Towards Dynamic Composition of Question Answering Pipelines

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von
Kuldeep Singh
aus
Sawai Madhopur, India

Bonn, 29.11.2018

1. Gutachter:	Prof. Dr. Sören Auer
2. Gutachter:	Prof. Dr. Jens Lehmann
Tag der Promotion:	17.01.2019
Erscheinungsjahr:	2019

Abstract

Question answering (QA) over knowledge graphs has gained significant momentum over the past five years due to the increasing availability of large knowledge graphs and the rising importance of question answering for user interaction. DBpedia has been the most prominently used knowledge graph in this setting. QA systems implement a pipeline connecting a sequence of QA components for translating an input question into its corresponding formal query (e.g. SPARQL); this query will be executed over a knowledge graph in order to produce the answer of the question. Recent empirical studies have revealed that albeit overall effective, the performance of QA systems and QA components depends heavily on the features of input questions, and not even the combination of the best performing QA systems or individual QA components retrieves complete and correct answers. Furthermore, these QA systems cannot be easily reused, extended, and results cannot be easily reproduced since the systems are mostly implemented in a monolithic fashion, lack standardised interfaces and are often not open source or available as Web services. All these drawbacks of the state of the art that prevents many of these approaches to be employed in real-world applications.

In this thesis, we tackle the problem of QA over knowledge graph and propose a generic approach to promote reusability and build question answering systems in a collaborative effort. Firstly, we define `qa` vocabulary and Qanary methodology to develop an abstraction level on existing QA systems and components. Qanary relies on `qa` vocabulary to establish guidelines for semantically describing the knowledge exchange between the components of a QA system. We implement a component-based modular framework called "Qanary Ecosystem" utilising the Qanary methodology to integrate several heterogeneous QA components in a single platform. We further present *Qaestro* framework that provides an approach to semantically describing question answering components and effectively enumerates QA pipelines based on a QA developer requirements. *Qaestro* provides all valid combinations of available QA components respecting the input-output requirement of each component to build QA pipelines. Finally, we address the scalability of QA components within a framework and propose a novel approach that chooses the best component per task to automatically build the QA pipeline for each input question. We implement this model within FRANKENSTEIN, a framework able to select QA components and compose pipelines. FRANKENSTEIN extends Qanary ecosystem and utilises qa vocabulary for data exchange. It has 29 independent QA components implementing five QA tasks resulting in 360 unique QA pipelines. Each approach proposed in this thesis (Qanary methodology, *Qaestro*, and FRANKENSTEIN) is supported by extensive evaluation to demonstrate their effectiveness. Our contributions target a broader research agenda of offering the QA community an efficient way of applying their research to a research field which is driven by many different fields, consequently requiring a collaborative approach to achieve significant progress in the domain of question answering.

Acknowledgements

This work would not have been possible without the support and guidance of many individuals. I am grateful to my supervisor Prof. Dr. Sören Auer for providing me an opportunity to do thesis under his guidance. I have been very lucky to have him as my supervisor. His trust in my potentials and capabilities have always inspired me to excel in research over time.

I am thankful to Prof. Dr. Jens Lehmann for the support provided by him during the course of this thesis. His valuable advises on many critical issues have helped me to vision new ideas while addressing the research problem. I am thankful to my colleagues in EIS department, Fraunhofer IAIS, SDA research group, University of Bonn, co-authors, and Marie Curie ITN network for all the good time we had in last 2.5 years, and mutual support on many day to day issues.

Dr. Christoph Lange has always inspired me with his sheer hardworking approach for putting effort in continuous improvement of my skills and capabilities. Thank you Dr. Lange. I extend my thanks to Dr. Ioanna Lytra for being such a wonderful mentor. Deep discussions together and your advises have been one of the foundations of this thesis.

I further extend thanks to my friends Isaiah Mulang', Dharmen Punjani, Akmal, Akhilesh, Yasher, Ahmad and Harsh Thakkar for all the support and good time. I am thankful to Arun for being a valuable addition to this research. I and him have complemented each other in our approaches towards research, and that has been our strength to target very challenging problems successfully. I would also thank Dr. Saeedeh Shekarpour for trusting me, and always motivating me to work hard.

Also, I would like to thank my family, especially my mother Mrs. Suman Lata, and father Mr. Arvind Kumar for their generous support, care, many sacrifices and love. Journey from a small rural village in India till Bonn was never easy, neither was part of my dreams. Your encouragement when the times got rough are much appreciated and duly noted. My wife Ankita has been a source of strength since we have been together. She sacrificed many things in personal life to shape myself professionally. She has been a silent contributor in this thesis. Thank you Ankita.

Prof. Dr. Andreas Both has nurtured my capabilities since I joined doctoral studies. In spite knowing my weaknesses, he made sure that I never fall back due to them and have always been there to motivate me to dream higher and higher. He helped me to overcome those weaknesses, and turned them into my strength. You are, and will always be someone I look upto as a role model. My supervisor and mentor Prof. Dr. Maria Esther Vidal has put a lot of effort in shaping the direction of this thesis. She has been my biggest critic, and her advises on nearly daily basis has improved my approach towards research a lot. I dedicate this thesis to my gurus: Prof. Dr. Maria Esther Vidal and Prof. Dr. Andreas Both.

Contents

CHAPTER 1

Introduction

In the era of Big Knowledge, Question Answering (QA) systems allow for responding natural language or voice-based questions posed against various data sources, e.g. knowledge graphs, videos, relational databases, or documents [1–3]. Particularly, with the advent of open knowledge graphs (e.g. DBpedia [4], Freebase [5], and Wikidata [6]), question answering over structured data gained momentum and researchers from different communities, e.g. semantic web, information retrieval, databases, and natural language processing have extensively studied this problem over the past decade [3, 7, 8]. Thus, since 2010, more than 62 QA systems have been published, and DBpedia is the underlying knowledge graph in 38 of them [3]. Those systems usually translate natural language questions to a formal representation of a query that extracts answers from the given knowledge graph. Figure 1.1 illustrates the layers of question answering process over knowledge graphs. Layer 3 comprises of underlying knowledge graph which is used as knowledge source to extract answers. Layer 2 presents several QA systems developed by researchers extract answers from the underlying knowledge base. The analysis of the architecture of these QA systems over DBpedia shows that the QA system architectures share similar question answering tasks on the abstract level [9] and abstract QA tasks are at the first level as illustrated in Figure 1.1. These tasks include Named Entity Recognition and Disambiguation (NER and NED), Relation Linking (RL), Class Linking (CL), dependency parsing, and Query Building(QB) [9, 10].
For instance, in question "What is the time zone of New York City?", an ideal QA system over DBpedia generates a formal representation of this question, that is a formal query (here expressed as SPARQL[1]), which retrieves all answers from the DBpedia endpoint[2] (i.e. `SELECT ?c {dbr:New_York_City dbo:timeZone ?c.}`). During this process, a QA system performs successive QA tasks. In the first step (i.e. NED), the QA system is expected to recognise and link the entity being present in the question to its candidate(s) from DBpedia (e.g. mapping `New York City` to `dbr:New_York_City`[3]). The next step is RL, where QA systems link the natural language predicate to the corresponding predicate in DBpedia (e.g. mapping `time zone` to `dbo:timeZone`[4]). In the final step, the QB component formulates a SPARQL query using these IRIs.
Research Objectives. Several independent QA components for various QA tasks (e.g. NED and RL) have been released by research community. Some of these components are reused in QA frameworks such as openQA [11], QALL-ME [12], OKBQA [13] to build QA systems in collaborative community efforts rather building a system from scratch. However, in existing frameworks, a user has to choose components

[1] https://www.w3.org/TR/rdf-sparql-query/
[2] http://dbpedia.org/sparql
[3] Prefix `dbr` is bound to http://dbpedia.org/resource/
[4] Prefix `dbo` is bound to http://dbpedia.org/ontology/

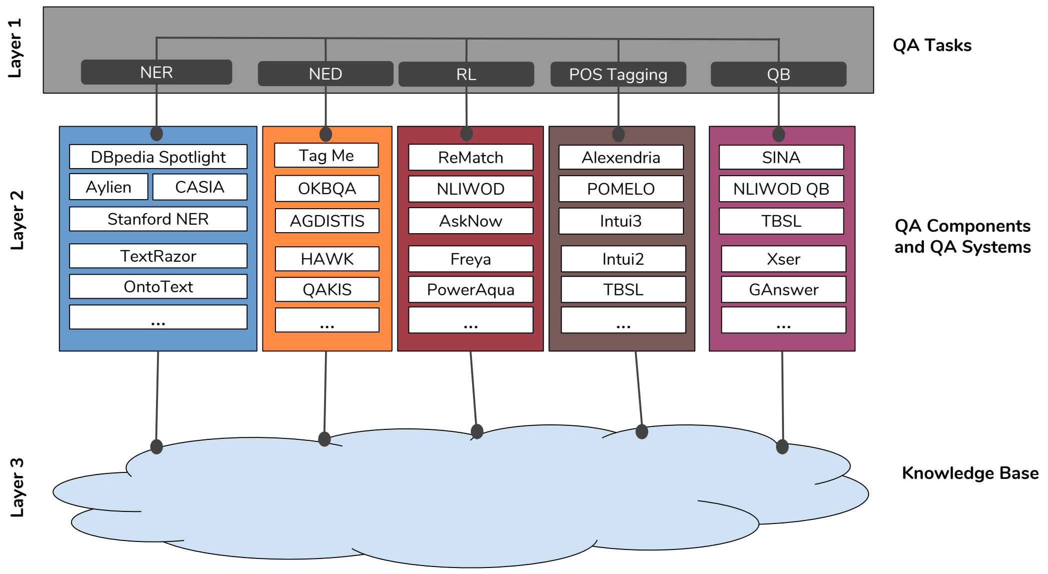


Figure 1.1: Layers of Semantic Parsing Based Question Answering. The Question Answering Systems implement similar tasks to translate a user defined natural language question to its formal representation.

manually and there is no automatic way to compose QA pipelines automatically. Recent empirical studies have revealed that albeit overall effective, the performance of QA systems and QA components depends heavily on the features of input questions such as question length, POS tags, question head word etc. [14, 15], and not even the combination of the best performing QA systems or individual QA components retrieves complete and correct answers [16]. Therefore, in order to advance the state of the art, and explore future research directions, it is important to combine QA components into a QA framework based on the strengths and weaknesses of the range of existing QA components.

1.1 Motivation, Problem Statement, and Challenges

The necessity of this research study emerged from our observations we have made on more than 60 QA systems and several other independent QA components which have been published until now. In fact, a great number of independent components perform QA tasks – either as part of QA systems or standalone. Figure 1.2 presents several QA components, implementing the QA tasks NED (Named Entity Disambiguation) implemented by (i) DBpedia Spotlight [17], (ii) Aylien API[5], and (iii) Tag Me API [18]), RL (Relation Linking) implemented by (i) ReMatch [19] and (ii) RelMatch [13]), and QB (Query Building) implemented by (i) SINA [20] and (ii) NLIWOD QB[6]).

Among these components, DBpedia Spotlight, ReMatch, and NLIWOD QB achieve the best performance for the tasks NED, RL, and QB, respectively [21]. When QA components are integrated into a QA pipeline, the overall performance of the pipeline depends on the individual performance of each component. The fact that a particular component gives superior performance for a task on a given set of questions does not imply that the component is superior for all types of questions. That is, it may be a

[5]http://docs.aylien.com/docs/introduction

[6]Component is based on https://github.com/dice-group/NLIWOD and [8].

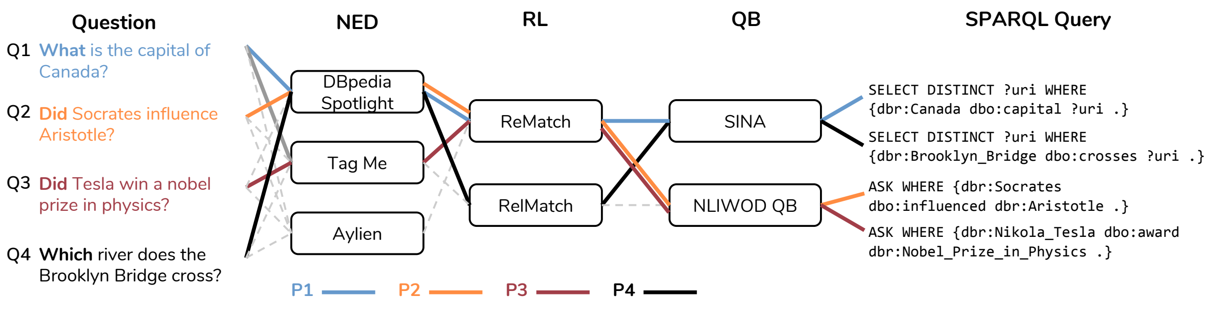

Question
Q1 What is the capital of Canada?
Q2 Did Socrates influence Aristotle?
Q3 Did Tesla win a nobel prize in physics?
Q4 Which river does the Brooklyn Bridge cross?
NED
DBpedia Spotlight
Tag Me
Aylien
RL
ReMatch
RelMatch
QB
SINA
NLIWOD QB
SPARQL Query
SELECT DISTINCT ?uri WHERE {dbr:Canada dbo:capital ?uri .}
SELECT DISTINCT ?uri WHERE {dbr:Brooklyn_Bridge dbo:crosses ?uri .}
ASK WHERE {dbr:Socrates dbo:influenced dbr:Aristotle .}
ASK WHERE {dbr:Nikola_Tesla dbo:award dbr:Nobel_Prize_in_Physics .}
P1
P2
P3
P4

Figure 1.2: Four natural language questions answered successfully by different pipelines composed of three NED, two RL, and two QB components. The optimal pipelines for each question are highlighted.

case that the performance of components varies depending on the type of question with varying number of words, number of POS tags etc.

For example, Figure 1.2 illustrates the best performing QA pipelines for four exemplary input questions. We observe that Pipeline P1 is the most efficient for answering Question Q_1: *"What is the capital of Canada?"* but it fails to answer Question Q_4: *"Which river does the Brooklyn Bridge cross?"*. This is caused by the fact that the RL component ReMatch in Pipeline P1 does not correctly map the relation `dbo:crosses` in Q_4 for the input keyword *"cross"*, while RelMatch maps this relation correctly. Although the overall precision of ReMatch on QALD-5 is higher than that of RelMatch, for Q_4, the performance of RelMatch is higher. Similarly, for Question Q_2 *"Did Socrates influence Aristotle?"* Pipeline P2 delivers the desired answer, while it fails to answer the similar question Q_3 *"Did Tesla win a nobel prize in physics?"*. Although questions Q_2 and Q_3 have a similar structure (i.e., Boolean answer type), DBpedia Spotlight NED succeeds for Q_2, but on Q_3 it fails to disambiguate the resource `dbr:Nobel_Prize_in_Physics`. At the same time, Tag Me can accomplish the NED task successfully. Although, the optimal pipeline for a given question can be identified experimentally by executing all possible pipelines, this approach is costly and even practically impossible, since covering all potential input questions is not feasible. Therefore, a heuristic approach to identify an optimal pipeline for a given input question is required.

Before aiming for an optimal pipeline for a given question, several other challenges need to be addressed. For example, components that are part of motivating example illustrated in Figure 1.1 are heterogeneous and have different interoperability issues such as heterogeneity at programming language, input/output requirements. Second, there is no systematic way to integrate these components into a single platform. However, if components from existing QA systems implementing subsequent steps of a QA pipeline are reused and integrated into a single architecture, it will result in a new question answering system. Several QA systems have been developed recently in the research community, for example, [20, 22–24]. While many of these systems 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 (for details, please refer to Chapter 3):

- **Potential of reusing the available components is very weak.** In spite of several overlapping QA tasks, reusability for further research is limited and remains an open challenge because of their focus on specific technologies, applications or datasets. As a result, creating new QA systems is currently still cumbersome and inefficient and needs to start from scratch. Particularly, the research community is not empowered to focus on improving particular components of the QA process, as developing a new question answering system and integrating a component is extremely resource-consuming.

- **The existing attempts for promoting reusability in QA systems lacks scalability.** Some first steps for developing flexible, modular question answering systems have started to address this challenge, e.g., [11, 12]. These frameworks follow a tightly coupled approach at the implementation level for reusing QA components. Therefore, existing QA frameworks do not tackle scalability of QA components within the framework and QA pipelines have to be composed manually.

- **Interoperability between the employed components is not systematically defined.** The existing QA frameworks 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 example, openQA [11] expects each component to be implemented in Java, whereas OKBQA [13] has strict input/output data format requirements.

- **Missing heuristic approach for selecting best component based on input question.** We have observed in our motivating example (Figure 1.2) that QA components exhibit different behaviour based on different types of question. There are many independent components implementing one QA task. Hence, assuming if all these components are integrated into a framework/platform, it is challenging to choose the best QA pipeline given all the viable possible combinations with other QA components implementing different QA tasks. Current frameworks lack a heuristic approach for selecting the best components per QA task as these frameworks have not considered the scalability of QA components within the framework.

- **Missing Semantics of QA components.** Existing QA frameworks lack an automatic way of composing QA pipelines. Currently, the user is expected to select the component manually from the state-of-the-art QA frameworks. Also, there is no way to semantically describe a QA component based on the input/output requirement of the component and the associated QA task. Due to the lack of semantic description, it is difficult for a QA system developer to choose a component and integrate it in the QA pipeline. QA systems developer is expected to understand the internal working of the component (i.e. the task it performs, the required input etc.) manually. There is no automatic process to compose QA pipelines on demand. Considering the observed shortcomings, the main research problem this thesis tackles is formulated as:

Research Problem Definition

How can existing components for question answering tasks be reused to build effective and seamless dynamic question answering pipelines?

1.1.1 Challenges for Building Effective Dynamic QA Pipelines

Based on the motivating example in the previous section, we identify four core challenges to address formulated research question. Each challenge correspond to a sub research question.

Challenge 1: Heterogeneity of Existing QA Components and Systems

Most of the state-of-the-art QA systems and components are developed in a span of the last ten years by different researchers [10]. These QA approaches have heterogeneity at different levels such as programming language, input/output format, data exchange within a QA system, architecture etc. Therefore, while aiming to reuse the existing QA components, the first challenge is to address the heterogeneity of these tools/components at different levels of granularity and make them interoperable.

Challenge 2: Reusability of QA Components to Build QA Systems

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 (i.e. pipeline). 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 standardised interfaces and are often not open source or available as Web services. Therefore, it is very challenging to reuse them easily and limited reusability constitutes towards the second challenge.

Challenge 3: Automatic Composition of QA Pipeline

Since QA process involves a vast number of (partially overlapping) subtasks, existing QA components can be combined in various ways to build tailored QA systems that perform better in terms of scalability and accuracy in specific domains and use cases. However, to the best of our knowledge, no systematic way exists to formally describe and automatically compose such components to build on-demand QA systems. With the growing number of QA components for a specific task, and aiming towards their integration in a single platform, it is challenging to foresee QA component composition in a QA pipeline manually. In other words, if there are many components available for each task, the challenge here is how to combine a given QA component with components performing other QA tasks, respecting high-level input/output dependencies.

Consider DBpedia Spotlight NED [17] which can perform disambiguation task in a question answering pipeline. It requires just the natural language question as an input, and provides an output as DBpedia URLs of entities present in the question. AGDISTIS [25] is an another component that performs named entity disambiguation task. However, the required input for AGDISTIS is the input question, and recognised spots of entities present in the question. These two tools have completely different input requirements, yet perform the same QA tasks. To utilise (reuse) either of these tools in a QA pipeline, the developer needs to understand the functionality of the tool, specific input/output requirements. In a real-world scenario, it is not expected from a QA developer to first learn about specific input/output of each component. Therefore, with the availability of a large number of QA components in a platform, manual composition of QA pipelines respecting such dependencies is cumbersome and constitutes the third challenge for defined research objective.

Challenge 4: Scalability of QA Components in a Framework

With a vision to integrate existing QA components in a single platform to build effective QA pipelines (i.e. systems), it is important to consider the scalability. For example, let us assume we have 10 components available for named entity disambiguation, five for relation linking, and two for query building task in a single platform. One option is to run all the possible viable combinations of the components to extract answer (in this case 10X5X2). Therefore, the challenge here is to select the best performing component per task for each input question based on the strengths and weaknesses of the QA components.

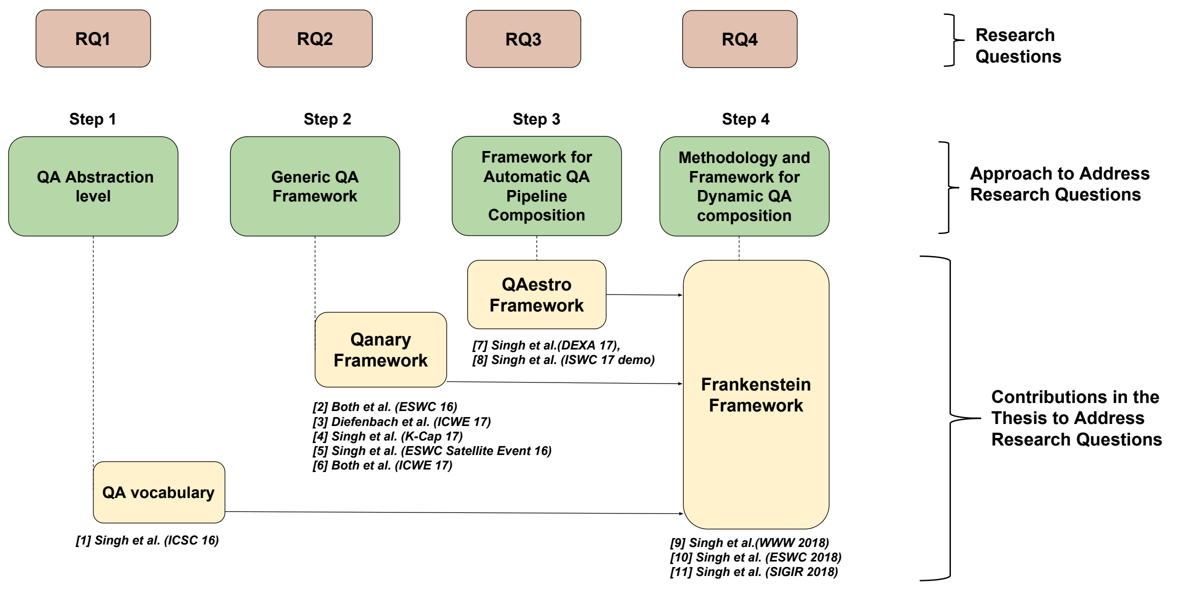

RQ1
RQ2
RQ3
RQ4
Research Questions
Step 1
Step 2
Step 3
Step 4
QA Abstraction level
Generic QA Framework
Framework for Automatic QA Pipeline Composition
Methodology and Framework for Dynamic QA composition
Approach to Address Research Questions
QAestro Framework
[7] Singh et al.(DEXA 17),
[8] Singh et al. (ISWC 17 demo)
Qanary Framework
[2] Both et al. (ESWC 16)
[3] Diefenbach et al. (ICWE 17)
[4] Singh et al. (K-Cap 17)
[5] Singh et al. (ESWC Satellite Event 16)
[6] Both et al. (ICWE 17)
Frankenstein Framework
Contributions in the Thesis to Address Research Questions
QA vocabulary
[1] Singh et al. (ICSC 16)
[9] Singh et al.(WWW 2018)
[10] Singh et al. (ESWC 2018)
[11] Singh et al. (SIGIR 2018)

Figure 1.3: Approach for addressing the main research problem comprises four steps. Each Step addresses individual challenges of the overall approach, and is supported by research publications.

Recent empirical studies have described that the performance of QA systems and QA components depends heavily on the features/type (such as question length, POS tags etc.) of input questions [14, 15]. This is because it may be a case that one NED component can effectively identify and disambiguate particular types of entities (for e.g. entities written in lower case in input question), may fail for another type of entities (for e.g. entities with upper case characters). Similarly, for other tasks, a component's performance may also vary based on the type of input questions. Therefore, it is not a wise idea to use same components per task for each input question. Assuming more and more components are added to a single platform, scalability becomes a key issue.

1.1.2 Approach

The approach that aims to address the four identified challenges has multiple stages as shown in Figure 1.3. The first step comprises the creation of an abstract level on top of existing QA systems and components. The existing state-of-the-art QA systems differ at multiple levels of granularity of their interoperability such as data exchange format, architecture, programming language, input/output format of intermediate steps etc. Hence the first stage (and *Challenge 1*) consists of modelling and conceptualising QA systems to make them interoperable.

The second level incorporates the methodology and framework to integrate the existing QA components in a single platform overcoming their heterogeneity to build reusable QA systems. The task of building such methodology and framework constitutes the second stage (and *Challenge 2*) of the proposed approach. The third step presents a way to assist the QA system developer to compose effective QA pipelines. With the possibility of increasing scalability of QA components, and foreseeing their integration in a single platform, it is quite challenging to manually combine the components to form QA pipelines. In turn, the third stage (*Challenge 3*) implies an effective way to compose QA pipelines. Once, the approach tackles the problem of QA pipeline composition in the previous step, the last stage envisages a dynamic composition of QA pipelines based on the type of question. In other words, the challenge

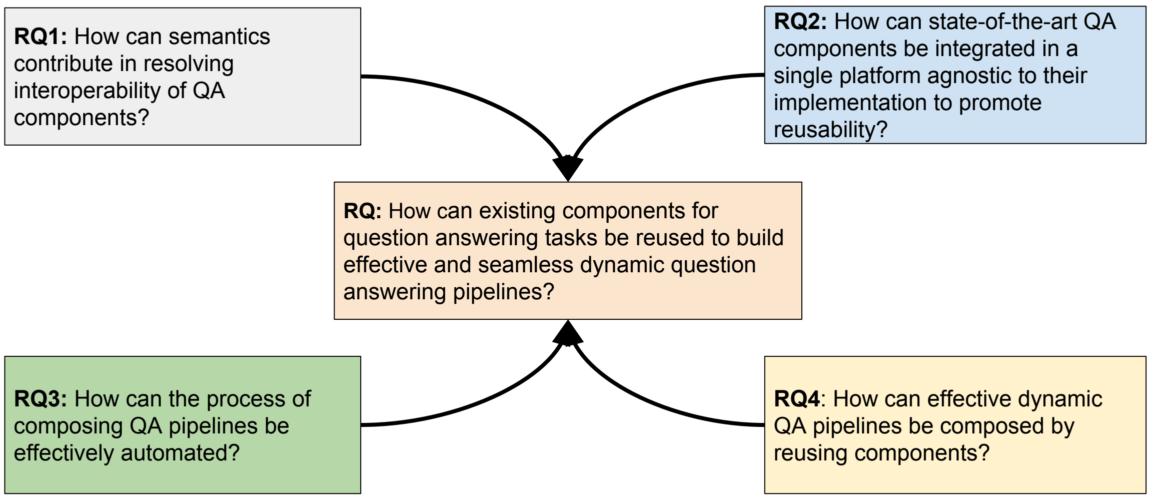

RQ1: How can semantics contribute in resolving interoperability of QA components?
RQ2: How can state-of-the-art QA components be integrated in a single platform agnostic to their implementation to promote reusability?
RQ: How can existing components for question answering tasks be reused to build effective and seamless dynamic question answering pipelines?
RQ3: How can the process of composing QA pipelines be effectively automated?
RQ4: How can effective dynamic QA pipelines be composed by reusing components?

Figure 1.4: Four sub research questions contribute to the overall research objective of the thesis

(*Challenge 4*) here is to select the best component per task for a given input question from the plethora of QA components integrated into the single platform.

1.2 Research Questions

Based on the revealed challenges we devise the following research questions to be addressed in the thesis. Each challenge is mapped to one sub-research question and collectively contributes towards the overall research question of the thesis as illustrated in the Figure 1.4.

Research Question 1 (RQ1)

How can semantics contribute in establishing interoperability of QA components?

The Web of Data has attracted the attention of the question answering community and recently, a number of schema-aware question answering systems have been introduced. Much research has been done w.r.t. specific QA applications, showing clearly that the problem is very complex from a scientific as well as technical point of view. While research achievements are individually significant yet, integrating different approaches is not possible due to the lack of a systematic approach for conceptually describing QA systems and tackle their heterogeneity at different dimensions (e.g. input/output requirements, programming language etc.). To address this research problem, we analyse the challenges for making existing QA systems and components interoperable. We study the problems that hinder the interoperability of QA systems. We then analyse the need of a generic approach to model and conceptualise QA systems and components. 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.

Research Question 2 (RQ2)

How can QA components be integrated in a single platform agnostic to their implementation to promote reusability?

QA systems are very complex and existing 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. In this question, we study a mechanism to promote reusability of QA components to build new QA systems instead of building a complete QA system from scratch. We define a methodology to integrate heterogeneous QA components in a single platform which is agnostic to implementation details of QA components. Additionally, we address the heterogeneity of existing QA components at different levels of granularity to integrate them within a single platform.

Research Question 3 (RQ3)

How can the process of composing QA pipelines be effectively automated?

An effective way to compose QA pipelines in an automatic manner is investigated in the third research question. Due to the increasing number of QA systems and components, question answering involves several tasks and subtasks, common in many systems. Existing components can be combined in various ways to build the tailored question answering pipelines. However, manual compositions of such pipelines are cumbersome and time-consuming. When we are aiming for integrating several QA components in a single framework, the problem may arise when many components are present for each task. We thus overview existing pitfalls of the manual composition of QA pipelines and devise an approach for automatic composition of QA pipelines in a seamless manner. This automates the process of combining QA components with minimal manual effort respecting input and output requirement of each component.

Research Question 4 (RQ4)

How can effective dynamic QA pipelines be composed by reusing components?

We then delve into the possibility and methodology for composing dynamic question answering pipelines. We consider the scalability of QA components and devise an approach for composing QA pipelines based on the type of question and call it dynamic QA pipeline.

1.3 Thesis Overview

To present a high-level but descriptive overview of the achieved results during the course of conducted research, this section emphasises the main contributions of the thesis and provides references to scientific articles covering these contributions published throughout the whole term.

1.3.1 Contributions

Contributions for RQ1

Vocabularies for promoting the interoperability of question answering systems.

To address the first research question, we present two generic vocabularies built upon an abstract level of existing QA systems. We initiate a step towards an interoperable approach that will be used to build systems which follow a philosophy of being actually open for extensions. Firstly, we present a QAV vocabulary [9] to semantically define the QA components and systems. This vocabulary helps us to define a component based on the task it performs, its input and output requirements on a higher level. This provides a clear picture of the component. We then collect and generalise the necessitated requirements from implementing the state-of-the-art QA systems. We model the conceptual view of QA systems using and extending the Web Annotation Data Model[7] while thereafter we show how these requirements are fulfilled while using the Web Annotation Data Model. This model empowers us for designing a knowledge-driven approach for QA systems and deals with the heterogeneity of existing question answering approaches. This resulted into a vocabulary which is concluded from conceptual views of different question answering systems. We call this ontology `qa` vocabulary [26]. In this way by proposing QAV and `qa` vocabularies, we are enabling researchers to implement knowledge-driven QA systems and to reuse and extend different approaches without interoperability and extension concerns.

Contributions for RQ2

A framework for knowledge-driven open question answering systems.

Establishing a QA system is time-consuming. One main reason is the involved fields, as solving a Question Answering task, i.e., answering a user's question with the correct fact(s), might require functionalities from different fields like information retrieval, natural language processing, and linked data. Therefore, it is cumbersome and inefficient to design and implement new or improved approaches, in particular as many components lack reusability and extensibility. 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 (i.e. `qa`) that is aligned to existing, powerful ontologies provided by domain-specific communities. We achieve this by a methodology for binding existing vocabularies to

[7] https://www.w3.org/TR/annotation-model/

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. We name the proposed methodology Qanary [27]. Qanary methodology is the first approach to open QA systems that are agnostic to implementation details and that inherently follow the Linked Data principles. The `qa` vocabulary is the foundation for Qanary methodology for implementing the QA processes.

Qanary Ecosystem is the implementation of the Qanary methodology where all knowledge related to questions, answers and intermediate results is stored in a central local 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 ecosystem, the components all implement the same service interface. Therefore, all components can be integrated into a QA system without manual engineering effort. Using its 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 organised 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 starts, 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. The `qa` vocabulary and Qanary methodology act as the foundation for the Qanary ecosystem [27–29] which is the framework consisting of components and web services integrated in a single platform using Qanary methodology. Here, our main contribution is a component-based architecture enabling developers to create or re-combine components following a plug-and-play approach. While aiming at an optimal system w.r.t. a given use case, (scientific) developers are enabled to rapidly create new/adapted QA systems from the set of Qanary components available. Hence, our component-based architecture enabling developers to create or recombine components following a plug-and-play approach. While aiming at an optimal system w.r.t. a given use case, (scientific) developers are enabled to rapidly create new/adapted QA systems from the set of Qanary components available.

Besides the methodology, and framework for creating QA systems, we also contribute an approach for creating relation linking components. The research community has developed many components for named entity recognition and disambiguation task, but little work has been done in the direction of independent relation linking components. To scale up the number of components in Qanary ecosystem, we developed an approach for creating relation linking component reusing the large corpus of natural language relational patterns.

Contributions for RQ3

A framework for semantic-based composition of question answering pipelines.

Examining reusability of QA components and systems, we took a detailed look at their implementation. Despite different architectural components and techniques used by the various QA systems, these systems have several high-level functions and tasks in common. However, to the best of our knowledge, no systematic way exists to formally describe and automatically compose QA pipelines from such components. Thus, we introduce *Qaestro*, a framework for semantically describing both QA components and developer requirements for QA component composition. *Qaestro* relies on a controlled vocabulary and the Local-as-View (LAV) approach to model QA tasks and components, respectively. Furthermore, the problem of QA component composition is mapped to the problem of LAV query rewriting [30], and state-of-the-art SAT solvers [31] are utilised to efficiently enumerate the solutions. We have formalised 51 existing QA components implemented in 20 QA systems using *Qaestro*. Our empirical results suggest that *Qaestro* enumerates the combinations of QA components that effectively implement QA developer requirements to compose on demand QA pipelines.

Contributions for RQ4

Methodology and framework for composing effective dynamic QA pipelines.

We have observed in motivating example that modern question answering (QA) systems need to flexibly integrate a number of components specialised to fulfil specific tasks in a QA pipeline. Since a number of different software components exist that implement different strategies for each of these tasks, it is a major challenge to select and combine the most suitable components into a QA system, given the characteristics of a question. We study this optimisation problem and train classifiers, which take features of a question as input and have the goal of optimising the selection of QA components based on those features. We then devise a greedy algorithm to identify the pipelines that include the suitable components and can effectively answer the given question. We implement this model within FRANKENSTEIN, a QA framework able to select QA components and compose QA pipelines. We evaluate the effectiveness of the pipelines generated by Frankenstein using question answering benchmarks. These results not only suggest that FRANKENSTEIN precisely solves the QA optimisation problem but also enables the automatic composition of optimised QA pipelines, which outperform the static Baseline QA pipeline. FRANKENSTEIN uses Qanary methodology to integrate QA components in its architecture. The modular architecture of FRANKENSTEIN allows developers to add more components to this platform just by following simple configuration steps. Overall, FRANKENSTEIN promotes reusability of components and tools performing different QA tasks by integrating them into a single platform. Question Answering is a domain which is driven by different fields, consequently, it requires a collaborative approach to achieve significant progress. Hence, by reusing infrastructure and tools provided by FRANKENSTEIN, researchers can build QA systems in collaboration with a focus on individual stages of QA tasks, and reuse components for other tasks from the FRANKENSTEIN.

1.3.2 Publications

The following list of publications contribute a scientific basis of this thesis and acts as a reference point for numerous figures, tables and ideas presented in the later chapters. Please note that the co-authors in the papers are either Professors, post-docs, or masters student. For the papers co-authored with other PhD student, individual contribution is clearly mentioned. Therefore, parts of the contributions of this dissertation which is mentioned below were achieved as the result of effective teamwork. The author (Kuldeep Singh) will use the “we” pronoun throughout this dissertation, but all of the contributions and materials presented in this work, except the below mentioned collaborative works with an another PhD student, originated from the work of the author solely by himself.

