Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marcos André Oliveira Ramos

Question and Answer Systems

A Closed-Domain Question and Answer System For the Python Language Domain

January 2016

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Marcos André Oliveira Ramos

Question and Answer Systems

A Closed-Domain Question and Answer System For the Python Language Domain

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Pedro Rangel Henriques
Maria João Varanda

January 2016

ABSTRACT

This document formally proposes a Master Thesis in Software Engineering, in the field of Language Engineering.
The main goal of this thesis is to create a Question Answering System that can automatically answer to questions presented in a natural language about the Python programming language.
A system of this kind aims at the interaction with a human. Since it is natural for a human to communicate in a natural language, such as Portuguese or English, there is a need for systems that can respond to the human user in the same language. When restricted to a target database of knowledge or information, these systems can offer satisfiable answers to the posed questions.
As a proof of concept it is expected the implementation of a tool that can present reasonable answers to questions about Python.

RESUMO

O principal objectivo desta tese é criar um sistema de perguntas e respostas que consiga, de forma automática, responder a perguntas apresentadas em linguagem natural sobre a linguagem de programação Python.
Um sistema deste tipo tenta sobretudo, interagir com um utilizador humano. Visto que para uma pessoa é normal e natural comunicar usando uma linguagem natural, como por exemplo, Português ou Inglês, existe uma grande procura de sistemas que possam comunicar com o utilizador usando a mesma linguagem.
Estes sistemas quando restringidos a uma certa área de conhecimento, podem responder de forma satisfatória às perguntas que lhe são apresentadas.
No final, como modelo prático de prova, espera-se a implementação de uma ferramenta que seja, razoavelmente capaz de apresentar respostas a questões sobre o domínio da linguagem de programação Python.

CONTENTS

1 INTRODUCTION

Question Answering (QA) Systems arose from the need to provide systems that could answer to the user in a natural language. Most users would think that questions such as "What is an integer variable?" are easy to interpret. This is not so, even if the complexity of the question seems low to the human user, interpreting and finding an answer can be extremely difficult (Hutchins and Somers, 1992).

Since fifty or forty years ago the world has seen the creation and growth of various QA systems (Fortnow and Homer, 2003). Most were more or less successful and even though some of them have been discontinued, the fact that this computer science discipline is so relevant today, shows how hard it is to build and maintain a system capable of understanding natural languages as a human does. Nonetheless, the increasing demand for such tools is growing at a fast rate, leading to a greater demand for research.

Examples of such systems, are:

- *Wolframalpha*[1]: based on the earlier product *Mathematica*, a mathematical computational software. WolframAlpha offers a way to get knowledge not by searching the web, but by doing computations on a collection of built-in data.
- *IBM Watson*[2]: it was primarily developed to answer questions of the quiz tv show *Jeopardy!* (Thompson, 2010). Although general purpose it must be customized to each application domain. It is now widely used by medical professionals.
- *SHRDLU*[3]: an early natural language software where the user could maintain a conversation with the computer about a "blocks world" (McTear and Raman, 2011). The idea was to tell the system to move blocks in this world, name collections and show the state of the world. The system was purely didactic, and its purpose was to show a different way to interact with a computer. In a sense, it was a world that the user could build using only geometric figures and the help of an automatic assistant.

These systems always fall under one of two categories (Andreasen et al., 2009):

1 http://www.wolframalpha.com
2 http://www.ibm.com/smarterplanet/us/en/ibmwatson/
3 https://en.wikipedia.org/wiki/SHRDLU

- Closed-domain: questions are restricted to a specific domain; usually supported on ontologies (Calvanese et al., 2006) and structured databases; answers only to a limited type of questions;
- Open-domain: allows questions about everything; relies on general ontologies and world knowledge (Prager, 2006); supported on great amounts of data from which answer are formulated.

1.1 OBJECTIVES

The main goal of this master's thesis is to create a closed-domain QA system that answers questions related with a specific and well defined programming language.

Python was chosen as the programming language for the QA system. Even tough Python has been around for more then twenty years, its popularity among both beginners and experienced programmers just started a few years ago and is now rapidly increasing. It is massively used in industry, scientific research (Zelle, 2004; Bird et al., 2009), and by people wanting to teach themselves a new programming language. Large amounts of textual information about it are available throughout the Internet, although significantly dispersed.

