In 2011, a yearly benchmark series QALD was introduced. In the latest advancements, QALD now focus on hybrid approaches using information from both structured and unstructured data. Many open domain QA systems now use QALD for the evaluation. PowerAqua [96] is an ontology based QA system which answers the question using the information that can be distributed across heterogeneous semantic resources. FREyA [36] is another QA system that increases system's QA precision by learning user's query formulation preferences. It also focuses to resolve ambiguity while using natural language interfaces. QAKiS [27] is an agnostic QA system that matches fragments of the question with binary relations of the triple store to address the problem of question interpretation as a relation-based match. SINA [132] is a semantic search engine which obtains either keyword-based query or natural language query as input. It uses a Hidden Markov model for disambiguation of mapped resources and then applies forward chaining for generating formal queries. It formulates the graph pattern templates using the knowledge base. TBSL [156] is a template based QA system over linked data that matches a question against a specific SPARQL query. It combines natural Language Processing capabilities (NLP) with linked data to produce good results w.r.t. QALD benchmark. The field of QA is so vast that the list of different QA systems can go long. Besides domain specific question answering, QA systems can be further classified on type of question (input), sources (structured data or unstructured data), and based on traditional intrinsic challenges posted by search environment (scalability, heterogeneity, openness, etc.) [103]. For a QA system, an input type can be anything ranging from keyword, factoids, temporal and spatial information (*e.g.* the geo-location of the user), audio, video, image etc. Many systems have been evolved for a particular input type. For example, DeepQA of IBM Watson [114], Swoogle [95], and Sindice [120] focus on keyword-based search whereas systems described in [83] integrates QA and automated speech recognition (ASR). Similarly, there are several examples of QA based on different sources used to generate an answer like Natural Language Interfaces to Data Bases (NLIDB) and QA over free text. Earlier in this section, we have observed that the field of QA is growing and new advancements are made in each of existing approaches over the short

period of time. However, there is a need of an open framework for generating QA system that integrates state-of-the-art of different approaches. Now we discuss some approaches for establishing an abstraction of QA systems and semantic search.

Research presented in [154] describes a Search Ontology to provide abstraction of user's question. User can create complex queries using this ontology without knowing the syntax of the query. This approach provides a way to specify and reuse the search queries but the approach is limited in defining properties represented within the ontology. Using Search Ontology, user can not define the dataset that should be used and other specific properties. The QALL-ME framework [67] is an attempt to provide a reusable architecture for multilingual, context aware QA frameworks. It is an open source software package and uses an ontology to model structured data of a targeted domain. This framework is restricted to closed domain QA, and finds limitation to get extended for heterogeneous data sources and open domain QA systems. The openQA [109] framework is an architecture that is dedicated to implement a QA pipeline. Additionally, here the implementation of a QA pipeline is limited to Java and cannot work agnostic to the programming language. For implementing an open framework for generating QA systems, it is important to define a generalized vocabulary for QA. It will be an abstraction level on top of all the existing QA approaches and will provide interoperability and exchangeability between them. This generalized vocabulary can be further used to integrate different components and web services within a QA system. DBpedia Spotlight [34] is an open source web service for annotating text documents with DBpedia resources. AIDA [182] is a similar project which uses the YAGO [140] ontology. While the last two examples address only specific ontologies, AGDISTIS [160] is an approach for named entity disambiguation that can use any ontology. To leverage the capabilities of different available web services and different tools for QA systems, a generalized vocabulary is needed.

5.1.4 Dimensions of Question Answering Systems

When we look at the typical pipeline of QA systems, the complete QA process is oriented to three main dimensions as follows: (i) *Query dimension:* covers all processing on input query. (ii) *Answer dimension:* refers to processing on the query answer. (iii) *Dataset dimension:* is related to the both characteristics of and processing over employed datasets. Figure 5.1 presents these high level dimensions. Generally, all of various known QA components or processes either associated or interacted with QA systems are corresponding to one of these dimensions. In the following, each dimension is discussed in more detail.

1. *Question Dimension:* The first aspect of this dimension refers to the characteristics of the input query. User-supplied questions can be issued through multifold interface such as voice-based, text-based, and form-based. Apart from the input interface, each query can be expressed in its full or partial form. For instance, full form of a

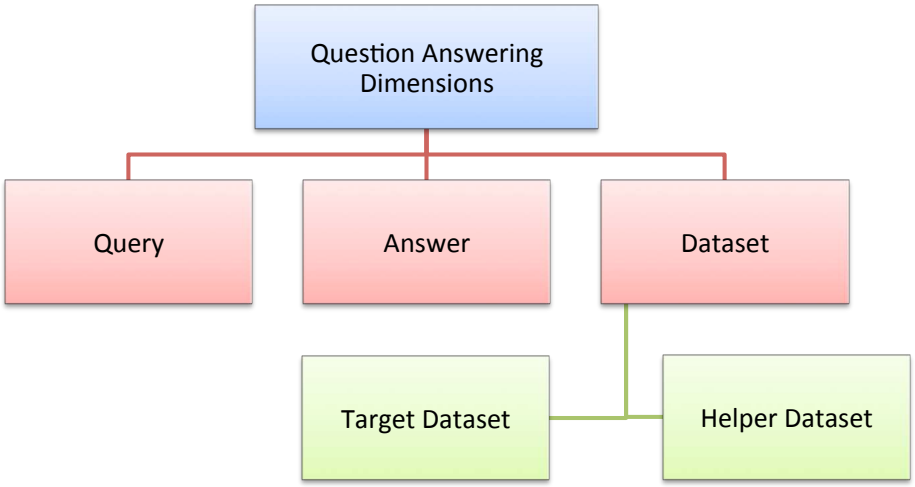

Figure 5.1: Main dimensions of question answering systems.

textual or voice query is a complete natural language query whereas partial form is an incomplete natural language query (i.e., either keyword-based query or phrase-based); or full form of a form-based query is a completion of all fields of the form. The second aspect of this dimension refers to processing techniques, which are run on the input query, *e.g.* query tokenization or Named Entity Recognition.

2. *Answer Dimension:* Similar to the input query, answer dimension (i.e., refers to the retrieved answer for the given input query) also can have its own characteristics. Answer can have different types (*e.g.* image, text, audio) also with full or partial form. For instance, a given query can have either a single resource or a list of items as answer (*e.g.* the query "islands of Germany" has a list of items as answer). The system might deliver a complete answer (the full list of items) or partial answer (i.e., a subset of items).

3. *Dataset Dimension:* This dimension also has a series of inherent characteristics such as (i) The type of a dataset refers to the format in which a dataset has been published, i.e., structured, semi-structured or unstructured. (ii) Domain of dataset specifies subject of information included (*e.g.* movies, sport, life-science and so forth). If the domain of a dataset covers all subjects, the dataset is open domain. In contrast, a closed domain dataset is limited to a few number of subjects. (iii) The size of data obviously shows how big the employed datasets are. Datasets hold two sub-dimensions as follows:

    a) *Helper Dataset Sub-dimension* This dimension includes all datasets required for (i) providing additional information, (ii) training the models. In other words, the helper dataset is used for annotating the input query. Dictionaries like WordNet, gazetteers are examples of this kind of dataset.

b) *Target Dataset Sub-dimension* Target datasets are leveraged for finding and retrieving answer of input query. For instance, Wikipedia can be a target dataset for search.

5.1.5 Problem and Idea

In this section we will present the problem observed from the state-of-the-art. Thereafter we will outline our idea for solving the problem. **Problem:** Considering the related work it is clear that three groups of problems exist:

1. *Lack of a generic conceptional view on QA systems:* While there are many different architectures for QA systems (*e.g.* [142, 83, 109, 96]), most of them are tailored to specific and limited use cases as well as applications. Reusability and interoperability was not (enough) in the focus of such approaches. Creating new QA systems is cumbersome and time consuming as well as limited to domains or programming languages.

2. *No standardized message format for QA systems:* While there are plenty of available tools and services employed by QA systems, yet, interoperability is not ensured due to a missing message format. However, there might be great synergy effects while creating QA systems w.r.t. the combination of different tools. For example, in a given architecture, Named Entity Recognition (NER) and Named Entity Disambiguation (NED) are integrated in a single tool. NER solutions might be evolved and thus, implemented in either a novel way (*e.g.* in [34]) or employ existing tools (*e.g.* the Stanford NER [70] used in [182]). Thus, integrating a (new) NER approach without a standardized message format is also cumbersome and time consuming. However, integrating different components is very difficult and causes a lot of work for each new implementation.

3. *Scalability and Coverage problem:* Existing schema-driven approaches are mainly focus on the input query (even limited to textual representations of input query), and aren't flexible for fulfilling emerging demands which are not discovered yet (*e.g.* [154]).

Hence, considering the implementations of current QA systems (and their components) it can be observed they do not achieve compatibility due to their concentration on the implementation instead of the conceptial view. Instead, implementation details need to be hidden and the focus has to be on the message format communicated between the components of the QA system. **Idea:** A conceptual view of QA systems has to be completely implementation-independent. Therefore, we introduce a vocabulary (i.e., schema) that addresses abstract definitions of the data needed for solving QA tasks. We will describe the data model by following the mentioned dimensions (cf., Sec. 5.1.4).

1. *Input Query:* The abstract definition of the input query along with its properties used for the interpretation and transformation leading towards a formal query of the given dataset.

2. *Answer:* The abstract definition of the answer (i.e., the search result for the given input query) covering all its associated properties.

3. *Dataset:* The abstract definition for all kinds of data being used as background knowledge (i.e., for interpreting the input query and retrieving the answer).

Please note that we do not describe a specific architecture. Instead our focus is the conceptual level, i.e., the format of the message that needs to be used as input and returned as output by the components of the QA system. Hence, properties need to be *annotated* to the question to make them available for the following components in the QA system pipeline. As each component of the QA system will use the message to retrieve and encode its knowledge about the question from the message, QA applications following this idea are called *message-driven*. Hence, information need to be *annotated* to the message to make it available for following components in the QA system pipeline.

We adapt the definition of annotations from the Web Annotation Data Model[3].

Definition 2 (Annotation)**.** *An annotation is an object having the properties* body *and* target. *There should be associated one or more instances of the body property of an annotation object, but there might be zero body instances. There must be one or more instances of the target property of an annotation.*

E.g. considering the question "Where was the European Union founded?" (target) it might be annotated that it contains the named entity "European Union" (body).

In many circumstances, it is required to retrieve the *annotator* (i.e., the creator) of *annotations*. Thus, we demand the provenance of each annotation to be expressable (*e.g.* while using several NED components and later pick one interpretation). We manifest this in the following requirement:

Req. 3 (Provenance of Annotations)**.** *The provenance of each annotation needs to be representable within the data model. The annotator needs to be a resource that identifies the agent responsible for creating the annotation.*

Hence, if annotations are available, then each atom of the question can be annotated with additional information.

[3]W3C First Public Working Draft 11 December 2014, `http://www.w3.org/TR/annotation-model/`

Definition 4 (Question Atom)**.** *The smallest identifiable part of a question (user query) is called question atom and denoted by q_a. Thus, each given user query Q independent of its type consist of a sequential set of atoms $Q = (q_1, q_2, \ldots, q_n)$.*

E.g. while considering the question to be a text, the characters of the string or the words of the query might be considered as question atoms, while a user query in the form of an audio input (*e.g.* for spoken user queries) might be represented as byte stream. Considering textual questions, main component might be parser or Part of Speech taggers. They are used to identify relations between the terms in a question. These relations can have a tree structure like in the case of dependency trees but also more complex ones like direct acyclic graphs (DAG) that are used for example in Xser [170]. Some QA systems such as gAnswer [188] use coreference resolution tools, i.e., finding phrases that refer to some entity in the question. Moreover, a typical goal of QA systems is to group phrases in triples that should reflect the RDF structure of the given dataset. These examples imply the following requirement:

Req. 5 (Relations between Annotations)**.** *(a) It has to be possible to describe relations between annotations. (b) Using these relations, it has to be possible to describe a directed or undirected graph (of annotations).*

Annotations of components do not always have boolean characteristics. It is also possible to assign a confidence, (un)certainty, probablity, or (in general) score for the annotations. Once again an example is the NED process, where for entity candidates also a certainty is computed (like in [34]). This implies the following requirement:

Req. 6 (Score of Annotations)**.** *It should be possible to assign a* score *to each annotation.*

Note: The type of the score is an implementation detail, *e.g.* in [34] the confidence (score) is within the range $[0, 1]$ while in [130] the score might be any decimal number.

In the next section the three main dimensions of QA system are described and necessary requirements are derived.

5.1.6 Requirements of Message-driven Approach

In this section while aiming for a conceptional level the requirements of message-driven approach for describing QA systems are derived from the state-of-the-art (cf., Section 5.3.3). We present them following the dimensions of QA systems as described in Section 5.1.4. Hence, on the one side a data model for messages in QA systems should be able to describe at least actual QA systems. On the other side the data model has to be flexible and extensible since it is not known how future QA systems will look like nor which kind of annotations their components will use. In general, there are two main attributes which we have to take into account:

- The proposed approach should be comprehensive in order to catch all known annotations used so far in QA systems.
- It should have enough flexibility for future extensions in order to be compatible with the upcoming annotations.

Input Query

The input query of a QA system can be of various types. For example it might be a query in natural language text (*e.g.* [156]), a keyword-based query (*e.g.* [132]), an audio stream (*e.g.* [105]), or a resource-driven input (*e.g.* [24]). In all these cases the parts of an input query need to be identifiable as a referable instance such that they can be annotated during the input query analysis. Hence, we define the following requirement:

Req. 7 (Part of the Input Query)**.** *The definitions of* part of the input query *satisfies the following conditions: (i)* Part consists of a non-empty set of parts and atoms: *each part might be an aggregation of atoms and other parts. However, the transitive closure of the aggregation of each part needs to contain at least one question atom. (ii) For an input query an arbitrary number of* parts *can be defined.*

Please note that as we mentioned before, all of the requirements or definitions are independent of any implementation details. Thus, for instance, input query, atom or part have to been interpreted conceptually.

Examples from an implementation view are as follows: for text queries the NIF [85] vocabulary might be used to identify each part by its start and end position within the list of characters. Similarly, the Open Annotation Data Model [131] selects parts of a binary stream by indicating the start and end position within the list of bytes. We leave the actual types of atoms and properties of parts open, as it is depending on the implementation of the actual QA system.

In a QA system the components of the analytics pipeline will annotate the parts of the query. Examples for such components are Part-of-Speech (POS) taggers (*e.g.* in [96]), Named Entity Recognition (NER) tools (*e.g.* in [27]) or Named Entity Disambiguation (NED) tools (like [160]). One possible scenario for a textual question is that first several parts are computed (*e.g.* with a POS-tagger). Thereafter a NED component might annotate the parts with the additional properties, expressing the current state of the question analytics, using the properties that are accepted in the NED community. As it is not known what kind of properties are annotated by which component, we will not define them here. Hence, we have to keep the annotations open for on-demand definition:

Req. 8 (Annotations of Parts)**.** *It has to be possible to define an arbitrary number of annotations for each part.*

E.g. for the input textual query "capital of Germany", the part "Germany" might be annotated by a NER tool as place (*e.g.* while using `dbo:place`[4]).

Answer

Each QA system is aiming at the computation of a result. However, considering the QA task there are some demands of the type of the answer. For example, the QA task might demand a boolean answer (*e.g.* for "Did Napoleon's first wife die in France?"), a set of resources (*e.g.* for "Which capitals have more than 1 mio. inhabitants?"), a list of resources, just one resource etc. Therefore, we define the following requirement:

Req. 9 (Answer)**.** *The question needs to be annotated with an object typed as answer. A resource of the type answer might be annotated with a property describing the type of the QA task (*e.g. *boolean, list, set, . . .).*

Of course, it is possible that the answer type is pre-configured by the user as well as that it needs to be derived automatically by the QA system from the given question.

Additionally only several types might be acceptable for the items been contained in the answer. *E.g.* given the question "Who was the 7th secretary-general of the United Nations?" only specific resource types are acceptable as answer items (w.r.t. the given data). Here it might be `dbo:person`. From this observation we derive the following requirement.

Req. 10 (Types of Answer Items)**.** *An arbitrary number of types can be annotated, to express the types acceptable for the items within the answer.*

As an answer is also an annotation of the question, the answer, its answer item types and any additional pieces of information might also be annotated with a score (cf., Req. 6), the annotator (cf., Req. 3) etc.

Dataset

The proposed data model needs to take into account information about the employed datasets. The dataset dimension and its sub-dimensions were introduced in the section 5.1.4. To include these dimensions to the data model, the following requirements are met:

Req. 11 (Dataset)**.** *A* dataset *provides an* endpoint *where the data can be accessed and statements about the dataset* format *can be gathered.*

Req. 12 (Helper Dataset)**.** *A question should be annotated by an arbitrary number of helper datasets (which are subclass of dataset class).*

[4] `@prefix dbo: <http://dbpedia.org/ontology/>`

Req. 13 (Target Dataset)**.** *At least there is one target dataset (which is subclass of dataset class). Both question and answer should be annotated by at least one target dataset (number of target datasets is arbitrary).*

These requirements enables QA system components to easily (1) spot data, (2) access data, (3) query data. Please note that target datasets might overlap with the helper data sets and vice versa.