- *Conference Papers (peer reviewed)*

  1. **K Singh**, AS Radhakrishna, A Both, S Shekarpour, I Lytra, R Usbeck, A Vyas, A Khikmatullaev, D Punjani, C Lange, ME Vidal, J Lehmann, S Auer. *Why Reinvent the Wheel- Lets Build Question Answering Systems Together.* In Proceedings of the Web Conference (formerly known as WWW), 2018, ACM;
  2. **K Singh**, A Both, AS Radhakrishna, S Shekarpour. *Frankenstein: a Platform Enabling Reuse of Question Answering Components.* In Proceedings of the 15th Extended Semantic Web Conference (ESWC), 2018, Springer;
  3. **K Singh**, IO Mulang, I Lytra, MY Jaradeh, A Sakor, ME Vidal, C Lange, S Auer. *Capturing Knowledge in Semantically-typed Relational Patterns to Enhance Relation Linking.* In Proceedings of the Knowledge Capture Conference (K-Cap), 2017, ACM;
  4. **K Singh**, I Lytra, ME Vidal, D Punjani, H Thakkar, C Lange, S Auer. *Qaestro -Semantic-based Composition of Question Answering Pipelines.* In Proceedings of 28th International Conference on Database and Expert Systems Applications (DEXA), 2017, Springer;
  5. D Diefenbach, **K Singh**, A Both, D Cherix, C Lange, S Auer. *The Qanary ecosystem: getting new insights by composing Question Answering pipelines.* In Proceedings of the 17th International Conference on Web Engineering (ICWE), 2017, Springer; The work was done jointly with PhD student Dennis Diefenbach (Universite Jean Monnet, France). In this paper, my contributions include designing and implementing the integration of various QA components in the core Qanary architecture. I have also contributed in designing the core Qanary Ecosystem using Springboot framework.
  6. A Both, D Diefenbach, **K Singh**, S Shekarpour, D Cherix, C Lange. *Qanary -a methodology for vocabulary-driven open question answering system.* In Proceedings of the 13th Extended Semantic Web Conference (ESWC), 2016, Springer; This work was predecessor work for Qanary ecosystem jointly done with Dennis Diefenbach (Universite Jean Monnet, France). I contributed in designing the fundamentals of the Qanary methodology, and implementing the core QA pipeline architecture for the evaluation of the proposed methodology.
  7. **K Singh**, A Both, D Diefenbach, S Shekarpour. *Towards a message-driven vocabulary for promoting the interoperability of question answering system.* In Proceedings of the 10th International Conference on Semantic Computing (ICSC), 2016, IEEE; The foundational step for Qanary is the `qa` vocabulary proposed in this paper, jointly done with Dennis Diefenbach (Universite Jean Monnet, France). My contributions in this paper was to collect all the requirements for designing open and scalable vocabulary by reviewing state of the art QA systems, and then jointly designing the concrete requirements wrt. Web Annotation Data model for conceptualising the QA systems.

- *Demo Papers (peer reviewed)*

  8. **K Singh**, I Lytra, A Sethupat, A Vyas, ME Vidal. *Dynamic Composition of Question Answering Pipelines With Frankenstein.* In Proceedings of the 41st International ACM SIGIR conference on research and development in Information Retrieval (SIGIR), 2018, ACM;
  9. A Both, **K Singh**, D Diefenbach, I Lytra. *Rapid Engineering of QA Systems Using the Light-Weight Qanary Architecture.* In Proceedings of the 17th International Conference on Web Engineering (ICWE), 2017.

10. **K Singh**, I Lytra, K Abhinav, ME Vidal. *Qaestro Framework- Semantic Composition of QA Pipelines.* In Posters and Demo Track, 16th International Semantic Web Conference (ISWC) 2017. CEUR Workshop Proceedings.
11. **K Singh**, A Both, D Diefenbach, S Shekarpour. *Qanary–the Fast Track to Creating a Question Answering System with Linked Data Technology.* In Posters and Demo Track at the 13th Extended Semantic Web Conference (ESWC), 2016.

- *Workshop Articles (peer reviewed)*

12. S Shekarpour, KM Endris, A Jaya Kumar, D Lukovnikov, **K Singh**, H Thakkar, and C Lange. *Question answering on linked data: Challenges and future directions.* In Proceedings of the 25th International Conference Companion on World Wide Web (WWW Companion). 2016.

- *Miscellaneous Papers (peer reviewed)*

  Following publications originated during and are related to this thesis but are not part of the thesis itself.

13. IO Mulang, **K Singh**, F Orlandi. *Matching Natural Language Relations to Knowledge Graph Properties for Question Answering.* In Proceedings of the Semantics, ACM, 2017.
14. **K Singh**, I Lytra, AS Radhakrishna, S Shekarpour, ME Vidal, J Lehmann. *No one is Perfect- Analysing the Performance of Question Answering Components over the DBpedia Knowledge Graph.* Submitted to Information Processing and Management Journal, Elsevier.

The full list of publications completed during the PhD term is available in Appendix A.

1.4 Thesis Structure

The thesis is structured into eight chapters. Chapter 1 introduces the thesis covering the main research problem, the motivation for the conducted study, research questions, scientific contributions that address research questions, and a list of published scientific papers that formally describe those contributions. Chapter 2 presents fundamental concepts and background in the fields of Semantic Web, Linked Data, and Question Answering for a holistic overview of the research problem. Chapter 3 describes state-of-the-art efforts in the domain of question answering. We describe QA components, systems and frameworks to provide a detailed understanding of their limitation, and gaps we identified in this thesis.

In Chapter 4 we describe two vocabularies to 1) conceptualise existing QA systems and components 2) capture knowledge generated in a QA process. These two vocabularies are foundations of our approach defined in Chapter 5 for automatic composition of QA pipelines, and solving interoperability issues of QA components. Therefore, in Chapter 5, we report the efforts aimed at the first step towards integrating existing QA components in a single platform. We describe Qanary, a methodology for creating question answering systems using Linked Data technologies. We also describe Qanary Ecosystem, which is a framework built using Qanary methodology. We show how independent QA components can be integrated into Qanary Ecosystem, and benchmarked to evaluate their performance. In next Chapter, We devise an approach for creating relation linking components by reusing a large corpus of natural language relational patterns and their corresponding DBpedia predicates. Chapter 6 describes this approach for capturing information in large knowledge sources such as PATTY and then utilise this knowledge to build an independent relation linking tool. We detailed *Qaestro* framework in Chapter 7 able to deal with the QA pipeline composition problem by casting it to the query rewriting problem and leveraging state-of-the-art SAT solvers. *Qaestro* helps QA developers to semantically describe QA components and developer requirements based on these semantic descriptions; a controlled vocabulary is utilised to model QA tasks and is exploited in the description of the QA components.

In Chapter 8, we present our approach for dynamic composition of QA pipelines considering the type of the question. We implement this approach in a framework known as FRANKENSTEIN. We describe FRANKENSTEIN and its architecture with a large scale evaluation of its 29 components. Finally, Chapter 9 concludes the thesis with directions of future work. We once more look through the research questions and answer them based on the obtained results.

CHAPTER 2

Background

The research problem of creating effective dynamic question answering pipelines by reusing existing QA components and systems defined in Chapter 1 require a comprehensive approach from different perspectives. The principles and concepts presented in this chapter lay the foundations for addressing posed challenges. Figure 2.1 illustrates basic building blocks for the defined research problem. Resource Description Framework (RDF) defined in Section 2.1.1 acts as a foundation for defining data in machine-readable format. We leverage properties and characteristics of RDF in answering research question **RQ1** and **RQ2**. Knowledge Graphs defined in Section 2.2 act as a rich source of structured information which a user may be interested in. Therefore, leveraging the strengths of Knowledge Graphs with SPARQL query processing on top to extract right information act as foundations for question answering process. Therefore, we leverage Knowledge Graph and SPARQL capabilities in **RQ2** and **RQ4**. Question Answering Tasks defined in Section 2.3 act as a conceptual representation of existing state-of-the-art QA components and systems. QA tasks define the foundations for **RQ3** and **RQ4**.

2.1 Semantic Web & Linked Data

On the Web, the documents and files can be identified by Uniform Resource Locators (URIs) and are accessible via the HyperText Transfer Protocol (HTTP). Berners-Lee et al. [32] proposed the idea of Semantic Web in 2001 to allow machines to understand the context of data. Semantic web describes resources (things) in a machine-readable format which are real-world entities such as "cars" but also define abstract concepts such as "transportation". Semantic web is an extension of existing Web with adding meaning to the information to make the data more accessible and structured. Extensible Markup Language (XML) [33] and its descendants as Turtle [1], N-Triples [2], N3 [3], TriG [4] are important technologies for developing Semantic Web. XML is W3C specification[5] that focuses on simplicity, generality, and usability over the Internet for textual data. Resource Description Framework (RDF) is the core of Semantic Web data representation. Meaning of data is expressed in RDF that uses a triple model `<subject verb object>` to provide a formal resource description. In RDF, a document makes statement about a resource (e.g. Barack Obama) has properties (e.g. wife of, president of) with certain values (another resource such as Michelle Obama, or United States of America) [32]. Standardised

[1]Turtle Specification https://www.w3.org/TR/turtle/
[2]N-Triples Specification https://www.w3.org/TR/n-triples/
[3]N3 Specification https://www.w3.org/TeamSubmission/n3/
[4]TriG Specification https://www.w3.org/TR/trig/
[5]https://www.w3.org/TR/REC-xml/

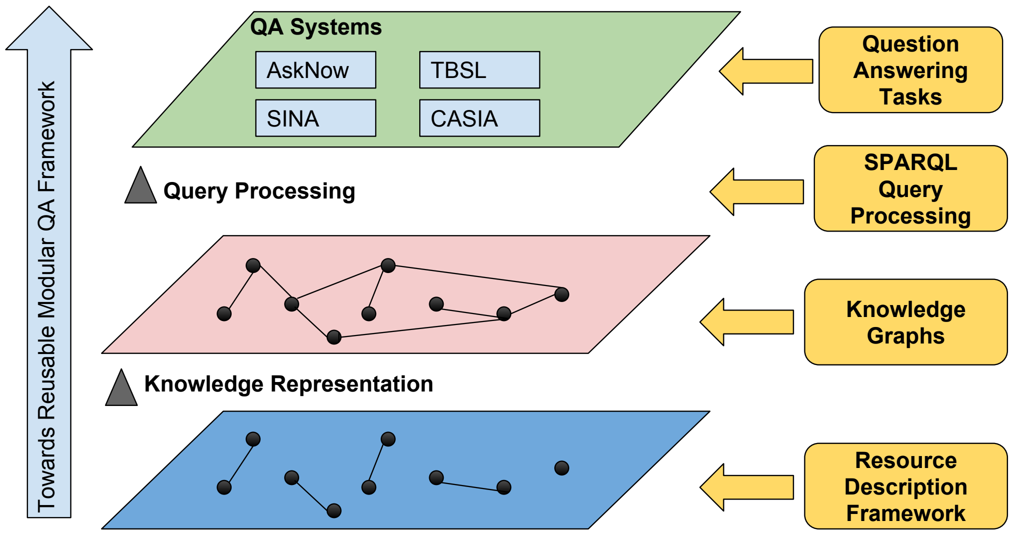


Figure 2.1: **Relevant foundations for the defined research problem.** Resource Description Framework lay foundations of knowledge graph creation and provides a machine-understandable knowledge representation. Knowledge Graphs act as a framework for uniform knowledge representation. SPARQL Query Processing is responsible for extraction of information from Knowledge Graphs. Question Answering tasks represent the existing QA systems on conceptual level.

vocabulary is used to achieve the formal semantics in RDF. These resources and concepts are represented by unique URIs to give better representation, uniqueness, and accessibility.

Furthermore, the main advantage of Semantic Web is its powerful structured representation for data consumption and publishing. Tim Berners-Lee proposed five Star deployment scheme[6] for open data as illustrated in Figure 2.2. In these five levels of data deployment scheme, Linked open Data (LOD) constitutes towards highest order of this deployment scheme. To promote reusability and add semantics in linked data, there are four principles proposed by Tim Berners-Lee to adhere to [34]:

- To use URIs as names of the things (i.e. resources);
- To use HTTP URIs for dereferencing such that user can look up for these names easily;
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL);
- Interlink URIs so that people can discover more things.

The core message behind these principles is to promote openness and interlinking of information over Web. Following these principles, more and more data providers adapted it, which lead to the development of *Linked Open Data Cloud* (LOD Cloud) [35], a network of interconnected 5 star datasets. Some of the examples include DBpedia [4] – a structured version of Wikipedia[7]; Wikidata [6] – an uniform source for Wikipedia articles; LinkedGeoData [36] – a collection of geospatial RDF data from OpenStreetMap[8].

[6]http://5stardata.info/en/
[7]http://www.wikipedia.org
[8]https://www.openstreetmap.org/

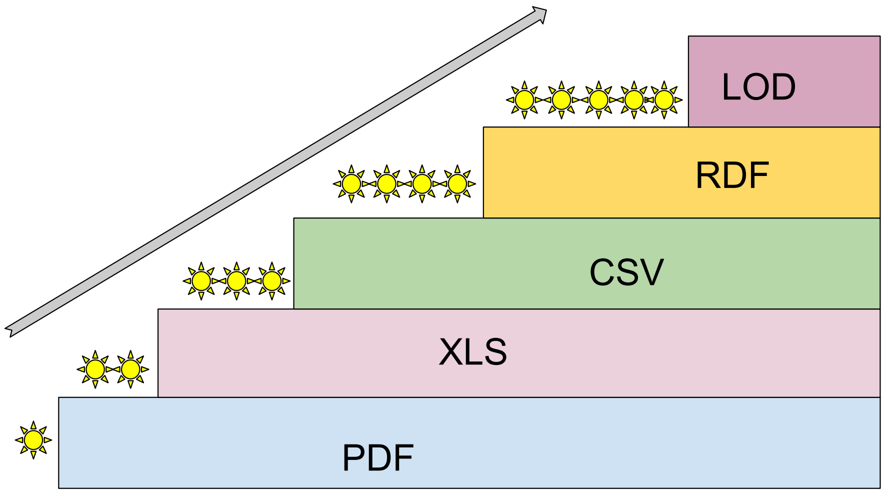


Figure 2.2: 5 Star Deployment Scheme. PDF, XSL, CSV, RDF, and Linked Open Data (LOD) represent five levels of data deployment schemes in increasing order of openness.

2.1.1 The Resource Description Framework (RDF) and Web Ontology Language (OWL)

The Resource Description Framework (RDF) is a W3C standard [37] that is originally designed as meta data model. It has been widely used for conceptual description or modelling of information using variety of syntax, notations, and web standards. RDF data models is similar to classical conceptual data modelling approaches such as entity relationships or class diagrams and information is represented as *triple* which can be described as follows:

- A triple consists of subject, object and verb to define a sentence. Consider the sentence "Michelle Obama is the spouse of Barack Obama", this sentence can be represented in RDF w.r.t DBpedia knowledge base as: `<dbr:Michelle_Obama, dbo:spouse, dbr:Barack_Obama>`.
- Subject denotes a resource to which verb and object belong; subjects are either blank node or URIs in RDF. In our example subject is Michelle Obama represented as `dbr:Michelle_Obama`.
- Verb (or predicate) denotes the relationship between subject and object in in this case, it is the ontology: `dbo:spouse`.
- Object specifies predicate with a particular value; and it can be URIs, blank nodes or string literals. For the given sentence it is: `dbr:Barack_Obama`.

In Semantic Web, vocabularies (or RDF vocabularies) are the set of terms defined using standard formats (e.g. RDF) for further reuse by the users. RDF Schema is used to define RDF vocabularies. RDF Schema (RDFS) is alternatively known as RDF Vocabulary Description Language[9]. To structure the RDF information (such as resources), RDF Schema is used which is a collection of classes and properties providing basic elements for the description of RDF vocabularies (ontologies). Meaning of terms and the relationship between those terms are represented by *Web Ontology Language (OWL)*. This

[9] https://www.w3.org/2001/sw/RDFCore/Schema/200203/

representation of terms is called Ontology [38]. OWL is built on top of RDFS to allow users to define ontologies. Ontologies can be defined as formal definitions of vocabularies that permit users to define complex structures as well as new relationships between the vocabulary terms and between members of the classes present in it. OWL is used to define and illustrate vocabularies and it is also in forms of triples. All the data expressed using ontologies can be stored in special datastores called *RDF triplestores*. RDF Triplestores have following properties[10]:

- RDF triplestores are flexible like NoSQL datastores with no pre-defined schema.
- They are fast and scalable to deal with large amount of data.
- Subjects and objects are stored as nodes, whereas edges are the predicates.
- Data can be interpreted easily.

Ontotext[11], Stardog[12], and rdf4j[13] etc. are few examples of such triplestores. RDF triplestores can be queried using formal query language known as SPARQL.

2.1.2 SPARQL as a Query Language

SPARQL is similar to SQL and used to query RDF data by querying unknown relationships. It can perform a complex join of disparate data in a simple query. SPARQL is a W3C recommendation that uses *triple pattern* as its foundation[14]. *Basic Graph Pattern (BGP)* denotes a set of triple patterns.

Triple Pattern, Basic Graph Pattern [39]

Definition 2.1.1 *Let $\mathcal{U}, \mathcal{B}, \mathcal{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively. Let V be a set of variables such that $V \cap (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) = \emptyset$. A triple pattern tp is member of the set $(\mathcal{U} \cup V) \times (\mathcal{U} \cup V) \times (\mathcal{U} \cup \mathcal{L} \cup V)$. Let $tp_1, tp_2, \ldots, tp_n$ be triple patterns. A Basic Graph Pattern (BGP) B is the conjunction of triple patterns, i.e., $B = tp_1$ AND tp_2 AND $\ldots$ AND tp_n.*

The conjunctions of triple patterns can be extended with filters (`FILTER`), optional patterns (`OPTIONAL`), logical operators (`UNION` and `AND`), aggregate functions in SPARQL. Furthermore, SPARQL defines following four query forms: `SELECT` query returns all, or a subset of the variables bound in query match pattern. `ASK` query returns a boolean value `TRUE` or `FALSE` depending on the query pattern matches with the given BGP or not. `CONSTRUCT` query returns an RDF graph constructed by substituting variables in a set of triple patterns. `DESCRIBE` query result is an RDF graph that describe the resource found. SPARQL expression and `SELECT` query is formally defined as:

[10] RDF Tutorial: https://www.fullstackacademy.com/
[11] https://ontotext.com
[12] https://www.stardog.com/
[13] http://rdf4j.org/
[14] https://www.w3.org/TR/rdf-sparql-query/

SPARQL Expression and SELECT Query [39]

Definition 2.1.2 *Let V be a set of variables. A SPARQL expression is built recursively as follows.*

1. *A tuple from* $(\mathcal{U} \cup V) \times (\mathcal{U} \cup V) \times (\mathcal{U} \cup \mathcal{L} \cup V)$ *is a triple pattern.*
2. *If* Q_1 *and* Q_2 *are graph patterns, then expressions* $(Q_1\ AND\ Q_2)$, $(Q_1\ UNION\ Q_2)$, $(Q_1\ OPT\ Q_2)$ *are graph patterns and SPARQL expressions, i.e., conjunctive graph pattern, union graph pattern, and optional graph pattern, respectively.*
3. *If Q is a SPARQL expression and R is a SPARQL filter condition, then (Q FILTER R) is a SPARQL expression, i.e., filter graph pattern.*

If Q is a SPARQL expression and $S \subset V$ *a finite set of variables, then SPARQL SELECT query is an expression of the form* $SELECT_S(Q)$.

Listing 2.3, Listing 2.2 and Listing 2.3 illustrate example of three different SPARQL queries. First Listing is a SPARQL `SELECT` query that returns answer of *Name the municipality of Roberto Clemente Bridge.* evaluated against DBpedia. Second SPARQL is a `ASK` query which expect boolean answer for the question *Is Nikolai Morozov the former coach of Stanislav Morozov?*:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?uri
WHERE { dbr:Roberto_Clemente_Bridge
dbo:municipality ?uri }
```

Listing 2.1: An Example of SPARQL SELECT query.

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
ASK WHERE { dbr:Stanislav_Morozov
dbo:formerCoach
dbr:Nikolai_Morozov_(figure_skater)> }
```

Listing 2.2: An Example of SPARQL ASK query.

2.2 Knowledge Graph and DBpedia

Google used the term *Knowledge Graph* for the first time in 2012 [40]. Knowledge graph is the structured representation of information collected about the objects in the real world. Objects could be person, car, movie, or any types of other things. Paulheim [41] describe the features of knowledge graphs as follows:

Knowledge Graph ([41])

Definition 2.2.1 *A knowledge graph:*

1. *mainly describes real world entities and their interrelations, organised in a structured graph;*
2. *defines possible classes and relations of entities in a schema;*
3. *allows for potentially interrelating arbitrary entities with each other;*
4. *covers various topical domains.*

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT DISTINCT COUNT(?uri)
WHERE{ ?x dbo:hometown
dbr:India . ?x
dbp:religion ?uri . }
```

Listing 2.3: An Example of SPARQL COUNT query.

Besides Google's knowledge graph, there are many publicly available knowledge graphs such as DBpedia, Wikidata [6], Yago [42] etc. DBpedia is a widely used knowledge graph that uses Wikipedia for extracting information to be represented in RDF format. DBpedia has more than 4 million entities, with over 3 billion RDF triples. Each Web page of Wikipedia is represented as unique URI in DBpedia. For example, consider the city of Delhi in India. DBpedia refer Delhi as `http://dbpedia.org/resource/Delhi` (in short dbr:Delhi). During the extraction process, related information (e.g. infobox fields, categories, and page links) about wiki page of Delhi are also extracted and stored as triples, described in the Listing 2.4. This information is added to the knowledge base as properties of the corresponding URI.

```
dbr:Delhi dbo:isoCodeRegion IN-DL .
dbr:Delhi dbo:part  dbr:List_of_districts_of_Delhi .
dbr:Delhi dbo:leaderName dbr:Alok_Verma .
dbr:Delhi dbp:east dbr:Uttar_Pradesh .
dbr:Delhi dbo:populationTotal "18686902" .
```

Listing 2.4: A snippet of the triples describing `http://dbpedia.org/resource/Delhi`.

The knowledge present in DBpedia can be easily extracted by querying its online endpoints using SPARQL queries. However, DBpedia or other knowledge graphs do not provide an inbuilt easy interface where users can type natural language questions to extract information. For this, question answering systems have been built by the research community to provide an easy to use interface in order to extract information from Knowledge Graphs.

2.3 Question Answering over Knowledge Graphs

The Web of Data is growing permanently as well as the industrial data sets. Induced by this movement the challenge for retrieving knowledge from such data sets has gained much importance in research and industry. Question answering research is tackling this challenge by providing an easy-to-use natural language interface for retrieving knowledge from large data sets. In recent years, QA systems have received much interest, since they manage to provide intuitive interfaces to humans for accessing distributed knowledge – structured, semi-structured, or unstructured – in an efficient and effective way. Since the first attempts to provide natural language interfaces to databases around 1970 [43], an increasing number of QA systems and QA related components have been developed by both industry and the research community [7, 44]. Question answering over unstructured data has been a field of continuous interest for the researchers in the last two decades [45]. In recent years, question answering over structured knowledge bases (e.g., DBpedia [4]) has also gained momentum. Despite different architectural components and techniques used by the various QA systems, these systems have several high-level functions and tasks in common [46].

The Question Answering Tasks Now we formally define a set of necessary QA tasks as $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ such as NED, RL, and QB. Each task ($t_i : q^* \rightarrow q^+$) transforms a given representation q^* of a question q into another representation q^+. For example, NED and RL tasks transform the input representation *"What is the time zone of India?"* into the representation *"What is the* `dbo:timeZone` *of* `dbr:India`*?"*. The entire set of QA components is denoted by $C = \{C_1, C_2, \dots, C_m\}$. It is also possible to have availability of multiple components per QA task. Therefore, while aiming to build QA pipelines reusing existing components, we define two levels of optimisation problems:

1. **Local Optimisation**: With availability of several components for each task, idea is to find best performing component per task rather blindly traversing the space of available components and running each of them. The problem of finding the best performing component for accomplishing the task t_i for an input question q, denoted as $\gamma_q^{t_i}$, is formulated as follows:

$$\gamma_q^{t_i} = \arg\max_{C_j \in C^{t_i}} \{Pr(\rho(C_j)|q)\} \tag{2.1}$$

   Where $Pr(\rho(C_j)|q, t_i)$ is a supervised learning problem to predict the performance of the given component C_j for the given question q; Each component C_j solves one single QA task; $\rho(C_j)$ corresponds to the QA task t_i in $\mathcal{T}$ implemented by C_j. For example, DBpedia Spotlight [17] implements the entity linking QA task, i.e., $\rho(DBpediaSpotlight) = NED$. In this work, we assume a single knowledge graph (i.e., DBpedia); thus, λ is considered a constant parameter that does not have any impact.

2. **Global optimisation:** the problem of finding the best performing pipeline of QA components ψ_q^{goal}, for a question q and a set of QA tasks called *goal*. Formally, this optimisation problem is defined as follows:

$$\psi_q^{goal} = \arg\max_{\eta \in \mathcal{E}(goal)} \{\Omega(\eta, q)\} \tag{2.2}$$

   where $\mathcal{E}(goal)$ represents the set of pipelines of QA components that implement *goal* and $\Omega(\eta, q)$ corresponds to the estimated performance of the pipeline η on the question q.

Figure 2.3 depicts an abstract pipeline of the QA tasks, which receives a question as input and outputs the answers to this question over a knowledge base [46]. We first list different question answering tasks

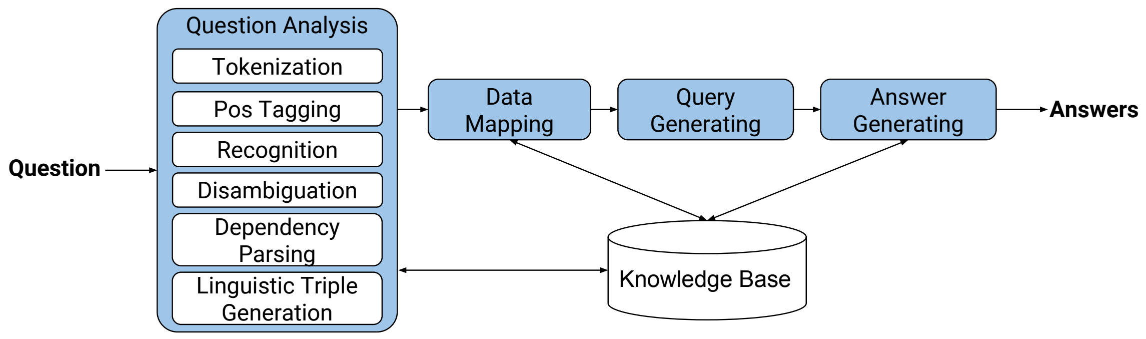


Figure 2.3: **Pipeline of QA Tasks.** A QA pipeline receives a question and outputs the question answers. Question Analysis allows for question linguistic and semantic analysis to identify question features. During Data Mapping, question features are mapped into concepts in a Knowledge Base. A SPARQL query is constructed and executed during Query Generation and Answer Generation respectively.

below and then in the next section we list the components performing different tasks. The below QA task definitions describe the logical structure of an abstract QA pipeline. However, QA systems implement these tasks differently, sometimes combining several of these tasks in a different order or skipping some of the tasks which are often a case in state of the art QA systems.

Question Analysis: Using different techniques, the input question is analysed linguistically to identify syntactic and semantic features. The following techniques form important subtasks:

Tokenisation: A natural language question is fragmented into words, phrases, symbols, or other meaningful units known as tokens. For the question "What is the time zone of India?", tokenisation task will provide keywords "time zone, India" as output.

POS Tagging: The part of speech, such as noun, verb, adjective, and pronoun, of each question word is identified and attached to the word as a tag. The Stanford POS tagger[15] converts "What is the time zone of India?" into *what/WP is/VBZ the/DT time/NN zone/NN of/IN India/NNP ?/.*

Dependency Parsing: An alternative form of syntactic representation of the question to form a tree-like structure is created where arcs (edges in the tree) indicate that there is a grammatical relation between two words, whereas the nodes in the tree are the words (or tokens) in the question. The sample output of such dependency parsing is describe below:

```
(ROOT
  (SBARQ
    (WHNP (WP what))
    (SQ (VBZ is)
      (NP
        (NP (DT the) (NN time) (NN zone))
        (PP (IN of)
          (NP (NNP India)))))
    (. ?)))
```

Listing 2.5: Dependency Parsing of the Question "What is the time zone of India"

[15] http://nlp.stanford.edu:8080/parser/index.jsp

Named Entity Recognition: An input question is parsed to identify the sequence of words that represent a person, a thing, or any other entity. For example question, the entity `India` is recognised by a NER component.

Named Entity Disambiguation: The identity of the entity in the text is retrieved and then linked to its mentions in knowledge graph. An ideal NED component will disambiguate *India* to `dbr:India`.

Linguistic Triple Generation: Based on the input natural language question, triple patterns of the form ⟨*query term*, *relation*, *term*⟩ are generated [46]. A sample triple is `<India,timeZone,?>`.

Data Mapping: Information generated by Query Analyser such as entities and tokens is mapped to its mentions in online knowledge bases such as DBpedia. The triple from triple generator is mapped to `<dbr:India,dbo:timeZone,?>`.

Query Generating: SPARQL queries are constructed; generated queries represent input questions over entities and predicates in online knowledge bases. A corresponding SPARQL query is constructed by a query builder component as:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?uri
WHERE { dbr:India
dbo:timeZone ?uri }
```

Answer Generating: The SPARQL queries are executed on the end points of knowledge bases to obtain the final answer. The expected correct answer of the SPARQL query is `http://dbpedia.org/resource/Indian_Standard_Time`.

Other QA Tasks

- **Question Type Identification:** This task identifies the type of the question. The input is the natural language question; the output is the type of the question, e.g., "yes-no", "location", "person","misc", "time", or "reason". For exemplary question, question type is misc.
- **Answer Type Identification:** This task identifies the desired type of the final answer. In our example, answer type is list. This task is sometimes performed as a part of the Question Analysis task or as a subtask of Answer Generation.
- **Query Ranking:** In some of the QA systems, the task Query Generation generates multiple candidate queries. This task ranks the generated SPARQL queries using a ranking function and it helps to select the best ranked query.

2.4 Local as View and Global as View

In data integration systems, the aim is to integrate data from various sources and provide a uniform view of them. These systems are generally defined as a triple <G,S,M> where G is the global schema, S is the heterogeneous groups of source schemas, and M is the mapping between G and S. The *View-based data integration* frameworks address the problem of data integration by coupling the data sources into a single

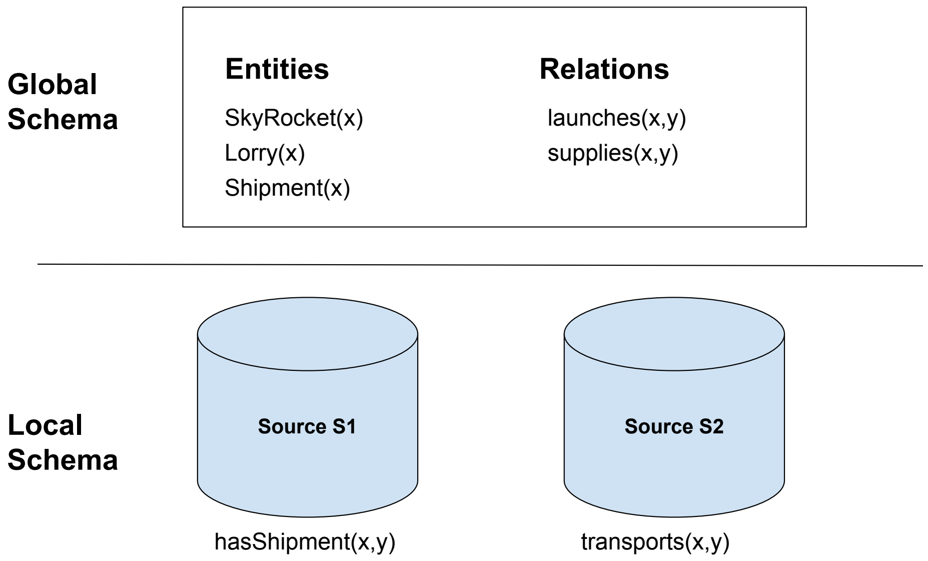


Figure 2.4: **Data Integration Example.** Two different sources S_1 and S_2 that consist of local schemas have to be aligned with the global schema. The global schema describes three entities and two relations between them.

unified view. In a data integration system, the view presented to the user as the *Global Schema*. This view is a unified view of all the heterogeneous data sources. The user runs queries on the Global schema, and system access many local views to combine data from these views to provide a final answer to the user query. There are two basic approaches to provide mapping in a data integration system [47]. These are: Global-As-View (GAV) and Local-As-View(LAV) approaches. The LAV mappings are formally defined as:

Local-as-View (LAV) [47]

Definition 2.4.1 *In a data integration system $\mathcal{IS} = \langle O, S, M \rangle$ based on LAV approach the mapping M associates to each source in S a query q_O in terms of the global schema O:*

$$s \rightarrow q_O$$

That is, the sources are represented as a view over the global schema.

In contrast with LAV mappings, GAV mappings define the principles about the creation of a global database from the local databases, querying in a GAV approach is straightforward. GAV approach performs well when sources are stable. Local As View (LAV) approach overcomes the limitations of GAV. The LAV approach requires each source to have an associated view over the global schema. The LAV approach follows the exact opposite methodology of a GAV and defines each local schema as a function of the global schema and assume global schema as fixed. GAV mappings are formally defined:

Global-as-View (LAV) [47]

Definition 2.4.2 *In a data integration system $\mathcal{IS} = \langle O, S, M \rangle$ based on GAV approach the mapping M associates to element in the global schema O a query q_S in terms of the sources S:*

$$o \rightarrow q_S$$

That is, the elements of the global schema are represented as a view over the sources.

We have utilised the power of LAV mappings to describe QA components in form of views to their semantic description. We then utilised these views to provide valid (in terms of input-output requirements of the components) composition of QA pipelines (for details please see Chapter 7). Figure 2.4 illustrates two local sources S1 and S2 with their individual schema which is expected to be aligned to global schema O. The global schema defines three concepts SkyRocket(x), Lorry(x), Shipment(x) and two relations, namely launch(x,y) that explains that some SkyRocket launches some Shipment, and supply(x,y) that describes that some Lorry supplies some Shipment. Source S1 contains relations in the form hasShipment(x,y) related to skyrocket and shipment but without explicit class assignment of variables x and y. Furthermore, source S2 contains relation transports(x,y) that implicitly links lorry and shipment. This example explains a typical data integration task. The LAV mappings are defined as below where relations in S1 and S2 are mapped as a conjunctive query over the concepts in global schema.

```
hasShipment(x,y) := launches(x,y),SkyRocket(x),Shipment(y)
transports(x,y) := supplies(x,y),Lorry(x),Shipment(y)
```

In contrast to LAV mappings, GAV performs well when source is stable. Applying GAV mapping over our data integration example illustrated in the Figure 2.4, the concepts are determined as a conjunctive query over local relations from S 1 or S 2. Some of GAV mappings are given below:

```
launches(x,y) := hasShipment(x,y)
SkyRocket(x) := hasShipment(x,y)
Shipment(y) := hasShipment(x,y)
supplies(x,y) := transports(x,y)
Lorry(x) := transports(x,y)
Shipment(y) := transports(x,y)
```

CHAPTER 3

Related Work

This chapter reviews the state of the art approaches related to the main research problem and research questions presented in Chapter 1. We show that the problem remains largely unexplored while existing state-of-the-art researches partly address the mentioned challenges in this thesis.

Section 3.1 presents a brief overview of current state-of-the-art question answering systems. We further describe independent components that can be part of question answering systems. While many of these systems 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 their components is very weak, (2) extension of the components is problematic, and (3) interoperability between the employed components is not systematically defined.

Section 3.2 describes some of the existing frameworks that promote reusability of QA components. We observe that all of them face limitations in terms of the easy configuration of QA systems just by reusing components, and do not support the dynamic composition of QA pipelines based on the type of the question. We take a closer look at their implementation to understand the pitfalls, and limitations in terms of scalability, reusability, and ease-of-use. We conclude that the potential of reusing existing QA components to build effective QA pipelines to be elicited.

3.1 Question Answering Systems

In 1970, first attempt of providing easy to use natural language interface to interact with data has been published [43]. Since then, an increasing number of QA systems and QA related components have been developed by both industry and the research community [7, 44]. The QA systems can be distinguished based on the 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 [48]. In this work, the authors presented a dedicated QA system for translating a natural language question to its corresponding SPARQL query to extract answers from structured medical data.

For a QA system, an input type can be anything ranging from the 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 [49], Swoogle [50], and Sindice [51] focus on natural language based search whereas systems described in [52] integrate 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. However, when the scope is limited to an explicit domain or ontology, there are

QA System	NER	NED	POS Tagging	Dependency Parsing	Query Builder
OKBQA [13]		✓			✓
Alexandria [54]	✓	✓	✓		✓
QAKIS [23]	✓	✓			✓
HAWK [55]	✓	✓			✓
AskNow [56]		✓			✓
CASIA [57]	✓	✓	✓	✓	✓
DEANNA [58]	✓	✓	✓	✓	✓
Intui2 [59]		✓	✓		✓
Intui3 [24]	✓	✓	✓		✓
ISOFT [60]	✓	✓	✓	✓	✓
POMELO [61]	✓	✓			✓
PowerAqua [62]		✓	✓		✓
QAnswer [63]		✓	✓	✓	✓
SemGraphQA [64]	✓	✓		✓	✓
SINA [20]	✓	✓			✓
SWIP [65]	?	✓		✓	✓
TBSL [8]		✓			✓
Xser [66]		✓			✓
UTQA [67]		✓			✓
gAnswer [68]		✓		✓	✓
FREyA [69]		✓	✓		✓

Table 3.1: **QA Systems and Tasks**. Analysis of the architecture of various QA systems shows that these systems implement similar tasks by dedicated components.

fewer chances of ambiguity and high accuracy of answers. It is also difficult and costly to extend closed domain systems to a new domain or reuse it in implementing a new system. Other types of QA systems described in [53] extract answers from an unstructured corpus (e.g., news articles) or other various forms of documents available over Web. They are known as corpus-based QA systems. QuASE [53] is one of such corpus based QA system that mines answers directly from the Web.