5.1.7 Case Study

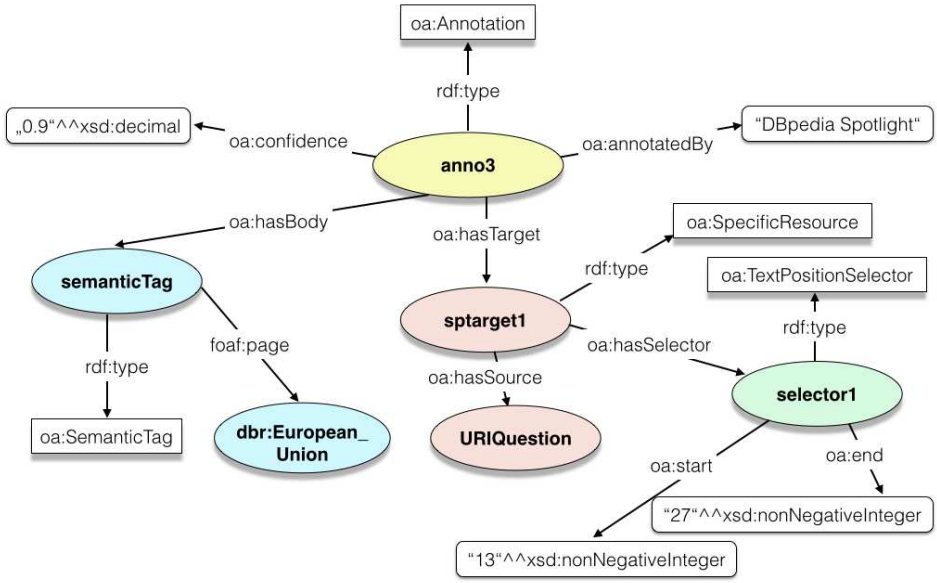

Figure 5.2: This picture represents an annotation of the question "Where was the European Union founded?". The part "European Union" is selected using a Specific Resource and a Selector. Moreover a semantic tag is associated to it.

In the previous section, we have collected the requirements for a data model describing the message of interoperable QA systems. As case study we know focus on an ontology that fulfills these requirements (although other formal representation will also comply with the requirements). Here, the Web Annotation Data Model (WADM[5]) is used as basis that is currently a W3C working draft. The WADM is an extensible, interoperable framework for expressing annotations and is well accepted. In the following it is shown that how the identified requirements are met.

The WADM introduces the class `Annotation` of the vocabulary `oa`[6]. Thus, any annotation can be defined as an instance of this class. The class `Annotation` has two major characteristics as the body and the target. The body is “about” the target and it can be

[5] W3C First Public Working Draft 11 December 2014, `http://www.w3.org/TR/annotation-model/`

[6] `@prefix oa: <http://www.w3.org/ns/oa#> .`

changed or modified according to the intention of the annotation. The basic annotation model is represented in Turtle format[7] as follows:

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

The above pseudocode describes an annotation instance which is identified by `anno`. The `anno` has the properties `target` and `body` (i.e., each one is a resource). In the following, we extend the WADM in order to meet all requirements. For this purpose, a new namespace is introduced:

```
@prefix qa: <urn:qa> .
```

In order to illustrate the implications, we use a running example with the question "Where was the European Union founded?". First an instance with the type `qa:Question` is instantiated with the identifier `URIQuestion`. We extended the data model as the input query needs to be defined as well as the answer and the dataset. These concepts are represented by the classes `qa:Question`, `qa:Answer` and `qa:Dataset` which are used to identify questions, answers and datasets. For the example also a URI for the answer `URIAnswer` and for the dataset `URIDataset` is introduced. Then one can establish the corresponding instances:

```
<URIQuestion> a qa:Question .
<URIAnswer> a qa:Answer .
<URIDataset> a qa:Dataset .
```

These annotations instantiates question, answer and dataset object. To establish annotation of a question instance we introduce a new type of annotation namely `qa:AnnotationOfQuestion`. It is defined as follows:

```
qa:AnnotationOfQuestion rdf:type owl:Class ;
           rdfs:subClassOf oa:Annotation ;
           owl:equivalentClass [
             rdf:type owl:Restriction ;
             owl:onProperty oa:hasTarget ;
             owl:someValuesFrom qa:Question ].
```

This means that annotations of this type need to have a target of type question. Analogously two new annotation types are introduced `qa:AnnotationOfAnswer` and `qa:AnnotationOfDataset`. In our example the question is annotated with an answer (`anno1`) and a dataset (`anno2`).

```
<anno1> a oa:AnnotationOfQuestion ;
      oa:hasTarget <URIQuestion> ;
```

[7] http://www.w3.org/TR/owl2-manchester-syntax/RDF 1.1 Turtle, W3C Rec. 2014-02-25, `http://www.w3.org/TR/turtle/`

```
      oa:hasBody <URIAnswer> .
<anno2> a oa:AnnotationOfQuestion ;
      oa:hasTarget <URIQuestion> ;
      oa:hasBody <URIDataset> .
```

Now, we will consider requirement 7. To select parts of a query the WADM introduces two concepts: *Specific Resources* and *Selectors*. In the WADM, there is a class called Specific Resource (`oa:SpecificResource`) for describing a specific region of another resource called source. We use this class for typing the concept of part of query in our data model. Assume "European Union" is a part of the input query. For this part, we instantiate a resource with the identifier `sptarget1` and the type `oa:SpecificResource`. The WADM introduces the property `oa:hasSource` which connects a specific resource to its source. In our example, the source of `sptarget1` is `URIQuestion` stating that "European Union" is a part of the input query. Another relevant class which can be captured from the WADM is the class `oa:Selector`. It describes how to derive the specific resource from the source. In our example we instanciate the resource `selector1` which is a particular type of selector, namely a `oa:TextPositionSelector`. It describes that the part "European Union" can be selected in the input query between the character 13 and 27. This is indicated using the properties `oa:start` and `oa:end`. This can be expressed via:

```
<sptarget1> a oa:SpecificResource;
         oa:hasSource <URIQuestion>;
         oa:hasSelector <selector1> .
<selector1> a oa:TextPositionSelector;
         oa:start 13 ;
         oa:end 27 .
```

WADM introduces other types of selectors like *Data Position Selectors* for byte streams and *Area Selectors* for images. Hence, the requirement 7 is fulfilled. It is obvious that we can instantiate an arbitrary number of annotations for each part of a question. Thus, the requirement 8 is also met. The WADM defines the property `oa:annotatedBy` to identify the agent responsible for creating the Annotation, s.t., requirement 3 is fulfilled. To comply with requirement 6 a new property `qa:score` with domain `oa:Annotation` and range `xsd:decimal` is introduced. *E.g.* if "European Union" is annotated by DBpedia Spotlight[8] with a confidence (score) of `0.9`, this can be expressed as:

```
<anno3> a oa:Annotation ;
          oa:hasTarget <sptarget1> ;
          oa:hasBody <semanticTag> .
<semanticTag> a oa:SemanticTag ;
          foaf:page dbr:European_Union .
<anno3> oa:annotatedBy DBpedia spotlight ;
          oa:score "0.9"^^xsd:decimal .
```

[8] `@prefix dbr: <http://dbpedia.org/resource/>`

To fulfill requirement 9, in our data model a new class `qa:AnswerFormat` and a new type of annotation `qa:AnnotationOfAnswerFormat` are introduced:

```
qa:AnnotationOfAnswerFormat a owl:Class ;
         rdfs:subClassOf oa:AnnotationOfAnswer;
         owl:equivalentClass [
           rdf:type owl:Restriction ;
           owl:onProperty oa:hasBody ;
           owl:someValuesFrom qa:AnswerFormat
         ].
```

If the expected answer format is a string, then this can be expressed with the following annotation:

```
<anno4> a qa:AnnotationOfAnswerFormat ;
       oa:hasTarget <URIAnswer> ;
       oa:hasBody <body4> .
<body4> a qa:AnswerFormat ;
       rdfs:label "String" .
```

Although later a resource will be used (instead of the `rdfs:label`), this shows that the requirement 9 is met. Requirement 10 is analogously satisfied. Now the requirements for datasets are considered. To fulfill requirement 11 a new class `qa:Endpoint` is introduced having the property `qa:hasFormat` and a new annotation `qa:AnnotationOfEndpointOfDataset`:

```
qa:AnnotationOfEndpointOfDataset a owl:Class ;
   rdfs:subClassOf oa:AnnotationOfDataset ;
   owl:equivalentClass [
     rdf:type owl:Restriction ;
     owl:onProperty oa:hasBody ;
     owl:someValuesFrom qa:Endpoint
   ].
```

If the question in the example should be answered using a SPARQL endpoint available under the URI `body5` (*e.g.* `http://dbpedia.org/sparql`), this might be expressed by:

```
<anno5> a oa:AnnotationOfEndpointOfDataset;
      oa:hasTarget <URIDataset> ;
      oa:hasBody <body5> .
<body5> a qa:Endpoint ;
      qa:hasFormat dbr:SPARQL .
```

To fulfill requirements 12 and 13 two new classes are introduced `qa:HelperDataset` and `qa:TargetDataset`. Both are subclasses of `qa:Dataset`. If DBpedia is considered to be a target dataset, this can be expressed as follows while `URIDataset` is `http://dbpedia.org`:

```
<URIDataset> a qa:TargetDataset ;
          rdfs:label "DBpedia version 2015" .
```

Relations between two annotations <annoX> and <annoY> with a label can be expressed using a new annotation in the following way:

```
<annoZ> a oa:Annotation ;
      oa:hasTarget <annoX> ;
      oa:hasBody <annoY> ;
      rdfs:label "my anotation label" .
```

This corresponds to a directed edge between the two annotations. Representing undirected edges is possible in a similar way. This shows that requirement 5 is also fulfilled.

To sum up, in this case study we expressed the demanded datamodel with a generalized ontology which is reusing the concepts of the Web Annotation Data Model. We have shown that it is able to address all the requirements identified in Section 5.1.6. Moreover, we have illustrated the usage with an example. While using annotation, the data model is extensible and open for improvements. The complete case study is available as online appendix at https://goo.gl/vECgK5.

5.1.8 Conclusion and Future Work

In this paper we have motivated the high demand for an ontology which covers the need for interoperability of QA systems on a conceptual level. We distinguish our approach from other attempts to establish a QA system architecture. Instead we focus on the message level, s.t., everything needed to establish a QA system is included within the data model. Given the requirements and the corresponding case study to the best of our knowledge, we have established for the first time a message-driven interoperable approach that follows a philosophy aiming for QA systems actually open for extension.

Consequently, we collected the requirements for the data model from the recent works to cover also the needs of existing QA systems. However, previous work consider only specific scopes of applicability of QA systems (particularly many focus on text-driven QA). We have chosen an approach that is agnostic to the implementation and the actual representation of the input question and the answer as well as the data sets. Hence, it is a descriptive and open approach. This is an important milestone on the way to actual open QA systems.

We have shown in our case study that our requirements can be covered by the Web Annotation Data Model. Hence, a logical, extensible, and machine-readable representation is now available fulfilling the collected requirements. Eventually the proposed approach can be used for all the existing QA systems while transforming them into a message-driven (and

therefore interoperable) implementations. We see this work as the first step in a larger research agenda. Based on the presented data model (or its implementation as an ontology) the research community is enabled to establish a new generation of QA systems and components of QA systems that are interoperable. Hence, actually open QA systems are in sight. Based on our contribution the research community and the industry can work with a best-of-breed paradigm while establishing future QA systems and integrate novel components into their QA systems. Additionally, our approach enables developers to integrate several components with the same task (*e.g.* NED) and thereafter compare the results (*e.g.* with ensemble methods) to achieve the best results w.r.t. the considered domain or data. Consequently we will also aim at an implementation of an open QA system that can be used for comparable benchmarks (like it was done for entity annotation in [161]) strengthening the competition of different approaches as well as measuring the influence of different data sets.

5.2 Qanary – A Methodology for Vocabulary-driven Open Question Answering Systems [9]

5.2.1 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

5.2.2 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.* [96, 35, 27, 156]). 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.

[9] Corresponding publications is: Andreas Both, Dennis Diefenbach, Kuldeep Singh, Saedeeh Shekarpour, Didier Cherix, Christoph Lange, *Qanary – A Methodology for Vocabulary-driven Open Question Answering Systems*,ESWC 2015 [22]

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 [109]. 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 [67]. Furthermore it will be possible to apply quality increasing approaches such as ensemble learning [57] 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` [133]. `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[10], YAGO[11]) or representing the analytics results of data (*e.g.* NLP Interchange Format [85], Ontology for Media Resources[12]). 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. 5.3.3 reviews related work. In Sec. 5.2.5 the problem is will be broken down to actual requirements. Thereafter, we present our approach (Sec. 5.2.6) followed by a methodology to align existing vocabularies to our `qa` vocabulary in Sec. 5.2.7. In Sec. 5.2.8, we present a case study where a QA system is created containing actual reusable and exchangeable components. Sec. 5.3.7 concludes, also describing future research tasks.

5.2.3 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.) [103].

Closed domain QA systems target specific domains to answer a question, for example, medicine [1] or biology [6]. 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 [8], and Google's non-public knowledge graph [136] are examples of semantically structured general-purpose *Knowledge Bases* used by open domain QA systems. Recent examples of such QA systems include PowerAqua [96], FREyA [35], QAKiS [27], and

[10] `http://dbpedia.org/services-resources/ontology`
[11] YAGO: A High Quality Knowledge Base; `http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`
[12] W3C Recommendation 09 February 2012, v1.0, `http://www.w3.org/TR/mediaont-10/`

TBSL [156]. QuASE [142] 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 provides better results in linguistic analysis [156]. It would thus be desirable to combine these functionalities of [156] and [97] into a new, more powerful system.

For example, the open source web service DBpedia Spotlight [34] analyzes texts leading to named entity identification (NEI) and disambiguation (NED), using the DBpedia ontology (cf., section 5.2.4). AIDA [88] is a similar project, which uses the YAGO ontology (cf., section 5.2.4). AGDISTIS [160] 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 [116] 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.

5.2.4 Related Work

We have already reviewed the state of the art of QA systems in section 5.3.4. 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.

Abstract QA frameworks

The QALL-ME framework [67] 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 [109] 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 [132] and TBSL to answer questions. openQA requires all components of the pipeline to be implemented in Java. The OKBQA Hackathon [13], on the other hand, is a collaborative effort to develop knowledge bases and question answering systems that are generic and independent of programming languages.

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

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".[14]

The YAGO ontology unifies semantic knowledge extracted from Wikipedia with the taxonomy of WordNet. The YAGO Knowledge Base contains more than 10 million 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 [154] 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.

5.2.5 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 5.3.4. Related work on abstract frameworks and QA ontologies has not yet solved the interoperability problem fully, as explained in section 5.3.3. 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.

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

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

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 5.3.4 and 5.3.3 that the number of reusable (components of) QA systems is negligible so far.

Requirements

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

Req. 14 (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. 15 (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. 16 (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. *[34]), other components might target the Named Entity Disambiguation (NED) only (*e.g. *[160]) and additionally there might exist components providing just an integrated interface for NEI and NED in a processing step.*

Req. 17 (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.

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[15]), 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. 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. 14. 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. 15, 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. 5.2.8) as it was introduced earlier for search-driven processes on hybrid federated data sources (cf., [23]). However, the methodology might be implemented by different architectures.

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

5.2.6 Approach

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.

Vocabulary for Question Answering Systems

In [133] 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 [133].

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 [133].

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 [115]. 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`.

5.2.7 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.

```
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 5.3: Aligning identified NE to a new `qa` annotation using SPARQL

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. 5.2.8). In the following the components are described briefly and also a possible alignment implementation of the custom vocabulary to `qa`.

NE Identification and Disambiguation via DBpedia Spotlight

DBpedia Spotlight [34] 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 following first-order rule:

$$\begin{aligned} &\text{rdf:type}(?s, \text{nif:String}) \wedge \text{nif:beginIndex}(?s, ?b) \wedge \text{nif:endIndex}(?s, ?e) \\ \Longrightarrow\ &(\exists ?x \bullet \text{rdf:type}(?x, \text{oa:TextPositionSelector}) \wedge \text{oa:start}(?x, ?b) \wedge \text{oa:end}(?x, ?e)) \end{aligned} \tag{5.1}$$

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:

$$\begin{aligned} &\text{itsrdf:taIdentRef}(?s, ?NE) \wedge \text{nif:confidence}(?s, ?conf)t \\ \Longrightarrow\ &\text{rdfs:subClassOf}(\text{qa:AnnotationOfEnitites}, \text{oa:AnnotationOfQuestion}) \wedge \\ &(\exists ?sp \bullet \text{rdfs:type}(?sp, \text{oa:SpecificResource}) \wedge \text{oa:hasSource}(?sp, \text{<URIQuestion>}) \wedge \\ &\text{oa:hasSelector}(?sp, ?s)) \ \wedge \ (\exists ?x \bullet \text{rdfs:type}(?x, \text{oa:AnnotationOfNE}) \wedge \\ &\text{oa:hasBody}(?x, ?NE) \wedge \text{oa:hasTarget}(?x, ?sp) \wedge \text{qa:score}(?x, ?conf)) \end{aligned} \tag{5.2}$$

Fig. 5.3 shows our SPARQL implementations of this rule. After applying this rule, named entities and their identified resources are available within the `qa` vocabulary.

Relation detection using PATTY lexicalization

PATTY [116] 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. 5.3.

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. 15 and Req. 17 are fulfilled).

Query Construction and Query Execution via SINA

SINA [132] 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.

$$\begin{aligned} & \text{sparqlSpec:select}(?x, ?t) \wedge \text{rdf:type}(?t, \text{xsd:string}) \\ \Longrightarrow\ & \text{rdfs:subClassOf}(\text{oa:SparqlQueryOfAnswer}, \text{oa:AnnotationOfAnswer}) \wedge \\ & (\exists ?x \bullet \text{rdfs:type}(?x, \text{oa:SparqlQueryOfAnswer}) \wedge \text{oa:target}(?x, \text{<URIAnswer>}) \wedge \\ & \text{oa:body}(?x, \text{“SELECT ...”})) \end{aligned} \tag{5.3}$$

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.

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. 17 is fulfilled, and we achieving Req. 15 by being able to exchange every component.

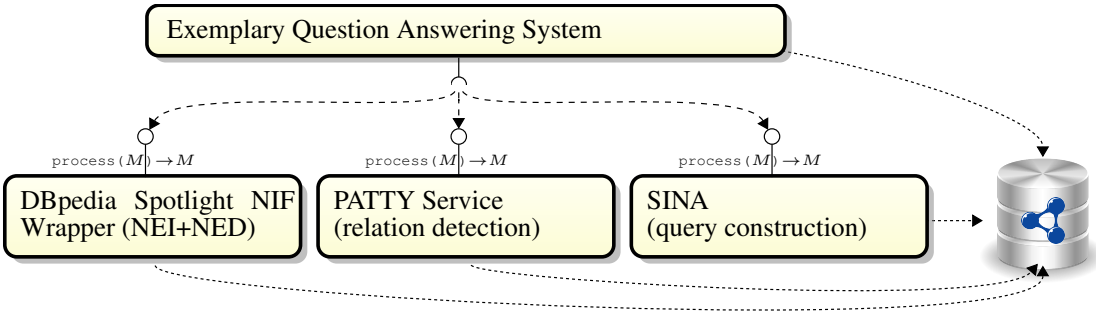

Figure 5.4: Architecture of the exemplary question answering system.

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. 5.2.7, implemented as an OWL axiom, is given in the online appendix[16]).

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).

5.2.8 Case Study

In this section we present a QA system that follows the idea presented in Sec. 5.2.5. 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 components is possible in an architecture derived from the Qanary methodology. In this paper, we have extended the vocabulary proposed in [133] 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[17].

The whole architecture is depicted in Fig. 5.4. 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

[16] alternative alignment: `https://goo.gl/hdsaq4`
[17] `http://stardog.com/`, community edition, version 4.0.2

Sec. 5.2.6. Step by step each component receives a message M (cf., Fig. 5.4) 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 Spotlight is transformed using the alignment from section 5.2.7 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. 5.2.7).

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. 17), exchangeable and reusable (cf., Req. 15), as the exchanged messages follow the `qa` vocabulary (cf., Req. 14), 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. 16 is also fulfilled.

The case study is available as online appendix[18].

5.2.9 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 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

[18] `https://github.com/WDAqua/Pipeline`

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.

5.3 The Qanary Ecosystem: getting new insights by composing Question Answering pipelines [19]

5.3.1 Abstract

The field of Question Answering (QA) is very multi-disciplinary as it requires expertise from a large number of areas such as natural language processing (NLP), artificial intelligence, machine learning, information retrieval, speech recognition and semantic technologies. In the past years a large number of QA systems were proposed using approaches from different fields and focusing on particular tasks in the QA process. Unfortunately, most of these systems cannot be easily reused, extended, and results cannot be easily reproduced since the systems are mostly implemented in a monolithic fashion, lack standardized interfaces and are often not open source or available as Web services. To address these issues we developed the knowledge-based Qanary methodology for choreographing QA pipelines distributed over the Web. Qanary employs the `qa`vocabulary as an exchange format for typical QA components. As a result, QA systems can be built using the Qanary methodology in a simpler, more flexible and standardized way while becoming knowledge-driven instead of being process-oriented. This paper presents the components and services that are integrated using the `qa`vocabulary and the Qanary methodology within the Qanary ecosystem. Moreover, we show how the Qanary ecosystem can be used to analyse QA processes to detect weaknesses and research gaps. We illustrate this by focusing on the Entity Linking (EL) task w.r.t. textual natural language input, which is a fundamental step in most QA processes. Additionally, we contribute the first EL benchmark for QA, as open source. Our main goal is to show how the research community can use Qanary to gain new insights into QA processes.

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

5.3.2 Introduction

The amount of data, information, and knowledge available on the Web and within enterprise environments is increasing constantly. Especially in enterprise environments a strong trend to better connected data can be observed, leading to interlinked and accessible data unlocking the company's information for intense data analytics and information retrieval. Novel interfaces are required for enabling users to retrieve information in such scenarios and interact with it. Natural language interfaces are being considered to bridge the gap between large amounts of (semi-structured) data and users' needs. Recent industrial applications

[19] Corresponding publications is: Dennis Diefenbach, Kuldeep Singh, Andreas Both, Didier Cherix, Christoph Lange, Sören Auer, *The Qanary Ecosystem: getting new insights by composing Question Answering pipelines*,ICWE 2017 [50]

show the capabilities and advantages of natural language interfaces in the field of Question Answering (QA). These include *Apple Siri*[20], *Microsoft Cortana*[21], and "Ok Google"[22]. However, these proprietary platforms do not facilitate experimentation with cutting-edge research approaches, they offer only limited interfaces for integrating third-party components and they are generally not open, reusable and extensible by developers and the research community.

Several QA systems have been developed recently in the research community, for example, [132, 71, 27, 58]. These systems perform well in specific domains, but their reusability for further research is limited because of their focus on specific technologies, applications or datasets. As a result, creating new QA systems is currently still cumbersome and inefficient. Particularly, the research community is not empowered to focus on improving particular components of the QA process, as developing a new question answering systemand integrating a component is extremely resource-consuming. Some first steps for developing flexible, modular question answering systems have started to address this challenge, *e.g.* [109, 67]. However, these approaches lack several key properties required for constructing QA systems in a community effort as they are, for example, bound to a particular technology environment and have rather static interfaces, which do not support the evolution of the inter-component data exchange models. For this reason we presented the `qa`vocabulary [133] as a flexible and extensible data model for QA systems. Based on the vocabulary, we developed the Qanary [22] methodology for integrating components into QA systems; it is independent from programming languages, agnostic to domains and datasets, as well as enabled for components on any granularity level within the QA process.

This work presents the Qanary *ecosystem*: the components and services currently implemented around the `qa`vocabulary by using the Qanary methodology. We present a general workflow that can be used to construct and particularly analyze as well as optimize future QA systems in a community effort using the Qanary ecosystem. It can be broken down into two phases: (1.) the identification and integration of existing state-of-the-art approaches to solve a particular task in the QA pipeline, and (2.) the derivation of benchmarks for sub-tasks of a QA process from well-known QA benchmarks such as the Question Answering over Linked Data (QALD) challenge[23]. Hence, the described approach is dedicated to support the engineering process to build components for a QA system and the system by itself, by using the knowledge-driven approach for flexible component integration and quality evaluations. In this paper, we show this workflow applied to the task of EL, which is key in the QA process. Therefore, we consider components dedicated to the tasks of named entity identification/recognition (NER) and named entity disambiguation (NED), which are integrated into the Qanary ecosystem. The included components are the NER tool of *DBpedia Spotlight* [112], the *Stanford NER tool* [70] and the *Federated knOwledge*

[20] `http://www.apple.com/ios/siri/`
[21] `http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana`
[22] `https://support.google.com/websearch/answer/2940021?hl=en`
[23] `http://greententacle.techfak.uni-bielefeld.de/cunger/qald`

eXtraction Framework (FOX) [139] as well as the NED components *Agnostic Disambiguation of Named Entities Using Linked Open Data* (AGDISTIS) [160] and the named entity disambiguator of DBpedia Spotlight. In addition two combined approaches for NER and NED are also provided as components: *IBM Alchemy*[24] and *Lucene Linker* – a component that we implemented following the idea of the QA system *SINA* [132]. Moreover, we devised a benchmark for entity linking (EL) based on the well-known Question Answering over Linked Data (QALD) challenge. 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 their novel functionality. Second, we provide the results of comparing existing tools, i.e., an expressive benchmark for the quality of entity linking components w.r.t. natural language questions, thus enabling the QA community to gain new insights into QA processes. Third, we compute a list of questions that are completely annotated w.r.t. the entity linking process. Hence, researchers investigating a processing step of a QA system that comes after entity linking can reuse these annotations to create an environment for conveniently testing and continuously improving their components.

As a result, the QA community is empowered to easily reuse entity linking functionality for QA systems (or for the development of other tools depending on named entities) and reuse a profound 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). However, the entity linking functionality and experiments presented in this paper are just a proof that Qanary's knowledge-driven and component-driven approach as well as the previously described general workflow provides key advantages particularly in contrast to existing systems and other benchmarks.

The next section describes related work. Sec. 5.3.4 gives an overview of our recent work which laid the groundwork for the Qanary ecosystem. Sec. 5.3.5 gives an overview of the components and services that are available in the Qanary ecosystem. Sec. 5.3.6 describes how the Qanary ecosystem can be used to gain new insights into QA processes w.r.t. the EL task. Sec. 5.3.7 concludes and points to future research areas.

5.3.3 Related Work

In the context of QA, a large number of systems and frameworks have been developed in the last years. This can be observed for example from the number of QA systems (> 20 in the last 5 years) that were evaluated against the QALD benchmark. Many QA systems use similar techniques. For example, there are services for named entity identification (NER) and disambiguation (NED) such as DBpedia Spotlight [112] and Stanford NER [70], which are reused across several question answering systems. These reasons led to the idea of

[24] http://www.alchemyapi.com/

Property vs. Tool	NERD	Gerbil	Gerbil-qa	Qanary
support for analyzing NER/NED task	■	■	□	■
support for analyzing QA process quality	□	□	◪	■
third party evaluations	□	■	■	■
fine-grained information	■	□	□	■
traceability of intermediate results	◪	□	□	■
computation of citable URIs	□	■	■	□

Table 5.1: Overview of tools related to benchmarks in the field of question answering in comparison to the benchmark functionality of Qanary.

developing component-based frameworks that make parts of QA systems reusable. We are aware of three frameworks that attempt to provide a reusable architecture for QA systems. The first is QALL-ME [67] which provides a reusable architecture skeleton for building multilingual QA systems. The second is openQA [109], which provides a mechanism to combine different question answering systemsand evaluate their performance using the QALD-3 benchmark. The third is the Open KnowledgeBase and Question-Answering (OKBQA) challenge[25]. It is a community effort to develop a QA system that defines rigid JSON interfaces between the components. Differently from these works we do not propose a rigid skeleton for the QA pipeline, instead we allow multiple levels of granularity and enable the community to develop new types of pipelines.

Recognizing named entities in a text and linking them to a knowledge base is an **essential task** in QA. DBpedia Spotlight [112], Stanford NER [70], FOX [139], and Alchemy API are a few of the tools dedicated to such tasks. Furthermore, tools such as DBpedia Spotlight, AGDISTIS [160], Alchemy API etc. not only identify information units in text queries but also point every named entity to a knowledge resource for disambiguation.

We are not aware of any work that has tried to compare in a systematic way existing approaches that tackle sub-processes of QA pipelines, for example EL. Atdag and Labatut [5] compare a few NER tools applied to bibliographic text, whereas researchers in [127] present NERD, a framework for evaluating NER tools in the context of Web data where a wrapper of NER/NED services was implemented but the independent registration of new services is not possible. Platforms such as GERBIL [161] and GERBIL for QA[26] offer benchmarks for EL tools and full QA systems and they generate persistent URIs for experiment results. This enables third-party evaluations and citable URIs. Their main goal is not to gain new insights into the underlying processes but only to generate one final metric that is publishable. For example, they do not generate a summary indicating in which cases the corresponding tool succeeded or failed. In contrast, the Qanary reference implementation is a full-featured framework for component-oriented QA process creation, which is *additionally* enabled to

[25] `http://www.okbqa.org/`
[26] `https://github.com/TortugaAttack/gerbil-qa`

support benchmarking of the included distributed components. We give a comparison of the described tools in Table 5.1.

5.3.4 The qa Vocabulary and the Qanary Methodology

To advance the QA process, researchers are combining different technologies to optimize their approaches. However, reusability and extensibility of QA components and systems remains a major hurdle. There are many components and services, which are provided as standalone implementations but can be useful in QA processes (*e.g.* the previously mentioned DBpedia Spotlight, AGDISTIS etc.), but there has so far not been a methodology to integrate them within QA pipelines. Instead substantial programming efforts had to be invested as each component provides its own API or integration method.

To address this challenge, and to promote reusability and extensibility of QA components, we introduced the qa vocabulary [133]. This vocabulary can represent information that is generated during the execution of a QA pipeline when processing a question given as speech or text input. Consider, for example, the question "When was Barack Obama born?". Typical information generated by components of a QA pipeline are the positions of named entities (NE) (such as "Barack Obama"), the ontological relations used to express the relational phrase in the question (that "born" refers to dbo:birthPlace[27]), the expected answer type (here: a date), the generated SPARQL query, the language of the question and possible ontologies that can be used to answer it.

The rationale of qais that all these pieces of information can be expressed as annotations to the question. Hence, these exposed pieces of information can be provided as an (RDF) knowledge base containing the full descriptive knowledge about the currently given question.

qais built on top of the *Web Annotation Data Model* (WADM)[28], a vocabulary to express annotations. The basic constructs of the WADM are annotations with at least a target indicating what is described and a body indicating the description.