To overcome the limitations of closed domain QA systems, researchers have shifted their focus to *open domain* QA systems. FreeBase [5], DBpedia [4], and WikiData [6] are few examples of open domain Knowledge Bases. However, KBs like Google's knowledge graph [40] are not publicly available. Open domain QA systems use the publicly available semantic information to answer questions. Question answering over structured knowledge bases has gained momentum since last one decade. The recent survey by Diefenbach et al. [10] provides a complete and holistic overview of existing QA systems and their employed techniques. 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 the type of question (input), sources (structured data or unstructured data), and based on traditional intrinsic challenges posted by search environment (scalability, heterogeneity, openness, etc.) [44].

In 2011, a yearly benchmark series QALD was introduced. In the latest advancements, QALD now focuses on hybrid approaches using information from both structured and unstructured data [70]. Many

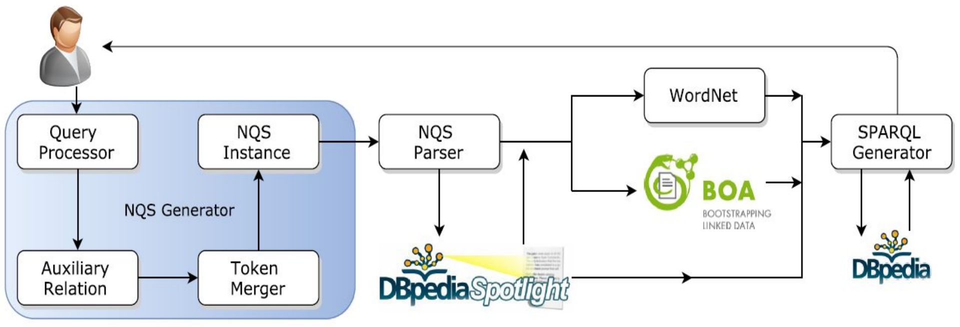


Figure 3.1: **AskNow Architecture [56].** AskNow is a monolithic QA system which has a QA pipeline using several QA components implementing various QA task. DBpedia Spotlight is reused in this architecture.

open-domain QA systems now use QALD for the evaluation. PowerAqua [62] is an ontology-based QA system which answers the question using the information that can be distributed across heterogeneous semantic resources. FREyA [69] 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 [23] 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 [20] is a semantic search engine which obtains either the 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 [8] 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. Figure 3.1 illustrates the architecture of Asknow QA system. It can be observed from the architecture that the system implements a sequence of several QA tasks and reuses DBpedia spotlight as entity linking component.

We deeply analysed the architecture of more than 30 QA systems and looked into the implementation of more than 50 QA components in total which is part of various QA systems. Table 3.1[1] summarises the snippet of our findings. The detailed list of analysed QA systems and components can be seen at `http://wdaqua.eu/QAestro/qasystems/`. We observe from our analysis that most of the QA systems have implemented similar QA tasks using various components in their architecture. Hence, the following observations were made:

- The QA system developers have employed various approaches while building QA systems, however on the logical level, these systems implement similar and overlapping QA tasks.
- In spite of a high number of overlapping components at an abstract level, there is no way/approach to address their interoperability and connect them physically.
- Some of the QA tasks such as NED and QB are implemented by all the QA systems by dedicating a separate component in its architecture.
- At the implementation level, the QA systems implementing similar tasks are heterogeneous at different levels of granularity such as different input/output format, input datasets, programming languages etc.

[1]For a few QA systems, results have been taken from [10]

- The components present in these QA systems are not easily reusable due to their monolithic implementation. If a developer is looking to reuse some of these components, she needs a major engineering effort to decouple the components present in these QA systems.

This motivated us to propose an approach to tackle heterogeneity of QA components, and aim towards building an infrastructure for QA pipeline composition to easily reuse QA components.

3.1.1 Components for Question Answering

In the previous section, we provide an overview of question answering systems over structured data. Besides the components which are part of existing QA systems, we surveyed the independent components available for different question answering tasks. This section provides a brief overview of some of these components. We have reused all these components in proposed Qanary Ecosystem and FRANKENSTEIN framework (Chapter 8).

Named Entity Recognition and Disambiguation Components The aim of the named entity recognition (NER) task is to recognise the entities present in the question and the aim of named entity disambiguation (NED) is to link these spotted entities to their knowledge base mentions (e.g., for DBpedia [4]). For instance, in the example question "Who is the mayor of Berlin?", an ideal component performing the NER task recognises `Berlin` as entity and components for NED task link it to its DBpedia mention `dbr:Berlin`[2]. Below we list some of the NER and NED components.

1. **Entity Classifier** uses rule base grammar to extract entities in a text [71]. Its REST endpoint is available for wider use for NER task.

2. **Stanford NLP Tool**: Stanford named entity recogniser is an open source tool that uses Gibbs sampling for information extraction to spot entities in a text [72].

3. **Babelfy** is a multilingual, graph-based approach that uses random walks and the densest subgraph algorithm to identify and disambiguate entities present in a text [73]. We have used public API[3] of Babelfy for NER and NED task as separate components.

4. **AGDISTIS** is a graph based disambiguation tool that couples the HITS algorithm with label expansion strategies and string similarity measures to disambiguate entities in a given text [25]. The code is publicly available[4].

5. **DBpedia Spotlight** is a web service[5] that uses vector-space representation of entities and using the cosine similarity, recognise and disambiguate the entities [17].

6. **Tag Me** matches terms in a given text with Wikipedia, i.e., links text to recognise named entities. Furthermore, it uses the in-link graph and the page dataset to disambiguate recognised entities to their Wikipedia URIs [74]. Tag Me is open source, and its REST API endpoint[6] is available for further (re-)use.

[2] `dbr` corresponds to `http://dbpedia.org/resource/`
[3] `http://babelfy.org/guide`
[4] `https://github.com/dice-group/AGDISTIS`
[5] `https://github.com/dbpedia-spotlight/dbpedia-spotlight`
[6] `https://services.d4science.org/web/TagMe/documentation`

7. **Other APIs**: Besides the open-source available components, there are many commercial APIs that also provide open access for the research community. Aylien API[7] is one of such APIs that use natural language processing and machine learning for text analysis. Its text analysis module also consists of spotting and disambiguation entities. TextRazor[8], Dandelion[9], Ontotext[10] [75], Ambiverse[11], and MeaningCloud[12] are other APIs that have been providing open access to researchers for their reuse.

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

1. **ReMatch** maps natural language relations to knowledge graph properties by using dependency parsing characteristics with adjustment rules [19]. It then carries out a match against knowledge base properties, enhanced with word lexicon Wordnet via a set of similarity measures. It is an open source tool, and the code is available for reuse as RESTful service[14].

2. **RelMatch**: The disambiguation module (DM) of OKBQA framework [13] provides disambiguation of entities, classes, and relations present in a natural language question. This module is the combination of AGDISTIS and disambiguation module of AutoSPARQL project [8]. The DM module is an independent component in OKBQA framework and available for reuse[15].

3. **RNLIWOD**: Natural Language Interfaces for the Web of Data ((NLIWOD) community group[16] provides reusable components for enhancing the performance of QA systems.

4. **Spot Property**: This component is the combination of RNLIWOD and OKBQA disambiguation module [13] for the relation linking task.

Components for Class Linking To correctly generate a SPARQL query for a NL query, it is necessary to also disambiguate classes against the ontology.[17] For example, considering the question "Which river flows through Seoul" the word "river" needs to be mapped to `dbo:River`[18]. A little work has been done for Class Linking, and the following two components can be reused for this task:

1. **NLIWOD CLS**: NLIWOD Class Identifier is one among the several other tools provided by NLIWOD community for reuse. The code for class identifier is available on GitHub[19].

2. **OKBQA Class Identifier**: This component is part of OKBQA disambiguation module[20].

[7]http://docs.aylien.com/docs/introduction
[8]https://www.textrazor.com/docs/rest
[9]https://dandelion.eu/docs/api/datatxt/nex/getting-started/
[10]http://docs.s4.ontotext.com/display/S4docs/REST+APIs
[11]https://developer.ambiverse.com/
[12]https://www.meaningcloud.com/developer
[13]http://dbpedia.org/ontology/leader
[14]https://github.com/mulangonando/ReMatch
[15]http://repository.okbqa.org/components/7
[16]https://www.w3.org/community/nli/
[17]https://www.w3.org/TR/rdf-schema/#ch_class
[18]http://dbpedia.org/ontology/River
[19]https://github.com/dice-group/NLIWOD
[20]http://repository.okbqa.org/components/7

Components for Query Builder A query builder generates SPARQL queries using disambiguated entities, relations and classes which can serve as input from previous steps in a QA pipeline. Little has been done also for query builder task. Following two components can be easily reused in a QA system:

1. **NLIWOD QB**: Template based query builders are widely used in QA community for SPARQL query construction (e.g., HAWK [55], TBSL [8] etc). NLIWOD reusable resources[21] provides a template based query builder.

2. **SINA Query Builder**: SINA is a keyword and natural language query search engine that is based on Hidden Markov Models for choosing the correct dataset to query [20]. The developers of SINA have decoupled the query builder module from its monolithic implementation for further reuse.

It can be observed from the existing QA systems and components that there are many independent components which can be part of a QA system and perform similar tasks as monolithic QA systems. However, a generic yet effective approach is missing to address their heterogeneity and integrate these components in a single platform.

3.1.2 Ontologies for Question Answering

Ontologies play an important role in question answering. First, they can be used as a knowledge source to answer the questions. Prominent examples are the DBpedia Ontology and YAGO [76]. 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”.[22]. Users can configure annotations to their specific needs using the DBpedia Ontology.

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.

NIF ontology were designed aiming for promoting interoperability for natural language processing tools, language resources, and annotations. For example, One of the output format of DBpedia Spotlight is using NIF Interchange Format ontology (NIF) [77].

Ontologies can also be used to model the search process in a question answering system. For example, the research presented in [78] 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. So far no ontology has been developed that would provide a common abstraction to model the whole QA process.

3.1.3 Question Answering Benchmarks

To evaluate question answering over knowledge graphs, several benchmarking datasets have been released by QA community. Question answering over Freebase have various datasets for evaluating QA systems, such as SimpleQuestion [79] and WebQuestions [80]. SimpleQuestion contain over 100,000 questions whereas WebQuestions contain 5810 questions. QA systems such as [80] and [81] are evaluated using

[21] https://github.com/dice-group/NLIWOD
[22] http://wiki.dbpedia.org/services-resources/ontology

these datasets. For Wikidata, recently proposed dataset [82] contains 21,957 questions with 689 questions in addition that are actual user questions asked to QA system deployed online[23].

Question answering systems over DBpedia have been mostly evaluated using Question Answering over Linked Data Challenge (QALD) [70] which is now in its 9th edition[24]. QALD series have various tasks involving performance evaluation of various types of questions such as bio-medical questions, multilingual questions, simple questions over DBpedia etc. However, comparing to Freebase or Wikidata datasets, a total number of questions in QALD series ranges from 50-400; and all the QA systems over DBpedia have been evaluated using a limited number of questions. To overcome this problem dataset called LC-QuAD [83] has been released in 2017 with 5000 questions. This dataset contains 80 percent complex questions (questions with more than one entity and one relation). It is important to note that question answering over linked data has been restricted to open knowledge extracted from Wikipedia, and fewer attempts have been made to extend question answering in specific domains such as geospatial. A recent attempt has been made in this direction, where researchers have released a linked data based question answering benchmark for geospatial question answering over linked data. The dataset has been made publicly available[25]. For this thesis, we rely on QALD and LC-QuAD datasets for evaluation.

3.2 Component Based Question Answering Frameworks

Earlier in this chapter, 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. There exist many independent components that can be reused in question answering. However, there is a need for open frameworks for generating QA systems that integrate 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 [78] describes a search ontology to provide the abstraction of user's question. The 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, a user can not define the dataset that should be used and other specific properties. Moreover, OAQA [84] follows several architectural commitments to components to enable interchangeability, however, it is restricted to the biomedical domain. QANUS [85] also provides capabilities for the rapid development of information retrieval based QA systems. In 2014, Both et al. [86] presented a first semantic approach for integrating components, where modules that carry semantic self-descriptions in RDF are collected for a particular query until the search intent can be served using the proposed approach.

The QALL-ME framework [12] is an attempt to provide a reusable architecture for multilingual, context aware QA framework. It is an open source software package. The QALL-ME framework uses an ontology to model structured data of a targeted domain. This framework is restricted to closed domain QA and finds the limitation to get extended for heterogeneous data sources and open-domain QA systems. The openQA [11] framework is an architecture that is dedicated to implementing a QA pipeline. Additionally, here the implementation of a QA pipeline is restricted to Java and cannot work agnostically to the programming language. Furthermore, openQA has a few numbers of components.

OKBQA [13] is a recent and effective attempt to develop question answering systems with a collaborative effort. OKBQA has 24 components targeting English and Korean language question answering. The limitation of OKBQA is that it divides the components into four tasks, namely template generation,

[23]http://wdaqua-frontend.univ-st-etienne.fr/
[24]http://2018.nliwod.org/home
[25]http://geoqa.di.uoa.gr/

disambiguation, query generation, and answer generation and follow strict input/output format. New components performing other QA tasks such as answer type detection, named entity recognition, relation linking etc cannot be easily added to OKBQA. With the missing flexibility both in terms of data format and implementations, OKBQA is restricted to a specific type of components respecting these requirements. Besides the frameworks described above, we are not aware of other QA framework that addresses reusability of components to build QA systems. Also, these frameworks are limited in their functionalities. None of the existing frameworks for building QA systems consider the scalability of QA components and add dynamicity i.e. composing question answering pipelines on demand based on the type of input question. For example, OKBQA has 24 components. Here, components have to be manually selected to be part of a QA pipeline. Also, it is highly unlikely that components which are part of a QA pipeline will answer all the questions. There are chances that if one question is not answered by one QA pipeline comprising some of these 24 components, it may be answered by another pipeline. Hence, components need to be dynamically chosen to be part of a QA pipeline to answer the question.

In 2017, QA4ML [15] is the first attempt to add dynamicity to the QA framework where a QA system is selected out of 6 QA systems based on the question type. QA4ML describes several features to label a question (question length, entity type etc.) and trains classifiers to predict the performance of QA systems per component but does not allow composition of QA pipelines reusing the components, or adds new components to the framework. However, the implementation is restricted to six existing QA systems. Below we summarise the limitations which have been observed in state-of-the-art QA frameworks:

- The existing QA frameworks are rigid in terms of a fixed number of tasks and input/output format, and specific data format, hence interoperability issue of various QA components is still open.
- There is no standard way to establish communication between the integrated components in the QA frameworks.
- It is not easy and flexible to integrate QA components at any stage of QA pipeline due to fixing pipeline architecture at the implementation level.
- Many components are integrated into these frameworks, yet QA pipelines are composed manually. There is no seamless way to compose QA pipeline within these frameworks automatically.
- With an increasing number of QA components developed by the research community, existing QA frameworks do not consider the scalability of components for various QA tasks. With the availability of many QA components in a single platform, it is not clear if it is expected to run all the possible viable combinations for each input question. For example, if a QA framework has 10 components for NED, five for relation linking, five for query builder task, the number of resulting pipelines in the framework is 250. In existing frameworks, there is no dynamic (on the fly) way to select the best components per task to be part of the QA pipeline for the given input question based on strength and weaknesses of these components.

OKBQA, openQA, and QALL-ME frameworks can be directly used to build customised QA pipelines for DBpedia. We summarise their characteristics and limitations in Table 3.2. Besides building the QA frameworks, the question answering researchers have also focused on building frameworks for question answering evaluations. QA frameworks, evaluation frameworks like GERBIL [87] have emerged over the last years. GERBIL provides means to benchmark several QA systems on multiple datasets in a comparable and repeatable way fostering the open science methodology. Using GERBIL, many entity disambiguation components can be evaluated using different datasets. Very recently, this framework is further extended for benchmarking complete QA pipelines [3]. Although GERBIL provides benchmarks

Features	**QALL-ME**	**openQA**	**OKBQA**
Promoted reusability of QA components	✓	✓	✓
Strict programming language dependencies	✓	✓	–
Strict input/output requirements	✓	✓	✓
Isolation of integrated components	–	–	✓
Number of reusable components	7	2	24
Manual QA pipeline composition	✓	✓	✓
Easy exchange of components within framework		–	✓

Table 3.2: Comparison of QA frameworks

for disambiguation components and QA systems, it does not provide benchmarking capabilities to evaluate components for relation linking, class linking, and other such tasks. Hence, individual benchmarks need to be created for benchmarking other tasks.

CHAPTER 4

Semantic Based Approach for Describing QA Systems and Processes

In the previous chapters, we describe research problems, challenges, and state-of-the-art QA systems and components. We also illustrated that QA systems are connected on abstract level in terms of the QA tasks, however there is a clear lack of conceptual representation of existing (QA) systems and components that prevents these QA components and systems to be integrated in a single platform, in order to build more powerful QA systems. In other words, there is no systematic way to describe existing QA components – either standalone or parts of other QA systems – based on their functionality, i.e., the task they perform. Therefore, we made following observations:

- Several independent QA components are available for various QA tasks. However, QA system developer needs to study the architecture of QA component to understand its functionality and other dependencies. This is due to the fact that QA components are not semantically described based on the task they perform and high level input/output requirements.
- With a vision to integrate several QA components in a single framework, it becomes challenging to manually compose QA pipelines. For instance, with the increasing number of QA components, identifying all viable combinations of components when creating new QA pipeline requires a manual and time-consuming search in the large combinatorial space of solutions when integrated in a single platform, for example in OKBQA framework [13]. OKBQA has 24 QA components, but there is no semantic description of these components based on which task they perform, what is the input/output requirement. Hence, it becomes difficult for a developer to utilise plethora of QA components for building QA systems.
- Besides the missing semantic description of QA components, many independent QA components are not interoperable due to heterogeneity at different levels (programming language, datasets, input format etc.
- In Chapter 3, we reviewed existing state-of-the-art QA systems and frameworks. We identified 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 generalised approach for architecture or ontology of a QA system to bring all state-of-the-art advancements of QA under a single umbrella.

Therefore, there is a need for a descriptive approach that defines a conceptual view of QA systems covering both 1) logical description of QA systems and components 2) an approach that must cover all needs of current QA systems design and be abstracted from implementation details. Moreover the approaches should be extensible such that they can be used in future QA systems. To address these problems, we first aim at defining and conceptualising the QA systems and components based on the task they perform. We define a QA component based on its input/output requirement and associated task using a controlled vocabulary QAV. We use concepts of Local as View (LAV) mappings to describe the components. These semantic descriptions allow us to understand the functionalities of QA components effectively and in a simple way.

The second contribution of this chapter is a generalised ontology named qa which covers the need for interoperability of QA systems on an implementation level. We initiate a step towards an interoperable approach that will be used to build QA systems which follow a philosophy of being actually open for extensions. Our approach collects and generalises the necessitated requirements from the state-of-the-art of QA systems. While thereafter we show how these requirements are fulfilled while using the Web Annotation Data Model. We model the conceptual view of QA systems using and extending the Web Annotation Data Model. This model empowers us for designing a knowledge-driven approach for QA systems as next logical step towards building reusable QA systems. The QAV vocabulary and qa vocabulary differ to each other in following aspects:

- The QAV vocabulary targets the problem of missing semantic representation and description of QA components. Such semantic description can be helpful in automatising the process of selection of components for the QA pipeline (c.f. Chapter 5). However, it does not consider the requirements of actually implementing a new QA system by reusing QA components. Also, QAV vocabulary does not solve the interoperability issue at implementation level. It solves the interoperability at logical level i.e. assuming when components are already integrated in the QA framework, how to semantically define these components to build automatic QA pipelines.
- The qa vocabulary on other hand concerns with solving the interoperability issues at implementation level. It collects all the requirements related to the QA process (i.e. complete process of extracting answers from the KGs for input question), and focuses on capturing knowledge generated during this process. It allows us to propose a knowledge driven approach for creating QA pipelines by reusing the heterogeneous components (c.f. Chapter 5).

Therefore, we semantically describe QA components using the proposed QAV vocabulary, and then introduce qa vocabulary to capture all the knowledge exchanged between the QA components during the QA process. In this way we will establish for the first time a conceptual view of the existing QA systems. Therefore, following research question is addressed in this chapter:

Research Question 1 (RQ1)

How can semantics contribute in resolving interoperability of QA components?

We made following contributions in this chapter towards the mentioned **RQ1** and the research problem in general (This chapter is based on [26][1] and [9].):

[1]The qa vocabulary proposed in this paper, jointly done with Dennis Diefenbach (Universite Jean Monnet, France). My contributions in this paper was to collect all the requirements for designing open and scalable vocabulary by reviewing state of the art QA systems, and then designing the concrete requirements wrt. Web Annotation Data model for conceptualising the QA systems.

- A controlled vocabulary following the Local as View mapping concepts to semantically describe existing QA components.
- Descriptive analysis of requirements of knowledge driven approach for QA systems;
- A generalised knowledge-driven vocabulary built upon an abstract level;

The rest of this chapter is structured as follows. Next Section describe the QAV vocabulary for conceptualising the QA components. Section 4.2 describes the dimensions of QA systems for conceptualising question answering systems and components. Section 4.3 describes the existing problem and our proposed idea of an implementation independent compatibility level in detail. Section 4.4 details the requirement of knowledge-driven QA systems which are derived from the state-of-the-art QA approaches. Section 4.5 illustrates our proposed ontology with a case study to address all the requirements. We summarise the chapter in Section 4.6.

4.1 Semantic Description of QA Components

To semantically describe the QA components based on their functionalities, we introduce a controlled vocabulary named QAV. This vocabulary (i.e. QAV) of the domain of QA tasks is described as a pair $\langle \delta, A \rangle$, where δ is a signature of a logical language and A is the set of axioms describing the relationships among vocabulary concepts. A signature δ is a set of predicate and constant symbols, from which logical formulas can be constructed, whereas the axioms A describe the vocabulary by illustrating the relationships between concepts. For instance, the term *disambiguation* is a predicate of arity four in δ; $disambig(x, y, z, t)$ denotes that the QA task disambiguation relates an entity x, a question y, a disambiguated entity z, and a template t. Furthermore, the binary predicate $questionAnalysis(x, y)$ models the question analysis task and relates an entity x to a question y. The following axiom states that the disambiguation task is a subtask of the question analysis task:

```
disambig(x,y,z,t) -> questionAnalysis(x,y)
```

We then define a new concept QAC, which is a set of predicate signatures $\{QAC_1, \ldots, QAC_n\}$ that model QA components. For example, AGDISTIS [25] is represented with predicate $Agdistis(x, y, z)$ where x, y, and z denote an entity, a question, and a disambiguated entity, respectively. AGDISTIS is a NED tool that accept question, recognised entities as input and provide DBpedia URLs of entities. Further, the QA component Stanford NER [72] is modelled with the predicate $StanfordNER(y, x)$, which relates a question y to an entity x.

We follow the Local-As-View (LAV) approach to define QA components in QAC based on predicates in QAV. LAV is commonly used by data integration systems to define semantic mappings between local schemas and views that describe integrated data sources and a unified ontology [88]. The LAV formulation allows us to scale up to a large number of QA components, as well as to easily be adjusted to new QA components or modifications of existing ones. This property of the LAV approach is particularly important in the area of question answering, where new QA systems and components are constantly being proposed by practitioners and the research community. Following the LAV approach, a QA component C is defined using a conjunctive rule R. The head of R corresponds to the predicate in QAC that models C, while the body of R is a conjunction of predicates in QAV that represent the tasks performed by C. LAV rules are safe, i.e., all the variables in the head of a rule are also variables in the predicates in the body of the rule. Additionally, input and output restrictions of the QA components can be represented in LAV rules. The following LAV rules illustrate the semantic description of the QA components AGDISTIS and Stanford NER in terms of predicates in QAV. The symbol "*$*" denotes an input attribute of the corresponding QA component.

```
Agdistis($x,$y,z):-disambig(x,y,z,t),entity(x),question(y),
                   disEntity(z)
StanfordNER($y,x):-recognition(y,x),question(y),entity(x)
```

These rules state the following properties of AGDISTIS and Stanford NER: (i) AGDISTIS implements the QA task of disambiguation; an entity and a question are received as input, and a disambiguated entity is produced as output; (ii) Stanford NER implements the QA task of recognition; it receives a question as input and outputs a recognised entity. We further consider semantic descriptions for DBpedia NER [17], Alchemy API, and the answer type generator component of the QAKiS QA system [23] (*Qakisatype*).

```
DBpediaNER($y,x):-recognition(y,x),question(y),entity(x)
Alchemy($y,z):-disambig(x,y,z,t),question(y),disEntity(z)
Qakisatype($y,a):-answertype(y,a,o),question(y),atype(a)
```

These rules state the following properties of the described QA components:

- The predicates *DBpediaNER*($\$y, x$), *Alchemy*($\$y, z$), and *Qakisatype*($\$y, a$) represent the QA components DBpedia NER, the Alchemy API, and Qakisatype, respectively. The symbol $ denotes the input restriction of these QA components, i.e., the three QA components receive a question as input. These predicates belong to QAC.

- The predicates *recognition*(y, x), *disambig*(x, y, z, t), and *answertype*(y, a, o) model the QA tasks: entity recognition, disambiguation, and answer type identification, respectively. These predicates belong to the QAV.

- An input natural language question is modelled by the predicate *question*(y), while *entity*(x) represents a named entity identified in a question.

- The QAV predicates *disEntity*(z) and *atype*(a) model the QA tasks of generating disambiguated entities and answer type identification, respectively.

- The variables x, y, z, and a correspond to instances of predicates *entity*, *question*, *disEntity*, and *atype*, respectively. The variable o is not bound to any predicate because *Qakisatype* does not produce ontology concepts.

Semantic description of the components using QAV as LAV mapping allows us to understand the functionalities of the components more clearly, their abstract level input and output requirements. Although the input/output format of these components differ at implementation level (i.e. JSON/XML etc.), the upper level abstraction using LAV mapping provides a clear understanding of the functionalities of these components. For instance, all the components implementing disambiguation task now can be described as LAV mapping rules respecting their input/output dependencies. This gives the components a semantic representation in a more formal way. We formalise 51 such components using the QAV vocabulary and the complete list is online at our website[2].

[2] http://wdaqua.eu/QAestro/qacomponents/

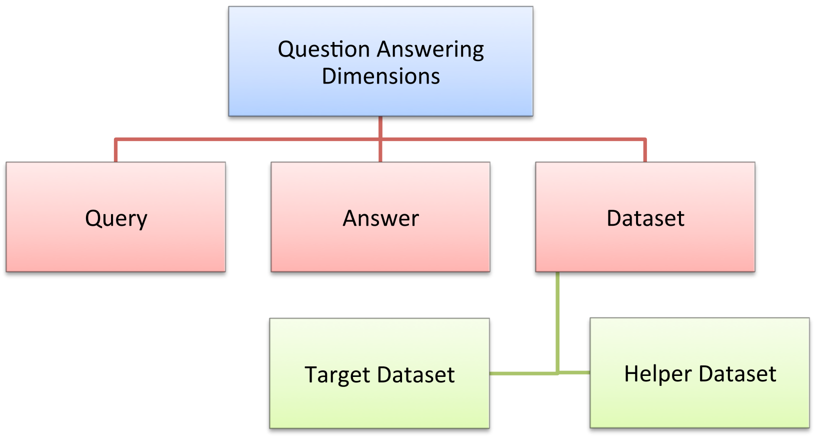


Figure 4.1: Main dimensions of question answering systems to define overall information need in QA process.

4.2 Dimensions of Question Answering Systems

In the previous section, we semantically describe the QA components at abstract level, but QAV vocabulary does not consider the implementation details of the components. Furthermore, QAV vocabulary does not focus on describing the knowledge generated and exchanged between various QA components during complete question answering process. When we look at the implementations of a 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 4.1 presents these high level dimensions. Generally, all of various known QA processes either associated or interacted with QA systems are corresponding to one of these dimensions. Now, we discuss each dimension in detail.

1. *Query Dimension:* The first aspect of this dimension refers to the characteristics of the input query. User-supplied queries 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 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 tokenisation or NER.

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 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 are the employed datasets. 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.

4.3 Addressing Interoperability of QA Components

In this section we will present the problem observed from the state of the art concerning interoperability issues which motivates the development of `qa`vocabulary. Thereafter we will outline our idea for solving the problem.

Problem: Considering the related work in the previous chapter, we identify three groups of problems:

1. *Lack of a generic conceptional view on QA systems:* While there are many different architectures for QA systems (e.g., [11, 52, 53, 62]), most of them are tailored to specific and limited use cases as well as applications. Reusability and interoperability were not (enough) in the focus of such approaches. Creating a new QA systems is cumbersome and time consuming as well as limited to domains or programming languages.

2. *No standardised knowledge exchange 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 knowledge format. However, there might be great synergy effect while creating QA systems w.r.t. the combination of different tools. For example, in a given architecture, NER and NED components are integrated in a single tool. NER solutions might be evolved and thus, implemented in either a novel way (e.g., in [17]) or employ existing tools (e.g., the Stanford NER [72] used in [89]). Thus, integrating a (new) NER approach without a standardised knowledge 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 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., [78]).

Hence, considering the implementations of current QA systems (and their components) it can be observed that they do not achieve compatibility due to their concentration on the implementation instead of the conceptual view. Instead, implementation details need to be hidden and the focus has to be on the knowledge 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 mentioned dimensions (Section 4.2).

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 information that needs to be used as input and returned as output by the components of the QA system. Hence, question needs to be *annotated* with properties to make them available for the following components in the QA system pipeline. As each component of the QA system will use the knowledge about the question, QA applications following this idea are called *knowledge-driven*. Hence, all the information in the question needs to be *annotated* to the knowledge to make it available for the next components in the QA system pipeline. We adapt the definition of annotations from the Web Annotation Data Model[3].

Definition 4.3.1 (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.*

For example, considering the question "Where was the European Union founded?" (target) it might be annotated that it contains with 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 expressible (e.g., while using several NED components and later pick one interpretation). We manifest this in the following requirement:

Req. 1 (Provenance of Annotations) *The provenance of each annotation needs to be re-presentable 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 to provide richer meta-information.

Definition 4.3.2 (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 consists of a sequential set of atoms $Q = (q_1, q_2, \dots, q_n)$.*

For example, 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, the 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 [66]. Some QA systems such as gAnswer [68] use co-reference 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. 2 (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 the respective defined annotations).*

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

Annotations of components do not always have boolean characteristics. It is also possible to assign a confidence, (un)certainty, probability, 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 [17]). This implies the following new requirement to be considered:

Req. 3 (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 [17] the confidence (score) is within the range [0, 1] while in [63] the score might be any decimal number.

4.4 Requirements for Knowledge-driven QA Vocabulary

In this section while aiming for a conceptional level on top of existing QA approaches, the requirements of knowledge-driven approach for describing QA systems are derived from the state-of-the-art (Chapter 3). We present collected requirements following the dimensions of QA systems as described in Section 4.2. Hence, on the one side a data model for knowledge generated in QA process 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 new annotations depending on the requirements.

4.4.1 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., [8]), a keyword-based query (e.g., [20]), an audio stream (e.g., [90]), or a resource-driven input (e.g., [91]). 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 requirements for the input query:

Req. 4 (Part of the Input Query) *The definitions of* part of the input query *satisfy 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 and independent of their implementation in various QA systems.

Examples from an implementation view are as follows: for text queries the NIF [77] 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 [92] 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 [93]), NER tools (e.g., in [23]) or NED tools (like [25]). 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. 5 (Annotations of Parts) *It has to be possible to define an arbitrary number of annotations for each part of the input question.*

For example, 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]).

4.4.2 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 million inhabitants?"), a list of resources, just one resource etc. Therefore, we define the following requirement:

Req. 6 (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 tasks (e.g., boolean, list . . .).*

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. For example, 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. 7 (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 (Requirement 3), the annotator (Requirement 1) and other.

4.4.3 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 4.2. To include these dimensions to the data model, the following requirements are met:

Req. 8 (Dataset) *A* dataset *provides an* endpoint *where the data can be accessed and statements about the dataset* format *can be gathered.*

Req. 9 (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. 10 (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 (note: the number of target datasets is arbitrary and depends on the requirement).*

These requirements enable 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.

4.5 Running Example

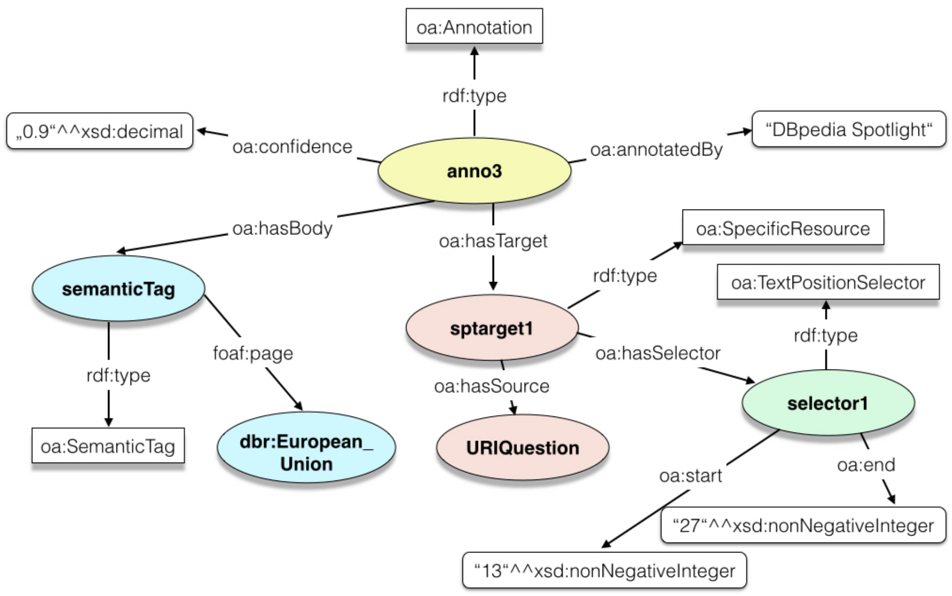


Figure 4.2: This picture represents an annotation of the question "Where was the European Union founded?". The part "European Union" is selected using a Specific Resource and a Selector. Moreover a semantic tag is associated.