```
PREFIX oa: <http://www.w3.org/ns/oa#>
<anno> a oa:Annotation ;
      oa:hasTarget <target> ;
      oa:hasBody <body> .
```

In qa, a question is assumed to be exposed at some URI (e.g. URIQuestion that can be internal and does not need to be public) and is of type qa:Question. Similarly other QA concepts (qa:Answer, qa:Dataset, etc.) are defined in the vocabulary; please see [133] for further details. As a result, when using qa, the knowledge of the question answering

[27] PREFIX dbo: <http://dbpedia.org/ontology/>
[28] W3C Candidate Recommendation 2016-09-06, http://www.w3.org/TR/annotation-model

systemis now representable independently from a particular programming language or implementation paradigm because everything is represented as direct or indirect annotations of a resource of type `qa:Question`. The `qa`vocabulary is published at the persistent URI `https://w3id.org/wdaqua/qanary#` under the CC0 1.0 license[29].

The `qa`vocabulary led to the Qanary methodology [22] for implementing processes operating on top of the knowledge about the question currently processed within a question answering system, leading to the possibility of easy-to-reuse QA components. All the knowledge related to questions, answers and intermediate results is stored in a central Knowledge Base (KB). The knowledge is represented in terms of the `qa`vocabulary in the form of annotations of the relevant parts of the question.

Within Qanary the components all implement the same service interface. Therefore, all components can be integrated into a QA system without manual engineering effort. Via it's service interface a component receives information about the KB (i.e., the endpoint) storing the knowledge about the currently processed question of the user. Hence, the common process within all components is organized as follows:

1. A component fetches the required knowledge via (SPARQL) queries from the KB. In this way, it gains access to all the data required for its particular process.

2. The custom component process is started, computing new insights of the user's question.

3. Finally, the component pushes the results back to the KB (using SPARQL).

Therefore, after each process step (i.e., component interaction), the KB should be enriched with new knowledge (i.e., new annotations of the currently processed user's question). This way the KB keeps track of all the information generated in the QA process even if the QA process is not predefined or not even known. A typical QA pipeline consists of several steps such as NER, NED, relation identification, semantic analysis, query computation and result ranking. Most recently we provided a reference implementation of Qanary [133]. We call this implementation *message-driven*; it follows the architectural pattern that we have previously described for search engines in [23]. The processing steps might be implemented in different components with dedicated technology provided by distinguished research groups. The message-driven implementation of Qanary [23] laid foundations for the QA ecosystem. The advantage of such an ecosystem is that it combines different approaches, functionality, and advances in the QA community under a single umbrella.

[29] `https://creativecommons.org/publicdomain/zero/1.0/`

5.3.5 The Qanary Ecosystem

The Qanary ecosystem consists of a variety of components and services that can be used during a QA process. We describe in the following what components and services are available.
The Qanary ecosystem includes various components covering a broad field tasks within QA systems. This includes different components performing NER like FOX [139] and Stanford NER [70] and components computing NED such as DBpedia Spotlight and AGDISTIS [160]. Also industrial services such as the Alchemy API are part of the ecosystem.

Furthermore, Qanary includes a language detection module [117] to identify the language of a textual question. A baseline automatic speech recognition component is also included in the reference implementation. It translates audio input into natural language texts and is based on Kaldi[30]. Additionally it should be noted that a monolithic QA system component [52] was developed in the course of the WDAqua project[31] and is integrated in Qanary. Additional external QA components are included in the ecosystem. In particular, Qanary includes two components from the OKBQA challenge[32] namely the template generation and disambiguation component. All components are implemented following the REST principles. Hence, these tools/approaches become easy to reuse and can now be invoked via transparent interfaces. To make it easy to integrate a new component we have created a Maven archetype that generates a template for a new Qanary component[33]. The main services are encapsulated in the *Qanary Pipeline*. It provides, for example, a service registry. After being started, each component registers itself to the *Qanary Pipeline* central component following the local configuration[34] of the component. Moreover, the *Qanary Pipeline* provides several web interfaces for machine and also human interaction (*e.g.* for assigning a URI to a textual question, retrieving information about a previous QA process, etc.). Particularly, as each component automatically registers itself to the *Qanary Pipeline*, a new question answering systemcan be created and executed just by on-demand configuration (a concrete one is shown in Fig. 5.5). Hence, the reference implementation already provides the features required for QA systems using components distributed over the Web.

An additional interface allows for benchmarking a QA system created on demand using Gerbil for QA[35], thus allowing third-party evaluation and citeable URIs. Fig. 5.6 illustrates the complete reference architecture of Qanary and a few of its components. Additional services include a user interface called Trill [46] for a fully working QA system. A running

[30] `http://kaldi-asr.org`
[31] `http://www.wdaqua.eu`
[32] `http://www.okbqa.org/`
[33] `http://github.com/WDAqua/Qanary/wiki/How-do-I-integrate-a-new-component-in-Qanary%3F`github.com/WDAqua/Qanary/wiki/How-do-I-integrate-a-new-component-in-Qanary%3F
[34] The configuration property `spring.boot.admin.url` defines the endpoint of the central component (and can be injected dynamically).
[35] `http://gerbil-qa.aksw.org`

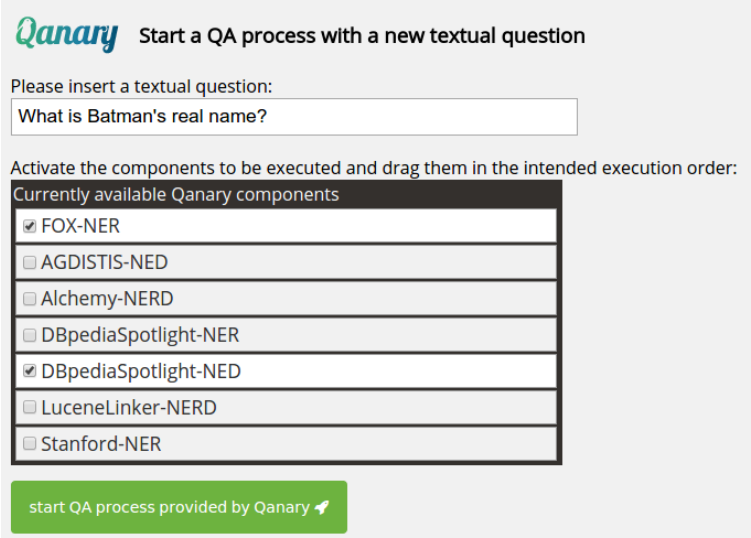

Figure 5.5: Snapshot of the Web interface for defining a textual question and a sequence of components to process it (here only NED/NER components where registered).

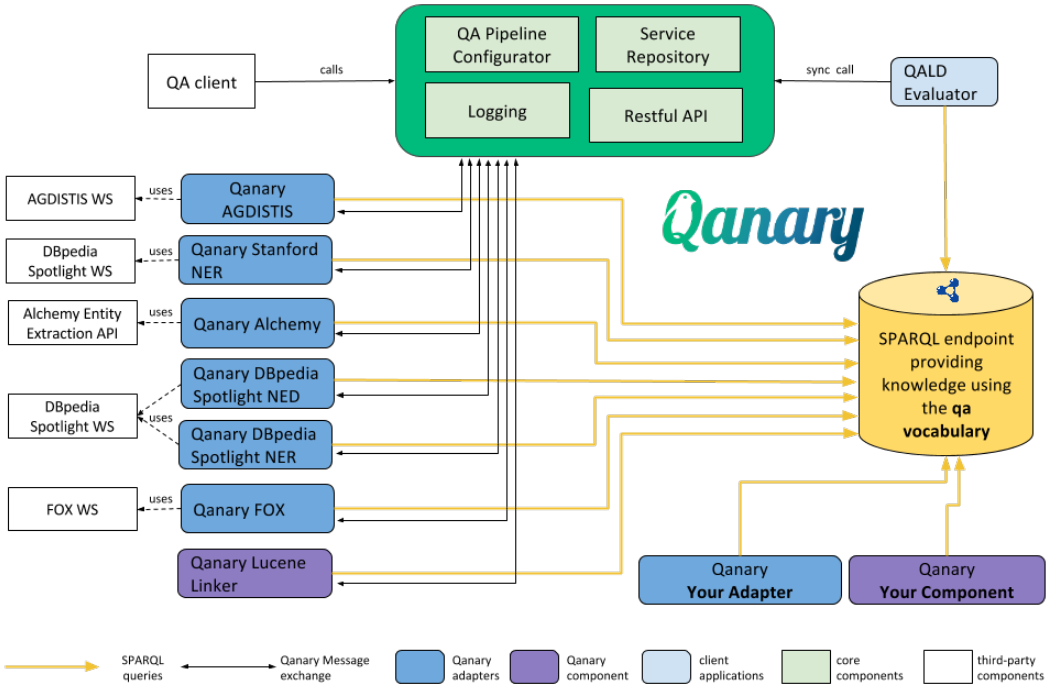

Figure 5.6: The Qanary reference architecture implementation highlighting the NER/NED components.

demo can be found at http://www.wdaqua.eu/qa. The code is maintained in the repository at https://github.com/WDAqua/Qanary under the MIT License[36].

5.3.6 Gaining new insights into the QA process: The EL task

To show how Qanary can be used to gain new insights into QA processes we focus here on the EL task. We present the qa vocabulary used to represent the information produced by NER and NED tools. Moreover, we describe the components of the Qanary ecosystem that are integrated using the Qanary methodology and that can be used for the EL task. We describe how we constructed a benchmark for EL out of QALD. The analysis of the benchmark will show: what are the best tools to tackle QALD, where are current research gaps, and for which questions do single tools fail and why.

Finally, we present a new dataset that can be used as a gold standard for a sub-task of the QA process.

The following workflow is not restricted to the EL task but can be applied to any other sub-task of the QA process to gain new insights into QA processes.

The Qanary vocabulary for the EL task

The qavocabulary is designed to be extensible so as not to constrain the creativity of the QA community developers. All information that can possibly be generated and that might need to be shared across QA components can be expressed using new annotations. This principle follows the understanding that standards that allow communication between QA components must be defined by the community. Taking into consideration the state-of-the-art (*e.g.* [160, 70]), the qavocabulary was extended with standard concepts for NER and NED representations. This in particular uniforms the representation of the input and output of every integrated component, making it easy to compare and analyze the integrated tools. Note that this does not only hold for tools that can be used for EL but for every tool integrated into the Qanary ecosystem.

To describe an entity spotted within a question we introduced a dedicated annotation:

```
qa:AnnotationOfSpotInstance a owl:Class;
   rdfs:subClassOf qa:AnnotationOfQuestion .
```

If in the question “When was Barack Obama born?” a spotter detects “Barack Obama” as an NE, this fact can be expressed by the following annotation, where oa:SpecificResource and oa:hasSelector are concepts of the WADM to select a part of a text.

[36] https://opensource.org/licenses/MIT

```
<anno1> a qa:AnnotationOfSpotInstance .
<anno1> oa:hasTarget [
   a oa:SpecificResource ;
      oa:hasSource <URIQuestion>;
      oa:hasSelector [
        a oa:TextPositionSelector;
        oa:start "9"^^xsd:nonNegativeInteger;
        oa:end "21"^^xsd:nonNegativeInteger
      ]
] .
```

For named entities, we define the new concept `qa:NamedEntity` and a corresponding annotation subclass (i.e., annotations of questions whose body is an instance of `qa:NamedEntity`):

```
qa:NamedEntity a owl:Class ;
qa:AnnotationOfInstance a owl:Class ;
   owl:equivalentClass [
      a owl:Restriction ;
      owl:onProperty oa:hasBody ;
      owl:someValuesFrom qa:NamedEntity
      ] ;
   rdfs:subClassOf qa:AnnotationOfQuestion .
```

If an NED tool detects in the question "When was Barack Obama born?" that the text "Barack Obama" refers to `dbr:Barack_Obama`[37], then this can be expressed (using `oa:hasBody`) as:

```
<anno1> a qa:AnnotationOfInstance ;
 oa:hasTarget [
   a oa:SpecificResource ;
   oa:hasSource <URIQuestion> ;
   oa:hasSelector [
      a oa:TextPositionSelector ;
      oa:start "9"^^xsd:nonNegativeInteger;
      oa:end "21"^^xsd:nonNegativeInteger
   ]
 ] ;
 oa:hasBody dbr:Barack_Obama .
```

Note that using annotations provides many benefits thanks to the inclusion of additional metadata such as the creator of the information, the time and a trust score. However, this information is omitted here for readability.

[37] `PREFIX dbr: <http://dbpedia.org/resource/>`

Reusable NER and NED components

The following components integrated into the Qanary ecosystem solve the task of NER, NED or the combined task of EL.

- **Stanford NER (NER)** is a standard NLP tool that can be used to spot entities for any ontology, but only for languages where a model is available (currently English, German, Spanish and Chinese) [70].
- **FOX (NER)** integrates four different NER tools (including the Stanford NER tool) using ensemble learning [139].
- **DBpedia Spotlight spotter (NER)** uses lexicalizations, i.e., ways to express named entities, that are available directly in DBpedia [112].
- **DBpedia Spotlight disambiguator (NED),** the NED part of DBpedia Spotlight, disambiguates entities by using statistics extracted from Wikipedia texts [112].
- **AGDISTIS (NED)** is a NED tool that uses the graph structure of an ontology to disambiguate entities [160].
- **ALCHEMY (NER + NED):** Alchemy API[38] is a commercial service (owned by IBM) exposing several text analysis tools as web services.
- **Lucene Linker (NER + NED)** is a component that we implemented following the idea of the SINA QA system [132], which employs information retrieval methods.

Note that thanks to the integration as Qanary components all these tools can be interwoven, i.e., each NER tool can be combined with each NED tool just by configuration.

A QALD-based benchmark for EL in QA

To compare the different entity linking approaches, we created a benchmark based on the QALD (Question Answering over Linked Data) benchmark used for evaluating complete QA systems. The QALD-6 training set[39], which is the recent successor of QALD-5 [155], contains 350 questions, including the questions from previous QALD challenges. For each question, it contains a SPARQL query that retrieves the corresponding answers. For example, the following SPARQL query corresponds to the question "Which soccer players were born on Malta?".

[38] `http://alchemyapi.com`
[39] Training Questions of Task 1: `http://qald.sebastianwalter.org/index.php?x=challenge&q=6`

$$\text{Precision}(q) = \frac{\text{\# correct URIs retrieved by the EL configuration for } q}{\text{\# URIs retrieved by the EL configuration identifying NE in } q}$$

$$\text{Recall}(q) = \frac{\text{\# correct URIs retrieved by the EL configuration for } q}{\text{\# gold standard answers for } q}$$

$$F_1\text{-measure}(q) = 2 \times \frac{\text{Precision}(q) \times \text{Recall}(q)}{\text{Precision}(q) + \text{Recall}(q)}$$

Figure 5.7: Metrics used in the EL benchmark

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?uri WHERE {
   ?uri a dbo:SoccerPlayer .
   ?uri dbo:birthPlace dbr:Malta .
}
```

EL tools should provide functionality to interlink the named entities present in the question with DBpedia (or other data), i.e., they should be able to identify "Malta" and link to it the resource `http://dbpedia.org/resource/Malta`. Our benchmark compares the URIs generated by an EL tool with the resource URIs in the SPARQL query (i.e., those in the `http://dbpedia.org/resource/` namespace)[40], which are obviously required for answering the question. Hence the gold standard for each question is given by all resource URIs in the SPARQL query.[41]

The metrics for a question q are calculated as defined in the QALD benchmark and are reported in Fig. 5.7[42]. The metrics over all questions are defined as the average of the metrics over the single questions. The corresponding benchmark component is available at `https://github.com/WDAqua/Qanary`.

Note that this procedure can be generalized and applied to many sub-processes of a QA pipeline. For example, one might establish a benchmark to recognize relations or classes, a benchmark to identify the type of the SPARQL query required to implement a question (i.e., a *SELECT* or an *ASK* query), a benchmark for identifying the answer type (i.e., list, single resource, ...) and so on.

[40] Note that resources of the type `http://dbpedia.org/ontology/*` would match a DBpedia property or class and therefore are not considered here.

[41] This definition of the gold standard ignores the *order* of the URIs. In practice this definition rarely causes problems, but in theory one could construct counter-examples that pinpoint the limitations. Imagine the question "What German actors were not born in Germany?", and imagine that the word "German" got linked to the entity `dbr:Germany` and "Germany" to `dbr:Germans` – clearly a wrong linking, but "correct" w.r.t. our gold standard definitions. However, in QALD (release 6) there are no questions in which such a mix-up is likely to happen.

[42] In the corner cases where the number of system answers or the number of gold standard answers is zero we follow the same rules that are used in the QALD evaluation; see `https://github.com/ag-sc/QALD/blob/master/6/scripts/evaluation.rb`.

Pipeline configuration	Fully detected	Correctly Annotated	Precision	Recall	F_1-measure
StanfordNER + AGDISTIS	200	195	0.76	0.59	0.59
StanfordNER + Spotlight disamb.	209	189	0.77	0.62	0.61
FOX + AGDISTIS	189	186	0.83	0.56	0.56
FOX + Spotlight disambiguator	199	192	0.86	0.59	0.58
Spotlight Spotter + AGDISTIS	209	204	0.75	0.62	0.62
Spotlight spotter + disambiguator	242	**213**	0.76	0.71	**0.68**
Lucene Linker	**272**	0	0.01	**0.78**	0.03
Alchemy	143	139	**0.91**	0.42	0.42

Table 5.2: Benchmark of the QALD-6 data using the Qanary reference implementation.

We used our benchmarking resource described above to evaluate EL tools. We have identified different strategies to annotate entities in questions. These include using the spotters Stanford NER, FOX, DBpedia Spotlight Spotter, the NED tools AGDISTIS, and the DBpedia Spotlight disambiguator, as well as the (monolithic w.r.t. NER and NED) EL tools Alchemy and Lucene Linker. Each of them is implemented as an independent Qanary component, as presented in Sec. 5.3.6. According to the Qanary methodology the computed knowledge about a given question is represented in terms of the `qa`vocabulary and can be interpreted by the benchmark component.

For the benchmark all three NER components are combined with each of the two NED components. All questions of QALD-6 are processed by each of the six resulting configurations, and by the two monolithic tools. The benchmark was executed exclusively using the service interface of the *Qanary Pipeline*.

Table 5.2 shows the benchmark results. The "Fully detected" column indicates the number of questions q where some resources were expected and the EL configuration achieved Recall(q)=1. The column "Correctly Annotated" indicates for how many questions we obtained Precision(q)=Recall(q)=1. Finally, the table shows for each configuration the precision and recall metrics over all questions.

Discussion

We presented the Qanary methodology, which allows to interweave the analysed tools. Thanks to the `qa`vocabulary we can collect (from the SPARQL endpoint) the results produced by every configuration. A detailed overview showing for each question if the pipeline configurations lead to a recall resp. F-measure of 1 can be found at:

`https://raw.githubusercontent.com/WDAqua/Qanary/master/ICWE-results/`

- `Recall_1.csv` and
- `F-measure_1.csv`.

We analysed both this data and the results presented in Table 5.2 to draw some conclusions on the performance of the used tools with respect to QALD.

For some QALD-6 questions none of the pipeline configurations is able to find the required resources, for example:

- *Q1:* "Give me all cosmonauts." with the following resources requested in the SPARQL query: `dbr:Russia`, `dbr:Soviet_Union`. For this question one should be able to understand that cosmonauts are astronauts born either in Russia or in the Soviet Union. Detecting such resources would require a deep understanding of the question. Q201 is similar: "Give me all taikonauts.".

- *Q13:* "Are tree frogs a type of amphibian?"; requested resources: `dbr:Hylidae`, `dbr:Amphibian`.
  The problem here is that the scientific name of "tree frogs" is Hylidae and there is no such information in the ontology except in the free text of the Wikipedia abstract.

- *Q311:* "Who killed John Lennon?"; requested resource: `dbr:Death_of_John_Lennon`.
  The problem is that one would probably assume that the information is encoded in the ontology as a triple like "John Lennon", "killed by", "Mark David Chapman" but this is not the case. Even if in the question the actual NE is "John Lennon", DBpedia happens to encode the requested information in the resource "Death of John Lennon". A similar case is Q316 ("Which types of grapes grow in Oregon?"), where the resource `dbr:Oregon_wine` is searched.

Spotter comparison An unexpected result is that FOX as a spotter has a lower recall than the Stanford NER tool, even though FOX also includes the results of Stanford NER. This can be seen from comparing the recall of the configurations that combine these tools with AGDISTIS or the DBpedia Spotlight disambiguator. The reason is that, for example, in Q101 ("Which German cities have more than 250,000 inhabitants?") the word "German" is tagged by the Stanford NER tool as "MISC" (miscellaneous). However, FOX only supports the tags "PERS" (person), "ORG" (organisation), and "LOC" (location). This explains why FOX has a lower recall but a higher precision than Stanford NER.

The spotters based on NLP (*e.g.* Stanford NER and FOX) perform worse than the DBpedia Spotlight Spotter, which is mainly based on vocabulary matching. Syntactic features do not

suffice to identify “Prodigy” in Q114 (“Give me all members of Prodigy.”) or “proinsulin” in Q12 (“Is proinsulin a protein?”). Moreover, there are cases like Q109 (“Give me a list of all bandleaders that play trumpet.”), where bandleaders is not an NE in the NLP sense but is modeled as a resource in the DBpedia *ontology*. Similarly, in Q133 (“Give me all Australian nonprofit organizations.”), the resource `dbr:Australia` is expected for the adjective “Australian”.

NED comparison The results show that the DBpedia Spotlight disambiguator performs better than AGDISTIS w.r.t. QA. AGDISTIS works on co-occurrences of NE. These occur often in longer texts but are rare in questions. If only one NE is spotted, AGDISTIS can only decide based on the popularity of the resources but not on the context as DBpedia Spotlight does.

EL comparison The best spotter, the DBpedia Spotlight spotter, and the best NED, the DBpedia Spotlight disambiguator, also perform best in the EL task. Only the Lucene Linker has a higher recall but must be followed by a disambiguation tool in the next step to increase precision. The Alchemy API shows the lowest recall.

Our evaluation does not permit the conclusion that the combination of DBpedia Spotlight spotter and disambiguator should be recommended in general. The best choice may depend on the questions and on the particular form of dataset. The DBpedia Spotlight disambiguator, for example, is tightly connected to Wikipedia; even its algorithm cannot be ported to other ontologies. Alchemy, despite showing a very low F_1-score and recall, could be a useful resource for QA over other datasets, such as Freebase or Yago. This is one of the many reasons that makes Qanary in general a valuable resource. For a new QA scenario, Qanary empowers developers to quickly combine existing tools and more easily determine the best configuration. Moreover, a detailed analysis of the configurations can help to detect the main problems of the different strategies to further improve the complete QA process or just individual components. Hence, using Qanary provides insights on the quality of each component w.r.t. the current use case, leading to an optimization of the system based on the given data.

A combination of tools solving the same task in an ensemble learning approach is now possible and is recommended as the benchmark results already indicate. Note that such an analysis is not possible using existing benchmarking tools such as Gerbil or Gerbil for QA since they only provide a final overall score. On the other hand such an analysis is needed to detect existing research gaps and push the advancement of QA further. Hence, following the Qanary methodology the research community is enabled to develop new QA processes and components in a joint engineering effort and to validate the given quality metrics within the specific QA scenario. This again proves the potential impact of Qanary within the engineering process of QA systems.

```
PREFIX qa: <http://www.wdaqua.eu/qa#>
PREFIX oa: <http://www.w3.org/ns/openannotation/core/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX git: <https://github.com/dbpedia-spotlight/>
<anno1> a qa:AnnotationOfInstance ;
      oa:annotatedAt "2016-04-30T15:00:43.687+02:00"^^xsd:dateTime ;
      oa:hasTarget [
            a oa:SpecificResource ;
            oa:Selector [
                     a oa:TextPositionSelector ;
                     oa:end 4 ;
                     oa:start 21
                   ] ;
            oa:hasSource <URIQuestion>
            ] .
      oa:hasBody dbr:Margaret_Thatcher ;
      oa:annotatedBy git:dbpedia-spotlight .
```

Figure 5.8: Example data of Q178: "Was Margaret Thatcher a chemist?"

Dataset of Annotated Questions for Processing in question answering systems

We provide a new dataset with questions of the QALD-6 benchmark, which are completely annotated with disambiguated named entities (DBpedia resource URIs) computed by applying our benchmarking to the EL configurations described in Sec. 5.3.6. This dataset contains 267 questions (out of 350 questions in the QALD-6 training set) because the components could not annotate the rest. A Turtle file, representing the results in terms of the `qa`vocabulary, is published at `https://github.com/WDAqua/Qanary` under the CC0 license. A typical fragment of the provided data is provided in Fig. 5.8.

We imagine the following usage scenarios for this dataset. It can be used as input for steps in a QA process *following* the EL task that require annotated named entities, such as relation detection or query computation. Consequently, QA components that depend on the output of the EL task can now be evaluated without depending on concrete EL components (and without the results being influenced by possible flaws). Hence, in conjunction with the SPARQL queries already defined in QALD-6, we established a new gold standard for evaluating parts of a QA process. We also provide the component for computing this dataset (cf., Sec. 5.3.6); it can be extended if improved EL configurations are available or when a new version of the QALD benchmark is released.

5.3.7 Conclusion and Future Work

We have presented the status of the Qanary ecosystem, which includes a variety of components and services that can be used by the research community. These include typical components for sub-tasks of a QA pipeline as well as a number of related services.

Since all messages exchanged between components are expressed using the `qa` vocabulary, all information generated during a QA process can be easily kept. Thanks to this uniform message format it is now possible to easily compare existing tools and integrate new ones. Moreover, the Qanary methodology allows to integrate independent, distributed components, implemented in different programming languages in a loosely-coupled fashion. This allows the creation of comprehensive QA systems in a community effort.

Driven by the demand for better QA technology, we propose a general workflow to develop future QA systems. It mainly breaks down into two parts: (1.) the identification and integration of existing state-of-the-art approaches to solve a particular sub-task in the QA pipeline, and (2.) the derivation of a benchmark from benchmarks for QA such as QALD. Additionally a new gold standard for the sub-task can be provided. In contrast to other approaches the `qa` vocabulary allows to analyse a QA process. Hence, full traceability of the information used in the QA process is ensured, enabling, for example, the optimization of the assigned components. Additionally, the Qanary methodology allows to create such processes in a flexible way. This allows researchers to focus on particular tasks taking advantage of the results of the research community and contributing to it directly in a reusable way.

We have demonstrated this workflow in the case of EL. This way we realized a set of reusable components as well as the first benchmark for EL in the context of QA. All together we have shown how Qanary can be used to gain deep insights in QA processes. While having such insights the engineering process can be steered efficiently towards the improvement of the QA components. Hence, the presented engineering approach is particularly well suited for experimental and innovation-driven approaches (*e.g.* used by research communities).

The Qanary ecosystem is maintained and used within the WDAqua ITN project[43] (2015–2018 [106]), where Qanary is the reference architecture for new components. All artifacts are published under permissive open licenses: MIT for the software, CC0 for the datasets and the vocabulary.

One of our future task is to populate the Qanary ecosystem with any component significant to the QA community. According to the literature, the tasks of relation and answer type detection are of particular relevance, but not yet sufficiently covered by existing components. Additionally, as Qanary provides easy access to different implementations having the same purpose, ensemble learning components for all steps within a QA process are becoming possible and will increase the flexibility as well as boost overall QA quality. Hence, our overall goal is to provide a fully-featured ecosystem for creating QA components and concurrently supporting the measuring and improvement of particular QA systems w.r.t. the considered use cases. This aim provides several research challenges, *e.g.* the (semi-)automatic creation of self-optimizing QA systems.

[43] `http://wdaqua.eu`

6 Acknowledgements

Three years is a long time and an occasion to meet many people. Many of them helped me and influenced me, without them this PhD would not have been possible or would have been very different.

sFor sure this thesis would not have been possible without my boyfriend Christian Moritz. There are many reasons for that. First, I would most probably never have started a PhD in this domain. Life is so unpredictable. I would have never thought 5 years ago that I would have started a PhD in Computer Science. You encouraged me to try new things and explore directions that I would not have taken. What would I have missed! Second, you always were there to hear all the ups and downs that arrive to a young researcher, the excitements, the disappointments and the pressure. Third, the fact to be there, the breakfasts, the lunches, dinners, merendas, hikes, travels, game evenings, films that we passed together. Thank you!

I want to thank my supervisors, Pierre Maret and Kamal Singh. First because you gave me the opportunity to start this challenge. You not only followed me as a researcher, but you also took care that I would feel comfortable in a new city and in a new country. As a researcher I appreciated most the freedom that you gave me. No strict directions, no time pressure and no restrictions. I could not have explored the directions I took without that. Moreover I appreciated a lot the direct contact with you, especially when I needed it. Thank you!

This PhD would not have been so a great experience without my friends. In particular without Jose Gimenez Garcia, Maísa Duarte, Oudom Kem, Radha Krishna Ayyasomayajula, Omar Qawesmeh, Adrian Villalba Weinberg, Alexandra Herczku, Carlos Arango and Justine Rogues. It was a great time I passed with you. The dinners, the board game evenings, the hikes, coffee breaks, the holidays and especially these carefree dinners full of discussions, jokes and laugh. Thank you for that!

Also I would like to thank my family that gave me this opportunity and that is there when I need them. In particular my mom, Elena Panzia Oglietti, my dad, Willy Diefenbach, my grandmother Rosanna Cerruti, my aunt Simona Panzia Oglietti, my uncle Mauro Aimino and my two cousins Gabriele and Riccardo Aimino.

Even if it is difficult to meet frequently I want to thank my Italian friends Marta Garrone, Silvia Causone, Claudio Nicolotti and Martina Viletto. Thank you for all your e-mails and the time passed when I was there that give me new energy for my work. Thank you!

There are many other people that I have met during this time that I have really enjoyed. That they didn't become closer friends was more a matter of circumstances and time. Especially I remember Maria Galvan, Jose Carlos Rangel Ortiz, Pierre-René Lhérisson, Yuta Suzuki, Niklas Petersen, Chung Lu, Maria Mislene, Priyanka Rawat, Laura Kosten, Emilia Kacprzak, Brisbane Ovilla, Richard Venditti and Raúl García Castro.

Many thanks to the Connected Intelligence Group. In particular to Antoine Zimmermann, Maxime Lefrancois, Olivier Boissier, Fabrice Muhlenbach, Frederique Laforest, Julien Subercaze, Christophe Gravier, Syed Gillani, Nicolas Cointe, Kadim Ndiaye, Tanguy Raynaud and Jules Chevalier.

I want to thank my internship students: Shanzay Amjad, Youseff Dirdi and Pierre Tardiveau. Your hunger to learn was an important motivation for me. In particular it was a pleasure to work with Shanzay Amjad both for her technical skills and the good mood that she spread around. Also through Youseff Dirdi and Pierre Tardiveau I learned a lot.

I want to mentions my office mates for the discussions we had, for the numerous interruptions you had to take, the questions you answered me, the debugging sessions you "lived with me", the breaks in the secret room, the discussions about cantine, faboulose cantine, kebab and boulangerie, the chess games and the fact that you took care about Ernestina.

I want to thank the other colleagues in the lab: Rémi Emonet, Elisa Formont, Damien Fourure, Valentina Zantedeschi, Mohamed Elawady, Bissan Audeh Issa, Jordan Fréry, and Baptiste Jeudy.

I want to thank the system administrators Thomas Gautrais, Christophe Vereecke and Fabrice Allibe. I don't know how many times they helped me out! Also thank you for your open mindset to explore new directions.

Particular thanks go to Andreas Both for having been my third unofficial supervisor. I learned a lot from you from a research, developer and leading perspective. How such a busy guy is still able to find time for my projects is and remains a mystery.

Andreas Thalhammer for the fastest collaboration ever. 10 minutes of discussion in a conference sufficed to integrate his great service LinkSum. And other fruitfull collaboration followed. I really enjoyed working with you.

Claus Stadler for introducing me LinkedGeoData and for helping me in the extraction process of OpenStreetMap. You did a great work, I really appreciate it and I hope you can defend soon!

Vanessa Lopez for her e-mails full of smiles and good mood. And most importantly for accepting to join our survey paper submission. You were of great help.

I want to thank the administrative stuff of the lab. In particular Julie Debiesse for being there when needed and always pleasant even if full of work, and Véronique Reveillé for examining my numerous missions.

I also want to remember the members of the WDAqua project.

Finally, as in every paper, I want to acknowledge 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). This time I want to thank the tax payers in Europe that financed my position and that made this possible. I hope I can give something back!

Bibliography

[1] A. B. Abacha and P. Zweigenbaum. „Medical question answering: translating medical questions into SPARQL queries“. In: *ACM IHI*. 2012 (cit. on pp. 122, 123, 139).

[2] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. „Automated Template Generation for Question Answering over Knowledge Graphs“. In: *Proceedings of the 26th International Conference on World Wide Web*. 2017, pp. 1191–1200 (cit. on p. 55).

[3] Nitish Aggarwal and Paul Buitelaar. „A system description of natural language query over dbpedia“. In: *Proc. of Interacting with Linked Data (ILD 2012)[37]* (2012) (cit. on pp. 28, 35–37, 41, 42, 47, 51, 53).

[4] Ali Mohamed Allam and Mohamed Hassan Haggag. „The Question Answering Systems: A Survey“. In: *Int Journal of Research and Reviews in Information Sciences (IJRRIS)* 2.3 (2012), (cit. on p. 23).

[5] S. Atdag and V. Labatut. „A comparison of named entity recognition tools applied to biographical texts“. In: *2nd International Conference on Systems and Computer Science (ICSCS)*. 2013 (cit. on p. 156).

[6] S. J. Athenikos and H. Han. „Biomedical question answering: A survey“. In: *Computer Methods and Programs in Biomedicine* 99.1 (2010), pp. 1–24 (cit. on p. 139).

[7] Maurizio Atzori, Giuseppe Mazzeo, and Carlo Zaniolo. „QA3@QALD-6: Statistical Question Answering over RDF Cubes“. In: *ESWC. to appear*. 2016 (cit. on p. 27).

[8] S. Auer, Ch. Bizer, G. Kobilarov, et al. „DBpedia: A Nucleus for a Web of Open Data“. In: *ISWC + ASWC*. 2007 (cit. on pp. 123, 139).

[9] Sören Auer, Christian Bizer, Georgi Kobilarov, et al. „DBpedia: A Nucleus for a Web of Open Data“. English. In: *The Semantic Web*. Vol. 4825. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 722–735 (cit. on p. 95).

[10] Junwei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao. „Knowledge-based question answering as machine translation“. In: *Cell* 2.6 (2014) (cit. on p. 55).

[11] Hannah Bast and Elmar Haussmann. „More accurate question answering on freebase“. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM. 2015 (cit. on pp. 55, 56).

[12] Petr Baudiš and Jan Šedivy. „QALD Challenge and the YodaQA System: Prototype Notes“. In: (2015) (cit. on pp. 27, 28, 80).

[13] Romain Beaumont, Brigitte Grau, and Anne-Laure Ligozat. „SemGraphQA@QALD-5: LIMSI participation at QALD-5@CLEF“. In: *Working Notes for CLEF 2015 Conference*. CLEF. 2015 (cit. on pp. 28, 35, 41, 47, 53, 64, 80, 81).

[14] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. „LOD laundromat: a uniform way of publishing other people's dirty data“. In: *International Semantic Web Conference*. Springer. 2014, pp. 213–228 (cit. on pp. 97, 105).