In the previous section, we have collected the requirements for a data model describing the knowledge of interoperable QA systems. As running example, we now focus on an ontology that fulfils 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 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 changed or modified according to the intention of the annotation. The basic annotation model is represented in Turtle format[7] is as follows where the below pseudo code describes an annotation instance which is identified by `anno`. The `anno` has the properties `target` and `body` (i.e., each one is a resource):

[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#> .`
[7]http://www.w3.org/TR/turtle/

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

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 instantiate question, answer and dataset object. To establish an 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> ;
        oa:hasBody   <URIAnswer>   .
<anno2> a             oa:AnnotationOfQuestion ;
        oa:hasTarget <URIQuestion> ;
        oa:hasBody   <URIDataset>  .
```

Now, we will consider Requirement 4. To select parts of a query, 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 instantiate 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, Requirement 4 is fulfilled. It is obvious that we can instantiate an arbitrary number of annotations for each part of a question. Thus, Requirement 5 is also met.

The WADM defines the property `oa:annotatedBy` to identify the agent responsible for creating the Annotation, s.t., Requirement 1 is fulfilled. To comply with Requirement 3 a new property `qa:score` with domain `oa:Annotation` and range `xsd:decimal` is introduced. For example, 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 .
```

To fulfil Requirement 6, 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" .
```

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

Although later a resource will be used (instead of the `rdfs:label`), this shows that Requirement 6 is met. Requirement 7 is analogously satisfied. Now the requirements for datasets are considered. To fulfil Requirement 8 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 fulfil requirements 9 and 10 two new classes are introduced, namely `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 2 is also fulfilled.

To sum up, in this example we expressed the demanded data model with a generalised ontology which is reusing the concepts of the Web Annotation Data Model. These concepts capture all knowledge generated in the QA process in a homogeneous way. We call the proposed ontology “qa” vocabulary. We have shown that it is able to address all the requirements identified in Section 4.4. Moreover, we have illustrated the usage with an example and the data model is extensible and open for improvements. The complete running example is also available as online appendix at `https://goo.gl/vECgK5`.

4.6 Summary

In this chapter we have motivated the high demand for vocabularies which cover the need for interoperability of QA systems on a conceptual and implementation level. We first present QAV vocabulary to semantically describe the QA components and systems based on the task they perform. This provides a homogeneous way to represent a QA component w.r.t the corresponding QA task along with input and output requirements at abstract level. We then focus on the knowledge generated in the QA process at the implementation level, s.t., everything needed to establish a QA system is included within the data model. Given the requirements and the corresponding running example, we have established for the first time a knowledge-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. We have chosen a vocabulary that is agnostic to the implementation and the actual representation of the input question and the answer as well as the datasets. Hence, it is a descriptive and open approach. Hence, a logical, extensible, and machine-readable representation is now available fulfilling the collected requirements. Eventually, the proposed `qa`vocabulary can be used for all existing QA systems while transforming them into knowledge-driven (and therefore interoperable) implementations.

The QAV vocabulary and `qa` vocabulary present the foundation for promoting interoperability within QA community. However, an approach that utilises these vocabularies to integrate the QA components in a single platform is missing in this chapter. We utilise semantics of the QA components, and the knowledge produced in the QA process Also, we believe the QAC mapping can be used to automatise the process of composing QA pipelines in QA frameworks. We present this chapter as the first step towards addressing interoperability issues of QA components and systems. Based on the presented data model (or its implementation as an ontology in the use case presented in Section 4.5) it enable us to establish a new generation of QA systems and components of QA systems that are interoperable and we address first sub-research question (**RQ1** successfully. Hence, actually open and reusable QA systems are in sight.

CHAPTER 5

Knowledge-Driven Creation of Question Answering Systems

In the previous chapter, we first laid the foundation to describe QA components and systems semantically using the QAV vocabulary. We also proposed the `qa` vocabulary for creating an abstraction level on top of existing QA systems and components. Although the field of QA is large and many state-of-the-art QA systems exist, we have observed in the study of state of the art in Chapter 3 that QA components are not easily reusable because the QA systems and frameworks focus more on implementation details. For example, PowerAqua [62] links information available across distributed semantic resources to answer queries whereas TBSL [8] presents an approach that parses the question to produce SPARQL templates that depict the internal structure of the question. However, TBSL provides better results regarding linguistic analysis of questions, whereas PowerAqua is limited w.r.t. linguistic coverage of questions. Combining the capabilities of both systems open up the chances to provide better functionalities.

In other research areas, such as service-oriented architectures or cloud computing, the vision of building an ecosystem of components within a dedicated field has already proven its significance for the rapid advancement of research. Therefore, establishing a methodology – on a conceptual and implementation level – is considered crucial for managing the identified challenges of question answering in previous chapters. We first describe a methodology for developing question answering systems driven by the knowledge available for describing the question and related concepts. In this methodology, 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. We aim at establishing a methodology for integrating external components into a QA system using the `qa` vocabulary. By this, we 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 knowledge driven 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[1], YAGO[2]) or representing the

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

[2]`http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/`

analytics results of data (e.g., NLP Interchange Format [77], Ontology for Media Resources[3]). 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.

To evaluate the effectiveness of Qanary methodology, we present the first reference implementation of components and services that are integrated using the `qa` vocabulary and the Qanary methodology within the *Qanary ecosystem- a framework for creating reusable question answering systems*. Using Qanary ecosystem, a sophisticated framework level is achieved while hiding implementation details of the integrated components and establishing the `qa` vocabulary as representation of the knowledge about the user's question and the search query derived from it. 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 NER and NED task w.r.t. textual natural language input, which is a fundamental step in most QA processes. Additionally, we contribute the first NED benchmark for QA, as open source. Our main goal is to show how research community can use Qanary to gain new insights into QA processes. At a higher level, the following research question is addressed in this chapter:

Research Question 2 (RQ2)

How can state-of-the-art QA components be integrated in a single platform agnostic to their implementation to promote reusability effectively?

The stated **RQ2** benefits from the following contributions of this chapter:

- *Qanary*, a methodology for vocabulary-driven open question answering systems;
- *Qanary Ecosystem*, a framework for creating fast track question answering system;
- Empirical evaluation of NER and NED components integrated in the *Qanary Ecosystem* using *Qanary* methodology.

The chapter is structured as follows. Section 5.1 presents Qanary, a methodology for creating open question answering systems by reusing existing QA components. Section 5.2 is dedicated to the description of Qanary Ecosystem, a framework built on top of Qanary and `qa` vocabulary for integrating heterogeneous QA components in a single platform. We integrate several NED and NER components, and provide benchmarking of these components using a question answering dataset and present the experimental results. Finally, in Section 5.3, we provide a summary of the achieved results and conclude whether **RQ2** holds true. This chapter is based on [27–29, 94][4].

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

[4]In these papers, my contributions include designing and implementing the integration of various QA components in the core Qanary architecture. I have also contributed in designing the core Qanary Ecosystem using Springboot framework. I then contributed in designing the fundamentals of the Qanary methodology, and implementing the core QA pipeline architecture for the evaluation of the proposed methodology. These papers have been a joint work with another PhD student Dennis Diefenbach (Universite Jean Monnet, France)

5.1 Qanary – A Knowledge-driven Methodology for Open Question Answering Systems

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.) which are identified in the recent QA survey [44].

Closed domain QA systems target specific domains to answer a question, for example, medicine [48] or biology [95]. 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 and Google's non-public knowledge graph [40] are examples of semantically structured general-purpose *Knowledge Bases* used by open domain QA systems. Recent examples of such QA systems include PowerAqua [62], FREyA [69], QAKiS [23], and TBSL [8] and others as described in Chapter 3.

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 [8]. It would thus be desirable to combine these functionalities of [8] and [93] into a new, more powerful system. Also, the open source web service DBpedia Spotlight [17] analyses texts leading to NER and NED, using the DBpedia ontology. AIDA [89] is a similar project, which uses the YAGO ontology. AGDISTIS [25] is an independent NED service, which, in contrast to DBpedia Spotlight and AIDA, can use any ontology, but does not provide an interface. The PATTY system [96] 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.

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 [11]. 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 fulfil 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 generalised question answering systems which increases the overall efficiency [12]. Furthermore it will be possible to apply quality increasing approaches such as ensemble learning [97] 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 to 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.

Existing Problem in State of the Art QA Systems and Frameworks

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

Requirements for Knowledge Driven QA systems

From the previous problem statement and our observations in state of the art, we derived the following concrete requirements for a vital ecosystem of QA system's components (note: we have already summarised problem in last chapter, now we concretely define them):

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

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

Requirement 3 (Flexible Granularity) *It should be possible to integrate components for each small or big step of a QA pipeline. For example, components might provide string analytics leading to NER (e.g., [17]), other components might target the NED only (e.g., [25]) and additionally there might exist components providing just an integrated interface for NER and NED in a processing step.*

Requirement 4 (Isolation) *Every component needs to be able to execute their specific step of the QA pipeline in isolation from other components. Hence, business, legal and other aspects of distributed ownership of data sources and systems can be addressed locally per component. This requirement targets the majority of the QA 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 fulfils 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 section, 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[5]), the `qa` vocabulary is defined. This generalised 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 generalised 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.

5.1.1 Approach

Web Annotation Framework

The Web Annotation Data Model (WADM)[6] is a framework for expressing annotations. A WADM annotation has at least a target and a body. The target indicates the resource that is described, while the body indicates the description. The basic structure of an annotation, in Turtle syntax, looks as follows:

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

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

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

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

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

Vocabulary for Question Answering Systems

In the last chapter (c.f. 4) we introduced the vocabulary for the knowledge driven 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.

We reuse the `qa` vocabulary in Qanary that 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`:

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

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 standardised 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 [98]. 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` annotations.

5.1.2 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

```
PREFIX itsrdf: <http://www.w3.org/2005/11/its/rdf#>
PREFIX nif: <http://persistence.uni-leipzig.org/..../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 5.1: Aligning identified named entities to a new `qa` annotation using SPARQL

existing vocabularies due to the variety of expressing information. However, here we provide three typical solution patterns matching standard use cases and presenting the intended behaviour. As running example we consider an implemented exemplary question answering system with a pipeline of three components (NER+NED, relation detection, and query generation and processing; section 5.1.3). In the following the components are described briefly and also a possible alignment implementation of the custom vocabulary to `qa`.

NER and NED via DBpedia Spotlight

DBpedia Spotlight provides the annotated information via a JSON interface [17]. 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:

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

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:

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

Figure 5.1 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 lexicalisation

PATTY [96] 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 Figure 5.1. For example, the question “Where did Barack Obama graduate?” will now contain the annotation:

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

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

Query Construction and Query Execution via SINA

SINA [20] 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 `dbr:Barack_Obama`[7] and `dbo:almaMater`[8] are

[7] http://dbpedia.org/resource/Barack_Obama
[8] http://dbpedia.org/ontology/almaMater

present and can be used. The SPARQL query generated by SINA as output is a formal representation of a natural language query given below:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?uri
WHERE { dbr:Barack_Obama
dbo:almaMater ?uri }
```

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.

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

The implementation of this rule as a SPARQL INSERT query is straightforward. 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 Requirement 4 *islolation* is fulfilled, and we achieve *exchangeability* (Requirement 2) by being able to exchange every component.

Note that the choice of how to implement the alignments depends on the power of the triple store used. Hence, more elegant vocabulary alignments are possible but are not necessarily usable within the given system environment (e.g., an alternative alignment for Section 5.1.2, implemented as an OWL axiom, is given in the online appendix[9]). 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).

5.1.3 Use Case

In this Section we present a QA system that follows the idea presented in Section 5.1. Note that in this section 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. We have extended the vocabulary proposed in Chapter 4 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.

[9]alternative alignment: `https://goo.gl/hdsaq4`

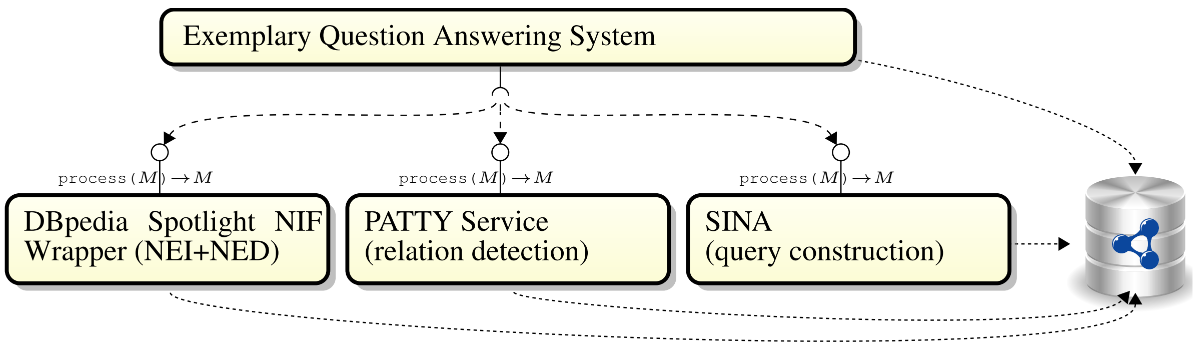

Exemplary Question Answering System
process(M) → M
process(M) → M
process(M) → M
DBpedia Spotlight NIF Wrapper (NEI+NED)
PATTY Service (relation detection)
SINA (query construction)

Figure 5.2: Architecture of the exemplary question answering system.

Our exemplary QA system consists of three components: DBpedia Spotlight for named entity identification and disambiguation, a service using the relational lexicalisations 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[10]. We can term it as local Knowledge Base (KB) of Qanary.

The whole architecture is depicted in Figure 8.4. 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 Section 5.1.1. Step by step each component receives a message M (8.4) 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 5.1.2 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 lexicalisations. These are then updated in the local triple store (or the local knowledge base) (Section 5.1.2). Finally, 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. Hence, the common process within all components is organised as follows:

1. A component fetches the required knowledge via (SPARQL) queries from the local Knowledge

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

Base (KB) i.e. triplestore. In this way, it gains access to all the data required for 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). 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 QA system 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 use case clearly shows the power of the approach. The knowledge representation is valid and consistent using linked data technology. Moreover, each component is now isolated (Requirement 4), exchangeable and reusable (Requirement 2), as the exchanged messages follow the `qa` vocabulary (Requirement 1), which contains the available pieces of information about the question, and their provenance and confidence. The components are independent and lightweight, as the central triple store holds all knowledge and takes care of querying and reasoning. As Qanary does not prescribe an execution order or any other processing steps, requirement of granularity (Requirement 3) is also fulfilled. The use case is available as online appendix[11].

5.2 Qanary Ecosystem

We describe in the previous section that the `qa` vocabulary laid foundation to the Qanary methodology. The message-driven implementation of Qanary foresees an 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. In this section, we present *Qanary ecosystem*, which 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 of tasks within QA systems. This includes different components performing NER like FOX [99] and Stanford NER [72] and components computing NED such as DBpedia Spotlight and AGDISTIS [25]. Also industrial services such as the Alchemy API are part of the ecosystem.

Furthermore, Qanary includes a language detection module [100] 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[12]. Additionally it should be noted that a monolithic QA system component [101] was developed and is integrated in Qanary. Additional external QA components are included in the ecosystem. In particular, Qanary includes two components from the OKBQA challenge[13] namely the template generation and disambiguation component. All components are implemented following the REST principles. Hence, these

[11] https://github.com/WDAqua/Pipeline
[12] http://kaldi-asr.org
[13] http://www.okbqa.org/

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[14]. 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[15] 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.)[16]. 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 the Figure 5.3). Hence, the reference implementation already provides the features required for QA systems using components distributed over the Web and can be accessed either as web service or provided as open source projects.

An additional interface allows for benchmarking a QA system created on demand using Gerbil for QA[17], thus allowing third-party evaluation and citeable URIs. Figure 5.4 illustrates the complete reference architecture of Qanary and a few of its components. The code is maintained in the repository[18] under the MIT License[19] for open source usage.

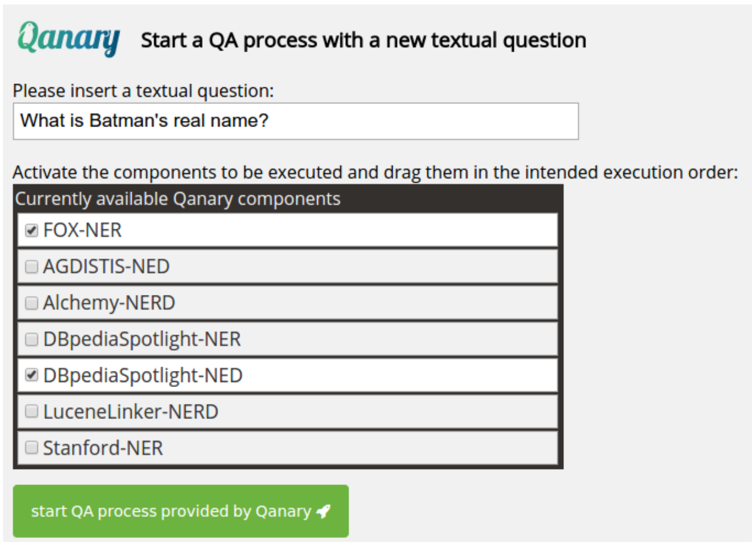
5.2.1 Gaining new insights into the QA process

To show how Qanary can be used to gain new insights into QA processes we focus here on the NED task. We present how we have extended the `qa` vocabulary 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 NED task. We describe how we constructed a benchmark for NED out of QALD-6[20]. 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 what are the reasons behind it. The following workflow is not restricted to the NED task but can be applied to any other sub-task of the QA process to gain new insights into QA processes and have utilised in next few chapters of the thesis.

The QA vocabulary for the NED task

The `qa` vocabulary 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 of standards that allow communication between QA components must be defined by the community. Taking into consideration the state of the art (e.g., [25, 72]), the `qa` vocabulary 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 NED but for every tool integrated into the Qanary ecosystem. To describe an entity spotted within a question we introduced a dedicated annotation which is describe below:

[14]https://github.com/WDAqua/Qanary/wiki
[15]The configuration property `spring.boot.admin.url` defines the endpoint of the central component.
[16]https://github.com/WDAqua/Qanary/wiki/Frequently-Asked-Questions
[17]http://gerbil-qa.aksw.org
[18]https://github.com/WDAqua/Qanary
[19]https://opensource.org/licenses/MIT
[20]https://qald.sebastianwalter.org/


Qanary Start a QA process with a new textual question

Please insert a textual question:

What is Batman's real name?

Activate the components to be executed and drag them in the intended execution order:

Currently available Qanary components

- [✓] FOX-NER
- [ ] AGDISTIS-NED
- [ ] Alchemy-NERD
- [ ] DBpediaSpotlight-NER
- [✓] DBpediaSpotlight-NED
- [ ] LuceneLinker-NERD
- [ ] Stanford-NER

start QA process provided by Qanary

Figure 5.3: 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).

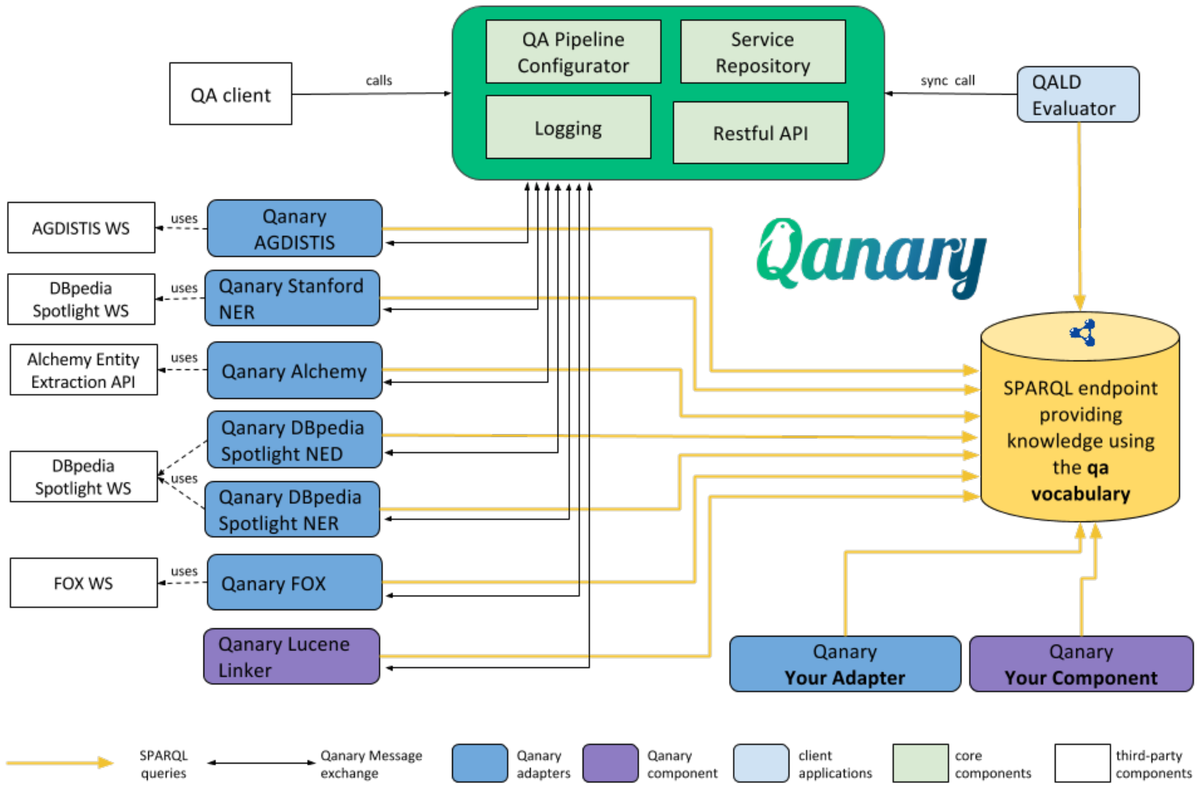


Figure 5.4: The Qanary reference architecture implementation highlighting the NER/NED components.

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

If in the question “When was Narendra Modi born?” a spotter detects “Narendra Modi” as an named entity, 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.

```
<anno1>  a  qa:AnnotationOfSpotInstance .
<anno1>  oa:hasTarget [
         a oa:SpecificResource ;
         oa:hasSource     <URIQuestion>;
         oa:hasSelector   [
         a oa:TextPositionSelector;
         oa:start "10"^^xsd:nonNegativeInteger;
         oa:end   "22"^^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 Narendra Modi born?” that the text “Narendra Modi” refers to `dbr:Narendra_Modi`, 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        "10"^^xsd:nonNegativeInteger;
         oa:end          "22"^^xsd:nonNegativeInteger
                       ]
                      ] ;
         oa:hasBody  dbr:Narendra_Modi .
```

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 improving the readability of the annotations.

5.2.2 Reusable NER and NED components

The following components were integrated into the Qanary ecosystem as Qanary components solve the task of NER and NED. The NER tool can be combined with each NED tool just by configuration utilising the power of Qanary methodology.

- **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 (it has currently support for English, German, Spanish and Chinese languages) [72].
- **FOX (NER)** integrates four different NER tools (including the Stanford NER tool) using ensemble learning techniques for spotting entities [99].
- **DBpedia Spotlight spotter (NER)** uses lexicalisations, i.e., ways to express named entities, that are available directly in DBpedia [17].
- **DBpedia Spotlight disambiguator (NED),** the NED part of DBpedia Spotlight, disambiguates entities by using statistics extracted from Wikipedia texts [17].
- **AGDISTIS (NED)** is a NED tool that uses the graph structure of an ontology to disambiguate entities present in the question [25].
- **ALCHEMY (NER + NED):** Alchemy API[21] 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 [20], which employs information retrieval methods.

5.2.3 A QALD-based benchmark for NED 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[22], which is the recent successor of QALD-5 [70], contains 350 questions, including 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 "Name the municipality of Roberto Clemente Bridge".

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?uri
WHERE { dbr:Roberto_Clemente_Bridge
dbo:municipality ?uri }
```

NED 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 "Roberto Clemente Bridge" and link to it the resource `dbr:Roberto_Clemente_Bridge`. Our benchmark compares the URIs generated by an NED tool with the resource URIs in the SPARQL query (i.e., those in the `dbr` namespace), 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.

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

$$
\begin{aligned}
\text{Precision}(q) &= \frac{\#\text{ correct URIs retrieved by the NED configuration for } q}{\#\text{ URIs retrieved by the NED configuration identifying entities in } q}\\
\text{Recall}(q) &= \frac{\#\text{ correct URIs retrieved by the NED configuration for } q}{\#\text{ gold standard answers for } q}\\
F_1\text{-measure}(q) &= 2 \times \frac{\text{Precision}(q) \times \text{Recall}(q)}{\text{Precision}(q) + \text{Recall}(q)}
\end{aligned}
$$

Figure 5.5: Metrics used in the NED benchmark

The metrics for a question q are calculated as defined in the QALD benchmark and are reported in Figure 5.5.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`. The metrics over all questions are defined as the average of the metrics over the single questions. The corresponding benchmark component is available at Github[23].

Note that this procedure can be generalised and applied to many sub-processes of a QA pipeline. For example, one might establish a benchmark to recognise 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. We used our benchmarking resource described above to evaluate NED 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 tools Alchemy and Lucene Linker. Each of them is implemented as an independent Qanary component. 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* and this process is automatic. Table 5.1 shows the benchmark results[24]. The "fully detected" column indicates the number of questions q where some resources were expected and the NED configuration achieved Recall(q)=1. 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

Thanks to the `qa` vocabulary we can collect (from the SPARQL endpoint) the results produced by every configuration. We analysed both this data and the results presented in Table 5.1 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:

- *Q1:* "Give me all cosmonauts." with the following resources requested in the SPARQL query:

[23] `https://github.com/WDAqua/Qanary`

[24] the benchmarking has been performed by co-author Dennis Diefenbach in our paper [94]. However author of the thesis (Kuldeep Singh) contributed in integrated several of these components in Qanary Ecosystem and results of the benchmark have been included in this chapter for completeness of Qanary Ecosystem.

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 5.1: Benchmarking of Entity Linking Components over QALD-6 data using the Qanary implementation.

`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.

5.3 Summary

In this chapter, we present our contributions for building flexible framework for creating QA systems by reusing existing QA components. Our first contribution of the chapter is a knowledge driven methodology named Qanary, which takes into account the major problems while designing (complex) question answering systems. Via alignments, the `qa` vocabulary (which is the foundation of Qanary) is extensible with well-known vocabularies while preserving standard information. This enables best-of-breed QA approaches where each component can be exchanged according to considerations about quality, domains or fields of application. Hence, the approach presented in this chapter provides a clear advantage in comparison to earlier closed monolithic approaches. 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

```
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 "2018-04-28T12:43.67+02:00"^^xsd:dateTime ;
        oa:hasTarget [
                a            oa:SpecificResource ;
                oa:Selector [
                            a        oa:TextPositionSelector ;
                            oa:end   10 ;
                            oa:start 22
                           ] ;
                oa:hasSource  <URIQuestion>
                ] .
        oa:hasBody     dbr:Narendra_Modi ;
        oa:annotatedBy git:dbpedia-spotlight .
```

Figure 5.6: Example data of Question: “When was Narendra Modi Born?”

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 (Req. 1–4 defined in Section 5.1) ultimately sets the foundation for rapidly establishing new QA systems. A main advantage of our approach is the reusable ontology alignments, increasing the efficiency and the exchangeability in an open QA system.

We presented the status of the Qanary ecosystem (c.f. Section 5.2), 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. We have integrated several NER and NED components using Qanary in Qanary ecosystem. This allows the first step towards 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 optimisation 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 NER and NED task. This way we realised a set of reusable components as well as the first benchmark for NED in the context of QA. All together we have shown how Qanary ecosystem 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 are maintained and used as open source projects, where Qanary methodology is the reference architecture for new components.

CHAPTER 6

Relation Linking using a Semantically Indexed Bi-Partite Knowledge Base

In the last chapter, we have presented Qanary methodology, and Qanary Ecosystem. We have integrated several independent tools for named entity recognition and disambiguation in Qanary Ecosystem. We observe in Chapter 3 that many independent tools are available for NER and NED tasks, but for relation linking, only four independent tools are there. In Section 5.1.2 we have (re)used PATTY [96] for relation linking tool to be part of our exemplary question answering system described in Section 5.1.3. Research shows that formal query formulation from natural language questions and, more specifically, linking relations to Knowledge Graph (KG) properties, often require extra knowledge sources that contain semantic descriptions or extensions of the underlying knowledge graphs. These knowledge bases (KBs) capture knowledge from large corpora or taxonomies, e.g., Wordnet [57], PATTY [57, 66], or the BOA pattern library [56], and allow for enhancing the accuracy of the process of mapping natural language relations to concepts in a specific knowledge graph. Hence, extracting knowledge from such background knowledge bases will also improve the effectiveness of the relation linking task and increase the overall performance of QA systems.

The idea of providing semantically typed patterns against the properties in a knowledge graph is a special feature. For example, PATTY [96] is a large knowledge base consisting of semantically typed relational patterns with their associated properties in open domain knowledge graphs such as DBpedia. Therefore, PATTY provides a rich source of relational patterns that can be used during relation linking. Höffner et al. [7] report that PATTY allows for flexible mapping of natural language relations to their KG properties. However, this flexibility implies that one relation can be matched to several patterns. For example, the natural language relational pattern *been playing with*, appears 12 times in PATTY and is associated with 11 relations. Hence, efficient methods are needed both for capturing knowledge from a large corpus like PATTY, and for exploiting their features in QA systems.

In this chapter, we devise an approach for capturing knowledge from collections of semantically typed relational patterns like PATTY; further, we present a relation linking method able to exploit these features. First, *SIBKB*, a semantic-index based representation of these knowledge bases is proposed; SIBKB provides searching mechanisms for accurately linking relational patterns to semantic types. The benefits of SIBKB have been empirically evaluated on existing QA benchmarks. Results suggest that SIBKB enhances the performance of relation linking methods by up to three times. This chapter provides an independent relation linking tool by capturing knowledge encoded in PATTY knowledge base. We further integrate this component in Qanary Ecosystem as next step for full end to end QA pipeline to translate natural language question to its formal representation (i.e. SPARQL). The remainder of the chapter is structured as follows. Section 6.1 motivates our work with an example. Section 6.2 elaborates on the specific problem of capturing knowledge from semantically typed knowledge bases; this is followed by a detailed illustration of the approach and the proposed solution. Section 6.3 presents the experiments to evaluate our approach and then we summarise the chapter in 6.4. The content of this chapter is derived from the article [102].

6.1 Reusability Issue of PATTY

We motivate the chapter by analysing the problem of extracting knowledge from a large knowledge base corpus during the relation (predicate) linking task in QA systems. PATTY [96] is one such large knowledge base of semantically-typed relational patterns; it contains 127,811 pairs of relational phrases and DBpedia predicates, involving 225 DBpedia relations in total. Many QA systems, such as Xser [66] and CASIA [57] use PATTY's relational phrases to match word patterns in an input question and find the corresponding DBpedia relation, as part of understanding a natural language question.

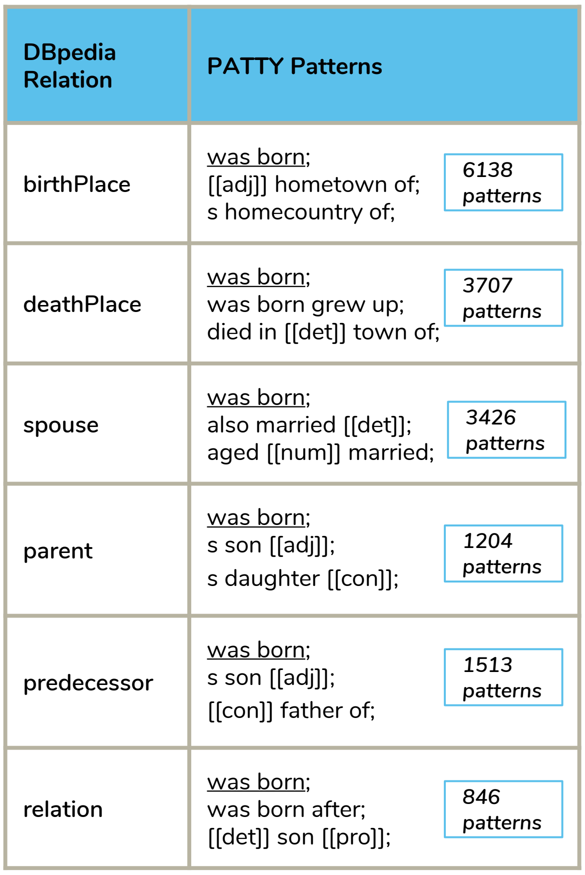
Let us consider the following question: *Where was Albert Einstein born?* Part of understanding this natural language question includes: (1) the extraction of the named entities and (2) the identification of the predicate(s). The successful completion of these tasks will allow a QA system to construct a formal query – e.g., a SPARQL query – in order to retrieve the answers from a knowledge graph like DBpedia.

For the first task of QA process, named entity recognition and disambiguation, tools such as DBpedia Spotlight [17] or AGDISTIS [25] can be used to identify the ALBERT EINSTEIN entity and disambiguate it to its DBpedia mention `dbr:Albert_Einstein`[1]. For the second task, the PATTY knowledge base can be used to link phrase patterns in the question such as *was born* to its associated DBpedia predicates (i.e., relations). In our exemplary question, the pattern *was born* can be mapped to six different DBpedia predicates of the PATTY corpus, namely `dbo:birthPlace`[2], `dbo:deathPlace`, `dbo:spouse`, `dbo:parent`, `dbo:relation`, and `dbo:predecessor`. In fact, all of these relations are linked to several textual patterns (e.g., `dbo:birthPlace` is related to more than 6,000 patterns of the PATTY corpus), which are often shared among different relations, as illustrated in Figure 6.1.

For example, in the PATTY corpus, the pattern *was born* appears 876 times and corresponds to six DBpedia relations like `dbo:birthPlace` (the correct answer in this case), `dbo:religion`, `dbo:parent`, `dbo:predecessor`, `dbo:spouse`, and `dbo:deathPlace`. Hence, if simple keyword based matching or generic similarity techniques are used to match phrase patterns of the question, multiple DBpedia relations for a given pattern are retrieved from the knowledge base. For the pattern *was born*, for instance, this will lead to six candidate DBpedia relations in PATTY knowledge base, as depicted in Figure 6.2.

Hakimov et al. [103] and Dubey et al. [56] describe this problem of PATTY and report noisy behaviour of PATTY patterns while building a QA system and relation linking tool. Many incomplete and ambiguous patterns in PATTY such as *s son [[adj]]*, *lt ref gt with* also cause noisy behaviour of PATTY. However, for our question, only `dbo:birthPlace` will provide the correct relation that will allow a QA developer to construct a SPARQL query to retrieve the correct answers. In the case of identifying the predicates `dbo:deathPlace`, `dbo:spouse`, and `dbo:parent`, a QA system which is utilising PATTY will retrieve wrong answers, while matching to the predicates `dbo:predecessor` and `dbo:relation` will lead to an empty answer set (see Figure 6.3). Therefore, relational pattern knowledge bases need to be exploited in an efficient way, in order to increase precision and recall in QA systems.

[1]`http://DBpedia.org/resource/Albert_Einstein`
[2]dbo is bound to `http://DBpedia.org/ontology`



DBpedia Relation	PATTY Patterns	
birthPlace	was born; [[adj]] hometown of; s homecountry of;	6138 patterns
deathPlace	was born; was born grew up; died in [[det]] town of;	3707 patterns
spouse	was born; also married [[det]]; aged [[num]] married;	3426 patterns
parent	was born; s son [[adj]]; s daughter [[con]];	1204 patterns
predecessor	was born; s son [[adj]]; [[con]] father of;	1513 patterns
relation	was born; was born after; [[det]] son [[pro]];	846 patterns

Figure 6.1: Excerpt of PATTY Knowledge Base; The natural language relational pattern "was born" is associated with six DBpedia predicates.

6.2 Bi-partite Graphs of Semantically-typed Relational Patterns

In this section, we present the problem of capturing knowledge in semantically-typed relational patterns. Further, we propose an index-based approach that allows for efficiently extracting the properties from a knowledge base that solves the relation linking task in question answering pipelines.

A collection of semantically-typed relational patterns corresponds to a bi-partite graph of patterns and properties in a knowledge base. A collection G of semantically-typed relational patterns is defined as a triple $G = (R, P, E)$, where:

- P and R are two disjoint sets representing semantic relational patterns and properties in a knowledge base (e.g., RDF properties from DBpedia or Yago), respectively.
- E is a set of pairs (r, p) in $R \times P$ representing a semantic type r of a relational pattern p, i.e., r is a property *semantically related* to p.

PATTY can be represented as a bi-partite graph $G = (R, P, E)$ where relational patterns in P are mined from large corpora, and properties in R correspond to the DBpedia predicates associated or *semantically related* to these patterns. Figure 6.5 illustrates a portion of a bi-partite graph for PATTY.

$$Rels(pattern(Q), G) = \{r \mid p \in pattern(Q) \text{ and } (p, r) \in E\} \tag{6.1}$$

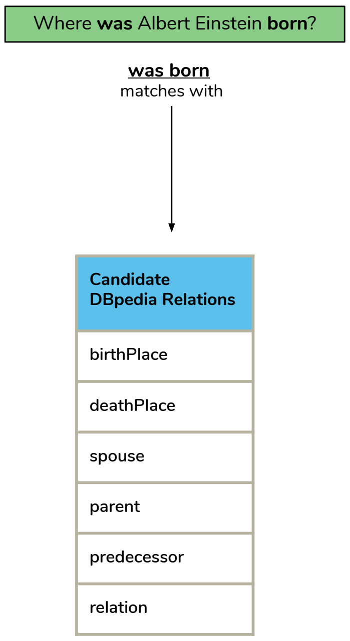
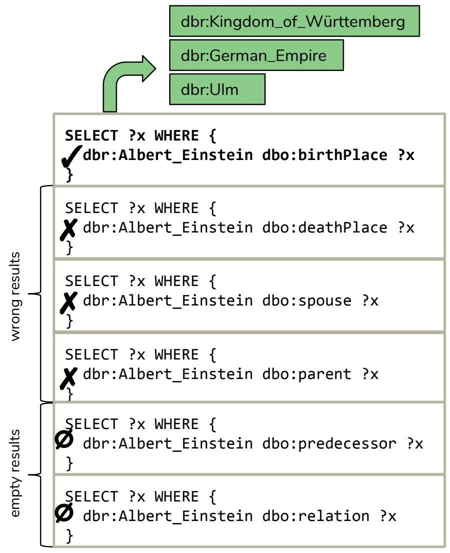


Figure 6.2: DBpedia predicates in PATTY associated with the pattern *was born* in the question.



```
SELECT ?x WHERE {
✓ dbr:Albert_Einstein dbo:birthPlace ?x
}
SELECT ?x WHERE {
✗ dbr:Albert_Einstein dbo:deathPlace ?x
}
SELECT ?x WHERE {
✗ dbr:Albert_Einstein dbo:spouse ?x
}
SELECT ?x WHERE {
✗ dbr:Albert_Einstein dbo:parent ?x
}
SELECT ?x WHERE {
∅ dbr:Albert_Einstein dbo:predecessor ?x
}
SELECT ?x WHERE {
∅ dbr:Albert_Einstein dbo:relation ?x
}
```

Figure 6.3: Potential SPARQL queries to answer the input question. Only the DBpedia predicate `dbo:birthPlace` allows for collecting correct answers.