[15] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. „Semantic Parsing on Freebase from Question-Answer Pairs.“ In: *EMNLP*. 2013 (cit. on pp. 24, 29, 38, 54–56, 65, 73, 107).

[16] Jonathan Berant and Percy Liang. „Imitation learning of agenda-based semantic parsers“. In: *Transactions of the Association for Computational Linguistics* (2015) (cit. on p. 55).

[17] Jonathan Berant and Percy Liang. „Semantic Parsing via Paraphrasing.“ In: *ACL (1)*. 2014 (cit. on p. 55).

[18] K. D. Bollacker, R. P. Cook, and P. Tufts. „Freebase: A Shared Database of Structured General Human Knowledge“. In: *Proc. 21st AAAI Conference on Artificial Intelligence*. 2007, pp. 1962–1963 (cit. on p. 123).

[19] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. „Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge“. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 1247–1250 (cit. on p. 95).

[20] Antoine Bordes, Sumit Chopra, and Jason Weston. „Question answering with subgraph embeddings“. In: *arXiv preprint arXiv:1406.3676* (2014) (cit. on pp. 54–56).

[21] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. „Large-scale simple question answering with memory networks“. In: *arXiv preprint arXiv:1506.02075* (2015) (cit. on pp. 24, 25, 55–57, 64, 65, 73, 82, 107).

[22] A. Both, D. Diefenbach, K. Singh, et al. „Qanary–a methodology for vocabulary-driven open question answering systems“. In: *International Semantic Web Conference*. Springer. 2016 (cit. on pp. 16, 18, 61, 65, 83, 88, 116, 117, 137, 154, 158, 199).

[23] A. Both, A.-C. N. Ngomo, R. Usbeck, et al. „A Service-oriented Search Framework for Full Text, Geospatial and Semantic Search“. In: *SEMANTiCS*. Leipzig, Germany, 2014 (cit. on pp. 143, 158).

[24] A. Both, V. Nguyen, M. Keck, et al. „Get Inspired: A Visual Divide and Conquer Approach for Motive-based Search Scenarios“. In: *13th Int. Conf. WWW/INTERNET (ICWI 2014)*. Porto, Portugal: Int. Association for Development of the Information Society (IADIS), 2014 (cit. on p. 129).

[25] A. Both, K. Singh, D. Diefenbach, and I. Lytra. „Rapid Engineering of QA Systems Using the Light-Weight Qanary Architecture“. In: *Web Engineering: 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings*. Ed. by Jordi Cabot, Roberto De Virgilio, and Riccardo Torlone. Cham: Springer International Publishing, 2017, pp. 544–548 (cit. on p. 199).

[26] Sergey Brin and Lawrence Page. „The Anatomy of a Large-scale Hypertextual Web Search Engine“. In: *Proceedings of the Seventh International Conference on World Wide Web 7*. WWW7. Brisbane, Australia: Elsevier Science Publishers B. V., 1998, pp. 107–117 (cit. on pp. 94, 96, 98).

[27] Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, et al. „QAKiS: an open domain QA system based on relational patterns“. In: *Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914*. CEUR-WS. org (cit. on pp. 28, 35, 39, 41, 47, 48, 53, 88, 123, 129, 137, 139, 154).

[28] Philipp Cimiano, Vanessa Lopez, Christina Unger, et al. „Multilingual question answering over linked data (qald-3): Lab overview“. In: *Information Access Evaluation. Multilinguality, Multimodality, and Visualization*. Springer, 2013 (cit. on pp. 23, 24, 27, 28, 58, 60, 80).

[29] Philipp Cimiano and Michael Minock. „Natural Language Interfaces: What Is the Problem?-A Data-Driven Quantitative Analysis.“ In: *NLDB*. Springer. 2009, pp. 192–206 (cit. on pp. 58–60).

[30] Daoud Clarke. „Simple, Fast Semantic Parsing with a Tensor Kernel“. In: *arXiv preprint arXiv:1507.00639* (2015) (cit. on p. 55).

[31] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. „GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications“. In: *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*. 2002 (cit. on p. 31).

[32] Diefenbach D., Dridi Y., Singh K., and Maret P. „SPARQLtoUser: Did the Question Answering System Understand me?“ In: *Joint Proceedings of BLINK2017: 2nd International Workshop on Benchmarking Linked Data and NLIWoD3: Natural Language Interfaces for the Web of Data co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21st - to - 22nd, 2017*. 2017 (cit. on pp. 15, 18, 83, 107, 200).

[33] Zihang Dai, Lei Li, and Wei Xu. „Cfo: Conditional focused neural question answering with large-scale knowledge bases“. In: *arXiv preprint arXiv:1606.01994* (2016) (cit. on pp. 57, 82).

[34] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N Mendes. „Improving efficiency and accuracy in multilingual entity extraction“. In: *Proceedings of the 9th International Conference on Semantic Systems*. ACM. 2013 (cit. on pp. 30, 124, 126, 128, 140, 142, 146).

[35] D. Damljanovic, M. Agatonovic, and H. Cunningham. „FREyA: An interactive way of querying Linked Data using natural language“. In: *ESWC*. 2011 (cit. on pp. 115, 116, 137, 139).

[36] Danica Damljanovic. „Towards portable controlled natural languages for querying ontologies“. In: *Second Workshop on Controlled Natural Languages*. Citeseer. 2010 (cit. on p. 123).

[37] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. „FREyA: An interactive way of querying Linked Data using natural language“. In: *The Semantic Web: ESWC 2011 Workshops*. Springer. 2012 (cit. on p. 46).

[38] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. „Identification of the Question Focus: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction.“ In: *LREC*. 2010 (cit. on pp. 28, 32, 35, 41, 47, 53).

[39] Diefenbach Dennis. *SimpleQuestions Linked Data*. `https://figshare.com/projects/SimpleQuestionsLinkedData/28530`. 2018 (cit. on p. 201).

[40] Diefenbach Dennis. *WDAquaCore0Questions*. `https://github.com/WDAqua/WDAquaCore0Questions`. 2017 (cit. on p. 201).

[41] Diefenbach Dennis, Both Andreas, Lytra Ioanna, et al. *Qanary*. `https://github.com/WDAqua/Qanary`. 2016 (cit. on p. 200).

[42] Diefenbach Dennis and Thalhammer Andreas. *PageRank scores of some RDF graphs*. `https://figshare.com/projects/PageRank_scores_of_some_RDF_graphs/28119`. 2018 (cit. on p. 201).

[43] Diefenbach Dennis and Thalhammer Andreas. *PageRankRDF*. `https://github.com/WDAqua/PageRankRDF`. 2018 (cit. on p. 200).

[44] Diefenbach Dennis and Thalhammer Andreas. *SummaServer*. `https://github.com/WDAqua/SummaServer`. 2018 (cit. on p. 200).

[45] Diefenbach Dennis, Amjad Shanzay, Dridi Youssef, and Tardiveau Pierre. *Trill*. `https://github.com/WDAqua/Trill`. 2016 (cit. on p. 200).

[46] D. Diefenbach, S. Amjad, A. Both, K. Singh, and P. Maret. „Trill: A reusable Front-End for QA systems“. In: *ESWC P&D*. 2017 (cit. on pp. 14, 18, 61, 83, 87, 94, 104, 108, 118, 159, 199).

[47] D. Diefenbach, N. Hormozi, S. Amjad, and A. Both. „Introducing Feedback in Qanary: How Users can interact with QA systems“. In: *ESWC P&D*. 2017 (cit. on pp. 14, 18, 91, 115, 199).

[48] D. Diefenbach, P. Lhérisson, F. Muhlenbach, and P. Maret. „Computing the Semantic Relatedness of Music Genre using Semantic Web Data.“ In: *SEMANTiCS (Posters, Demos, SuCCESS)*. 2016 (cit. on p. 199).

[49] D. Diefenbach, V. Lopez, K. Singh, and P. Maret. „Core techniques of question answering systems over knowledge bases: a survey“. In: *Knowledge and Information systems* (), pp. 1–41 (cit. on pp. 5, 18, 21, 64, 65, 68, 107, 108, 199).

[50] D. Diefenbach, K. Singh, A. Both, et al. „The Qanary Ecosystem: getting new insights by composing Question Answering pipelines“. In: *ICWE*. 2017 (cit. on pp. 17, 19, 65, 83, 108, 117, 153, 199).

[51] D. Diefenbach, K. Singh, A. Both, et al. „The Qanary Ecosystem: getting new insights by composing Question Answering pipelines“. In: *ICWE*. 2017 (cit. on pp. 61, 88).

[52] D. Diefenbach, K. Singh, and P. Maret. „WDAqua-core0: A Question Answering Component for the Research Community“. In: *ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7)*. 2017 (cit. on pp. 89, 94, 103, 108, 118, 159, 199, 200).

[53] D. Diefenbach, T. Tanon, K. Singh, and P. Maret. „Question Answering Benchmarks for Wikidata“. In: *ISWC 2017*. 2017 (cit. on pp. 73, 79, 200).

[54] D. Diefenbach, R. Usbeck, K. Singh, and P. Maret. „A scalable approach for computing semantic relatedness using semantic web data“. In: *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*. ACM. 2016, p. 20 (cit. on pp. 13, 200).

[55] Dennis Diefenbach, Andreas Both, Kamal Singh, and Pierre Maret. „Towards a Question Answering System over the Semantic Web“. In: *Semantic Web Journal (under review)* (2018) (cit. on pp. 13, 18, 63).

[56] Dennis Diefenbach and Andreas Thalhammer. „PageRank and Generic Entity Summarization for RDF Knowledge Bases“. In: *ESWC*. Springer. 2018 (cit. on pp. 14, 18, 93, 200).

[57] T. G. Dietterich. „Ensemble learning“. In: *The Handbook of Brain Theory and Neural Networks*. 2002 (cit. on p. 138).

[58] Corina Dima. „Answering natural language questions with Intui3“. In: *Conference and Labs of the Evaluation Forum (CLEF)*. 2014 (cit. on pp. 28, 32, 35, 41, 47, 48, 53, 80, 154).

[59] Corina Dima. „Intui2: A prototype system for question answering over linked data“. In: *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF* (2013) (cit. on pp. 28, 32, 35, 41, 47, 53, 80).

[60] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. „Question Answering over Freebase with Multi-Column Convolutional Neural Networks.“ In: *ACL (1)*. 2015 (cit. on pp. 55, 56).

[61] Mohnish Dubey, Sourish Dasgupta, Ankit Sharma, Konrad Höffner, and Jens Lehmann. „AskNow: A Framework for Natural Language Query Formalization in SPARQL“. In: *International Semantic Web Conference*. Springer. 2016 (cit. on pp. 64, 80).

[62] Sanjay K. Dwivedi and Vaishali Singh. „Research and Reviews in Question Answering System“. In: *Procedia Technology* 10 (2013), pp. 417–424 (cit. on p. 23).

[63] Basil Ell, Denny Vrandečić, and Elena Simperl. „Spartiqulation: Verbalizing sparql queries“. In: *Extended Semantic Web Conference*. Springer. 2012, pp. 117–131 (cit. on pp. 107, 108).

[64] Anthony Fader, Stephen Soderland, and Oren Etzioni. „Identifying relations for open information extraction“. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011 (cit. on p. 38).

[65] Anthony Fader, Luke S Zettlemoyer, and Oren Etzioni. „Paraphrase-Driven Learning for Open Question Answering.“ In: *ACL (1)*. Citeseer. 2013 (cit. on p. 56).

[66] Javier D Fernández, Miguel A Martinez-Prieto, Claudio Gutierrez, Axel Polleres, and Mario Arias. „Binary RDF representation for publication and exchange (HDT)“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 19 (2013), pp. 22–41 (cit. on pp. 76, 96).

[67] Oscar Ferrandez, Christian Spurk, Milen Kouylekov, et al. „The QALL-ME framework: A specifiable-domain multilingual question answering architecture“. In: *Web semantics: Science, services and agents on the world wide web* (2011) (cit. on pp. 61, 65, 88, 116, 124, 138, 140, 154, 156).

[68] Sébastien Ferré. „Sparklis: an expressive query builder for SPARQL endpoints with guidance in natural language“. In: *Semantic Web* 8.3 (2017), pp. 405–418 (cit. on p. 27).

[69] Sébastien Ferré. „squall2sparql: a Translator from Controlled English to Full SPARQL 1.1“. In: *Work. Multilingual Question Answering over Linked Data (QALD-3)*. 2013 (cit. on p. 27).

[70] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. „Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling“. In: *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proc. of the Conf.* 2005 (cit. on pp. 126, 154–156, 159, 161, 163).

[71] A. Freitas, J. Oliveira, E. Curry, S. O'Riain, and J. da Silva. „Treo: combining entity-search, spreading activation and semantic relatedness for querying linked data". In: *1st Workshop on Question Answering over Linked Data (QALD-1)*. 2011 (cit. on p. 154).

[72] Andre Freitas and Edward Curry. „Natural language queries over heterogeneous linked data graphs: A distributional-compositional semantics approach". In: *Proceedings of the 19th international conference on Intelligent User Interfaces*. ACM. 2014 (cit. on pp. 28, 31, 35, 41, 42, 47, 51, 53).

[73] André Freitas, Edward Curry, João Gabriel Oliveira, and Seán O'Riain. „Querying Heterogeneous Datasets on the Linked Data Web: Challenges, Approaches, and Trends". In: *IEEE Internet Computing* (2012) (cit. on pp. 23, 58, 59).

[74] Andre Freitas, Juliano Efson Sales, Siegfried Handschuh, and Edward Curry. „How hard is this query? Measuring the Semantic Complexity of Schema-agnostic Queries". In: *Proceedings of the 11th International Conference on Computational Semantics*. London, UK: Association for Computational Linguistics, Apr. 2015 (cit. on pp. 58, 60).

[75] Daniel Gerber and A-C Ngonga Ngomo. „Bootstrapping the linked data web". In: *1st Workshop on Web Scale Knowledge Extraction@ ISWC*. Vol. 2011. 2011 (cit. on pp. 38, 77).

[76] Cristina Giannone, Valentina Bellomaria, and Roberto Basili. „A HMM-based approach to question answering against linked data". In: *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF* (2013) (cit. on pp. 28, 35, 41, 43, 47, 49, 53, 80).

[77] Jose M Gimenez-Garcia, Antoine Zimmermann, and Pierre Maret. „NdFluents: An Ontology for Annotated Statements with Inference Preservation". In: *European Semantic Web Conference*. Springer. 2017, pp. 638–654 (cit. on p. 83).

[78] David Golub and Xiaodong He. „Character-level question answering with attention". In: *arXiv preprint arXiv:1604.00727* (2016) (cit. on pp. 57, 82).

[79] Google. *Freebase Data Dumps*. `https://developers.google.com/freebase/data`. 2016 (cit. on p. 24).

[80] Sherzod Hakimov, Soufian Jebbara, and Philipp Cimiano. „AMUSE: Multilingual Semantic Parsing for Question Answering over Linked Data". In: *International Semantic Web Conference*. Springer. 2017, pp. 329–346 (cit. on pp. 73, 81).

[81] Sherzod Hakimov, Christina Unger, Sebastian Walter, and Philipp Cimiano. „Applying semantic parsing to question answering over linked data: Addressing the lexical gap". In: *Natural Language Processing and Information Systems*. Springer, 2015 (cit. on pp. 28, 35, 41, 47, 49, 50, 53, 80).

[82] Thierry Hamon, Natalia Grabar, Fleur Mougin, and Frantz Thiessard. „Description of the POMELO System for the Task 2 of QALD-2014." In: *CLEF (Working Notes)*. 2014 (cit. on pp. 28, 35, 41, 47, 53, 54).

[83] S.M. Harabagiu, D.I. Moldovan, and J. Picone. „Open-Domain Voice-Activated Question Answering". In: *19th Int. Conf. on Computational Linguistics, COLING 2002*. 2002 (cit. on pp. 123, 126).

[84] Shizhu He, Yuanzhe Zhang, Kang Liu, and Jun Zhao. „CASIA@ V2: A MLN-based Question Answering System over Linked Data". In: *Proc. of QALD-4* (2014) (cit. on pp. 28, 30, 35, 38, 39, 41, 45, 47, 50, 53, 80).

[85] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer. „Integrating NLP Using Linked Data". In: *The Semantic Web - ISWC 2013 - 12th Int. Semantic Web Conf., Proceedings, Part II*. 2013, pp. 98–113 (cit. on pp. 129, 139).

[86] Konrad Höffner and Jens Lehmann. „Question Answering on Statistical Linked Data". In: (2015) (cit. on p. 27).

[87] Konrad Hoffner, Sebastian Walter, Edgard Marx, et al. „Survey on Challenges of Question Answering in the Semantic Web". In: *Semantic Web Journal* (2016) (cit. on pp. 23, 58, 60).

[88] Y. Ibrahim, M.A. Yosef, and G. Weikum. „AIDA-Social: Entity Linking on the Social Stream". In: *Exploiting Semantic Annotations in Information Retrieval*. 2014 (cit. on p. 140).