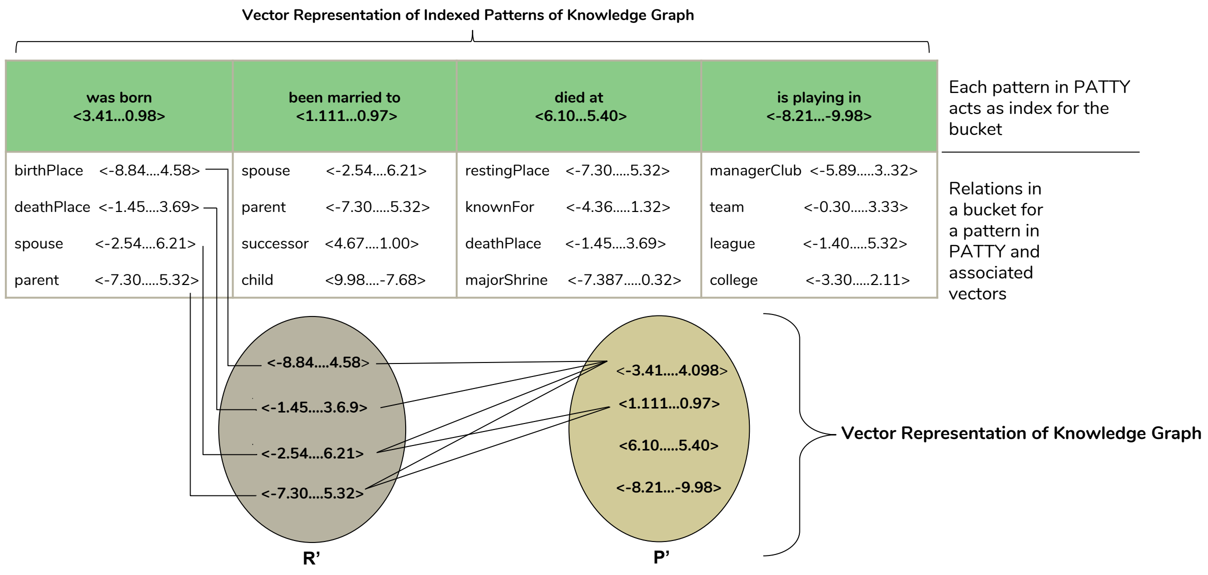

Vector Representation of Indexed Patterns of Knowledge Graph
was born <3.41...0.98>
been married to <1.111...0.97>
died at <6.10...5.40>
is playing in <-8.21...-9.98>
birthPlace <-8.84....4.58>
deathPlace <-1.45....3.69>
spouse <-2.54....6.21>
parent <-7.30.....5.32>
spouse <-2.54....6.21>
parent <-7.30.....5.32>
successor <4.67....1.00>
child <9.98....-7.68>
restingPlace <-7.30.....5.32>
knownFor <-4.36.....1.32>
deathPlace <-1.45....3.69>
majorShrine <-7.387.....0.32>
managerClub <-5.89.....3..32>
team <-0.30.....3.33>
league <-1.40.....5.32>
college <-3.30.....2.11>
Each pattern in PATTY acts as index for the bucket
Relations in a bucket for a pattern in PATTY and associated vectors
<-8.84....4.58>
<-1.45....3.6.9>
<-2.54....6.21>
<-7.30....5.32>
<-3.41....4.098>
<1.111...0.97>
<6.10.....5.40>
<-8.21...-9.98>
Vector Representation of Knowledge Graph
R'
P'

Figure 6.4: **Example of SIBKB on PATTY**. A portion of a Semantically Indexed Bi-partite Knowledge Base (SIBKB) for PATTY.

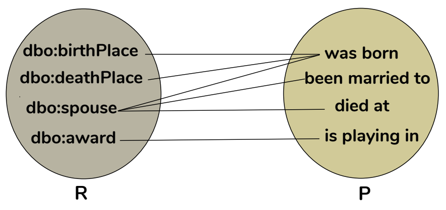


Figure 6.5: A portion of a Bi-partite Graph for PATTY

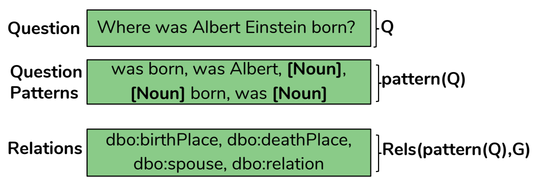
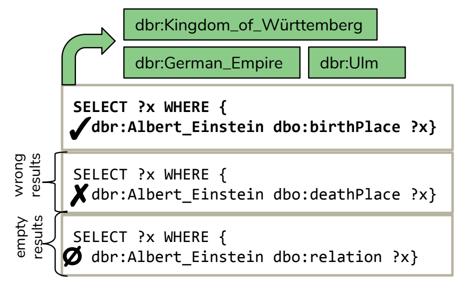

Question
Where was Albert Einstein born?
Q
Question Patterns
was born, was Albert, [Noun], [Noun] born, was [Noun]
pattern(Q)
Relations
dbo:birthPlace, dbo:deathPlace, dbo:spouse, dbo:relation
Rels(pattern(Q),G)

Figure 6.6: A question, its patterns, and corresponding relations (DBpedia properties) from PATTY

Figure 6.6 presents relational patterns of the question *Where was Albert Einstein born?*, as well as their associated semantic types in DBpedia. Semantic types associated with a pattern are used in question answering pipelines for building SPARQL queries whose evaluation will provide the answers of a question Q. For example, Figure 6.7 shows three SPARQL queries that can be built from the DBpedia predicates `dbo:birthPlace`, `dbo:deathPlace`, and `dbo:relation`.

Given a set Rel of semantic types or RDF properties in $Rels(pattern(Q), G)$, $f(Rel, D, Q, G)$ denotes a set of SPARQL queries over the knowledge base D that use predicates in Rel and that provide the *correct*

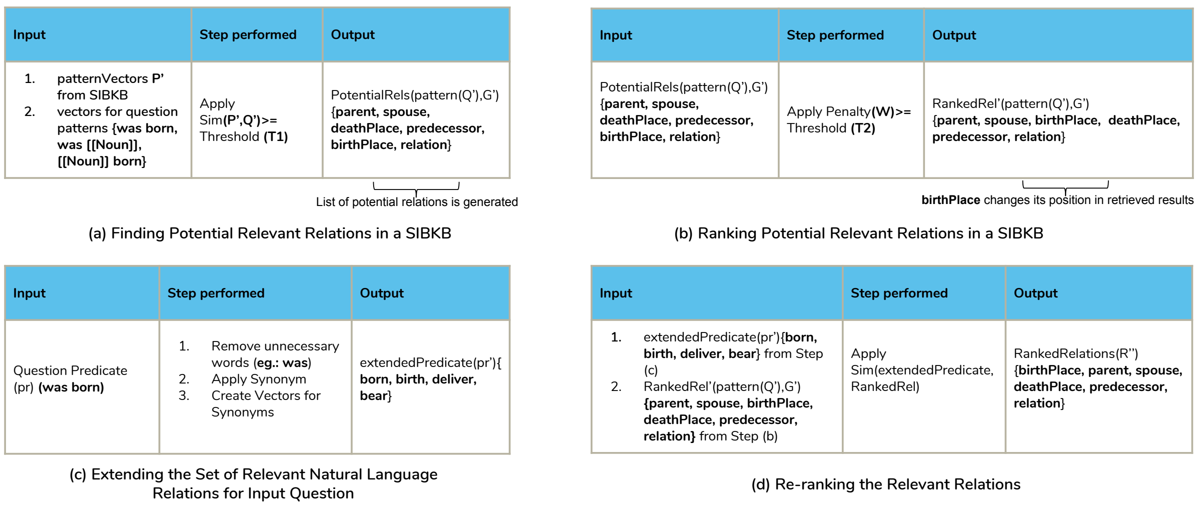


```
SELECT ?x WHERE {
✓dbr:Albert_Einstein dbo:birthPlace ?x}

SELECT ?x WHERE {
✗dbr:Albert_Einstein dbo:deathPlace ?x}

SELECT ?x WHERE {
∅ dbr:Albert_Einstein dbo:relation ?x}
```

Figure 6.7: Potential SPARQL queries from the selected DBpedia properties



Input	Step performed	Output
1. patternVectors P' from SIBKB 2. vectors for question patterns **{was born, was [[Noun]], [[Noun]] born}**	Apply Sim**(P',Q')**>= Threshold **(T1)**	PotentialRels(pattern(Q'),G') **{parent, spouse, deathPlace, predecessor, birthPlace, relation}**

List of potential relations is generated

(a) Finding Potential Relevant Relations in a SIBKB

Input	Step performed	Output
PotentialRels(pattern(Q'),G') **{parent, spouse, deathPlace, predecessor, birthPlace, relation}**	Apply Penalty**(W)**>= Threshold **(T2)**	RankedRel'(pattern(Q'),G') **{parent, spouse, birthPlace, deathPlace, predecessor, relation}**

(b) Ranking Potential Relevant Relations in a SIBKB

Input	Step performed	Output
Question Predicate (pr) **(was born)**	1. Remove unnecessary words **(eg.: was)** 2. Apply Synonym 3. Create Vectors for Synonyms	extendedPredicate(pr'){ **born, birth, deliver, bear}**

(c) Extending the Set of Relevant Natural Language Relations for Input Question

Input	Step performed	Output
1. extendedPredicate(pr')**{born, birth, deliver, bear}** from Step (c) 2. RankedRel'(pattern(Q'),G') **{parent, spouse, birthPlace, deathPlace, predecessor, relation}** from Step (b)	Apply Sim(extendedPredicate, RankedRel)	RankedRelations(R'') **{birthPlace, parent, spouse, deathPlace, predecessor, relation}**

(d) Re-ranking the Relevant Relations

Figure 6.8: **A SIBKB Relation Linking Pipeline**. Four-step pipeline exploiting SIBKB indices and captured knowledge.

answers for the question Q; $f(Rel, D, Q, G)$ is defined as follows:

$$\begin{aligned} f(Rel, D, Q, G) = \{ &Q(r) \mid r \in Rel \land \\ &Rel \subseteq Rels(Pattern(Q), G) \land \\ &Q(r) \in IdealQueries(Q, D)\} \end{aligned} \tag{6.2}$$

- $Q(r)$ is a SPARQL query composed of a triple pattern whose predicate is r;
- $IdealQueries(Q, D)$ represents a set that *only includes* the SPARQL queries that need to be run over D to produce the complete answer to the question Q.

In our running example, the resources `dbr:German_Empire`, `dbr:Kingdom_of_Württemberg`, and `dbr:Ulm` correspond to the complete answers of Q in DBpedia; one SPARQL query produces all these results, i.e., $IdealQueries(Q, D)$ is *only* composed of this query. Thus, although $f(Rel, D, Q, G)$ in Figure 6.7 includes this query, the other two queries in this set produce either incorrect or empty results for the input question.

Problem Statement Given a question Q and a collection G of semantically typed relational patterns, the problem of *linking relational patterns* in Q to semantic types from a knowledge base D corresponds to selecting a subset *Rel* of *Rels*(*pattern*(*Q*), *G*) from which the *maximal number* of SPARQL queries that produce the *correct answers* of *Q* can be generated. We define the problem of linking relational patterns in a question as the following optimisation problem:

$$\operatorname*{argmax}_{Rel \subseteq Rels(Pattern(Q),G)} \frac{|f(Rel, D, Q, G)|}{\max(|Rel|, |IdealQueries(Q, D)|)} \tag{6.3}$$

Since the set *IdealQueries*(*Q*, *D*) only includes one query in our running example, the optimal solution to this optimization problem corresponds to the set *Rel* that is only composed of the DBpedia property `dbo:birthPlace`. This property is part of the only triple pattern of the SPARQL query that produces the complete answer for the question *Q*.

Proposed Solution For matching the correct relations from a knowledge base for a given input question *Q*, we follow a two-step process. In the first step, a semantically indexed bi-partite knowledge graph (SIBKB) is built. In the second step, SIBKB is utilised in a pipeline for relation linking.

Semantically Indexed Bi-Partite Knowledge Base (SIBKB) : In the first step, we applied the GloVe [104] model to PATTY and built a vector representation of its bi-partite graph $G = (R, P, E)$, i.e., each node in R and P is replaced by its vector representation. PATTY is converted into $G' = (R', P', E')$ where R', P' are the vector representations of the semantically typed relational patterns and their associated DBpedia relations, respectively. Furthermore, a dynamic hashing [105] on semantically typed relational patterns is built; each entry in the hash table corresponds to a bucket composed of the predicates, e.g., in DBpedia, associated with the pattern in the *key* of the bucket. Figure 6.4 illustrates a portion of the SIBKB built on top of PATTY.

6.2.1 Pipeline for Relation Linking using a Semantically Indexed Bi-Partite Knowledge Base (SIBKB)

For finding the associated relation set *Rel* which is part of the set *Rels*(*pattern*(*Q*), *G*) (see Section 6.2), a four-step process is followed; Figure 6.8 illustrates the steps of this pipeline.

Finding potential relevant relations in SIBKB In this first step of the pipeline, we convert *pattern*(*Q*) into its vector representation *pattern*(*Q*′). We then calculate the cosine similarity between *pattern*(*Q*′) and the indexed semantically typed relational patterns *P*′ such that

$$Sim(pattern(Q'), P') \geq Threshold(T) \tag{6.4}$$

where *Threshold*(*T*) is the minimum admissible limit of the cosine similarity value. This results into a set of potential relevant relation vectors *potentialRels*′(*pattern*(*Q*′), *G*′) in SIBKB. In our example, the input for this step is the vector of question patterns, e.g., where, where was, was born, was [Noun], [Noun] born; the output is the list of vectors associated with potential relevant relations: `dbo:parent`, `dbo:spouse`, `dbo:relation`, `dbo:deathPlace`, `dbo:birthPlace`, `dbo:predecessor`.

Ranking potential relevant relations in SIBKB The numbers of occurrence of a particular pattern in PATTY is not uniform as illustrated in section 6.1. Therefore, it is likely that, while calculating the cosine similarity, some relations are ranked higher than others due to a higher number of associated matched patterns. To solve this issue, we applied a penalty function. For each relation *R* in PATTY, we first count the number of relational patterns associated with it; then this value is normalised by the total number of patterns in PATTY. The penalty function W is defined as follows:

$$W = 1 - \begin{bmatrix} count(P_{r,1})/count(P_{all}) \\ \cdots \\ count(P_{r,n})/count(P_{all}) \end{bmatrix}$$

$P_{r,1}, \cdots P_{r,n}$ are numbers of patterns for a relation, and P_{all} is the total number of relational patterns in PATTY. This step changes the ranking of the retrieved relations in Step 6.2.1. Therefore, *potentialRels*$'$(*pattern*(Q'), G') is now turned into the ranked relations *RankedRel*$'$(*pattern*(Q'), G'), which is the output of this pipeline step. In our example, the ranked list of relevant relations is updated from the list (`dbo:parent`, `dbo:spouse`, `dbo:deathPlace`, `dbo:predecessor`, `dbo:birthPlace`, `dbo:relation`) to (`dbo:parent`, `dbo:spouse`, `dbo:birthPlace`, `dbo:deathPlace`, `dbo:predecessor`, `dbo:relation`), i.e., the DBpedia predicate `dbo:birth-Place` is ranked in a higher position.

Extending the set of relevant natural language relations for the input Question Many times an irrelevant pattern appearing in a question, matches higher in number while calculating cosine similarities in the previous step. For example, the word ‘where’ appears 1,498 times in PATTY; this will negatively impact on the overall results. Therefore, to overcome this problem, we extract NL relations from the input question. In DBpedia, it is very likely that the DBpedia predicate associated has similar names with the NL predicate. For example, the NL relation ‘was born’ is associated with `dbo:birthPlace`, the relation ‘president of’ is associated with `dbo:President`, the relation ‘wife of’ is associated with `dbo:spouse` in the ranked list of DBpedia properties, and so on. Therefore, we extract *Predicate*(*Pr*) from the question Q; furthermore, we expand this list with synonyms from Wordnet. We then create vector representation of each of the relations in *extendedPredicate*(Pr') using the GloVe model. In our running example, the relation ‘was born’ is expanded to the list (born, birth, bear, deliver); it is converted further into its vector representation.

Re-ranking the relevant relations In the last step of the pipeline, we take the outputs of the second and third step, which correspond to the vector representation of ranked potential relations (*RankedRel*$'$(*pattern*(Q'), G')) and extended predicate patterns (*extendedPredicate*(Pr')). We again calculate cosine similarities between them to re-rank the list of obtained relations in *RankedRel*$'$(*pattern*(Q'), G').

In our example, the extended question predicate list from the third step is (born, birth, bear, deliver) and the ranked list of potential relations from the second step of the pipeline is (`dbo:parent`, `dbo:spouse`, `dbo:birthPlace`, `dbo:predecessor`, `dbo:relation`, `dbo:deathPlace`). After this step, the relation `dbo:birthPlace` has the highest similarity with *birth*, changing its position in the ranked list of relations. Therefore, our final re-ranked list of relations associated with the pattern *was born* is the following: (`dbo:birthPlace`, `dbo:parent`, `dbo:spouse`, `dbo:deathPlace`, `dbo:predecessor`, `dbo:relation`). The DBpedia predicate `dbo:birthPlace` is the top-1.

6.3 Experimental Study

We empirically study the efficiency and effectiveness of SIBKB for extracting properties from a knowledge base to solve the relation linking task. For this, we have integrated our tool in Qanary Ecosystem described in Section 5.2. In the first experiment, we assess the precision, recall, and F-Score of our approach using the QALD-7 benchmark. We address the following research questions: **RQi1)** What is

the impact of using an SIBKB on a relation linking task? **RQi2)** What is the impact of using an SIBKB on the relation linking execution time? **RQi3)** What is the impact of an SIBKB on the size of a collection of semantically-typed relational patterns?
The experimental configuration is as follows:
Relation Linking Benchmark In Section 5.2, we created a benchmark for entity linking task based on the QALD (Question Answering over Linked Data) benchmark used for evaluating complete QA systems. We devised a similar approach for the relation linking benchmark using the QALD-7 training set[3] that contains 215 questions.

Metrics: *i*) `Execution Time`: Elapsed time between the submission of a question to an engine and the delivery of the relevant DBpedia relations. Timeout is set to 300 seconds. *ii*) `Inv.Time`: It is calculated as: 1- (average execution time for BaseLine/average execution time for SIBKB) *iii*) `In Memory Size`: The Total size of the PATTY knowledge base and size of its corresponding SIBKB. *iv*) `Inv.Memory`: It is calculated as: 1- (Memory Size of PATTY/Memory Size of SIBKB) *v*) `Global Precision`: The number of correct relations retrieved at first rank in the list of retrieved relations out of the total number of questions. *vi*) `Global Recall`: The number of questions answered at any position (in our case till the 5th position of occurrence of a relation in the retrieved list) out of the total number of questions. *vii*) `F-Score`: Harmonic mean of global precision and global recall. *viii*) `Precision @ K`: The cumulative precision at position K. *ix*) `Recall @ K`: The correct relations for questions recommended in top K position out of total number of questions. *x*) `F-Score @ K`: Harmonic mean of precision and recall at position K.
Implementation: The pipeline for relation linking has been implemented in Python 2.7.12. Experiments were executed on a laptop with a quad-core 1.50 GHz Intel i7-4550U processor and 8GB RAM, running Fedora Linux 25. The word to vector conversion was done using GloVe [104]. Furthermore, for extracting NL predicates from the input question in the third step of the pipeline in section 6.2.1, we used the TextRazor API[4]. The source code and evaluation results can be downloaded from `https://github.com/WDAqua/ReMatch` for independent use.

		Cumulative Frequency at Rank Positions					**Precision @k**		**Recall @k**	**F-Score @k**
Num Properties	**Total**	**Rank#1**	**Rank#2**	**Rank#3**	**Rank#4**	**Rank#5**	**#1**	**#5**	**#5**	**#5**
1 Property	116	55	71	78	84	87	47.4%	57.6%	75.0%	63.2%
Properties	21	10	11	13	13	13	47.6%	53.1%	61.9%	57.2%
		5	10	14	14	15	23.80%	44.9%	71.4%	52.6%
Properties	6	1	1	1	1	1	16.6%	16.6%	16.6%	16.6%
		1	1	2	4	4	16.6%	30.5%	66.6%	41.8%
		1	1	2	2	2	16.6%	22.2%	33.3%	26.64%

Table 6.1: **SIBKB Performance.** Cumulative Frequency at Rank Positions 1 to 5; Precision, Recall, and F-Score are also reported at Top-1 and Top-5. Accuracy of the SIBKB-based relation linking method is enhanced whenever Top-5 results are considered.

[3] `https://github.com/ag-sc/QALD/blob/master/7/data/qald-7-train-multilingual.json`
[4] `https://www.textrazor.com/`

	Global		
	Precision	**Recall**	**F-Score**
Baseline	17%	37%	23%
SIBKB	51%	73%	60%

Table 6.2: **Comparison of SIBKB and Baseline.** Top-1 predicates are considered; SIBKB enhances accuracy of the proposed relation linking method.

6.3.1 Experiment 1: Performance Evaluation Using Relation Linking Benchmark

Evaluation of Relation Linking Task Using SIBKB

To evaluate the impact of the SIBKB on the relation linking task, we first calculate the performance of PATTY using a similarity measurement between question patterns and PATTY relational patterns using cosine similarity [104]; we call it 'BaseLine' approach. In the BaseLine approach, PATTY is directly used without indices. However, in our approach, we use SIBKB i.e., PATTY with indices along with the pipeline described in Section 6.2.1. Out of 215 questions of QALD-7, using PATTY patterns, we can answer 143 questions. The remaining 72 questions do not have any associated relational patterns for QALD questions in PATTY, and are therefore out of the scope for evaluation. Table 6.2 illustrates the results. Using our approach, the global precision increases from 17% to 51% compared to the BaseLine, which means a significant improvement of nearly three folds. The same analysis can be also seen in terms of the global recall and F-score.

We further observed the impact of our approach on capturing knowledge from the knowledge base by calculating the precision and recall values till the first five occurrences in the obtained list of relations. We divided questions with two or three properties into different groups as shown in Table 6.1. For example, the question 'Which professional surfers were born in Australia?', contains two DBpedia properties, namely, `dbo:occupation` and `dbo:birthPlace`.

Table 6.1 has two or three rows depending on the number of relations in a question. Using our SIBKB approach for relation linking, precision and recall at the first position are high enough to prove that our implementation can be easily used as relation linking tool in modular question answering frameworks such as OKBQA[13]; it will significantly improve the overall performance of QA systems in general. For example, QA system CASIA, which uses PATTY, shows an average precision of 0.35 over QALD-3 [57]. If its relation linking tool is replaced by our approach, this will improve the overall performance of the CASIA system. Furthermore, we have excluded a performance comparison with state-of-the-art relation linking tool presented in [19] because this work does not use the background knowledge base PATTY; it relies on modelling natural language relations with their underlying part of speech. The part of speech is then enhanced with Wordnet and dependency parsing. In contrast, our approach focuses on enhancing efficient knowledge capturing from knowledge bases for relation linking, which can further be extended for other similar knowledge bases like PATTY. However, combining both approaches will result in better performance of the relation linking task since relational patterns in PATTY are limited.

6.3.2 Experiment 2: Trade-offs between Different Metrics

We illustrate a trade-off between different dimensions of performance metrics for the SIBKB-based approach compared to the baseline. We choose global precision, global recall, F-score, in-memory size, and execution time as five different dimensions. The in-memory size of the PATTY knowledge base has increased from 7.34 MB to 22.44 MB as we have converted PATTY (two column corpus of relational

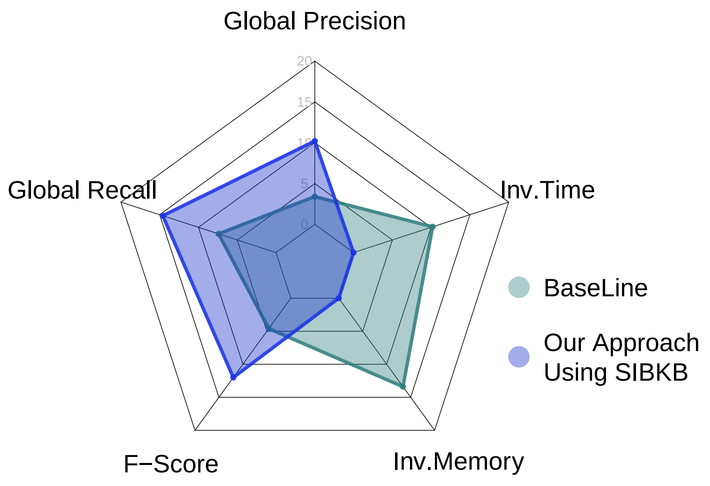


Figure 6.9: **Performance of SIBKB**. SIBKB and the Baseline are compared in terms of Global Precision, Global Recall, Global F-Score, Inv.Time, Inv.Memory; higher values are better. SIBKB increases Precision, Recall, and F-Score at the cost of evaluation time and memory consumption; Precision, Recall, and F-Score are improved by up to three times.

pattern and associated DBpedia relations) into SIBKB (indexed bipartite knowledge base from PATTY) using the GloVe model. Also, the average execution time per question is increased from 0.64 seconds to 5.96 seconds. A large portion of the total execution time of our implementation per question includes calling the TextRazor API (nearly 20 percent) for extracting NL predicates from the question. Figure 6.9 illustrates the trade-offs between these five dimensions. Although the SIBKB-based approach is more expensive by an order of magnitude in terms of memory consumption and execution time, it shows a drastic improvement with regard to precision, recall, and F-Score.

6.4 Summary

In this chapter, we addressed the need of building new components for relation linking task by reusing the knowledge encoded in PATTY knowledge base. Due to unavailability of independent components for other tasks except NED and NED, it is challenging to populate the Qanary Ecosystem with working full QA pipelines returning answer of the input questions. For this, we have presented the novel approach SIBKB – a semantic-based index which is able to capture knowledge encoded in background knowledge bases such as PATTY for the relation linking task. Many QA systems (e.g., [56] could not completely rely on PATTY owing to inherent noise, poor baseline performance, and in memory overheads. SIBKB is an approach that can be generalised for application on similar knowledge bases to alleviate these limitations. SIBKB indices allow not only for speeding up the search but also for reducing irrelevant relations appearing in the selection while efficiently and effectively matching natural language patterns to semantic relational patterns of knowledge bases. We demonstrate a case where semantically typed knowledge bases can now be fully utilised to a comparable degree to already successful graph types. SIBKB can, therefore, be integrated with other successful techniques for semantic disambiguation such as Wordnet similarity measures besides the inclusion of synonyms to extend precision of relation linking tools. We have integrated SIBKB component in Qanary ecosystem as a relation linking component.

CHAPTER 7

Semantic Composition of Question Answering Pipelines

Previous chapters, i.e., Chapter 4 and Chapter 5, focused on building a modular question answering (QA) framework for creating QA pipelines by reusing independent components. We introduced The Qanary ecosystem (c.f. Section 5.2) that supports the reusability of such QA components. The Qanary Ecosystem provides a framework for developing or even integrating QA systems but fails to systematically address how to formally describe and automatically compose existing QA components. In chapter 4, we defined QA components in form of Local As View (LAV) mappings of various components provide a systematic and homogeneous way to represent QA components using QAV as controlled vocabulary. Frameworks such as openQA [106], OKBQA [13], and QALL-ME [12] have attempted integration of QA components in a single platform, but there is no way to automatically compose QA pipelines within these QA frameworks. Similar problem continued in Qanary framework. QA system developer is expected to select the components manually to include them in the pipeline. This chapter studies the problem of effective composition of QA pipelines (Challenge 3 defined in Section 1.1). For this, we introduce *Qaestro*, a framework to semantically describe QA components and QA developer requirements and to produce QA component compositions based on these semantic descriptions. In particular, we utilise the QAV controlled vocabulary to model QA tasks and exploit the LAV approach [107] to express QA components. Furthermore, QA developer requests are represented as conjunctive queries involving the concepts included in the vocabulary. The QA Component Composition problem can be afterwards cast to the LAV *Query Rewriting Problem* (QRP) [30]. Then, state-of-the art SAT solvers [31] can find the solution models in the combinatorial space of all solutions which eventually correspond to QA component compositions. Using *Qaestro*, we formalised 51 QA components included in 20 distinct QA systems. In an empirical study, we show that *Qaestro* effectively enumerates possible combinations of QA components for different developer requirements to compose question answering pipelines. The content and details present in this chapter is based on following published articles: [9, 108]. At a higher level, the following research question is addressed in this chapter:

Research Question 3 (RQ3)

How can the process of composing QA pipelines be effectively automated ?

Towards addressing **RQ3** the following contributions are made:

- *Qaestro* - a framework for semantic-based composition of question answering pipelines;
- A controlled vocabulary to describe a QA system and developer requirements.
- An empirical evaluation of QAESTRO behaviour on QA developer requirements over the formalised components which are part of a QA framework.

The structure of the chapter is as follows: we introduce the problem of QA Component Composition in the context of a motivating example in 7.1. In Section 7.1.1 and 7.1.3, we introduce the QAESTRO framework and present its details respectively. The results of our evaluation are reported in 7.2. We provide a summary of the Chapter in Section 7.3.

7.1 Seamless Composition of Question Answering Pipelines

We motivate our work by discussing the problem of QA component composition in the context of the Open Knowledge Base and Question Answering (OKBQA) framework[1]. OKBQA considers QA as a predefined workflow consisting of four core modules providing Web service interfaces: (1) Template Generation Module for analysing a question in natural language and producing SPARQL query skeletons, (2) Disambiguation Module for mapping words or word sequences to Linked Data resources, (3) Query Generation Module for producing SPARQL queries based on modules (1) and (2), and finally, (4) Answer Generation Module for executing SPARQL queries to get the answers. Figure 7.1 illustrates an instantiation of a QA pipeline with the components OKBQA TGM v.2, OKBQA AGDISTIS, Sparqlator, and OKBQA AGM 2016 which implement the aforementioned modules (1)–(4), respectively[2]. Although OKBQA provides a public repository comprising several QA components that can be composed in the OKBQA pipeline, still several issues remain open for the QA system developer. First of all, there is no systematic way to identify other existing components – either standalone or parts of other QA systems – that could be part of the OKBQA pipeline. Secondly, there is no way to exploit OKBQA QA components in existing QA systems systematically. Thirdly, it is not clear whether and how other QA-related tasks and/or subtasks can be integrated in the OKBQA framework. For instance, let us consider the disambiguation task. Several components, such as Alchemy API[3], and DBpedia NED [17] may replace OKBQA AGDISTIS in the QA pipeline of Figure 7.1 since they perform conceptually the same QA task. Similarly, OKBQA AGDISTIS could serve the purpose of disambiguation in other QA systems as well. The same observation holds for other QA tasks that can participate in a QA pipeline. Based on the motivating example in previous section we identify following problems in existing QA frameworks:

- Due to missing semantic description of QA components within QA frameworks, it is challenging to combine QA components
- With the growing number of QA components in a framework, identifying all viable combinations of QA components that perform one or more tasks in combination requires a complex search in the large combinatorial space of solutions, which until now has to be performed manually.

To address the identified problems, and for seamless composition of a QA pipeline, we introduce *Qaestro*, which is a QA framework that allows for the composition of QA components into QA pipelines.

[1] http://www.okbqa.org/
[2] All components can be found at http://repository.okbqa.org.
[3] http://alchemyapi.com

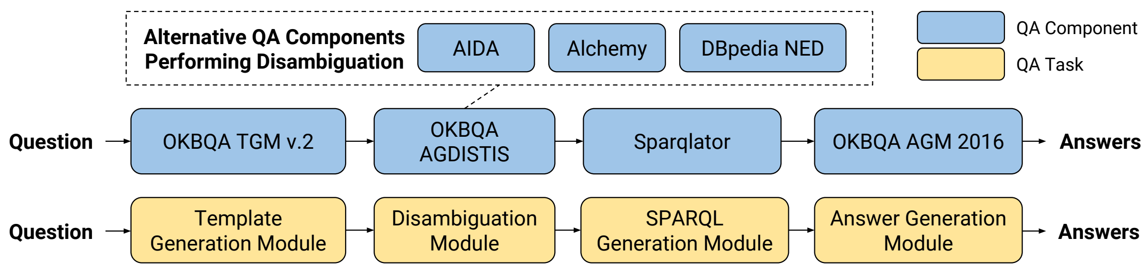


Figure 7.1: **OKBQA QA Pipeline and Pipeline Instance.** OKBQA pipeline consists of four components that implement four core modules: Template Generation Module, Disambiguation Module, Query Generation Module, and Answer Generation Module. In this example, the disambiguation task can be performed by OKBQA AGDISTIS, Alchemy, and DBpedia NED interchangeably.

Qaestro is based on QAV vocabulary (c.f. Section 4.1) which encodes the properties of generic QA tasks and is utilised to semantically describe QA components. *Qaestro* exploits semantic descriptions of QA components, and enumerates the compositions of the QA components that implement a given QA developer requirement. Thus, *Qaestro* provides a semantic framework for QA systems that not only enables a precise description of the properties of generic QA tasks and QA components, but also facilitates composition, integration, and reusability of semantically described QA components.

7.1.1 *Qaestro* Framework

Formally, *Qaestro* is defined as a triple ⟨QAV, QAC, QACM⟩, where: *i*) QAV is a domain vocabulary composed of predicates describing QA tasks, e.g., disambiguation or entity recognition; *ii*) QAC is a set of existing QA components that implement QA tasks, e.g., AGDISTIS [25] or Stanford NER [72]; and *iii*) QACM is a set of mappings that define the QA components in QAC in terms of the QA tasks that they implement. Mappings in QACM correspond to conjunctive rules, where the head is a predicate in QAC and the body is a conjunction of predicates in QAV. QA developer requirements are also represented as conjunctive queries over the predicates in QAC. Moreover, the problem of QA component composition corresponds to the enumeration of combinations of QA components that implement a QA developer requirement. In the following sections, *Qaestro* and the problem of QA composition are described.

7.1.2 Question Answering Developer Requirements

A QA developer requirement expresses the QA tasks that are required to be implemented by compositions of existing QA components. QA developer requirements are represented as conjunctive rules, where the body of a rule is composed of a conjunction of QA tasks. Similarly as for LAV mapping rules, input and output conditions can be represented; the symbol "*$*" denotes attributes assumed as input in the QA developer requirement. For instance, consider a developer who is interested in determining those compositions of QA components that, given a question q, perform entity recognition and disambiguation, and produce as output an entity e; the question q will be given as input to the pipeline.

```
QADevReq($q,e)-:recognition(q,e),disambig(e,q,de,t)
```

Now, suppose another developer requires also to know the compositions of QA components able to perform the pipeline of entity recognition and disambiguation. However, given the question as input, she requires to check all the intermediate results produced during the execution of the two tasks. In this case, the body of the rule remains the same, while the head of the rule (*QADevReq*) includes *all* variables corresponding to the arguments of the disambiguation task.

```
QADevReq($q,e,de,t):-recognition(q,e),disambig(e,q,de,t)
```

7.1.3 Composing QA Pipelines with Qaestro

In this section, we describe *Qaestro* as solution to the problem of QA component composition. We then describe the *Qaestro* architecture, and the main features of the *Qaestro* components.

Qaestro uses the QAV vocabulary that formalises QA tasks, and allows for the definition of QA components using LAV rules and QA developer requirements using conjunctive queries based on QAV. In this subsection, we will show how *Qaestro* solves the problem of QA Component Composition, i.e., how different QA components are automatically composed for a given developer requirement based on LAV mappings that semantically describe existing QA components. Consider the LAV mappings of few QA components described below:

```
Agdistis($x,$y,z):-disambig(x,y,z,t),entity(x),question(y),
                   disEntity(z)
StanfordNER($y,x):-recognition(y,x),question(y),entity(x)
DBpediaNER($y,x):-recognition(y,x),question(y),entity(x)
Alchemy($y,z):-disambig(x,y,z,t),question(y),disEntity(z)
Qakisatype($y,a):-answertype(y,a,o),question(y),atype(a)
```

Additionally, consider the following QA developer requirement for QA component compositions in a pipeline of entity recognition, disambiguation, and answer type identification, which receives a question *q* and outputs an entity *e*.

```
QADevReq($q,e):-recognition(q,e),disambig(e,q,de,t),answertype(q,a,o)
```

Qaestro generates two QA compositions as solutions to the problem of QA Component Composition. These compositions correspond to the enumeration of those combinations of QA components that implement the pipeline of the QA tasks of recognition, disambiguation, and answer type identification. Further, each composition satisfies the input restrictions of each QA component.

```
QADevReq($q,e):-StanfordNER($q,e),Agdistis($e,$q,de),
                Qakisatype($q,a)                      (1)
QADevReq($q,e):-DBpediaNER($q,e),Agdistis($e,$q,de),
                Qakisatype($q,a)                      (2)
```

Composition (1) indicates that the combination of the QA components Stanford NER, AGDISTIS, and Qakisatype implements the pipeline of recognition, disambiguation, and answer type identification. The input restriction of *StanfordNER*($\$q, e$) is satisfied by the question that is given as input in the pipeline. The QA component *Agdistis*($\$e, \$q, de$) is next in the composition; both the entity *e* produced by Stanford NER and the question *q* given by input to the pipeline, satisfy the input restrictions of this QA component. Similarly, input restriction of *Qakisatype*($\$q, a$) is satisfied by the question *q*. Additionally, Composition (2) implements the pipeline, but the QA component DBpedia NER is utilised for the QA task of entity recognition. The input restriction of DBpedia NER is also satisfied by the question received as input of the concerned question answering pipeline. Consider the following compositions for the same QA developer requirement:

```
QADevReq($q,e):-StanfordNER($q,e),Alchemy($q,de),
                Qakisatype($q,a)                 (3)
QADevReq($q,e):-DBpediaNER($q,e),Alchemy($q,de),
                Qakisatype($q,a)                 (4)
```

Both compositions implement the pipeline of recognition, disambiguation, and answer type identification; also the input restrictions of the QA components are satisfied. However, these compositions are not valid because the argument e that represents an entity is not generated by Alchemy. This argument is required to be joined with the entity produced by the QA component that implements the entity recognition task and to be output by the compositions.

Formally, the problem of QA Component Composition is cast to the problem of Query Rewriting using LAV views [109]. An instance of QRP receives a set of LAV rules on a set P of predicates that define sources in V, and a conjunctive query Q over predicates in P. The output of Q is the set of valid rewritings of Q on V. Valid rewritings QR of Q on V are composed of sources in V that meet the following conditions:

- Every source in QR implements at least one subgoal of Q.
- If S is a source in QR and implements the set of subgoals SG of Q, then
  - The variables in both the head Q and SG are also in the head of S.
  - The head of the LAV rule where S is defined, includes the variables in SG that are in other subgoals of Q.