[89] Sarthak Jain. „Question Answering over Knowledge Base using Factual Memory Networks". In: *Proceedings of NAACL-HLT*. 2016 (cit. on pp. 25, 55, 64).

[90] Guyonvarc'H Joris and Sébastien Ferré. „Scalewelis: a scalable query-based faceted search system on top of sparql endpoints". In: *Work. Multilingual Question Answering over Linked Data (QALD-3)*. 2013 (cit. on p. 27).

[91] Lucie-Aimée Kaffee, Alessandro Piscopo, Pavlos Vougiouklis, et al. „A Glimpse into Babel: An Analysis of Multilinguality in Wikidata". In: *Proceedings of the 13th International Symposium on Open Collaboration*. ACM. 2017, p. 14 (cit. on p. 79).

[92] E. Kaufmann, A. Bernstein, and R. Zumstein. „Querix: A natural language interface to query ontologies based on clarification dialogs". In: *ISWC*. 2006 (cit. on pp. 115, 116).

[93] Oleksandr Kolomiyets and Marie-Francine Moens. „A Survey on Question Answering Technology from an Information Retrieval Perspective". In: *Inf. Sci.* 181.24 (2011), pp. 5412–5434 (cit. on p. 23).

[94] C. Lange, S. Shekarpour, and S. Auer. „The WDAqua ITN: Answering Questions using Web Data". In: *EU project networking session at ESWC*. 2015 (cit. on p. 106).

[95] L.Ding, T.W. Finin, A. Joshi, et al. „Swoogle: a search and metadata engine for the semantic web". In: *Proc. of the 2004 ACM CIKM Int. Conf. on Information and Knowledge Management*. 2004, pp. 652–659 (cit. on p. 123).

[96] V. Lopez, M. Fernández, E. Motta, and N. Stieler. „PowerAqua: Supporting users in querying and exploring the semantic web“. In: *Semantic Web* 3.3 (2011), pp. 249–265 (cit. on pp. 122, 123, 126, 129, 137, 139).

[97] V. Lopez, E. Motta, M. Sabou, and M. Fernandez. *PowerAqua: A Multi-Ontology Based Question Answering System–v1*. OpenKnowledge Deliverable D8.4. 2007 (cit. on p. 140).

[98] V. Lopez, P. Tommasi, S. Kotoulas, and J. Wu. „QuerioDALI: Question Answering Over Dynamic and Linked Knowledge Graphs“. In: *International Semantic Web Conference*. Springer. 2016, pp. 363–382 (cit. on pp. 62, 65, 80).

[99] V. Lopez, C. Unger, P. Cimiano, and E. Motta. „Evaluating question answering over linked data“. In: *Web Semantics Science Services And Agents On The World Wide Web* (2013) (cit. on p. 115).

[100] Vanessa Lopez, Miriam Fernández, Enrico Motta, and Nico Stieler. „Poweraqua: Supporting Users in Querying and Exploring the Semantic Web“. In: *Semant. web* 3.3 (Aug. 2012) (cit. on pp. 28, 30, 31, 35–37, 41, 42, 47, 48, 52–54).

[101] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. „Evaluating question answering over linked data“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2013) (cit. on pp. 23, 24, 27, 28, 58, 60).

[102] Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. „AquaLog: An ontology-driven question answering system for organizational semantic intranets“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.2 (2007), pp. 72–105 (cit. on p. 31).

[103] Vanessa Lopez, Victoria Uren, Marta Sabou, and Enrico Motta. „Is question answering fit for the semantic web? a survey“. In: *Semantic Web* 2.2 (2011) (cit. on pp. 22–24, 58, 59, 63, 123, 139).

[104] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. „Neural Network-based Question Answering over Knowledge Graphs on Word and Character Level“. In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2017, pp. 1211–1220 (cit. on pp. 57, 64, 82).

[105] J. Luque, D. Ferres, J. Hernando, J.B. Mariño, and H. Rodriguez. „GeoVAQA: a voice activated geographical question answering system“. In: () (cit. on p. 129).

[106] Ioanna Lytra, Maria-Esther Vidal, Christoph Lange, Sören Auer, and Elena Demidova. „WDAqua – Answering Questions using Web Data“. In: (cit. on p. 169).

[107] Rahmad Mahendra, Lilian Wanzare, Raffaella Bernardi, Alberto Lavelli, and Bernardo Magnini. „Acquiring relational patterns from wikipedia: A case study“. In: *Proc. of the 5th Language and Technology Conference*. 2011 (cit. on p. 39).

[108] Anca Marginean. „Question answering over biomedical linked data with grammatical framework“. In: *Semantic Web* 8.4 (2017), pp. 565–580 (cit. on pp. 28, 35, 41, 47, 49, 53, 54).

[109] Edgard Marx, Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, et al. „Towards an open question answering architecture“. In: *Proceedings of the 10th International Conference on Semantic Systems*. ACM. 2014 (cit. on pp. 61, 65, 88, 116, 124, 126, 138, 140, 154, 156).

[110] Giuseppe M Mazzeo and Carlo Zaniolo. „Answering Controlled Natural Language Questions on RDF Knowledge Bases“. In: (2016) (cit. on p. 27).

[111] Giuseppe M Mazzeo and Carlo Zaniolo. „Question Answering on RDF KBs using Controlled Natural Language and Semantic Autocompletion“. In: *Semantic Web Journal (under review)* (2016) (cit. on pp. 115, 116).

[112] P. N. Mendes, M. Jakob, A. Garcia-Silva, and C. Bizer. „DBpedia Spotlight: Shedding Light on the Web of Documents“. In: *Proceedings of the 7th International Conference on Semantic Systems*. I-Semantics ’11. Graz, Austria, 2011 (cit. on pp. 154–156, 163).

[113] Donald Metzler and W Bruce Croft. „Linear feature-based models for information retrieval“. In: *Information Retrieval* 10.3 (2007), pp. 257–274 (cit. on p. 70).

[114] A. Moschitti, J. Chu-Carroll, S. Patwardhan, J. Fan, and G. Riccardi. „Using Syntactic and Semantic Structural Kernels for Classifying Definition Questions in Jeopardy!“ In: *Proc. of the 2011 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2011*. 2011, pp. 712–724 (cit. on p. 123).

[115] T. Mossakowski, O. Kutz, and C. Lange. „Three Semantics for the Core of the Distributed Ontology Language“. In: *Formal Ontology in Information Systems*. 7th International Conference (FOIS 2012). (Graz, Austria, July 24–27, 2012). Amsterdam, 2012 (cit. on p. 145).

[116] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. „PATTY: a taxonomy of relational patterns with semantic types“. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics. 2012 (cit. on pp. 33, 38, 140, 147).

[117] Shuyo Nakatani. *Language Detection Library for Java*. https://github.com/shuyo/language-detection. 2010 (cit. on pp. 83, 159).

[118] Axel-Cyrille Ngonga Ngomo, Michael Hoffmann, Ricardo Usbeck, and Kunal Jha. „Holistic and Scalable Ranking of RDF Data“. In: *2017 IEEE International Conference on Big Data*. to appear. 2017 (cit. on p. 105).

[119] Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. „Sorry, i don't speak SPARQL: translating SPARQL queries into natural language". In: *Proceedings of the 22nd international conference on World Wide Web*. ACM. 2013, pp. 977–988 (cit. on pp. 107, 108).

[120] E. Oren, R. Delbru, M. Catasta, et al. „Sindice.com: a document-oriented lookup index for open linked data". In: *IJMSO* 3.1 (2008), pp. 37–52 (cit. on p. 123).

[121] Seonyeong Park, Hyosup Shim, and Gary Geunbae Lee. „ISOFT at QALD-4: Semantic similarity-based question answering system over linked data". In: *CLEF*. 2014 (cit. on pp. 28, 35, 41, 47, 48, 53, 80).

[122] Amir Pouran-ebn-veyseh. „Cross-Lingual Question Answering Using Profile HMM & Unified Semantic Space". In: *ESWC. to appear*. 2016 (cit. on pp. 28, 31, 35, 41, 47, 53, 64, 78, 80).

[123] Seyedamin Pouriyeh, Mehdi Allahyari, Krzysztof Kochut, Gong Cheng, and Hamid Reza Arabnia. „ES-LDA: Entity Summarization using Knowledge-based Topic Modeling". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, 2017, pp. 316–325 (cit. on p. 105).

[124] Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. „A semantic web interface using patterns: the SWIP system". In: *Graph Structures for Knowledge Representation and Reasoning*. Springer, 2012 (cit. on pp. 28, 35, 41, 47, 53, 80).

[125] Siva Reddy, Mirella Lapata, and Mark Steedman. „Large-scale semantic parsing without question-answer pairs". In: *Transactions of the Association for Computational Linguistics* (2014) (cit. on pp. 55, 56).

[126] Siva Reddy, Oscar Täckström, Michael Collins, et al. „Transforming dependency structures to logical forms for semantic parsing". In: *Transactions of the Association for Computational Linguistics* (2016) (cit. on pp. 54–56).

[127] G. Rizzo and R. Troncy. „NERD: A Framework for Unifying Named Entity Recognition and Disambiguation Extraction Tools". In: *13th Conference of the European Chapter of the Association for Computational Linguistics*. Avignon, France, 2012 (cit. on p. 156).

[128] Antonio J. Roa-Valverde and Miguel-Angel Sicilia. „A survey of approaches for ranking on the web of data". In: *Information Retrieval* 17.4 (2014), pp. 295–325 (cit. on p. 105).

[129] Antonio J. Roa-Valverde, Andreas Thalhammer, Ioan Toma, and Miguel-Angel Sicilia. „Towards a formal model for sharing and reusing ranking computations". In: *Proceedings of the 6th International Workshop on Ranking in Databases (DBRank 2012) held in conjunction with the 38th Conference on Very Large Databases (VLDB 2012)*. 2012 (cit. on p. 101).

[130] Stefan Ruseti, Alexandru Mirea, Traian Rebedea, and Stefan Trausan-Matu. „QAnswer-Enhanced Entity Matching for Question Answering over Linked Data“. In: *CLEF (Working Notes)*. CLEF. 2015 (cit. on pp. 28, 35, 39, 41, 47, 49, 53, 80, 128).

[131] R. Sanderson, P. Ciccarese, H. Van de Sompel, et al. „Open annotation data model“. In: *W3C Community Draft* (2013) (cit. on p. 129).

[132] Saeedeh Shekarpour, Edgard Marx, Axel-Cyrille Ngonga Ngomo, and Sören Auer. „Sina: Semantic interpretation of user queries for question answering on interlinked data“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* (2015) (cit. on pp. 8, 28, 30, 35, 36, 41, 43, 47, 51, 53, 54, 64, 80, 88, 123, 129, 140, 148, 154, 155, 163).

[133] K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. „Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems“. In: *ICSC 2016*. 2016 (cit. on pp. 16, 18, 65, 88, 116, 117, 121, 138, 144, 145, 149, 154, 157, 158, 199).

[134] K. Singh, A. Both, D. Diefenbach, et al. „Qanary – The Fast Track to Creating a Question Answering System with Linked Data Technology“. In: *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers*. Ed. by Harald Sack, Giuseppe Rizzo, Nadine Steinmetz, et al. Cham: Springer International Publishing, 2016, pp. 183–188 (cit. on p. 199).

[135] Kuldeep Singh, Arun Sethupat Radhakrishna, Andreas Both, et al. „Why Reinvent the Wheel – Let's Build estion Answering Systems Together“. In: *The Web Conference*. Springer. 2018 (cit. on pp. 17, 18).

[136] A. Singhal. „Introducing the knowledge graph: things, not strings“. In: *Official Google Blog, May* (2012) (cit. on pp. 123, 139).

[137] Dezhao Song, Frank Schilder, Charese Smiley, et al. „TR Discover: A Natural Language Interface for Querying and Analyzing Interlinked Datasets“. In: *The Semantic Web-ISWC 2015*. Springer, 2015 (cit. on pp. 28, 35, 41, 47, 49, 53, 54, 62).

[138] Daniil Sorokin and Iryna Gurevych. „End-to-end Representation Learning for Question Answering with Weak Supervision“. In: *ESWC 2017 Semantic Web Challenges*. Portoroz, Slovenia, 2017 (cit. on p. 81).

[139] R. Speck and A. Ngonga Ngomo. „Ensemble learning for named entity recognition“. In: *The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. Springer International Publishing, 2014 (cit. on pp. 155, 156, 159, 163).

[140] F. M. Suchanek, G. Kasneci, and G. Weikum. „Yago: a core of semantic knowledge“. In: *16th WWW Conf.* 2007, pp. 697–706 (cit. on p. 124).

[141] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. „YAGO: A Core of Semantic Knowledge“. In: *Proceedings of the 16th International Conference on World Wide Web*. WWW '07. Banff, Alberta, Canada: ACM, 2007, pp. 697–706 (cit. on p. 99).

[142] H. Sun, H. Ma, W.-T. Yih, et al. „Open Domain Question Answering via Semantic Enrichment“. In: *WWW*. 2015 (cit. on pp. 123, 126, 140).

[143] Andreas Thalhammer. „Linked Data Entity Summarization“. Phdthesis. Karlsruhe: KIT, Fakultät für Wirtschaftswissenschaften, 2016 (cit. on p. 105).

[144] Andreas Thalhammer, Nelia Lasierra, and Achim Rettinger. „LinkSUM: Using Link Analysis to Summarize Entity Data“. In: *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*. Vol. 9671. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 244–261 (cit. on pp. 84, 105).

[145] Andreas Thalhammer, Nelia Lasierra, and Achim Rettinger. „LinkSUM: using link analysis to summarize entity data“. In: *International Conference on Web Engineering*. Springer. 2016 (cit. on p. 90).

[146] Andreas Thalhammer and Achim Rettinger. „Browsing DBpedia Entities with Summaries“. English. In: *The Semantic Web: ESWC 2014 Satellite Events*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 511–515 (cit. on pp. 94, 101).

[147] Andreas Thalhammer and Achim Rettinger. „ELES: combining entity linking and entity summarization“. In: *International Conference on Web Engineering*. Springer. 2016, pp. 547–550 (cit. on p. 94).

[148] Andreas Thalhammer and Achim Rettinger. „PageRank on Wikipedia: Towards General Importance Scores for Entities“. In: *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers*. Cham: Springer International Publishing, Oct. 2016, pp. 227–240 (cit. on pp. 94, 95, 98, 99, 105, 106).

[149] Andreas Thalhammer and Achim Rettinger. „PageRank on Wikipedia: towards general importance scores for entities“. In: *International Semantic Web Conference*. Springer. 2016 (cit. on p. 91).

[150] Andreas Thalhammer and Steffen Stadtmüller. „SUMMA: A Common API for Linked Data Entity Summaries“. English. In: *Engineering the Web in the Big Data Era*. Vol. 9114. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 430–446 (cit. on pp. 94, 96, 99, 100, 105).

[151] Felix Tristram, Sebastian Walter, Philipp Cimiano, and Christina Unger. „Weasel: a machine learning based approach to entity linking combining different features“. In: *Proceedings of 3th International Workshop on NLP and DBpedia, co-located*

with the 14th International Semantic Web Conference (ISWC 2015), October 11-15, USA. Bethlehem, Pennsylvania, USA, 2015 (cit. on p. 94).

[152] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. „LC-QuAD: A corpus for complex question answering over knowledge graphs“. In: *International Semantic Web Conference*. Springer. 2017, pp. 210–218 (cit. on p. 79).

[153] Ferhan Ture and Oliver Jojic. „Simple and Effective Question Answering with Recurrent Neural Networks“. In: *arXiv preprint arXiv:1606.05029* (2016) (cit. on pp. 55, 56).

[154] A. Uciteli, Ch. Goller, P. Burek, et al. „Search Ontology, a new approach towards Semantic Search“. In: *Workshop on Future Search Engines*. 2014 (cit. on pp. 124, 126, 141).

[155] C. Unger, C. Forascu, V. Lopez, et al. „Question Answering over Linked Data (QALD-5)“. In: *CLEF (Working Notes)*. 2015 (cit. on p. 163).

[156] Christina Unger, Lorenz Bühmann, Jens Lehmann, et al. „Template-based question answering over RDF data“. In: *Proceedings of the 21st international conference on World Wide Web*. ACM. 2012, pp. 639–648 (cit. on pp. 28, 35, 37, 39, 41, 47, 49, 50, 53, 122, 123, 129, 137, 140).

[157] Christina Unger, Corina Forascu, Vanessa Lopez, et al. „Answering over Linked Data (QALD-5).“ In: *Working Notes for CLEF 2015 Conference*. 2015 (cit. on pp. 23, 24, 27, 28, 80).

[158] Christina Unger, Corina Forascu, Vanessa Lopez, et al. „Question answering over linked data (QALD-4)“. In: *Working Notes for CLEF 2014 Conference*. 2014 (cit. on pp. 23, 24, 27, 28, 58, 60, 80).