Note that the QA component $Alchemy(q, de)$ violates these conditions in Composition (3) and (4), i.e., $Alchemy(q, de)$ does not produce an entity e for a question q. Thus, compositions that implement the QA task of disambiguation with $Alchemy(q, de)$ are not valid solutions for this QA developer requirement.

Qaestro casts the problem of QA Component Composition into the Query Rewriting Problem (QRP). Deciding if a query rewriting is a solution of QRP is NP-complete in the worst case [88]. However, given the importance of QRP in data integration systems and query optimisation, QRP has received a lot of attention in the Database area, and several approaches are able to provide effective and efficient solutions to the problem, e.g., MCDSAT [109, 110] or GQR [111]. Thus, building on existing solutions for QRP, we devise a solution to the problem of QA Component Composition that is able to efficiently and effectively enumerate valid compositions of a QA developer requirement. *Qaestro* implements a two-fold approach, where first, solutions to the cast instance of QRP are enumerated. Then, input and output restrictions of QA components are validated. Valid compositions of QA components that both implement a QA developer requirement and respect the input and output restrictions, are produced as solutions of an instance of the problem of QA Component Composition.

7.1.4 The *Qaestro* Architecture

Qaestro relies on MCDSAT, a state-of-the-art solver of QRP to efficiently enumerate the compositions of QA components that correspond to implementations of a QA developer requirement. Figure 8.4 depicts the *Qaestro* architecture. *Qaestro* receives as input a QA developer requirement QADR expressed as a conjunctive query over QA tasks in a vocabulary QAV. Furthermore, a set QACM of LAV rules describing QA components in terms of QAV is given as input to *Qaestro*. QACM and QADR correspond to an instance of the QA Component Composition which is *cast* into an instance of QRP and passed to MCDSAT, a solver of QRP. MCDSAT encodes the instance of QRP into a CNF theory in a way that

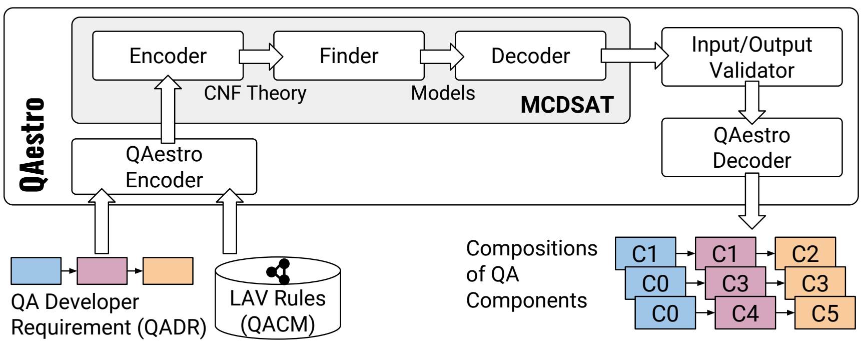


Figure 7.2: ***Qaestro* Architecture.** *Qaestro* receives as input a QA developer requirement QADR and a set QACM of LAV rules describing QA components, and produces all the valid compositions that implement QADR.

models of this theory correspond to solutions of QRP. MCDSAT utilises an off-the-shelf SAT solver to enumerate all *valid* query rewritings that correspond to *models* of the CNF theory. The output of the SAT solver is decoded, and input and output restrictions are validated in each query rewriting. Finally, *Qaestro* decodes valid query rewritings where input and output restrictions are satisfied, and generates the compositions of QA components that implement the pipeline of QA tasks represented by QADR.

7.2 Empirical Study

We empirically study the behavior of *Qaestro* in generating possible QA component compositions given QA developer requirements. We assess the following research questions: 1. **RQj1**: Given the formal descriptions of QA components using QAV and QA developer requirements are we able to produce sound and correct compositions? 2. **RQj2**: Are we able to produce efficiently solutions to the problem of QA Component Composition? The experimental configuration is as follows:

QA Components and developer requirements To evaluate *Qaestro* empirically, we have semantically described 51 QA components implemented by 20 QA systems which have participated in the first five editions of the QA over Linked Data Challenge (QALD1–5)[4]. Additionally, we studied well-known QA systems such as AskNow [56], TBSL [8], and OKBQA to semantically describe their components. After closely examining more than 50 components of these QA systems, we broadly categorised the components based on the QA tasks they perform, as defined in Section 2.3. For defining the LAV mappings, we selected only those QA components, for which there is a clear statement about input, output, and the QA tasks they perform in a publication (i.e., scientific paper, white paper, or source repository) about the respective QA system. Furthermore, we constructed manually 30 QA developer requirements for standalone QA tasks and QA pipelines integrating various numbers of tasks.

Metrics *i) Number of QA component compositions*: Number of QA component compositions given the semantic descriptions of QA components in QACM and a QA developer requirement; *ii) Processing Time*: Elapsed time between the submission of a QA developer requirement and the arrival of all the QA component compositions produced by *Qaestro*.

[4] http://qald.sebastianwalter.org/

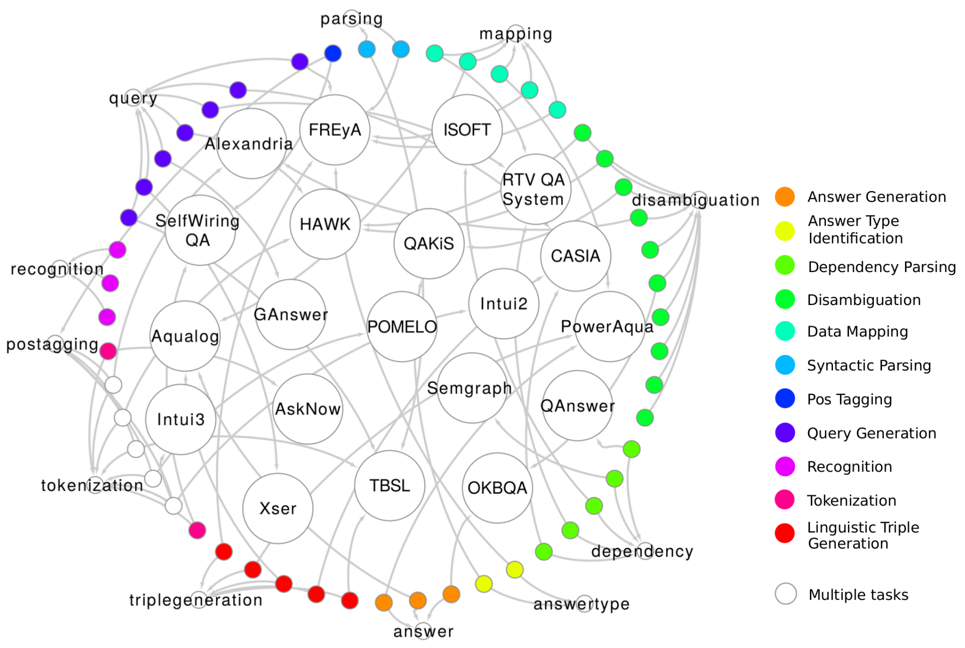


Figure 7.3: **QA Systems, Components, and Tasks** 51 QA components from 20 QA systems over DBpedia, implementing 11 distinct QA tasks are depicted as a directed graph.

Implementation *Qaestro* is implemented in Python 2.7 on top of MCDSAT [109], which solves QRP with the use of the off-the-shelf model compilation and enumeration tool c2d[5]. *Qaestro* source code can be downloaded from `https://github.com/WDAqua/Qaestro` and the evaluation results can be viewed at `https://wdaqua.github.io/Qaestro/`. Experiments were executed on a laptop with Intel i7-4550U, 4x1.50GHz and 8GB RAM, running Fedora Linux 25.

7.2.1 Evaluation Results

Analysis of QA Components In Figure 7.3, we illustrate all QA components that have been formalised using *Qaestro* along with their connections to the QA tasks they implement and the QA systems they belong to as an undirected graph[6]. In total, the resulting graph consists of 82 nodes and 102 edges. From the 82 nodes, 20 correspond to QA systems, 11 represent QA tasks, and 51 refer to concrete QA components – 43 are part of the QA systems while 8 are provided also as standalone components (e.g., AGDISTIS, DBpedia NER, etc.). It can be observed in Figure 7.4 that the majority of the analyzed QA components implement the Disambiguation task (10 in total) followed by the Query Generation (8), Tokenisation (8), and POS Tagging (7) tasks. Many of these components are reused among the different QA systems. In addition, Figure 7.5 shows that in almost half of the QA systems, components that implement Tokenisation and Query Generation are included, while some less popular QA tasks like Answer Type Identification and Syntactic Parser are part of only two QA systems.

[5]`http://reasoning.cs.ucla.edu/c2d/`
[6]The graph visualisation was generated with cytoscape - `http://www.cytoscape.org`.

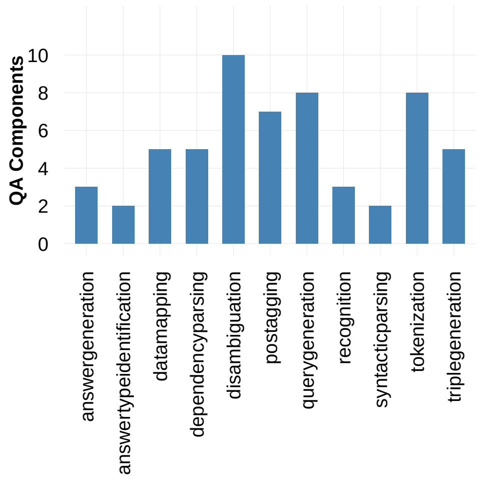


Figure 7.4: QA Components per QA Task

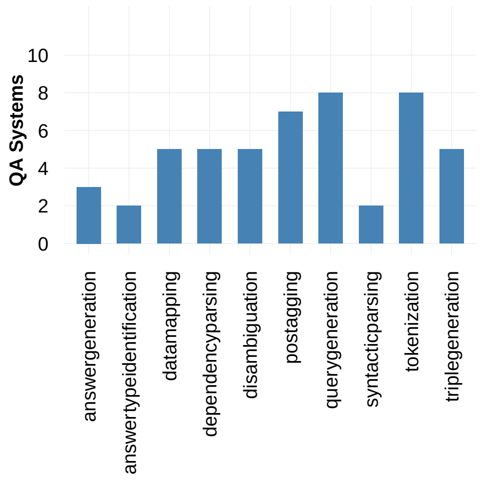


Figure 7.5: QA Systems per QA Task

7.2.2 Discussion

QA Component Compositions In order to evaluate the efficiency of *Qaestro*, we edited 30 QA developer requirements with different number of QA tasks to be included in the QA pipeline and different expected inputs and outputs. Given these requirements and the semantic descriptions of QA components *Qaestro* produced possible QA component compositions. Figure 7.6 reports on the number of different compositions for all 30 requirements grouped according to the number of QA tasks they include. Figure 7.7 demonstrates the time needed by *Qaestro* to process each of the requirements and generate QA component compositions. We performed the measurements 10 times and calculated the mean values.

While for standalone QA components or components that perform two tasks the solution space is relatively limited – from one to 30 combinations – for QA developer requirements that include three or more QA tasks the number of QA compositions may increase significantly. For instance, we notice

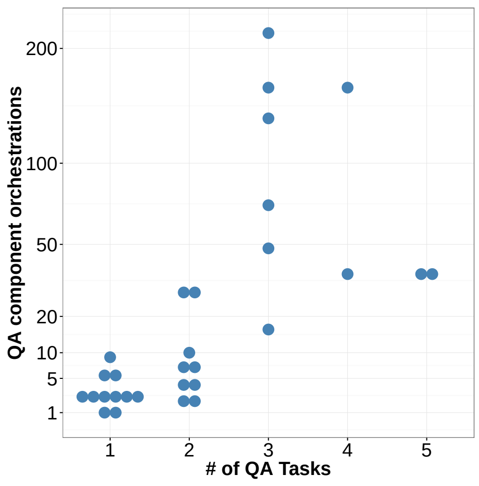


Figure 7.6: QA Component Compositions per # of QA Tasks in the Pipeline

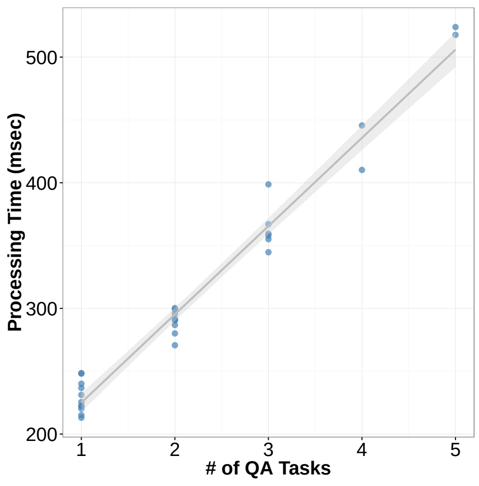
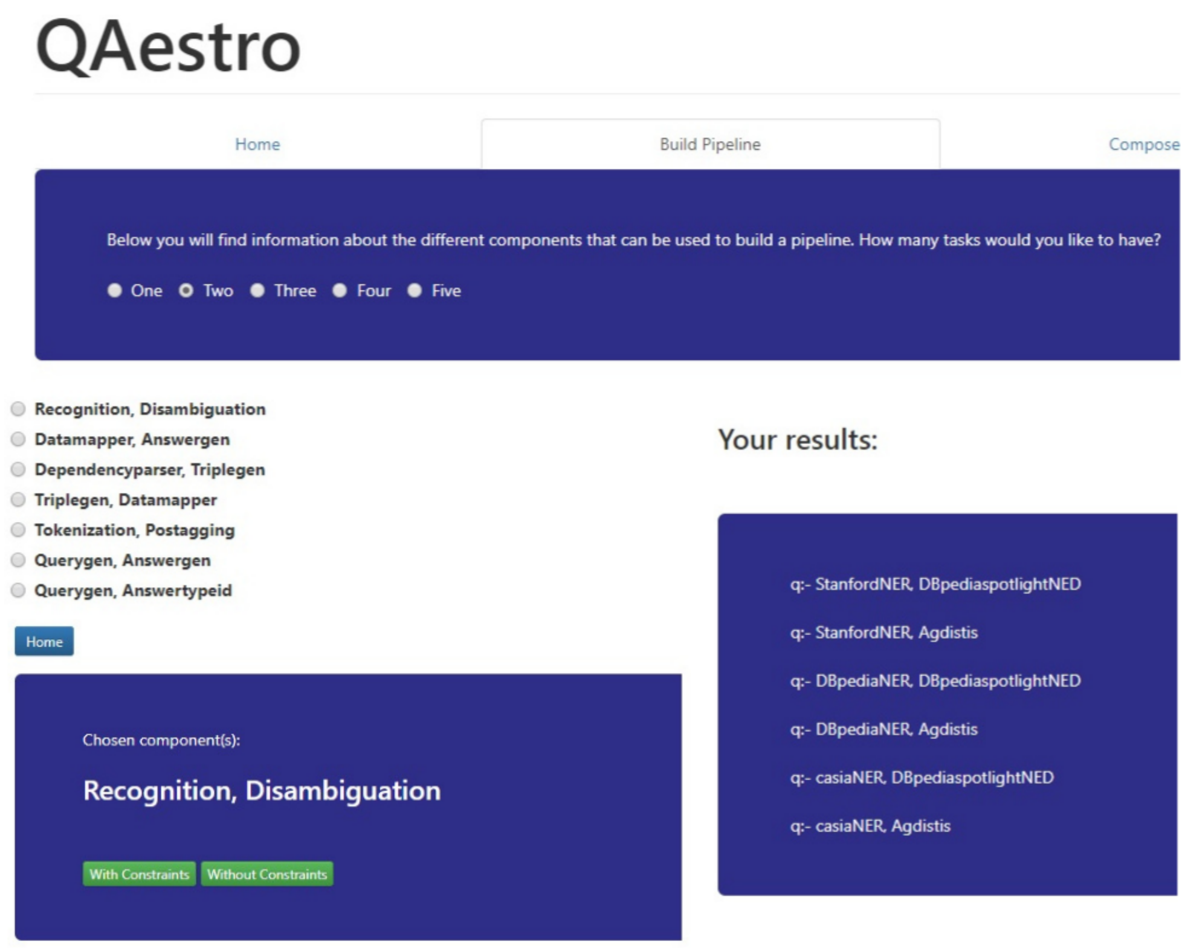


Figure 7.7: Execution Time for Generating QA Component Compositions

that for a few requirements with three and four QA tasks the possible compositions are more than 100. In these cases, the requirements do not foresee input or output dependencies between QA components, hence, the number of possible combinations increases significantly. All solutions produced by *Qaestro* are sound and complete, since MCDSAT is able to produce every valid solution and all solutions that it provides are valid [109]. Furthermore, the processing time is for all requirements less than half a second and relates linearly to the number of QA tasks, since MCDSAT can perform model enumeration in linear time. Consequently, the experimental results allow us to positively answer **RQj1** and **RQj2** described in this section.



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

● One [✓ Two] ● Three ● Four ● Five

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

Chosen component(s):

Recognition, Disambiguation

Your results:

q:- StanfordNER, DBpediaspotlightNED

q:- StanfordNER, Agdistis

q:- DBpediaNER, DBpediaspotlightNED

q:- DBpediaNER, Agdistis

q:- casiaNER, DBpediaspotlightNED

q:- casiaNER, Agdistis

Figure 7.8: Qaestro UI: Qaestro is integrated with Qanary and the developer can compose and run QA pipelines using a user interface.

7.3 Summary

In this chapter, we have tackled the problem of QA Component Composition by casting it to the Query Rewriting Problem. We introduced QAESTRO, a framework that enables QA developers to semantically describe QA components and developer requirements by exploiting the LAV approach. Moreover, QAESTRO computes compositions of QA components for a given QA developer requirement by taking advantage of SAT solvers. In an empirical evaluation, we tested QAESTRO with various QA developer requirements for QA pipelines of varying complexity, containing from two to five tasks. We observed that QAESTRO can not only produce sound and valid compositions of QA components, but also demonstrates efficient processing times. QAESTRO can successfully deal with the growing number of QA systems and standalone QA components, that is, the appearance of a new QA component only causes the addition of a new mapping describing the QA component in terms of the concepts in the QA vocabulary. Automated composition of QA components will enable subsequent research towards determining and executing best-performing QA pipelines that achieve better performance in terms of accuracy (precision, recall) and execution time. Currently, QAESTRO is not capable of implementing the QA pipeline in an automated way to answer an input question, however,we have integrated QAESTRO in Qanary Ecosystem to automatically

Functionality	**Qaestro**	**Qanary Ecosystem**
Resolves interoperability at implementation level	–	✓
Resolves interoperability at logical level	✓	–
Vocabulary used	QAV vocabulary	`qa` vocabulary
Promotes reusability	✓	✓
Allows benchmarking of components	–	✓
Automatic composition of QA pipelines	✓	–

Table 7.1: Comparison of *Qaestro* and Qanary Ecosystem

retrieve all feasible combinations of available QA components and to realise the best-performing QA pipeline in concrete use cases. Using a user interface as illustrated in Figure 7.8, user can choose the components, and compose and execute the QA pipelines.

It is important to note that *Qaestro* framework and Qanary solve two different dimensions of the overall identified problem in this thesis. Firstly, Qanary methodology and Ecosystem are concerned with resolving interoperability issues at the implementation level, and aim towards bringing heterogeneous QA components under a single umbrella to connect them physically. However, aim here is not to allow a QA developer to assist in QA pipeline composition. Once QA components are integrated in the Qanary Ecosystem using Qanary methodology, QA developer (or user) needs to run the pipelines by selecting the components manually. There is no semantic description of the components that allows QA developer to understand the functionalities of integrated component, and exhibit easy exchange of the components performing the similar tasks. *Qaestro* addresses this problem and assist QA developer to compose the pipelines automatically. In other words, just by specifying QA developer requirements as semantic mappings and using QAV vocabulary, *Qaestro* compose pipelines of integrated components within a framework (Qanary Ecosystem or OKBQA). Also, *Qaestro* is independent of QA framework. It can be easily reused with any of the available QA framework (OKBQA, openQA etc.) on top of it using an in built User Interface. Therefore, *Qaestro* framework is the solution to resolve interoperability at logical level when components are integrated in a single platform. We summarise differences/functionalities of *Qaestro* and Qanary in the Table 7.1.

CHAPTER 8

Dynamic Composition of Question Answering Pipelines

Modern QA systems need to flexibly integrate a number of components specialised to fulfil specific tasks in a QA pipeline. These systems typically include components building on Artificial Intelligence, Natural Language Processing, and Semantic Technology; they implement common tasks such as Named Entity Recognition and Disambiguation, Relation Extraction, and Query Building. In Chapter 4 we present a generalised ontology (i.e. `qa` vocabulary) which covers the need for interoperability of QA systems on a conceptual level. Using the `qa` vocabulary as foundation, we devise Qanary methodology for integrating heterogeneous QA components in a single platform. Qanary led to the development of Qanary Ecosystem, which is a framework for integrating QA components. On top of Qanary Ecosystem, we integrated QAestro framework which we have described in Chapter 7 for automatic composition of QA pipelines. Evaluation studies have shown that there is no best performing QA system for all types of Natural Language (NL) questions; instead, there is evidence that certain systems, implementing different strategies, are more suitable for certain types of questions [14]. Hence, modern QA systems need to flexibly integrate a number of components specialised to fulfil specific tasks in a QA pipeline. This is also necessary to tackle scalability of components i.e. in case when large number of QA components are available for same task. Therefore, we address following research question in this chapter:

Research Question 4 (RQ4)

How can effective dynamic QA pipelines be composed by reusing components?

To address this question, we devise FRANKENSTEIN, a framework able to dynamically select QA components in order to exploit the properties of the components to optimise the F-Score. We consider the scalability of QA components in the framework and FRANKENSTEIN implements a classification based learning model, which estimates the performance of QA components for a given question, based on its features. Given a question, the FRANKENSTEIN framework implements a greedy algorithm to generate a QA pipeline consisting of the best performing components for the particular question from the user. We empirically evaluate the performance of FRANKENSTEIN using two renowned benchmarks from the Question Answering over Linked Data Challenge[1] (QALD) and the Large-Scale Complex Question

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

Answering Dataset[2] (LC-QuAD). We observe that FRANKENSTEIN is able to combine QA components to produce optimised QA pipelines outperforming the static Baseline pipeline.
Towards addressing **RQ4** the following contributions are made:

- FRANKENSTEIN framework relying on machine learning techniques for dynamically selecting suitable QA components and composing QA pipelines based on the input question, thus optimising the overall F-Score.
- A collection of 29 reusable QA components that can be combined to generate 360 distinct QA pipelines, integrated in the FRANKENSTEIN framework.
- An in-depth analysis of advantages and disadvantages of QA components in QA pipelines after a thorough benchmarking of the performance of the FRANKENSTEIN pipeline generator using over 3,000 questions from the QALD and LC-QuAD QA benchmarks.

8.1 Predicting Best Performing Components

A full QA pipeline is composed of all the necessary tasks to transform a user-supplied Natural Language (NL) question into a query in a formal language (e.g., SPARQL), whose evaluation retrieves the desired answer(s) from an underlying knowledge graph. Correctly answering a given input question q requires a QA pipeline that, ideally, uses those QA components that deliver the best precision and recall for answering q. Identifying the best performing QA pipeline for a given question q requires: (i) a prediction mechanism to predict the performance of a component given a question q, a required task, and a knowledge graph λ; (ii) an approach for composing an optimised pipeline by integrating the most accurate components in the pipeline.

In this context, we formally define a set of necessary QA tasks as $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ such as NED, RL, and QB. Each task $(t_i : q^* \rightarrow q^+)$ transforms a given representation q^* of a question q into another representation q^+. For example, NED and RL tasks transform the input representation *"What is the capital of Canada?"* into the representation *"What is the* `dbo:capital` *of* `dbr:Canada`*?"*. The entire set of QA components is denoted by $C = \{C_1, C_2, \ldots, C_m\}$. Each component C_j solves one single QA task; $\theta(C_j)$ corresponds to the QA task t_i in $\mathcal{T}$ implemented by C_j. For example, ReMatch implements the relation linking QA task, i.e., $\theta(ReMatch) = RL$. Let $\rho(C_j)$ denote the performance of a QA component, then our first objective is to predict the likelihood of $\rho(C_j)$ for a given representation q^* of q, a task t_i, and an underlying knowledge graph λ. This is denoted as $Pr(\rho(C_j)|q^*, t_i, \lambda)$. In this work, we assume a single knowledge graph (i.e., DBpedia); thus, λ is considered a constant parameter that does not impact the likelihood leading to:

$$Pr(\rho(C_j)|q^*, t_i) = Pr(\rho(C_j)|q^*, t_i, \lambda) \tag{8.1}$$

Moreover, for each individual task t_i and question representation q^*, we predict the performance of all pertaining components. In other words, for a given task t_i, the set of components that can accomplish t_i is $C^{t_i} = \{C_j, \ldots, C_k\}$. Thus, we factorise t_i as follows:

$$\forall C_j \in C^{t_i}, [Pr(\rho(C_j)|q^*) = Pr(\rho(C_j)|q^*, t_i)] \tag{8.2}$$

Further, we assume that the given representation q^* is equal to the initial input representation q for all the QA components, i.e., $q^* = q$. Finally, the problem of finding the best performing component for

[2] `http://lc-quad.sda.tech/`

accomplishing the task t_i for an input question q, denoted as $\gamma_q^{t_i}$, is formulated as follows:

$$\gamma_q^{t_i} = \arg\max_{C_j \in C^{t_i}} \{Pr(\rho(C_j)|q)\} \tag{8.3}$$

Solution Suppose we are given a set of NL questions Q with the detailed results of performance for each component per task. We can then model the prediction goal $Pr(\rho(C_j)|q, t_i)$ as a supervised learning problem on a training set, i.e., a set of questions Q and a set of labels $\mathcal{L}$ representing the performance of C_j for a question q and a task t_i. In other words, for each individual task t_i and component C_j, the purpose is to train a supervised model that predicts the performance of the given component C_j for a given question q and task t_i leveraging the training set. If $|\mathcal{T}| = n$ and each task is performed by m components, then $n \times m$ individual learning models have to be built up. Furthermore, since the input questions $q \in Q$ have a textual representation, it is necessary to automatically extract suitable features, i.e., $\mathcal{F}(q) = (f_1, \dots, f_r)$. The details of the feature extraction process are presented in Subsection 8.4.2.

8.1.1 Identifying Optimal QA Pipelines

The second problem deals with finding a best performing pipeline of QA components ψ_q^{goal}, for a question q and a set of QA tasks called *goal*. Formally, we define this optimisation problem as follows:

$$\psi_q^{goal} = \arg\max_{\eta \in \mathcal{E}(goal)} \{\Omega(\eta, q)\} \tag{8.4}$$

where $\mathcal{E}(goal)$ represents the set of pipelines of QA components that implement *goal* and $\Omega(\eta, q)$ corresponds to the estimated performance of the pipeline η on the question q.

Solution We propose a greedy algorithm that relies on the *optimisation principle* that states that an optimal pipeline for a goal and a question q is composed of the best performing components that implement the tasks of the goal for q. Suppose that $\oplus$ denotes the composition of QA components, then an optimal pipeline ψ_q^{goal} is defined as follows:

$$\psi_q^{goal} := \oplus_{t_i \in goal} \{\gamma_q^{t_i}\} \tag{8.5}$$

The proposed greedy algorithm works in two steps: *QA Component Selection* and *QA Pipeline Generation*. During the first step of the algorithm, each task t_i in *goal* is considered in isolation to determine the best performing QA components that implement t_i for q, i.e., $\gamma_q^{t_i}$. For each t_i an ordered set of QA components is created based on the performance predicted by the supervised models that learned to solve the problem described in Equation 8.3. Figure 8.1 illustrates the QA component selection steps for the question q=*"What is the capital of Canada?"* and $goal = \{NED, RL, QB\}$. The algorithm creates an ordered set OS_{t_i} of QA components for each task t_i in *goal*. Components are ordered in each OS_{t_i} according to the values of the performance function $\rho(.)$ predicted by the supervised method trained for questions with the features $\mathcal{F}(q)$ and task t_i; in our example, $\mathcal{F}(q)$={(QuestionType:What), (AnswerType:String), (#words:6), (#DT:1), (#IN:1), (#WP:1), (#VBZ:1), (#NNP:1), (#NN:1)} indicates that q is a *WHAT* question whose answer is a *String*; further, q has six words and POS tags such as determiner, noun etc. Based on this information, the algorithm creates three ordered sets: OS_{NED}, OS_{RL}, and OS_{QB}. The order in OS_{NED} indicates that Dandelion[3], Tag Me, and DBpedia Spotlight are the top 3

[3]https://dandelion.eu/docs/api/datatxt/nex/getting-started/

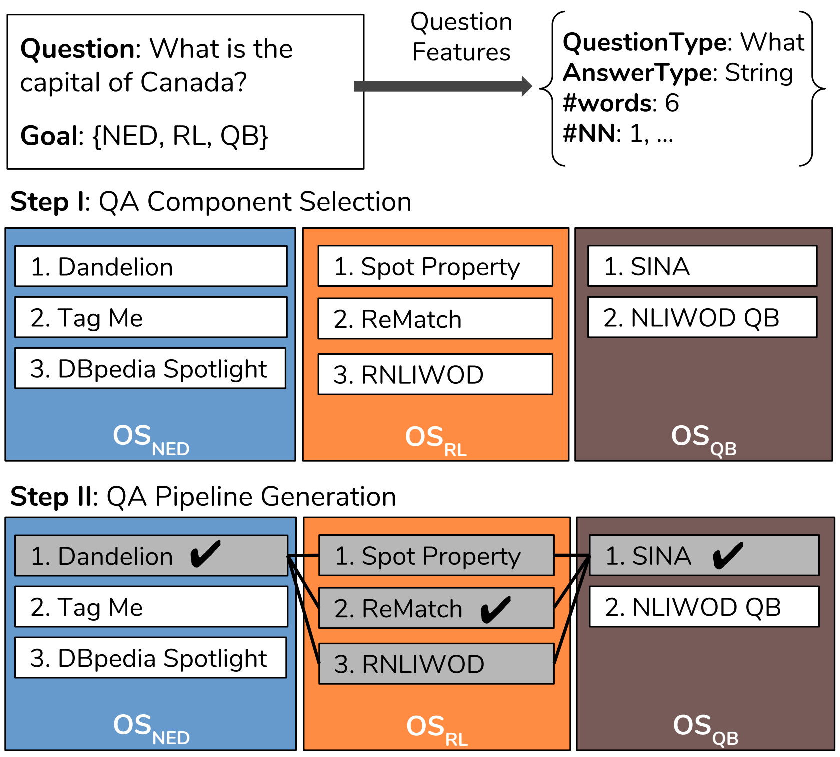


Figure 8.1: **QA Optimisation Pipeline Algorithm**. The algorithm performs two steps: First, QA components are considered in isolation; supervised methods are used to predict the top k best performing QA components per task and question features. Second, the QA Pipelines are generated from the best performing QA component of the tasks NED and QB, and the top 3 QA components of RL. The QA pipeline formed of Dandelion, ReMatch, and SINA successfully answers q.

best performing QA components for queries with the features $\mathcal{F}(q)$ in the QA task NED; similarly, for OS_{RL} and OS_{QB} sets.

In the second step, the algorithm follows the optimisation principle in Equation 8.5 and combines the top k_i best performing QA components of each ordered set. Values of k_i can be configured; however, we have empirically observed that for all studied types of questions and tasks, only the relation linking (RL) task requires considering the top 3 best performing QA components; for the rest of the tasks, the top 1 best performing QA component is sufficient to identify a best performing pipeline. Once the top k_i QA components have been selected for each ordered set, the algorithm constructs a QA pipeline and checks if the generated pipeline is able to produce a non-empty answer. If so, the generated QA pipeline is added to the algorithm output. In Equation 8.5, the algorithm finds that only the QA pipeline Dandelion, ReMatch, and SINA produces results; the other two pipelines fail because the QA components RNLIWOD[4] and Spot Property[5] are not able to perform the relation linking task of the question q=*"What is the capital of Canada?"*. The algorithm ends when the top k_i QA components have been combined and checked; the output is the union of the best performing QA pipelines that produce a non-empty answer.

[4]Component based on `https://github.com/dice-group/NLIWOD`.
[5]This component is the combination of the NLIWOD and RL components of [13].

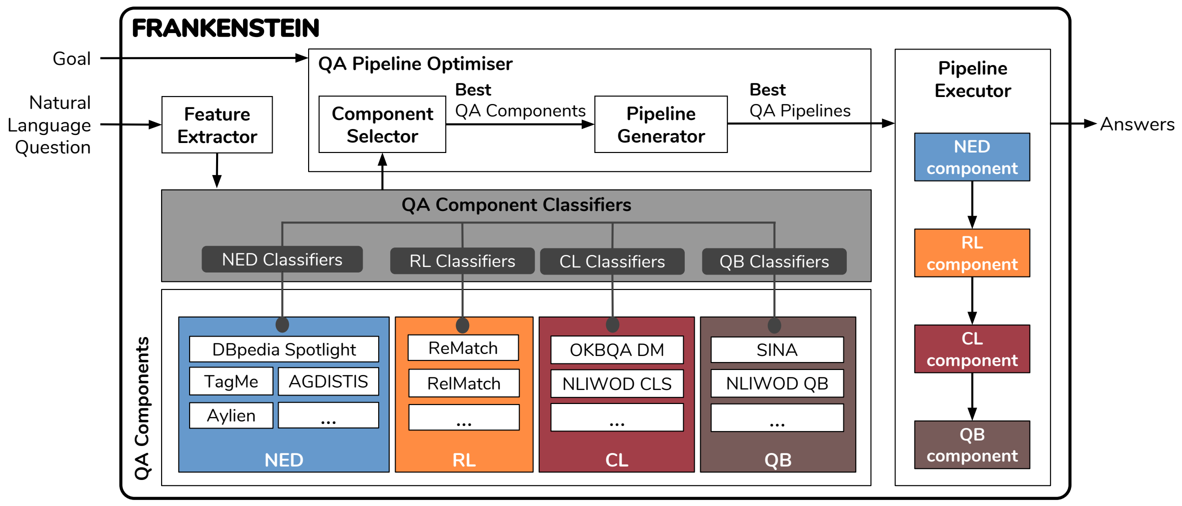


Figure 8.2: FRANKENSTEIN architecture comprising separate modules for question feature extraction, pipeline generation and optimisation, as well as pipeline execution.

8.2 Scalability of QA Components

In the previous chapter, we have presented Qanary methodology and Qanary framework. To address the fourth research question, we integrate 29 QA components using Qanary methodology implementing five QA tasks, namely Named Entity Recognition (NER), Named Entity Disambiguation (NED), Relation Linking (RL), Class Linking (CL), and Query Building (QB) in Qanary ecosystem. To the best of our knowledge, only two reusable CL and QB components, and five reusable RL components are available, therefore the component distribution among tasks is uneven. In most of the cases NED, RL and QB components are necessary to generate the SPARQL query for a NL question. However, to correctly generate a SPARQL query for certain NL questions, it is sometimes necessary to also disambiguate classes against the ontology. For example, in the question *"Which comic characters are painted by Bill Finger"*, *"comic characters"* needs to be mapped to `dbo:ComicsCharacter`[6]. Table 8.1 provides a list of QA components integrated in proposed FRANKENSTEIN. FRANKENSTEIN framework is build on top of Qanary framework. The 11 NER components are used with AGDISTIS to disambiguate entities as AGDISTIS requires the question and spotted position of entities as input [25]. Henceforth, any reference to NER tool, will refer to its combination with AGDISTIS, and we have excluded individual performance analysis of NER components. However, other 7 NED components recognise and disambiguate the entities directly from the input question.

8.3 FRANKENSTEIN Framework

FRANKENSTEIN is a framework that implements the QA optimisation pipeline algorithm and generates the best performing QA pipelines based on the input question features and QA goal.

[6]http://dbpedia.org/ontology/ComicsCharacter

8.3.1 FRANKENSTEIN Architecture

Figure 8.2 depicts the FRANKENSTEIN architecture. FRANKENSTEIN receives, as input, a natural language question as well as a goal consisting of the QA tasks to be executed in the QA pipeline. The features of an input question are extracted by the *Feature Extractor*; afterwards the *QA Component Classifiers* predict best performing components per task for the given question; these components are passed to the *Pipeline Generator*, which generates best performing pipelines to be executed, eventually, by the *Pipeline Executor*. The FRANKENSTEIN architecture comprises the following modules:
Feature Extractor. This module extracts a set of features from a question. Features include question length, question and answer types, and POS tags. Features are discussed in Section 8.4.2.
QA Components. FRANKENSTEIN currently integrates 29 QA components implementing five QA tasks (NED, NER, RL, CL, and QB). Qanary ecosystems is now integrated into FRANKENSTEIN framework to provide a single framework to solve our research problem. Hence, `qa` vocabulary, Qanary methodology, and Qanary framework serve as the foundations for FRANKENSTEIN framework.
QA Component Classifiers. For each QA component, a separate Classifier is trained; it learns from a set of features of a question and predicts the performance of all pertaining components.
QA Pipeline Optimiser. Pipeline optimisation is performed by two modules. The **Component Selector** selects the best performing components for accomplishing a given task based on the input features and the results of the QA Component Classifiers; the selected QA components are afterwards forwarded to the **Pipeline Generator** to dynamically generate the corresponding QA pipelines.
Pipeline Executor. This module executes the generated pipelines for an input question in order to extract answers from the knowledge base (i.e., DBpedia in our case).