[159] Christina Unger, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, and Cimiano. „6th Open Challenge on Question Answering over Linked Data (QALD-6)“. In: *The Semantic Web: ESWC 2016 Challenges.* 2016 (cit. on pp. 23, 24, 27, 28, 81, 107).

[160] R. Usbeck, A.-C. Ngonga Ngomo, M. Roder, et al. „AGDISTIS – Graph-Based Disambiguation of Named Entities Using Linked Data“. In: *ISWC*. 2014 (cit. on pp. 124, 129, 140, 142, 155, 156, 159, 161, 163).

[161] R. Usbeck, M. Röder, A.-C. Ngonga Ngomo, et al. „GERBIL – General Entity Annotation Benchmark Framework“. In: *24th WWW conference*. 2015 (cit. on pp. 136, 156).

[162] Ricardo Usbeck, Röder Michael, Christina Unger, et al. *Benchmarking question answering systems*. Tech. rep. Technical report, Leipzig University, 2016 (cit. on pp. 17, 83).

[163] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, and Christina Unger. „HAWK–Hybrid Question Answering Using Linked Data“. In: *The Semantic Web. Latest Advances and New Domains*. Springer, 2015 (cit. on p. 27).

[164] Denny Vrandecic and Markus Krotzsch. „Wikidata: A Free Collaborative Knowledgebase“. In: *Communications of the ACM* 57.10 (2014), pp. 78–85 (cit. on p. 95).

[165] Sebastian Walter, Christina Unger, and Philipp Cimiano. „M-ATOLL: a framework for the lexicalization of ontologies in multiple languages“. In: *The Semantic Web–ISWC 2014*. Springer, 2014 (cit. on p. 38).

[166] Sebastian Walter, Christina Unger, Philipp Cimiano, and Daniel Bär. „Evaluation of a layered approach to question answering over linked data“. In: *The Semantic Web–ISWC 2012*. Springer, 2012 (cit. on pp. 28, 35, 41, 47, 49, 53).

[167] Zhenghao Wang, Shengquan Yan, Huaming Wang, and Xuedong Huang. *An overview of Microsoft deep QA system on Stanford WebQuestions benchmark*. Tech. rep. Technical report, Microsoft Research, 2014 (cit. on p. 55).

[168] Wilkinson, M. D. *et al.* „The FAIR Guiding Principles for scientific data management and stewardship“. In: *Scientific Data* 3 (2016) (cit. on p. 105).

[169] Fei Wu and Daniel S Weld. „Open information extraction using Wikipedia“. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010 (cit. on p. 38).

[170] Kun Xu, Yansong Feng, and Dongyan Zhao. *Xser@ QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing*. 2014 (cit. on pp. 28, 29, 31, 33, 35, 38, 40, 41, 45, 47, 49, 53, 64, 65, 78, 80, 128).

[171] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. „Robust question answering over the web of linked data“. In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM. 2013 (cit. on p. 48).

[172] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, et al. „Natural language questions for the web of data“. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics. 2012 (cit. on pp. 28, 31, 35, 41, 44, 47, 53, 64, 80).

[173] Min-Chul Yang, Nan Duan, Ming Zhou, and Hae-Chang Rim. „Joint Relational Embeddings for Knowledge-based Question Answering.“ In: *EMNLP*. 2014 (cit. on pp. 55, 56).

[174] Min-Chul Yang, Do-Gil Lee, So-Young Park, and Hae-Chang Rim. „Knowledge-based question answering using the semantic embedding space“. In: *Expert Systems with Applications* (2015) (cit. on pp. 55, 56).

[175] Xuchen Yao. „Lean Question Answering over Freebase from Scratch.“ In: *HLT-NAACL*. 2015 (cit. on p. 55).

[176] Xuchen Yao and Benjamin Van Durme. „Information Extraction over Structured Data: Question Answering with Freebase.“ In: *ACL (1)*. Citeseer. 2014 (cit. on pp. 55, 56).

[177] Alexander Yates, Michael Cafarella, Michele Banko, et al. „Textrunner: open information extraction on the web“. In: *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Association for Computational Linguistics. 2007 (cit. on p. 38).

[178] Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, and Xifeng Yan. „Improving Semantic Parsing via Answer Type Inference.“ In: *EMNLP*. 2016, pp. 149–159 (cit. on pp. 55, 56).

[179] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. „Semantic parsing via staged query graph generation: Question answering with knowledge base“. In: (2015) (cit. on pp. 55, 56).

[180] Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. „The Value of Semantic Parse Labeling for Knowledge Base Question Answering.“ In: *ACL (2)*. 2016 (cit. on p. 25).

[181] Wenpeng Yin, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze. „Simple question answering by attentive convolutional neural network“. In: *arXiv preprint arXiv:1606.03391* (2016) (cit. on pp. 57, 82).

[182] Mohamed Amir Yosef, Johannes Hoffart, Ilaria Bordino, Marc Spaniol, and Gerhard Weikum. „Aida: An online tool for accurate disambiguation of named entities in text and tables“. In: *Proceedings of the VLDB Endowment* 4 (2011) (cit. on pp. 30, 124, 126).

[183] Dridi Youssef and Diefenbach Dennis. *SPARQLtoUser*. `https://github.com/WDAqua/SPARQL_to_User`. 2017 (cit. on p. 200).

[184] Luke S Zettlemoyer and Michael Collins. „Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars“. In: *arXiv preprint arXiv:1207.1420* (2012) (cit. on p. 50).

[185] Yuanzhe Zhang, Kang Liu, Shizhu He, et al. „Question Answering over Knowledge Base with Neural Attention Combining Global Knowledge Information“. In: *arXiv preprint arXiv:1606.00979* (2016) (cit. on pp. 55, 56, 65).

[186] Yuanzhe Zhange, Shizhu He, Kang Liu, and Jun Zhao. „A Joint Model for Question Answering over Multiple Knowledge Bases“. In: (2016) (cit. on pp. 28, 35, 41, 47, 52–54).

[187] Chenhao Zhu, Kan Ren, Xuan Liu, et al. „A Graph Traversal Based Approach to Answer Non-Aggregation Questions Over DBpedia“. In: *arXiv preprint arXiv:1510.04780* (2015) (cit. on pp. 28, 35, 40, 41, 47, 53, 80).

[188] Lei Zou, Ruizhe Huang, Haixun Wang, et al. „Natural language question answering over RDF: a graph data driven approach“. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014 (cit. on pp. 8, 28, 33, 35, 38, 40–42, 47, 48, 53, 64, 65, 80, 128).

List of Figures

List of Tables

List of Publications

K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. „Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems“. In: *ICSC 2016*. 2016.

K. Singh, A. Both, D. Diefenbach, S. Shekarpour, D. Cherix, and C. Lange. „Qanary – The Fast Track to Creating a Question Answering System with Linked Data Technology“. In: *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers*. Ed. by Harald Sack, Giuseppe Rizzo, Nadine Steinmetz, Dunja Mladenić, Sören Auer, and Christoph Lange. Cham: Springer International Publishing, 2016, pp. 183–188.

D. Diefenbach, P. Lhérisson, F. Muhlenbach, and P. Maret. „Computing the Semantic Relatedness of Music Genre using Semantic Web Data.“ In: *SEMANTiCS (Posters, Demos, SuCCESS)*. 2016.

A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, and C. Lange. „Qanary–a methodology for vocabulary-driven open question answering systems“. In: *International Semantic Web Conference*. Springer. 2016.

D. Diefenbach, S. Amjad, A. Both, K. Singh, and P. Maret. „Trill: A reusable Front-End for QA systems“. In: *ESWC P&D*. 2017.

D. Diefenbach, K. Singh, and P. Maret. „WDAqua-core0: A Question Answering Component for the Research Community“. In: *ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7)*. 2017.

D. Diefenbach, N. Hormozi, S. Amjad, and A. Both. „Introducing Feedback in Qanary: How Users can interact with QA systems“. In: *ESWC P&D*. 2017.

D. Diefenbach, V. Lopez, K. Singh, and P. Maret. „Core techniques of question answering systems over knowledge bases: a survey“. In: *Knowledge and Information systems* (), pp. 1–41.

D. Diefenbach, K. Singh, A. Both, D. Cherix, C. Lange, and S. Auer. „The Qanary Ecosystem: getting new insights by composing Question Answering pipelines“. In: *ICWE*. 2017.

A. Both, K. Singh, D. Diefenbach, and I. Lytra. „Rapid Engineering of QA Systems Using the Light-Weight Qanary Architecture“. In: *Web Engineering: 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings*. Ed. by Jordi Cabot,

Roberto De Virgilio, and Riccardo Torlone. Cham: Springer International Publishing, 2017, pp. 544–548.

Diefenbach D., Dridi Y., Singh K., and Maret P. „SPARQLtoUser: Did the Question Answering System Understand me?“ In: *Joint Proceedings of BLINK2017: 2nd International Workshop on Benchmarking Linked Data and NLIWoD3: Natural Language Interfaces for the Web of Data co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21st - to - 22nd, 2017.* 2017.

D. Diefenbach, T. Tanon, K. Singh, and P. Maret. „Question Answering Benchmarks for Wikidata“. In: *ISWC 2017*. 2017.

D. Diefenbach, R. Usbeck, K. Singh, and P. Maret. „A scalable approach for computing semantic relatedness using semantic web data“. In: *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*. ACM. 2016, p. 20.

D. Diefenbach, K. Singh, and P. Maret. „WDAqua-core0: A Question Answering Component for the Research Community“. In: *ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7)*. 2017.

Dennis Diefenbach and Andreas Thalhammer. „PageRank and Generic Entity Summarization for RDF Knowledge Bases“. In: *ESWC*. Springer. 2018.

Software Repositories

Diefenbach Dennis, Amjad Shanzay, Dridi Youssef, and Tardiveau Pierre. *Trill*. `https://github.com/WDAqua/Trill`. 2016.

Diefenbach Dennis, Both Andreas, Lytra Ioanna, Singh Kuldeep, Cherix Didier, and Mielke Stephan. *Qanary*. `https://github.com/WDAqua/Qanary`. 2016.

Diefenbach Dennis and Thalhammer Andreas. *PageRankRDF*. `https://github.com/WDAqua/PageRankRDF`. 2018.

Diefenbach Dennis and Thalhammer Andreas. *SummaServer*. `https://github.com/WDAqua/SummaServer`. 2018.

Dridi Youssef and Diefenbach Dennis. *SPARQLtoUser*. `https://github.com/WDAqua/SPARQL_to_User`. 2017.

Datasets

Diefenbach Dennis. *WDAquaCore0Questions*. https://github.com/WDAqua/WDAquaCore0Questions. 2017.

Diefenbach Dennis and Thalhammer Andreas. *PageRank scores of some RDF graphs*. https://figshare.com/projects/PageRank_scores_of_some_RDF_graphs/28119. 2018.

Diefenbach Dennis. *SimpleQuestions Linked Data*. https://figshare.com/projects/SimpleQuestionsLinkedData/28530. 2018.

Internship Students

Shanzay Amjad : Designing a front-end for QA systems (September 16 - December 16)

Youssef Dridi : Generating a user understandable representation of a SPARQL query (May 17 - July 17)

Shanzay Amjad : Transform an industrial dataset in RDF to make QA on it (May 17 July 17)

Pierre Tardiveau: Improving a front-end of a QA systems (December 17 - June 18)

Résumé Question Answering (QA) est un domaine de l'informatique qui se préoccupe de la construction d'un système qui peut répondre automatiquement à une question posée par un utilisateur en langage naturel. Les systèmes Question Answering utilisent principalement trois sources pour trouver une réponse: du texte libre (documents de bureau, pages web et livres), des services Web (services météorologiques, prix des actions, horaire, etc.) et bases de connaissances. Une base de connaissances est une collection structurée d'informations qui peuvent être consultées et interprétées facilement par des machines. Le sujet principal de cette thèse est d'étudier les systèmes de QA sur de telles bases de connaissances. Les systèmes de QA sur les bases de connaissances sont importants car ces bases contiennent de très grandes quantités d'informations et en même temps ces informations sont a priori très difficiles d'accès pour les utilisateurs.

Les systèmes QA sur bases de connaissances sont utilisés dans différentes applications industrielles telles que Google Search, Siri, Alexa et Bing. Ces applications ont en commun d'interroger des bases de connaissances propriétaires.

Cette thèse apporte plusieurs contributions dans le domaine des systèmes QA sur bases de connaissances.

D'abord, nous présentons une étude complète des systèmes QA existants. Nous présentons une analyse détaillée et une comparaison de tous les systèmes QA qui ont été évalués sur un benchmark populaire appelé QALD, et nous ciblons les systèmes de QA évalués par rapport à deux autres benchmarks très populaires, à savoir WebQuestions et SimpleQuestions. L'analyse contient également une liste des défis importants dans ce domaine.

Dans un deuxième temps, nous présentons un nouvel algorithme pour construire des systèmes de QA sur des bases de connaissances. Cet algorithme se caractérise par le fait qu'il peut être facilement adapté à de nouvelles langues, qu'il peut être facilement adapté aux nouvelles bases de connaissances, qu'il prend en charge les questions sous forme de mots clés et les questions en langage naturel, et qu'il est robuste aux questions mal formées. Nous prouvons ces revendications en appliquant ce nouvel algorithme à 5 langues différentes (à savoir anglais, allemand, français, espagnol et italien), à 6 bases de connaissances différentes (à savoir Wikidata, DBpedia, Freebase, Scigraph, Dblp et MusicBrainz) et en comparant les performances sur les benchmarks populaires QALD et SimpleQuestions.

Dans une troisième partie, nous montrons comment la réponse d'un système QA peut être présentée à un utilisateur. Les bases de connaissances contiennent beaucoup d'informations contextuelles qui peuvent être présentées avec la réponse elle-même. Ces informations sont par exemple des descriptions textuelles, des liens externes, des images et des vidéos. Nous montrons comment ces informations peuvent être présentées à l'utilisateur pour enrichir la réponse. En outre, nous proposons un mécanisme permettant à l'utilisateur d'interagir lorsque la question est ambiguë, un algorithme pour présenter à l'utilisateur l'interprétation que le système QA a de la question reçue et un algorithme pour générer des résumés.

La dernière partie décrit une architecture pour les systèmes QA qui permet de construire des systèmes QA de manière modulaire et collaborative. Cette approche permet de créer des systèmes QA distribués sur le Web et constitue une proposition pour standardiser le processus d'un système QA.

Toutes les contributions de cette thèse sont intégrées dans une démonstration en ligne disponible sous `www.wdaqua.eu/qa`.

Abstract Question Answering (QA) is a field in computer science, which is concerned about building a system, which can automatically answer a given question posed by user in natural language. There are mainly three sources that Question Answering systems use to find an answer: free text (like office documents, web-pages and books), web-services (like services for weather, stock prices and time) and knowledge bases (KB). KBs are a structured collection of information that can be accessed and interpreted easily by machines. The main topic of this thesis is to study QA systems over such KB. QA systems over KBs are important because there is a large amount of information available in KBs and at the same time the information in KBs is very difficult to access by end users.

QA systems over KBs are used in diverse industrial applications like Google Search, Siri, Alexa and Bing. One thing is common between these applications that they query proprietary KBs.

This thesis provides several contributions in the domain of QA systems over KBs.

First, it presents a comprehensive study of existing QA systems. It presents a detailed analysis and comparison of all QA systems that were evaluated over a popular benchmark called QALD, and points to the QA systems evaluated over other two very popular benchmarks, namely WebQuestions and SimpleQuestions. The analysis also contains a list of important challenges in this domain.

Second, it presents a novel algorithm to construct QA systems over KBs. It is characterized by the fact that it can be easily adapted to new languages as well as to new KBs. It supports both keyword questions and full natural language questions. Moreover, it is robust to malformed questions. We prove these claims by applying this novel algorithm to 5 different languages (namely English, German, French, Spanish and Italian), to 6 different knowledge bases (namely Wikidata, DBpedia, Freebase, Scigraph, Dblp and MusicBrainz) and by comparing its performance over popular benchmarks like QALD and SimpleQuestions.

Third, it shows how the answer of a QA system can be presented to a user. KBs contain main possible contextual information that can be presented together with the answer itself. These include textual descriptions, external links, images and videos. This thesis show how this information can be presented to the user together with the answer. Moreover, this thesis contains a mechanism to let the user interact when the question is ambiguous. This is done by using an algorithm which presents the interpretation of the question as understood by the QA system. Another algorithm is used to generate summaries for the answer to deliver additional contextual information.

The last part describes a possible architecture for QA systems that allows to construct QA systems in a modular and collaborative way. It allows to build QA systems which can be distributed over the web and is a tentative approach to standardize typical QA work flows.

All the contributions of this thesis are integrated in an online demo that is available under `www.wdaqua.eu/qa`.