8.3.2 Implementation Details

The code for FRANKENSTEIN including all 29 integrated components and empirical study results can be found in our open source GitHub repository[7]. The integration of the 29 new components with the *Qanary* methodology in FRANKENSTEIN is implemented in Java 8. Remaining FRANKENSTEIN modules are implemented in Python 3.4 which include learning module, feature extraction module, pipeline generator, and component selector module.

8.4 Corpus Creation

In this section, we describe the datasets used in our study and how we prepare the training dataset for our classification experiments. All experiments were executed on 10 virtual servers, each with 8 cores, 32 GB RAM and the Ubuntu 16.04.3 operating system. It took us 22 days to generate training data by executing questions of considered datasets for all 28 components, as some tools such as ReMatch[19] and RelationMatcher [102] took approximately 120 and 30 seconds, respectively, to process each question in these components.

8.4.1 Description of Datasets

Throughout our experiment, we employed the Large-Scale Complex Question Answering Dataset[8] (LC-QuAD) [83] as well as the 5th edition of Question Answering over Linked Data Challenge[9] (QALD-5)

[7] https://github.com/WDAqua/Frankenstein
[8] http://lc-quad.sda.tech/
[9] https://qald.sebastianwalter.org/index.php?x=home&q=5

Component/ Tool	**QA Task**	**Year**	**Open Source**	**RESTful Service**	**Publi- cation**
Entity Classifier [71]	NER	2013	✗	✓	✓
Stanford NLP [72]	NER	2005	✓	✓	✓
Ambiverse [112][i]	NER/NED	2014	✗	✓	✓
Babelfy [73]	NER/NED	2014	✗	✓	✓
AGDISTIS [25]	NED	2014	✓	✓	✓
MeaningCloud[iii]	NER/NED	2016	✗	✓	✗
DBpedia Spotlight [17]	NER/NED	2011	✓	✓	✓
Tag Me API [18]	NER/NED	2012	✓	✓	✓
Aylien API[iv]	NER/NED	-	✗	✓	✗
TextRazor[v]	NER	-	✗	✓	✗
OntoText [75][vi]	NER/NED	-	✗	✓	✓
Dandelion[vii]	NER/NED	-	✗	✓	✗
RelationMatcher ([102]	RL	2017	✓	✓	✓
ReMatch [19]	RL	2017	✓	✓	✓
RelMatch [13]	RL	2017	✓	✓	✓
RNLIWOD[viii]	RL	2016	✓	✗	✗
Spot Property [13][ix]	RL	2017	✓	✓	✓
OKBQA DM CLS[ix]	CL	2017	✓	✓	✓
NLIWOD CLS[viii]	CL	2016	✓	✗	✗
SINA [20]	QB	2013	✓	✗	✓
NLIWOD QB[viii]	QB	2016	✓	✗	✗

[i] https://developer.ambiverse.com/
[iii] https://www.meaningcloud.com/developer
[iv] http://docs.aylien.com/docs/introduction
[v] https://www.textrazor.com/docs/rest
[vi] https://www.ontotext.com/
[vii] https://dandelion.eu/
[viii] similar to https://github.com/dice-group/NLIWOD.
[ix] similar to http://repository.okbqa.org/components/7.

Table 8.1: **29 QA components integrated in FRANKENSTEIN using Qanary methodology**: 8 QA components are not available as open source software, 25 provide a RESTful service API and 19 are accompanied by peer-reviewed publications.

dataset [70].
LC-QuAD has 5,000 questions expressed in natural language along with their formal representation (i.e., SPARQL query), which is executable on DBpedia. W.r.t. the state of the art, this is the largest available benchmark for the QA community over Linked Data. We ran the entire set of SPARQL queries (on 2017-10-02) over the DBpedia endpoint[10], and found that only 3,252 of them returned an answer. Therefore, we rely on these 3,252 questions throughout our experiment.
QALD-5. Out of the QALD challenge series, we chose the 5th version (QALD-5) because it provides the largest number of questions (350 questions). However, during the experimental phase the remote Web service of the ReMatch component went down and we were only able to obtain proper results for 204 of the 350 questions. Therefore, we took these 204 questions into account to provide a fair and comparable setting (although, we obtained the results for all 350 questions for all other components).

[10]https://dbpedia.org/sparql

8.4.2 Preparing Training Datasets

Since we have to build an individual classifier for each component in order to predict the performance of that component, it is required to prepare a single training dataset per component. The whole sample set within the training dataset was formed by using the NL questions included from the datasets described previously (from both QALD and LC-QuAD). In order to obtain an abstract and concrete representation of NL questions, we extracted major features enumerated below.

1. *Question Length:* The length of a question w.r.t. the number of words has been introduced as a lexical feature by Blunsom et al. [113] in 2006. In our running example *"What is the capital of Canada?"*, this feature has the numeric value 6.

2. *Question Word:* Huang et al. [114, 115] considered the question word ("wh-head word") as a separate lexical feature for question classification. If a specific question word is present in the question, we assign the value 1, and 0 to the rest of the question words. We adapted 7 Wh-words: "what", "which", "when", "where", "who", "how" and, "why". In *"What is the capital of Canada?"*, *"What"* is assigned the value 1, and all the other words are assigned 0.

3. *Answer Type:* This feature set has three dimensions, namely "Boolean", "List/Resource", and "Number". These dimensions determine the category of the expected answer [14]. In our running example, we assign "List/Resource" for this dimension because the expected answer is the resource `dbr:Ottawa` (i.e correct answer of input question).

4. *POS Tags:* Part of Speech (POS) tags are considered an independent syntactical question feature that can affect the overall performance of a QA system [113]. We used the Stanford Parser[11] to identify the POS tags, where the number of occurrences is considered as a separate dimension in the question feature extraction.

We prepared two separate datasets from LC-QuAD and QALD. We adopted the methodology presented in Section 5.2.3 for the benchmark creation of the subsequent steps of the QA pipelines. Furthermore, the accuracy metrics are *micro F-Score (F-Score)* as a harmonic mean of micro precision and micro recall. Thus, the label set of the training datasets for a given component was set up by measuring the micro F-Score (F-Score) of every given question.

8.5 Evaluating Component Performance

The aim of this experiment is to evaluate the performance of components on the micro and macro levels and then train a classifier to accurately predict the performance of each component.

Metrics *i)* `Answered Questions`: The number of questions for which the QA pipeline returns an answer. *ii)* `Micro Precision (MP)`: The ratio of correct answers vs. total number of answers retrieved by a component for a particular question. *iii)* `Precision (P)`: For a given component, the average of the Micro Precision over all questions. *iv)* `Micro Recall (MR)`: For each question, the number of correct answers retrieved by a component vs. gold standard answers for the given question. *v)* `Recall (R)`: For a given component, the average of Micro Recall over all questions. *vi)* `Micro F-Score (F-Score)`: For each question, the harmonic mean of MP and MR. *vii)* `Macro F-Score (F)`: For each component, harmonic mean of P and R.

[11] `http://nlp.stanford.edu:8080/parser/`

8.5.1 Macro-level Performance of Components

In this experiment, we measured the performance of the reusable components from the QA community that are part of FRANKENSTEIN. We executed each component for each individual query from both LC-QuAD and QALD datasets. Then, for each dataset we calculated the macro accuracy per component and selected those representing highest macro performance. The performance results of the best components are shown in Table 8.2. For brevity, detailed results for each component are placed in our GitHub repository[12].

QA Task	Dataset	Best Component	P	R	F
QB	LC-QuAD	NLIWOD QB	0.48	0.49	0.48
	QALD-5	NLIWOD QB	0.49	0.50	0.49
CL	LC-QuAD	OKBQA DM CLS	0.47	0.59	0.52
	QALD-5	OKBQA DM CLS	0.58	0.64	0.61
NED	LC-QuAD	Tag Me	0.69	0.66	0.67
	QALD-5	DBpedia Spotlight	0.67	0.75	0.71
RL	LC-QuAD	RNLIWOD	0.25	0.22	0.23
	QALD-5	ReMatch	0.54	0.74	0.62

Table 8.2: The macro accuracy of the best components for each task on the QALD and LC-QuAD corpora.

Key Observation: Dependency on Quality of Input Question. From Table 8.2, it is clear that the performance considerably varies per dataset. This is because the quality of questions differs across datasets. Quality has various dimensions, such as complexity or expressiveness. For example, only 728 (22 %) questions of LC-QuAD are simple (i.e., with single relation, single entity), compared to 108 questions (53 %) of QALD. The average length of a question in LC-QuAD is 10.63, compared to 7.41 in QALD. Therefore, components that perform well for identifying an entity in a simple question may not perform equally well on LC-QuAD, which is also evident from Table 8.2 considering the NED task. The same holds for RL components. ReMatch, which is the clear winner on QALD, is outperformed by RNLIWOD on LC-QuAD. Hence, there is no overall best performing QA component for these two tasks, and the definition of the best performing QA component differs across datasets. However, this does hold true neither for CL components nor for QB components (note, these two tasks only have two components each), even though the Macro F-Score values on both datasets have significant differences.

8.5.2 Training the Classifiers

The aim of this part is to build up classifiers which efficiently predict the performance of a given component for a given question w.r.t. a particular task. As observed in the micro F-Score values of the components, these values are not continuous but usually discrete, e.g., 0.0, 0.33, 0.5, 0.66 or 1. Hence, we adopted five classification algorithms (treating it as a classification problem) namely 1) Support Vector Machines (SVM), 2) Gaussian Naive Bayes, 3) Decision Tree, 4) Random Forest, and 5) Logistic Regression. During the training phase, each classifier was tuned with a range of regularisation parameters to optimise the performance of the classifier on the available datasets. We used the cross-validation approach with 10 folds on the LC-QuAD dataset. Figure 8.3 illustrates the details of our experiment for

[12]https://github.com/WDAqua/Frankenstein

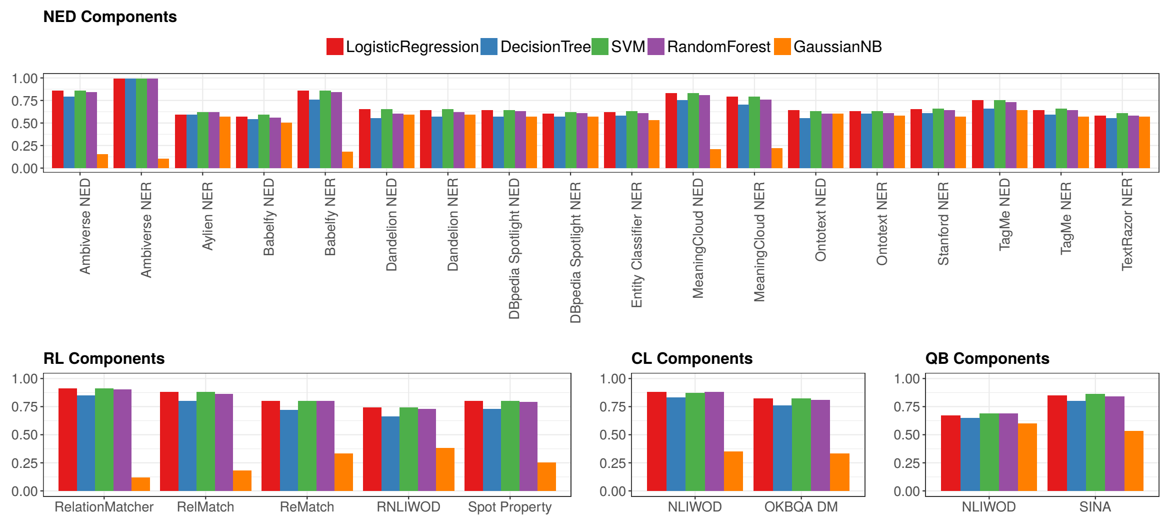


Figure 8.3: **Comparison of Classifiers for all QA Components**. Five Classifiers, namely Logistic Regression, Support Vector Machines, Decision Tree, Gaussian Naive Bayes, and Random Forest are compared wrt accuracy.

training classifiers. Predominantly, Logistic Regression and Support Vector Machines expose higher accuracy as illustrated in Figure 8.3.

8.6 Evaluating Pipeline Performance

In this experiment, we pursue the evaluation question *"Can an approach that dynamically combines different QA components taking the question type into account (such as* FRANKENSTEIN*) take advantage of the multitude of components available for specific tasks?"* To answer this question, we measure the FRANKENSTEIN performance on the (i) task level and (ii) pipeline level. Throughout our experiment, we adopt a component selector strategy as follows:

1. *Baseline-LC-QuAD*: The best component for each task in terms of Macro F-Score on the LC-QuAD dataset (8.5.1).
2. *Baseline-QALD*: The best component for each task in terms of Macro F-Score on the QALD dataset (8.5.1).
3. FRANKENSTEIN-*Static*: The QA pipeline consisting of the best performing components for each task on the QALD dataset (8.5.1).
4. FRANKENSTEIN-*Dynamic*: The QA pipeline consisting of the top performing components from the learning approach.
5. FRANKENSTEIN-*Improved*: Similar to the dynamic FRANKENSTEIN pipeline with a different setting (top 3 pipelines).

8.6.1 Task-level Experiment

Our major goal is to examine whether or not we can identify the N-best components for each QA task. Accordingly, we utilised the following metrics for evaluation *i*) *Total Questions*: the average number of

QA Task	Total Questions	Answer-able	FRANKENSTEIN Top1	Top2	Top3	Baseline
QB	324.3	175.4	162.7	175.4	–	159.6
CL	324.3	76	68.1	76	–	68.2
NED	324.3	294.2	245.2	270.9	284.3	236.3
RL	324.3	153.1	90.3	118.9	134.4	84.2

Table 8.3: 10-fold Validation on LC-QuAD

QA Task	Answer-able	FRANKENSTEIN Top1	Top2	Top3	Baseline QALD	Baseline LC-QuAD
QB	119	91	119	–	102	102
CL	55	52	55	–	52	52
NED	168	132	153	163	144	109
RL	138	83	107	121	105	46

Table 8.4: Performance comparison on task level using LC-QuAD as training and QALD as test dataset.

questions in the underlying test dataset. *ii*) *Answerable*: the average number of questions for which at least one of the components has an F-Score greater than 0.5. *iii*) *Top N*: the average number of questions for which at least one of the Top N components selected by the Classifier has an F-Score greater than 0.5. Furthermore, we rely on a top-*N* approach for choosing the best performing component during our judgement for each QA task.

Experiments on LC-QuAD. This experiment was run on the questions from the LC-QuAD dataset by applying a cross-validation approach. We compare the component selector approach in (i) learning-based manner – called FRANKENSTEIN, and (ii) Baseline-LC-QuAD manner – called Baseline. Table 8.3 shows the results of our experiment. FRANKENSTEIN's learning-based approach selects the top-*N* components with the highest predicted performance values for a given input question. Obviously, this approach outperforms the Baseline approach for the NED, RL, and QB tasks and equals the Baseline for CL task. When we select the top-2 or top-3 best performing components, FRANKENSTEIN's performance improves further.

Cross Training Experiment. The purpose of this experiment is similar to the previous experiment but in order to verify the credibility of our approach, we extended our dataset by including questions from QALD. In fact, questions from QALD are utilised as the test dataset. The results of this experiment are shown in Table 8.4. We observe that FRANKENSTEIN significantly outperforms the LC-QuAD Baseline components for the NED (i.e., Tag Me) and RL (i.e., RNLIWOD) tasks while it achieves comparable results for the CL task.

8.6.2 Pipeline-level Experiment

In this experiment, we greedily arranged a pipeline by choosing the best performing components per task from three strategies. We use the same settings as cross training experiments by utilising QALD questions as test dataset. The first one is the FRANKENSTEIN-Static pipeline composed of Baseline components driven by QALD (i.e., DBpedia Spotlight for NED, ReMatch for RL, OKBQA DM for CL, and NLIWOD QB for QB). The other two strategies are the FRANKENSTEIN-Dynamic and FRANKENSTEIN-Improved

FRANKENSTEIN-Pipeline	**Answered Questions**	**P**	**R**	**Macro F-Score**
Static	37	0.17	0.19	0.18
Dynamic	29	0.14	0.14	0.14
Improved	41	0.20	0.21	0.20

Table 8.5: Comparison with the Baseline Pipeline

pipelines composed by the learning-based component selector with top-1 setting (top-3 for RL of the improved strategy). The results of the comparison are demonstrated in Table 8.5. We conclude that the accuracy metrics for the dynamic pipeline are lower than for the static pipeline. (Note: As performance metrics for state-of-the-art QA systems on the same set of questions in QALD were not available, we excluded this comparison in the table.)

We noticed that the failure of the RL component significantly affects the total performance of both static or dynamic pipelines. Thus, we selected the top-3 components for the RL task to compensate for this deficiency. This setting yields the FRANKENSTEIN-Improved pipeline, where we ran three dynamically composed pipelines out of the 360 possible ones per question. Although this strategy is expected to affect the total accuracy negatively, this did not happen in practice; we even observed an increase in the overall precision, recall, and F-Score. Typically, the available RL and QB components have a full accomplishment or full failure for the input question. For example, considering the question *"What is the capital of Canada?"*, two of the top-3 selected RL components do not process the question. Hence, eventually, only one of the three pipelines returns a final answer. Thus, this simple modification in the setting significantly improves overall pipeline performance and the number of answered questions. With the static pipeline, the number of answered questions is fixed, however, with dynamic pipelines, the number of answered questions can be increased.

8.7 Insights on Evaluation Results

Despite the significant overall performance achieved by FRANKENSTEIN, we investigated erroneous cases in performance specifically w.r.t. classifiers. For instance, in our exemplary question *"What is the capital of Canada?"*, the top-1 component predicted by the Classifier for the RL task fails to map *"capital of"* to `dbo:capital`. Similarly, for the question (*"Who is the mayor of Berlin?"*), the predicted Dandelion NED component can not recognise and disambiguate *"Berlin"*. One of the reasons is related to the primary features extracted from questions. We plan to extend the feature set especially using the recent embedding models and also using different features per task as we have currently used the same features for all tasks. This can be done by associating features with component performance and calculating Cramér's V-coefficient for each feature and a component's ability to answer the given question [15]. One more extension is about switching to more sophisticated learning approaches like HMM, or deep learning approaches which require significantly larger datasets. Another observation is that the existing RL and QB components generally result in poor performance. The QB components need improvement in cases where previous tasks yield a low F-Score for a given question (i.e. returning more than one DBpedia URL as an answer). Hence, QB components should intuitively learn based on available URLs of entities and relations, and then form the right query. The current QB components fail to do so, which severely affected the overall performance of the complete QA pipelines (cf. Table 8.5). Further, RL and QB components need significant improvements in runtime efficiency and performance on complex questions. Thus, the QA community has to pay more attention to improve the components accomplishing these two tasks. To the best of our knowledge, currently very few independent components are available for these tasks (also for Class Linking) and the QA community can contribute building more independent components for these tasks. The FRANKENSTEIN architecture is not rigid and not restricted to the tasks considered in this paper. With the availability of more components performing new QA tasks (e.g., answer type prediction, syntactic parsing, query re-ranking, etc.), just by extending concepts of the `qa`vocabulary, new components can be added to the platform. Furthermore, in real world settings, a greedy approach may negatively affect the runtime of the pipeline. Hence, to provide a more efficient framework for creating QA pipelines, we plan to replace the greedy approach with concepts similar to web service composition [116], where we assign cost metrics (e.g., precision, runtime, or memory consumption) to select components using a pipeline optimiser in an automatic way.

8.8 FRANKENSTEIN as Resource Platform

In the last section, we describe FRANKENSTEIN, which is concerned with (1) a prediction mechanism to predict the performance of a component given a question and a required task; (2) an approach for composing performance-optimised pipelines[13] by integrating the most accurate components for the current QA tasks (i.e., the user's question). FRANKENSTEIN uses Qanary methodology to integrate state-of-the-art QA components within its architecture. However, we disregarded implementation details, reusability, configuration details, integration advantages of FRANKENSTEIN in the previous Section, while this Section introduces FRANKENSTEIN as an application/platform addressing a) how to build a new QA pipeline using 29 integrated components b) how each component can be reused independently c) how to evaluate the questions/texts. Hence, we *disassemble* the FRANKENSTEIN implementation to present a large set of reusable components from the QA community which can be run, evaluated and compared using the additional tools FRANKENSTEINis offering. In other words, by decoupling Frankenstein architecture, the overall architecture becomes collection of 29 components as reusable resources, which can be either used to build QA pipeline or text analysis. We introduce major modules of FRANKENSTEINwhich not only enable detecting optimum pipelines but also enable us to easily *run*, *evaluate* and *compare* any configured QA system. FRANKENSTEINintegrates 29 QA components by developing an individual wrapper for each component. Thus, end-user does not need to get involved in configuration and implementation details of components. In fact, these components can be directly reused to build QA systems. Consequently, just by using the QA components described in this paper 380 reasonable QA pipelines can be created with little effort. Hence, many new insights w.r.t. the performance of QA might be derived using these components and pipelines which also providing support for analytics as well as adopting additional components.

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

$\boxed{\text{R 1}}$ **Reusable QA Components**: We collected 29 QA components accomplishing various QA tasks, i.e., named entity identification/recognition (NER), Named Entity Disambiguation (NED), Relation Linking (RL), Class Linking (CL), and Query Builder (QB). Then, we implemented a wrapper for every included component which enables these popular tools to be easily integrated and reused in the FRANKENSTEINframework. Therefore, these components can be used for building

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

modular question answering systems which might analyse text, provide knowledge extraction etc. Furthermore, the wrapper annotates the output of the components using the `qa`vocabulary to provide machine readability and homogeneity among outputs of all components.

$\boxed{\text{R 2}}$ **Evaluators for components and benchmarks**: We have automatised the process of running and evaluating any component integrated within FRANKENSTEIN. Thus, it enables evaluating and comparing QA components for individual stages of QA pipeline. Consequently, it is possible to analyse the performance of each QA component as well as of the whole QA systems which lead to completely new insights on the performance of particular QA tasks. Hence, researchers are enabled to easily uncover quality flaws and improve the performance while aiming at existing or novel fields of applicability. The evaluator components are independent of the input benchmark, and it is configurable in easy steps based on the requirement of the user.

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

8.8.1 Broader Impact

Impact on QA Community. Recently, QA community was supported by the modular approaches such as openQA [11], OKBQA [13], QALL-ME [12], aiming at integrating and reusing the existing QA core components. FRANKENSTEINis a smart solution on top of Qanary to the limitations observed in the prior approaches. For example, openQA expects Java implementation of the components which is not possible in most of the cases. Also, openQA and QALL-ME have configuration difficulties and its components are not directly reusable in other approaches. More importantly these frameworks do not support a dynamic pipeline methodology. Moreover, the distinguished features united within FRANKENSTEINmakes it scalable, user-friendly and fully automatic which are rare in the prior approaches. Apart from these general characteristics, FRANKENSTEINresources make the researchers needless of developing a QA full pipeline. In fact, researchers can focus on improving individual stages of QA pipelines while reusing $\boxed{\text{R 1}}$ for other QA tasks to complete their pipeline.

For example, recent work on query builder component [117] has reused results of components for building and evaluating QA pipeline for its empirical study. In this way, QA researchers can focus on independent stages to make it more accurate and intelligent. Furthermore, using the automated process of evaluation (i.e., $\boxed{\text{R 2}}$) within FRANKENSTEINassists researchers to easily integrate their newly developed component and evaluate its performance against the-state-of-the-art components over any given benchmark.

Impact beyond QA Community. Although the primary contribution of our work targets the QA community, other disciplines – particularly information extraction (IE) and Natural language processing (NLP) communities – are beneficial of FRANKENSTEINbecause of the common tasks such as NED, RL, and CL. For example, 11 of NER components and 9 of NER components are integrated into FRANKENSTEINand coupled with tools (i.e., $\boxed{\text{R 1}}$ and $\boxed{\text{R 2}}$). These components are also utilised in information retrieval and social media analytics for entity recognition and disambiguation on large textual corpora or a tweet corpus. Any given benchmark can be uploaded to $\boxed{\text{R 2}}$ therefore possibly, a domain-specific evaluation of performance is published. Enabling these communities to reuse the existing

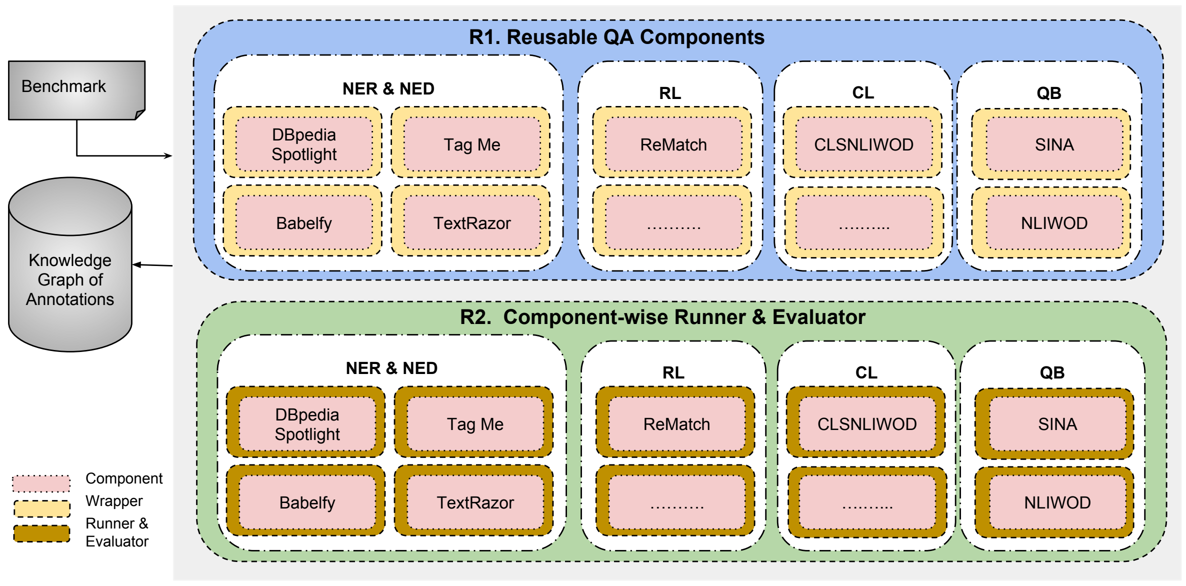


Figure 8.4: Modules of Frankenstein (i) Reusable QA Component Wrappers and (ii) Evaluators.

components opens new perspectives for the future steps. For example, there is no meticulous study about the performance details as for where each component is well-performed or what is its pitfalls.

8.8.2 Approach for Building Reusable QA Components within FRANKENSTEIN

Figure 8.4 represents the resource-wise (module-wise) architecture of FRANKENSTEIN. It is decoupled into two independent categories, (i) $\boxed{\text{R 1}}$ which provides an individual wrapper for each component, and (ii) $\boxed{\text{R 2}}$ which provides an individual runner & evaluator for every integrated component. In the following, these two sets of resources are described in more details.

Integration Approach and its Challenges

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

Employing Qanary methodology and qa Vocabulary. Qanary follows a micro service-based architecture where all components are accessible as RESTful services to be possibly integrated into a Qanary QA process. A QA process within Qanary is a knowledge-driven process where input/output about *question*, *answer*, *annotations* generated in different steps of QA pipeline is conceptualised and annotated by the qavocabulary. Each component integrated into a QA pipeline populates a local knowledge graph (typically its output is annotated by qa vocabulary) shared with other components within Qanary.

In order to be able to annotate outputs generated by all the QA tasks, we had to extend the original version of qa vocabulary by adding new concepts for RL, CL, and QB tasks, and reuse annotations of NER and NED s from Section 5.2.3. E.g., to describe relations appeared in the natural language question, we introduce the annotation:

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

For instance, for the given question `Who is the mayor of Berlin?`, the annotated output of RNLIWOD component for RL task is shown below[14]. There, the output i.e., `http://dbpedia.org/ontology/leaderName` is annotated by the `qa` vocabulary. However w.r.t. `qa`extension, we also introduce further annotations `qa:AnnotationOfClass` and `qa:AnnotationOfAnswerSPARQL` for CL and QB tasks. We reused `qa:AnnotationOfSpotInstance` for NER task and the annotation `qa:AnnotationOfInstance` for NED tasks from Section 5.2.3.

```
<tag:stardog:api:0.9278702836234858>
  a          <http://www.wdaqua.eu/qa#AnnotationOfRelation> ;
  oa:core/hasTarget    :bnode_5daa0259 ;
  oa:hasBody           <http://dbpedia.org/ontology/leaderName>;
  oa:annotatedBy       <http://RNLIWOD.com>;
  oa:AnnotatedAt       "2017-10-02T13:04:21.27"^^xsd:dateTime .
```

Alignment of QA Component Annotations. To ensure interoperability of a new component with existing ones, component has to express the semantics of its generated information using the `qa` vocabulary. We call this the *alignment* of the component to `qa`. There are at least three ways to align the knowledge of a component about the given question. (1) SPARQL queries: A component can execute SPARQL INSERTs in the knowledge base to generate new annotations expressed using the `qa` vocabulary, (2) OWL axioms: When a component already generates information in a specific vocabulary like the NIF vocabulary used by DBpedia Spotlight OWL axioms might express the semantic relation to the specific vocabulary to the `qa` vocabulary (e.g., by defining `owl:sameAs` rules) [17], (3) Distributed Ontology Language (DOL): It enables heterogeneous combinations of ontologies written in different languages and logics [98]. We presented alignments of some existing components using the Qanary approach in Section 5.1 and reused similar alignments.

Wrapping Components and Challenges. During the development of Qanary wrappers in FRANKENSTEINfor different components, we encountered several challenges. The first challenge was to deal with interoperability issues among the components. For instance, a number of components were available as RESTful service, while a few ones [19] had the open source code. Thus, we implemented a RESTful service on top of their source codes to make them easily reusable. The second challenge was associated with the heterogeneous output formats of the components. While some just provide output in JSON (e.g., [73, 74]), some provide output in their own specific vocabulary (e.g., OntoText [75]). A more challenging case was decoupling SINA from its monolithic implementation required to change the complete package structure, dependencies, input format etc. of the original code to make it reusable.

Integrating Evaluation Module

Another set of valuable resources in FRANKENSTEINis its evaluation modules. These modules have three configurations 1) benchmark creation, 2) pipeline configuration, and 3) evaluators. The configurations

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

are briefly described below:

Creating Benchmarks for QA Tasks

We follow the methodology provided in Section 5.2.3 to create benchmarks for each individual stage of QA pipeline. For our running example `Who is the mayor of Berlin?` the corresponding SPARQL query in QALD-5[15] is:

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

For NED and RL tasks, our modules compare the detected named entities and relations by the components with the entities or relations mentioned in the corresponding SPARQL query (e.g., `res:Berlin` for NED and `dbo:leader` for RL). For class linking (CL) components, a similar approach is applied when questions contain class references. To assess the performance of QB components, we run the generated SPARQL query and the benchmark SPARQL query, then we compare the return answers. For evaluating the complete pipeline, answers of the pipeline can be compared with the gold standard answers. In future, we plan to provide a simple configuration that directs the SPARQL results to GERBIL [3] which is an evaluation platform for complete QA processes.

Pipeline Configuration and Runner

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

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

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

Pipeline Execution.

We implemented an independent module called LC-Evaluator within FRANKENSTEINfor executing pipelines. This module is customised in an automatic way for evaluating every individual component of the pipeline. This module obtains questions from a text file (supports csv and txt formats). A user can run a single component or a pipeline containing multiple components. However, the pipeline executor automatically passes the questions sequentially to the associated components. Relying on Qanary methodology, the outputs of components (annotations) are stored in a knowledge graph (i.e., Stardog v4.1.3[16]). Then, the pipeline executor reads the annotations of a particular question from the

[15] https://qald.sebastianwalter.org/index.php?x=home&q=5
[16] https://www.stardog.com/

triplestore and creates an independent file using the turtle format (TTL) for the given input question with the label “questionID_component.ttl”. This process is efficient in case of a large number of questions or text. The user can upload the text file containing annotations of a question, then execute the LC-Evaluator component, and all the output is automatically generated in form of `.ttl` extension for each question.

Pipeline Evaluator.

We developed individual benchmarks for each step of QA pipeline used in evaluation module. Currently, since LC-QuAD [83] and QALD-5 [70] are the most popular and largest state-of-the-art gold standards, thus we developed individual benchmarks out of them for each separate QA task. In the future, we plan to provide additional benchmark files (e.g., for other QALD series). For NED task and for full pipeline evaluation, we plan to integrate pipeline evaluator to GERBIL platform [87]. Using these benchmarks, users are enabled to evaluate the performance of their components for any step of QA pipeline w.r.t. other QA components in FRANKENSTEIN.

8.9 Summary

Functionality	**Frankenstein**	**QALL-ME**	**openQA**	**OKBQA**
Promotes Reusability	✓	✓	✓	✓
Programming Language Independent	✓	-	-	✓
Input/Output Format Independent	✓	-	-	-
Number of Reusable Components	29	7	2	24
Automatic QA pipeline Composition	✓	-	-	-
Microservice Based Architecture	✓	✓	-	✓
Use of Linked Data Technologies	✓	✓	✓	✓

Table 8.6: Comparison of Various QA Frameworks

FRANKENSTEIN is the first framework of its kind for integrating all state-of-the-art QA components to build more powerful QA systems with collaborative efforts. The comparison of various functionalities of FRANKENSTEIN with other QA frameworks is given in the Table 8.6. It simplifies the integration of emerging components and is sensitive to the input question. The rationale was not to build a QA system from scratch but instead to reuse the currently existing 29 major components being available today to the QA community. Furthermore, our effort demonstrates the ability to integrate the components released by the research community in a single platform. FRANKENSTEIN’s loosely coupled architecture enables easy integration of newer components in the framework and implements a model that learns from the features extracted from the input questions to direct the user’s question to the best performing component per QA task. Also, FRANKENSTEIN’s design is component agnostic; therefore, FRANKENSTEIN can easily be applied to new knowledge bases and domains. Thus, it is a framework for automatically producing intelligent QA pipelines. Our experimental study provides a holistic overview on the performance of state-of-the-art QA components for various QA tasks and pipelines. In addition, it measures and compares the performance of FRANKENSTEIN from several perspectives in multiple settings and demonstrates the beneficial characteristics of the approach. FRANKENSTEIN can be extended in following directions: (i) improving quality as well as quantity of extracted features, (ii) improving the learning algorithm, (iii) extending our training datasets, and (iv) including more emerging QA components. In conclusion, our component-oriented approach enables the research community to further improve the performance of state-of-the-art QA systems by adding new components to the ecosystem, or to extend the available data (i.e., gold standard) to adapt the training process.

CHAPTER 9

Conclusion

With the advent knowledge graphs, question answering has been a continuous field of research since last one decade. However, implementing a QA system was cumbersome and time-consuming as most of the QA systems were developed as black boxes and the possibility of reuse was limited. Our vision is initiated by the fact that so far the research community has focused deeply on various QA tasks such as question classification, named entity recognition and disambiguation, relation extraction and has released many independent components and tools accomplishing these tasks. Combining these tools in a single platform eventually leads to the development of modular QA systems where researchers can reuse few components and focus on building specific components for other tasks. Considering prior attempts in this regard and their limitations, we also have a strong focus towards developing automatic and intelligent ways to build QA pipelines on demand. There are concrete pieces of evidence that there is no best performing QA system for all types of natural language questions with different features like question length, POS tags, capitalisation of entities etc.; instead, there are studies suggesting that certain systems, implementing different strategies, are more suitable for certain types of questions [14, 15]. Hence, modern QA systems need to flexibly integrate a number of components specialised to fulfil specific tasks in a QA pipeline and that also laid the foundation for our overall research problem definition of the thesis:

Research Problem Definition

How can existing components for question answering tasks be reused to build effective and seamless dynamic question answering pipelines?

In this thesis, we are concerned with utilising semantics associated with each QA component by semantically describing its functionality, and utilising its strength and weaknesses to build effective dynamic QA pipelines. We divided the research problem into four sub-research questions. We first focused on foundations [26–29] which are the essential steps to solve the interoperability, integration and reusability issues of QA components. We proposed the `qa` vocabulary [26] (Chapter 4) as a flexible and extensible data model for annotating outputs of QA components. The `qa` vocabulary creates an abstraction layer on top of the implementation of the QA components, and provide a homogeneous way to represent all the knowledge (e.g. question, question type, entities and relation in question etc.) produced in QA process. We have also developed a controlled QAV vocabulary [9] to semantically describe the QA components using the high-level input/output requirements of it, and the associated task. We conclude that `qa` and QAV vocabularies can successfully describe the QA components and the associated process semantically and set the foundation for our approach for building an infrastructure to reuse QA components. These two vocabularies collectively contribute towards our approach for addressing the first research question:

RQ1

How can semantics contribute in resolving interoperability of QA components

Thereafter, we describe Qanary [27–29], a methodology for integrating components of QA systems which (i) utilises `qa` vocabulary for annotation, (ii) is independent of programming languages, (iii) is agnostic to domains and datasets, (iv) integrates components on different granularity levels. We provide Qanary ecosystem as a framework for creating reusable question answering systems. Qanary has the limitation that it does not describe QA components based on the tasks it performs and there is no way to provide valid combinations of QA pipelines. For the same, we utilise the proposed QAV vocabulary in *Qaestro* [9, 108], a framework to semantically describe QA components and QA developer requirements and to produce QA component compositions based on these semantic descriptions. Specifically, we employed QAV controlled vocabulary to model QA tasks and exploit the Local As View (LAV) approach to express QA components. The *Qaestro* framework solves the problem of QA pipeline composition problem by casting it to the query rewriting problem and leveraging state-of-the-art SAT solvers. It helps QA developers to semantically describe QA components and developer requirements based on these semantic descriptions. We have integrated *Qaestro* and Qanary ecosystem to provide a seamless composition of QA pipelines. Therefore, Qanary, Qanary ecosystem, and *Qaestro* framework contribute towards addressing the second and third research questions:

RQ2

How can state-of-the-art QA components be integrated in a single platform agnostic to their implementation to promote reusability effectively?

RQ3

How can the process of composing QA pipelines be effectively automated?

We further extended our work to develop SIBKB approach [102] (described in Chapter 6) to build a new relation linking component by reusing the already existing knowledge graph PATTY. We also scale up the number of components in Qanary Ecosystem to 18 NED, five RL, two CL, and two QB components. Considering the scalability of the QA component within the framework, we developed Frankenstein [21], which is concerned with (1) a prediction mechanism to predict the performance of a component given a question and a required task; (2) an approach for composing performance-optimised pipelines by integrating the most accurate components for the current QA tasks (i.e., the user's question). Frankenstein uses Qanary methodology to integrate state-of-the-art QA components within its architecture and build on

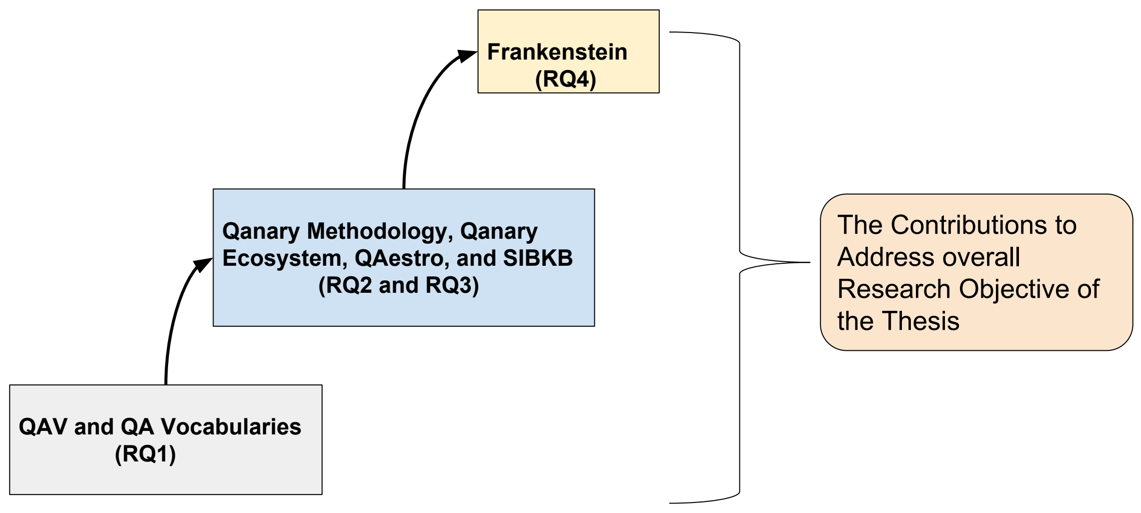

Frankenstein (RQ4)
Qanary Methodology, Qanary Ecosystem, QAestro, and SIBKB (RQ2 and RQ3)
QAV and QA Vocabularies (RQ1)
The Contributions to Address overall Research Objective of the Thesis

Figure 9.1: Contributions made to address the research questions are interconnected. Each contribution acts as foundation for the other. Collectively they are addressing the overall research objective successfully.

top of Qanary ecosystem. Hence, FRANKENSTEIN has utilises 29 QA components of Qanary Ecosystem with a learning mechanism implemented on top of it for predicting best components per task. This provides an automatic way to compose QA pipelines per question and provides an effective way of composing pipelines respecting the strengths and weaknesses of each QA component per task. This successfully addresses the fourth research question defined in this thesis:

RQ4

How can effective dynamic QA pipelines be composed by reusing components?

This thesis is contributing to a broader research agenda of offering the QA community an efficient way of applying their research to a research field which is driven by many different fields, consequently requiring a collaborative approach to achieve significant progress. It is important to note that the applicability of the approaches such as Qanary, `qa` vocabulary, and Frankenstein are currently applied to a single knowledge graph i.e. DBpedia. This is due to the reasons that most of the available independent components are bound to DBpedia as underlying KG. Qanary is independent of knowledge graph and implementation of the components. Although in this thesis the experiments are limited to DBpedia, we trust that the results may generalise to many other large cross domain knowledge graphs, for instance YAGO [76] and Wikidata [6][1]; the QA pipelines for other graphs are also similar to the QA pipelines defined in this thesis, and the graphs share similar structures and overlap in content (Wikipedia is common source of content in many publicly available knowledge graphs). Also, there is no specific assumption in our work on the structure or schema of the underlying knowledge graph, and our method should be

[1]18 components evaluated in this study were not exclusively released for DBpedia, but were for both DBpedia and Wikipedia in general; some of them additionally provide dismabiguated URIs for Yago and Freebase.

equally applicable and can be extended to any other knowledge graph.

We release integration of *Qaestro*, and Frankenstein providing an effective user interface which can be accessible at the URL: `http://frankenstein.qanary-qa.com`. It contains 28 various QA components, and *Qaestro* is used to generate valid combinations of QA pipelines by respecting input and output requirements of various QA components.

9.1 Limitations

Despite the overall achieved research objective, there are few limitations of this research which have not been covered in the scope of the thesis. We list the following limitations:

- The QAV vocabulary is a domain restricted vocabulary which is not extensible to other approaches. However, `qa` vocabulary is extensible to other question answering approaches, for example, specific domain. Work presented in [118] has extended `qa` vocabulary for representing all the knowledge generated in geospatial question answering process. The `qa` vocabulary is not extensible in other research areas such as information extraction, data management etc.

- Improving the efficiency of the overall QA pipeline has been out of the scope of this thesis. This is due to the fact that proposed approaches heavily rely on the individual performance of the re-used components. The QA pipelines composed after reusing QA components are not competitive in terms of overall precision and recall with state of the art monolithic QA systems. However, we have performed an intensive evaluation of the 29 QA components to understand the strength and weaknesses of used QA components. For example, we identified that relation linking and query builder components demonstrate the poorest performance and that impact negatively on overall QA pipelines. Researchers have already started contributing in this direction, and recent research work significantly improve relation linking performance (we report 0.23 as highest F-score for the state-of-the-art relation linking component in Chapter 8, EARL [119] improves it to 0.36, Sakor et al. [120] reports 0.58 for the same task). Similar work has been done to improve query builder component by Zafar et. al. [117] based on identified gaps in this thesis. We believe our work can be the foundation for improving the identified challenges in the state of the art, and the QA community can collectively work towards improving QA over KGs.

- Overall run time of the QA pipeline is another limitation of the work presented in the thesis. Many of the existing QA components are very slow (e.g. ReMatch for RL, and SINA for QB tasks describe in Chapter 8), consequently, they affect the overall run time of the QA pipeline and also restrict many of these approaches to be re-used in real world scenarios. However, we have considered run time performance for SIBKB approach and *Qaestro* framework to illustrate effectiveness and efficiency of these approaches.

9.2 Future Directions

Based on our findings, and the contributions made in this thesis, we now present some of the future directions for the research community:

- Our work can be extended in multiple directions. Firstly, the work presented in the thesis can be utilised for other knowledge graphs such as Wikidata and Yago by including components based on these KGs in Qanary and FRANKENSTEIN framework.

- SIBKB approach for capturing knowledge encoded in PATTY knowledge graph can be extended to multiple knowledge sources such as WordNet and others to improve the overall QA process, by improving relation linking components.
- Components from other domains such as components for biomedical, geographical, temporal question answering can be included in the FRANKENSTEIN framework to enhance FRANKENSTEIN as domain agnostic QA framework. Punjani et al. [118] have extended Frankenstein framework to develop geospatial question answering system where new components for geo functionalities have been added, and components for NER and NED tasks have been reused.
- FRANKENSTEIN and Qanary approaches are agnostic of the implementation detail. It is also possible to integrate end to end QA systems as QA pipelines in these frameworks targeting specific types of questions and train supervised learning methods to choose best pipelines (end to end or components based) for each question type.
- Stato-dynamic QA pipelines can be one of the solutions of improving overall QA process. Recently published query builder tool overcoming the limitations pointed in this thesis significantly improves the performance of the state of the art QB components [117]. Also, this tool answers all the questions answered by other QB components (i.e. other QA component's answered questions are the subset of the proposed tool). Therefore, making the QA pipeline static at the QB task, and choose other components dynamically is one solution for stato-dynamic QA pipelines. This can be done by combining the capabilities of *Qaestro* framework and FRANKENSTEIN. Using *Qaestro*, a component selected by the dynamic pipeline can be overruled by the pre-defined QB component in developers requirement.
- We have evaluated 19 NED tools and highest F-score for complex questions is 0.69. These entity linking tools are not specifically developed for question answering. Therefore, tools suffer from many specific problems. For example, all the tools are sensitive to the character cases and their performance drops sharply if the entity is written in lower case. Also, the performance of the questions with a single entity is about 0.65 (for TagMe). Therefore, it can be observed that generic entity linking tools have a lot of room for improvement. Researchers can target the problem of entity linking for question answering as an independent research area.
- In NLP community, a lot of work has been done for relation extraction from free text. Relation extraction and linking it to knowledge graphs for a given question is more challenging due to limited contextual information available in the question considering a limited number of words. We observe that existing relation linking tools have performed poorly and also a number of tools are limited, and this is another research direction for future research. We did first work towards it in question answering domain by proposing SIBKB [102], but linking relation of an input sentence in a document to knowledge graph is yet to be explored. We believe this would be an important step in the direction of knowledge graph completion and population.
- Several masters thesis have been emerged based on the contributions made in this thesis. Sakor [120] have developed an independent entity and relation linking tool[2] and integrated in Frankenstein. In second masters thesis [121], the author has studied the impact of each input feature on the prediction model employed in Frankenstein and found that there are few features which impact most on the performance of prediction model, and there are several features that do not have any

[2] `https://labs.tib.eu/falcon/`

impact. Work present in [122] extended the Qaestro model to implement a MIN MAX SAT solver to provide composed pipelines in its decreasing order of overall F-score.

As result of this work, we expect a new class of QA systems to emerge. Currently, QA systems are tailored to a particular domain (mostly common knowledge), a source of background knowledge (most commonly DBpedia [4]) and benchmarking data (most commonly QALD). Based on the approaches proposed in this thesis, more flexible, domain-agnostic QA systems can be built and quickly adapted to new domains. This will eventually improve the question answering research in collaborative effort.

Bibliography

[1] E. Choi et al., "Coarse-to-Fine Question Answering for Long Documents", *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 2017 209 (cit. on p. 1).

[2] F. Li and H. V. Jagadish, *Constructing an Interactive Natural Language Interface for Relational Databases*, PVLDB **8** (2014) 73 (cit. on p. 1).

[3] R. Usbeck et al., *Benchmarking Question Answering Systems*, Semantic Web Journal (to be published) () (cit. on pp. 1, 34, 111).

[4] S. Auer et al., "DBpedia: A Nucleus for a Web of Open Data", *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.* 2007 (cit. on pp. 1, 16, 21, 28, 30, 120).

[5] K. D. Bollacker, R. P. Cook and P. Tufts, "Freebase: A Shared Database of Structured General Human Knowledge", *AAAI 2007*, 2007 (cit. on pp. 1, 28).

[6] D. Vrandecic, "Wikidata: a new platform for collaborative data collection", *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, 2012 1063 (cit. on pp. 1, 16, 20, 28, 117).

[7] K. Höffner et al., *Survey on Challenges of Question Answering in the Semantic Web*, Semantic Web Journal (2016) (cit. on pp. 1, 21, 27, 71).

[8] C. Unger et al., "Template-based question answering over RDF data", *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, ACM, 2012 639 (cit. on pp. 1, 2, 28, 29, 31, 32, 44, 51, 53, 88).

[9] K. Singh et al., "QAestro - Semantic-Based Composition of Question Answering Pipelines", *Database and Expert Systems Applications - 28th International Conference, DEXA 2017, Lyon, France, August 28-31, 2017, Proceedings, Part I*, Springer, 2017 19 (cit. on pp. 1, 9, 38, 83, 115, 116).

[10] D. Diefenbach et al., *Core techniques of question answering systems over knowledge bases: a survey*, Knowledge and Information systems (2017) (cit. on pp. 1, 5, 28, 29).

[11] E. Marx et al., "Towards an open question answering architecture", *SEMANTiCS*, 2014 (cit. on pp. 1, 4, 33, 42, 53, 108).

[12] Ó. Ferrández et al., *The QALL-ME Framework: A specifiable-domain multilingual Question Answering architecture*, Web Semantics: Science, Services and Agents on the World Wide Web **9** (2011) (cit. on pp. 1, 4, 33, 53, 83, 108).

[13] J.-D. Kim et al., *OKBQA Framework for collaboration on developing natural language question answering systems*, (2017) (cit. on pp. 1, 2, 4, 28, 31, 33, 37, 80, 83, 98, 101, 108).

[14] M. Saleem et al., "Question Answering Over Linked Data: What is Difficult to Answer? What Affects the F scores?", *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.* CEUR-WS.org, 2017 (cit. on pp. 2, 6, 95, 102, 115).

[15] R. Usbeck et al., *Using Multi-Label Classification for Improved Question Answering*, CoRR (2017) (cit. on pp. 2, 6, 34, 106, 115).

[16] W. Cui et al., *KBQA: Learning Question Answering over QA Corpora and Knowledge Bases*, PVLDB **10** (2017) 565 (cit. on p. 2).

[17] P. N. Mendes et al., "DBpedia spotlight: shedding light on the web of documents", *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, ACM, 2011 1 (cit. on pp. 2, 5, 21, 30, 40, 42, 44, 53, 54, 57, 65, 72, 84, 101, 110).

[18] P. Ferragina and U. Scaiella, "TAGME: on-the-fly annotation of short text fragments (by wikipedia entities)", *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, 2010 1625 (cit. on pp. 2, 101).

[19] I. O. Mulang, K. Singh and F. Orlandi, "Matching Natural Language Relations to Knowledge Graph Properties for Question Answering", *Semantics 2017*, 2017 (cit. on pp. 2, 31, 80, 100, 101, 110).

[20] *SINA: Semantic interpretation of user queries for question answering on interlinked data*, Web Semantics: Science, Services and Agents on the WWW **30** (2015) (cit. on pp. 2, 3, 28, 29, 32, 44, 58, 65, 101).

[21] K. Singh et al., "Why Reinvent the Wheel–Let's Build Question Answering Systems Together", *The Web Conference (WWW 2018)*, 2018 (cit. on pp. 2, 116).

[22] A. Freitas et al., "Treo: combining entity-search, spreading activation and semantic relatedness for querying linked data", *1st Workshop on Question Answering over Linked Data (QALD-1)*, 2011 (cit. on p. 3).

[23] E. Cabrio et al., "QAKiS: an Open Domain QA System based on Relational Patterns", *Proc. of the ISWC 2012 Posters & Demonstrations Track*, 2012 (cit. on pp. 3, 28, 29, 40, 44, 53).

[24] C. Dima, "Answering Natural Language Questions with Intui3.", *CLEF (Working Notes)*, 2014 (cit. on pp. 3, 28).

[25] R. Usbeck et al., "AGDISTIS - Graph-Based Disambiguation of Named Entities Using Linked Data", *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, Springer, 2014 457 (cit. on pp. 5, 30, 39, 45, 53, 54, 61, 62, 65, 72, 85, 99, 101).

[26] K. Singh et al., "Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems", *Tenth IEEE International Conference on Semantic Computing, ICSC 2016, Laguna Hills, CA, USA, February 4-6, 2016*, IEEE Computer Society, 2016 386 (cit. on pp. 9, 38, 115).

[27] A. Both et al.,
"Qanary – a methodology for vocabulary-driven open question answering systems",
The Semantic Web. Latest Advances and New Domains: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 – June 2, 2016, Proceedings, 2016
(cit. on pp. 10, 52, 115, 116).

[28] K. Singh et al., "Qanary - The Fast Track to Creating a Question Answering System with Linked Data Technology", *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, 2016 183 (cit. on pp. 10, 52, 115, 116).

[29] A. Both et al.,
"Rapid Engineering of QA Systems Using the Light-Weight Qanary Architecture", *ICWE 2017*, 2017 (cit. on pp. 10, 52, 115, 116).

[30] A. Y. Halevy, *Answering queries using views: A survey*, VLDB J. **10** (2001) (cit. on pp. 11, 83).

[31] C. P. Gomes et al., "Satisfiability Solvers", *Handbook of Knowledge Representation*, 2008 (cit. on pp. 11, 83).

[32] T. Berners-Lee, J. Hendler and O. Lassila, *The semantic web*, Scientific american **284** (2001) 34 (cit. on p. 15).

[33] T. Bray et al., *Extensible markup language (XML).*, World Wide Web Journal **2** (1997) 27 (cit. on p. 15).

[34] T. Berners-Lee, *Linked data, 2006*, 2006 (cit. on p. 16).

[35] C. Bizer et al., "Linked data on the web (LDOW2008)",
Proceedings of the 17th international conference on World Wide Web, ACM, 2008 1265 (cit. on p. 16).

[36] C. Stadler et al., *LinkedGeoData: A core for a web of spatial open data*,
Semantic Web **3** (2012) 333 (cit. on p. 16).

[37] G. Klyne and J. J. Carroll,
Resource description framework (RDF): Concepts and abstract syntax, (2006) (cit. on p. 17).

[38] G. Antoniou and F. van Harmelen, "Web Ontology Language: OWL", *Handbook on Ontologies*, 2004 67 (cit. on p. 18).

[39] J. Pérez, M. Arenas and C. Gutiérrez, *Semantics and complexity of SPARQL*,
ACM Trans. Database Syst. **34** (2009) 16:1 (cit. on pp. 18, 19).

[40] A. Singhal, *Introducing the knowledge graph: things, not strings*, Official google blog (2012) (cit. on pp. 19, 28, 53).

[41] H. Paulheim, *Knowledge graph refinement: A survey of approaches and evaluation methods*,
Semantic Web **8** (2017) 489 (cit. on pp. 19, 20).

[42] F. Mahdisoltani, J. Biega and F. M. Suchanek,
"YAGO3: A Knowledge Base from Multilingual Wikipedias",
CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings, 2015 (cit. on p. 20).

[43] I. Androutsopoulos, G. D. Ritchie and P. Thanisch,
Natural Language Interfaces to Databases – an Introduction,
Natural Language Engineering **1** (1995) 29 (cit. on pp. 21, 27).

[44] V. López et al., *Is Question Answering fit for the Semantic Web?: A survey*, Semantic Web **2** (2011) (cit. on pp. 21, 27, 28, 53).

[45] E. M. Voorhees, "Question Answering in TREC", *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 2001*, 2001 535 (cit. on p. 21).

[46] C. Unger, A. Freitas and P. Cimiano, "An Introduction to Question Answering over Linked Data", *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th Int'l Summer School, Proceedings*, 2014 (cit. on pp. 21, 23).

[47] M. Lenzerini, "Data Integration: A Theoretical Perspective", *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, 2002 233 (cit. on pp. 24, 25).

[48] A. B. Abacha and P. Zweigenbaum, "Medical question answering: translating medical questions into SPARQL queries", *ACM IHI*, 2012 (cit. on pp. 27, 53).

[49] A. Moschitti et al., "Using Syntactic and Semantic Structural Kernels for Classifying Definition Questions in Jeopardy!", *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2011 712 (cit. on p. 27).

[50] L. Ding et al., "Swoogle: a search and metadata engine for the semantic web", *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA*, 2004 652 (cit. on p. 27).

[51] E. Oren et al., *Sindice.com: a document-oriented lookup index for open linked data*, IJMSO **3** (2008) 37 (cit. on p. 27).

[52] S. M. Harabagiu, D. I. Moldovan and J. Picone, "Open-Domain Voice-Activated Question Answering", *19th International Conference on Computational Linguistics, COLING 2002, Howard International House and Academia Sinica, Taipei, Taiwan*, 2002 (cit. on pp. 27, 42).

[53] H. Sun et al., "Open Domain Question Answering via Semantic Enrichment", *WWW*, 2015 (cit. on pp. 28, 42).

[54] V. Lopez et al., *Evaluating question answering over linked data*, Web Semantics: Science, Services and Agents on the World Wide Web **21** (2013) 3 (cit. on p. 28).

[55] R. Usbeck et al., "HAWK - Hybrid Question Answering Using Linked Data", *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, 2015 (cit. on pp. 28, 32).

[56] M. Dubey et al., "AskNow: A Framework for Natural Language Query Formalization in SPARQL", *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, Springer, 2016 300 (cit. on pp. 28, 29, 71, 72, 81, 88).

[57] S. He et al., "CASIA@ V2: A MLN-based Question Answering System over Linked Data.", *CLEF (Working Notes)*, 2014 (cit. on pp. 28, 71, 72, 80).

[58] M. Yahya et al., "Robust question answering over the web of linked data", *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, ACM, 2013 1107 (cit. on p. 28).

[59] C. Dima, "Intui2: A Prototype System for Question Answering over Linked Data.", *CLEF (Working Notes)*, 2013 (cit. on p. 28).

[60] S. Park et al., "ISOFT at QALD-5: Hybrid Question Answering System over Linked Data and Text Data.", *CLEF (Working Notes)*, 2015 (cit. on p. 28).

[61] T. Hamon et al., "Description of the POMELO System for the Task 2 of QALD-2014.", *CLEF (Working Notes)*, 2014 1212 (cit. on p. 28).

[62] V. Lopez et al., *PowerAqua: Supporting users in querying and exploring the semantic web*, Semantic Web **3** (2011) 249 (cit. on pp. 28, 29, 42, 51, 53).

[63] S. Ruseti et al., "QAnswer-Enhanced Entity Matching for Question Answering over Linked Data", CLEF, 2015 (cit. on pp. 28, 44).

[64] R. Beaumont, B. Grau and A. L. Ligozat, *SemGraphQA@QALD-5: LIMSI participation at QALD-5@CLEF*, CEUR Workshop Proceedings (2015) (cit. on p. 28).

[65] C. Pradel et al., "SWIP at QALD-3: Results, Criticisms and Lesson Learned", *Working Notes for CLEF 2013 Conference , Valencia, Spain, September 23-26, 2013.* 2013 (cit. on p. 28).

[66] K. Xu, Y. Feng and D. Zhao, *Xser@ QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing*, 2014 (cit. on pp. 28, 43, 71, 72).

[67] A. P. B. Veyseh, "Cross-lingual question answering using common semantic space", *Proceedings of TextGraphs-10: the Workshop on Graph-based Methods for Natural Language Processing*, 2016 15 (cit. on p. 28).

[68] L. Zou et al., "Natural language question answering over RDF: a graph data driven approach", *Proc. of the 2014 ACM SIGMOD international conference on Management of data*, ACM, 2014 313 (cit. on pp. 28, 43).

[69] D. Damljanovic, M. Agatonovic and H. Cunningham, "FREyA: An Interactive Way of Querying Linked Data Using Natural Language", *ESWC Workshops*, 2011 (cit. on pp. 28, 29, 53).

[70] C. Unger et al., "Question Answering over Linked Data (QALD-5)", *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum, Toulouse, France, September 8-11, 2015.* CEUR-WS.org, 2015 (cit. on pp. 28, 33, 65, 101, 112).

[71] M. Dojchinovski and T. Kliegr, "Entityclassifier.eu: Real-time Classification of Entities in Text with Wikipedia", *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECMLPKDD'13, Springer-Verlag, 2013 654 (cit. on pp. 30, 101).

[72] J. R. Finkel, T. Grenager and C. D. Manning,
"Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling",
ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA,
The Association for Computer Linguistics, 2005 363 (cit. on pp. 30, 39, 42, 61, 62, 65, 85, 101).

[73] A. Moro, A. Raganato and R. Navigli,
Entity Linking meets Word Sense Disambiguation: a Unified Approach, TACL **2** (2014) 231
(cit. on pp. 30, 101, 110).

[74] P. Ferragina and U. Scaiella, *Fast and Accurate Annotation of Short Texts with Wikipedia Pages*, IEEE Software **29** (2012) 70 (cit. on pp. 30, 110).

[75] A. Kiryakov et al., *Semantic annotation, indexing, and retrieval*, J. Web Sem. (2004) 49
(cit. on pp. 31, 101, 110).

[76] F. M. Suchanek, G. Kasneci and G. Weikum, "Yago: a core of semantic knowledge",
Proc. of the 16th Int. Conf. on World Wide Web, 2007 697 (cit. on pp. 32, 117).

[77] S. Hellmann et al., "Integrating NLP Using Linked Data",
The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II, 2013 98 (cit. on pp. 32, 44, 52).

[78] A. Uciteli et al., "Search Ontology, a new approach towards Semantic Search",
Workshop on Future Search Engines, 2014 (cit. on pp. 32, 33, 42).

[79] A. Bordes et al., *Large-scale Simple Question Answering with Memory Networks*, CoRR (2015)
(cit. on p. 32).

[80] J. Berant et al., "Semantic Parsing on Freebase from Question-Answer Pairs",
Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL, ACL, 2013 1533 (cit. on p. 32).

[81] M. Yu et al., "Improved Neural Relation Detection for Knowledge Base Question Answering",
Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers, 2017 571 (cit. on p. 32).

[82] D. Diefenbach et al., "Question Answering Benchmarks for Wikidata", *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017.* 2017
(cit. on p. 33).

[83] P. Trivedi et al.,
"LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs",
The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II, Springer, 2017 210 (cit. on pp. 33, 100, 112).

[84] Z. Yang et al., "Building optimal information systems automatically: configuration space exploration for biomedical information systems",
22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013, ACM, 2013 1421 (cit. on p. 33).

[85] J. Ng and M. Kan, *QANUS: An Open-source Question-Answering Platform*, CoRR (2015)
(cit. on p. 33).

[86] A. Both et al.,
"A service-oriented search framework for full text, geospatial and semantic search",
Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014, ACM, 2014 65 (cit. on p. 33).

[87] R. Usbeck et al., "GERBIL: General Entity Annotator Benchmarking Framework",
Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015, 2015 (cit. on pp. 34, 112).

[88] J. D. Ullman, *Information integration using logical views*, Theor. Comput. Sci. **239** (2000) (cit. on pp. 39, 87).

[89] Y. Ibrahim, M. Yosef and G. Weikum, "AIDA-Social: Entity Linking on the Social Stream",
Exploiting Semantic Annotations in Information Retrieval, 2014 (cit. on pp. 42, 53).

[90] J. Luque et al., *GeoVAQA: a voice activated geographical question answering system*, () (cit. on p. 44).

[91] A. Both et al.,
"Get Inspired: A Visual Divide and Conquer Approach for Motive-based Search Scenarios",
13th International Conference WWW/INTERNET (ICWI 2014),
International Association for Development of the Information Society (IADIS), 2014 (cit. on p. 44).

[92] R. Sanderson et al., *Open annotation data model*, Community Draft, W3C, 2013 (cit. on p. 44).

[93] V. Lopez et al., *PowerAqua: A Multi-Ontology Based Question Answering System–v1*,
OpenKnowledge Deliverable D8. 4 Pp1 **14** (2007) (cit. on pp. 44, 53).

[94] D. Diefenbach et al.,
"The Qanary Ecosystem: Getting New Insights by Composing Question Answering Pipelines",
Web Engineering - 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings, Springer, 2017 171 (cit. on pp. 52, 66).

[95] S. J. Athenikos and H. Han, *Biomedical question answering: A survey*,
Computer Methods and Programs in Biomedicine **99** (2010) 1 (cit. on p. 53).

[96] N. Nakashole, G. Weikum and F. M. Suchanek,
"PATTY: A Taxonomy of Relational Patterns with Semantic Types", *EMNLP-CoNLL*, 2012 (cit. on pp. 53, 58, 71, 72).

[97] T. G. Dietterich, "Ensemble learning", *The Handbook of Brain Theory and Neural Networks*, 2002 (cit. on p. 53).

[98] T. Mossakowski, O. Kutz and C. Lange,
"Three Semantics for the Core of the Distributed Ontology Language",
Formal Ontology in Information Systems,
7$^{\text{th}}$ International Conference (FOIS 2012), (Graz, Austria, 24th–27th July 2012), 2012,
ISBN: 978-1-61499-084-0 (cit. on pp. 56, 110).

[99] R. Speck and A. Ngonga Ngomo, "Ensemble learning for named entity recognition",
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. 61, 65).

[100] S. Nakatani, *Language Detection Library for Java*, https://github.com/shuyo/language-detection, 2010 (cit. on p. 61).

[101] D. Diefenbach, K. Singh and P. Maret,
"WDAqua-core0: A Question Answering Component for the Research Community",
ESWC, 7th Open Challenge on Question Answering over Linked Data (QALD-7), 2017
(cit. on p. 61).

[102] K. Singh et al.,
"Capturing Knowledge in Semantically-typed Relational Patterns to Enhance Relation Linking",
K-Cap 2017, 2017 (cit. on pp. 72, 100, 101, 116, 119).

[103] S. Hakimov et al.,
"Semantic question answering system over linked data using relational patterns",
EDBT/ICDT '13, 2013 (cit. on p. 72).

[104] J. Pennington, R. Socher and C. D. Manning, "Glove: Global vectors for word representation.",
EMNLP, 2014 (cit. on pp. 77, 79, 80).

[105] P. Larson, *Dynamic Hashing*, BIT (1978) (cit. on p. 77).

[106] E. Marx et al., "An Open Question Answering Framework", *ISWC-poster and Demo*, 2015
(cit. on p. 83).

[107] A. Y. Levy, A. Rajaraman and J. J. Ordille,
"Querying Heterogeneous Information Sources Using Source Descriptions",
Proceedings of 22th Int'l Conference on Very Large Data Bases, 1996 (cit. on p. 83).

[108] K. Singh et al., "QAestro Framework - Semantic Composition of QA Pipelines",
Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017. 2017 (cit. on pp. 83, 116).

[109] Y. Arvelo, B. Bonet and M. Vidal,
"Compilation of Query-Rewriting Problems into Tractable Fragments of Propositional Logic",
Proceedings of the 21st National Conference on Artificial Intelligence & the 18th Innovative Applications of Artificial Intelligence Conference, 2006 (cit. on pp. 87, 89, 91).

[110] D. Izquierdo, M. Vidal and B. Bonet,
"An Expressive and Efficient Solution to the Service Selection Problem",
9th Int'l Semantic Web Conference ISWC, 2010 (cit. on p. 87).

[111] G. Konstantinidis and J. L. Ambite, "Scalable query rewriting: a graph-based approach",
Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, 2011
(cit. on p. 87).

[112] J. Hoffart et al., "Robust Disambiguation of Named Entities in Text",
Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL, 2011 782 (cit. on p. 101).

[113] P. Blunsom, K. Kocik and J. R. Curran, "Question classification with log-linear models",
SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006,
ACM, 2006 615 (cit. on p. 102).

[114] Z. Huang, M. Thint and Z. Qin,
"Question Classification using Head Words and their Hypernyms",
2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL, ACL, 2008 927 (cit. on p. 102).

[115] Z. Huang, M. Thint and A. Çelikyilmaz,
"Investigation of Question Classifier in Question Answering", *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2009 (cit. on p. 102).

[116] R. Berbner et al., "Heuristics for QoS-aware Web Service Composition",
2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA, 2006 72 (cit. on p. 107).

[117] H. Zafar, G. Napolitano and J. Lehmann,
"Formal query generation for question answering over knowledge bases",
European Semantic Web Conference, Springer, 2018 714 (cit. on pp. 108, 118, 119).

[118] D. Punjani et al., "Template-Based Question Answering over Linked Geospatial Data",
Proceedings of the 12th Workshop on Geographic Information Retrieval in conjunction with ACM SIGSPATIAL 2018, GIR'18, ACM, 2018 7:1, ISBN: 978-1-4503-6034-0 (cit. on pp. 118, 119).

[119] M. Dubey et al.,
"EARL: Joint Entity and Relation Linking for Question Answering over Knowledge Graphs",
ISWC 2018, 2018 (cit. on p. 118).

[120] A. Sakor, *Relation and Entity extraction and linking over DBpedia*,
Masters Thesis, University of Bonn (to appear) (2018) (cit. on pp. 118, 119).

[121] M. Y. Jeradeh,
Improving Dynamic question answering pipeline composition within Frankenstein,
Masters Thesis, University of Bonn (to appear) (2018) (cit. on p. 119).

[122] K. A. Aditya, *Seamless Composition of Question Answering Pipelines*,
Masters Thesis, University of Bonn (to appear) (2018) (cit. on p. 120).

APPENDIX **A**

List of Publications

- *Conference Papers (peer reviewed)*
  1. **K Singh**, AS Radhakrishna, A Both, S Shekarpour, I Lytra, R Usbeck, A Vyas, A Khikmatullaev, D Punjani, C Lange, ME Vidal, J Lehmann, S Auer. *Why Reinvent the Wheel- Lets Build Question Answering Systems Together.* In Proceedings of the Web Conference (WWW), 2018, ACM;
  2. **K Singh**, A Both, AS Radhakrishna, S Shekarpour. *Frankenstein: a Platform Enabling Reuse of Question Answering Components.* In Proceedings of the 15th Extended Semantic Web Conference (ESWC), 2018, Springer;
  3. **K Singh**, IO Mulang, I Lytra, MY Jaradeh, A Sakor, ME Vidal, C Lange, S Auer. *Capturing Knowledge in Semantically-typed Relational Patterns to Enhance Relation Linking.* In Proceedings of the Knowledge Capture Conference (K-Cap), 2017, ACM;
  4. **K Singh**, I Lytra, ME Vidal, D Punjani, H Thakkar, C Lange, S Auer. *Qaestro -Semantic-based Composition of Question Answering Pipelines.* In Proceedings of 28th International Conference on Database and Expert Systems Applications (DEXA), 2017, Springer;
  5. D Diefenbach, **K Singh**, A Both, D Cherix, C Lange, S Auer. *The Qanary ecosystem: getting new insights by composing Question Answering pipelines.* In Proceedings of the 17th International Conference on Web Engineering (ICWE), 2017, Springer;
  6. A Both, D Diefenbach, **K Singh**, S Shekarpour, D Cherix, C Lange. *Qanary -a methodology for vocabulary-driven open question answering system.* In Proceedings of the 13th Extended Semantic Web Conference (ESWC), 2016, Springer;
  7. **K Singh**, A Both, D Diefenbach, S Shekarpour. *Towards a message-driven vocabulary for promoting the interoperability of question answering system.* In Proceedings of the 10th International Conference on Semantic Computing (ICSC), 2016, IEEE;
- *Demo Papers (peer reviewed)*
  8. **K Singh**, I Lytra, A Sethupat, A Vyas, ME Vidal. *Dynamic Composition of Question Answering Pipelines With Frankenstein.* In Proceedings of the 41st International ACM SIGIR conference on research and development in Information Retrieval (SIGIR), 2018, ACM;
  9. A Both, **K Singh**, D Diefenbach, I Lytra. *Rapid Engineering of QA Systems Using the Light-Weight Qanary Architecture.* In Proceedings of the 17th International Conference on Web Engineering (ICWE), 2017.

10. **K Singh**, I Lytra, K Abhinav, ME Vidal. *Qaestro Framework- Semantic Composition of QA Pipelines.* In Posters and Demo Track at the 16th International Semantic Web Conference (ISWC) 2017.
11. **K Singh**, A Both, D Diefenbach, S Shekarpour. *Qanary–the Fast Track to Creating a Question Answering System with Linked Data Technology.* In Posters and Demo Track at the 13th Extended Semantic Web Conference (ESWC), 2016.

- *Workshop Paper (peer reviewed)*

12. S Shekarpour, KM Endris, A Jaya Kumar, D Lukovnikov, **K Singh**, H Thakkar, and C Lange. *Question answering on linked data: Challenges and future directions.* In Proceedings of the 25th International Conference Companion on World Wide Web (WWW Companion). 2016.

- *Miscellaneous Papers (peer reviewed)*

Following publications originated during and are related to this thesis but are not directly part of the thesis itself. However, few chapters take inspirations from these publications.

13. IO Mulang, **K Singh**, F Orlandi. *Matching Natural Language Relations to Knowledge Graph Properties for Question Answering.* In Proceedings of the Semantics Conference, ACM, 2017.
14. **K Singh**, I Lytra, AS Radhakrishna, S Shekarpour, ME Vidal, J Lehmann. *No one is Perfect-Analysing the Performance of Question Answering Components over the DBpedia Knowledge Graph.* Submitted to Information Processing and Management Journal, Elsevier.

Abbreviations and Acronyms

QA	Question Answering
NER	Named Entity Recognition
NED	Named Entity Disambiguation
RL	Relation Linking
CL	Class Linking
QB	Query Builder
LAV	Local As View
GAV	Global As View
SAT Solver	Satisfiability Solver
RDF	Resource Description Framework
HTML	HyperText Markup Language
HTTP	The Hypertext Transfer Protocol
KG	Knowledge Graph
KB	Knowledge Base
WADM	Web Annotation Data Model
RDBMS	Relational Database Management System
XML	Extensible Markup Language

List of Figures

List of Tables