Therefore, this system could give a more personalized and enjoyable experience to the user, by eliminating or reducing the task of dealing directly with such an amount of scattered information.

Other languages, like Java, Perl, C++ or C#, are as well suited for this purpose.

This master thesis has the following objectives:

- research about the various methods used to build QA systems;
- discuss and chose the best methods to build and implement a QA system best suited to answer question concerning a programming language;
- discuss about the way the QA system should interpret and answer the user's questions;
- find and choose possible textual sources of information about Python;
- build a prototype system that can give reasonable answers to the most simple questions;
- add complexity and size to the QA system;
- make exhaustive and relevant tests with the created tool.

As a final result, it is expected to have a computer program that can be easily used by software engineers wanting to learn more about python, and that is useful for demonstration purposes about question and answer systems and techniques.

1.2 RESEARCH HYPOTHESIS

When learning a programing language the programmer may wonder why it is so difficult and morose to find succinct answers to questions that are apparently easy to answer. A more experienced programmer does not face this problem so often, but still, not even the experts know everything. Some people would propose a simple solution to all this problems: "Ask Google". Search engines such as Google or Yahoo are incredibly powerful but do not exactly answer questions. Simple factoid answers could take minutes to be found by a common programmer, using a traditional search engine.

This problem is not exclusive of programmers. Finding simple world facts are also time consuming, but for these, there has been an increasingly offer of alternatives referred to as QA systems.

If QA systems are adequate at finding answers posed in a natural language about world facts, could they not be capable of answering questions about a certain programming language?

This master thesis tries to prove that there is a positive answer to the previous question. A system that only deals with questions about a programming language like Python should be possible to be built in the time assigned for this master's thesis.

Also, Python is a widely known programming language with large amounts of documentation available, hence, it should be possible to use this data as a knowledge base for such a system.

1.3 DOCUMENT STRUCTURE

This document starts by explaining the different approaches and tools used by the majority of QA systems, in chapter 2. Concepts will be introduced and followed by examples of QA systems with a short explanation that prove and explain the ideas that are being introduced.

Chapter 3, proposes a QA system for the Python programming language. First it is given a description of the chosen domain (Python) and the reasons behind this choice, followed by an explanation of the information sources that will be used to retrieve all knowledge about Python. A graphical depiction of the proposed system's architecture is then presented to introduce section 3.3, where the most important components of the systems will be explained.

Chapter 4 closes this document with a conclusion about the work already completed and an explanation of what should be accomplished in the next months and phases assigned for this master's thesis.

2

QUESTION & ANSWERING SYSTEMS: APPROACHES AND TOOLS

This chapter starts by explaining the different domain types of QA systems and proceeds by introducing some techniques used by current systems.

Various QA systems and a brief explanation of these, will be mentioned along the chapter to help understand concepts and the complexity of QA systems.

2.1 QA DOMAIN TYPES

The methods used to create a Question Answering system can be varied and do not necessarily follow a strict or conventional structure, nonetheless, when regarding the domain of its knowledge, these tools are mostly separated by **closed** and **open domain**. It is important to mention that, although a system is always classified as open or closed domain, it can sometimes be adapted to work with a different kind of domain. A rather generic example would be an open-domain system that when restricted to questions of a specific domain would be considered as closed-domain. Another example, is a closed-domain QA system about the "apples" domain that can be changed to work with the "peers" domain. Since both domains are very similar, the changes would only affect the knowledge data.

2.1.1 *Closed-Domain*

Closed-domain systems answer questions within a specific knowledge domain or to only a certain type of questions. They target precision, rather than coverage.

Dealing with a restricted scope of knowledge allows QA systems to resort to smaller amounts of information, usually structured data such as ontologies. With such limited, formalized and structured data, systems try to take great advantages of natural language processing techniques in order to be the most accurate possible in their answers.

2.1.2 *Open-Domain*

Open-domain systems answer questions about almost everything. These system rely on much larger amounts of data than closed-domain QA systems, using mostly unstructured data and general ontologies.

Their primarily goal is to provide factoid answers to questions about world knowledge. While closed-domain systems aim at accuracy, open-domain systems intend to cover the greater scope of information that is possible. The ambition is to offer more than a conventional web search engine by answering the user's questions, rather than presenting a simple list of documents/web pages that match the search's query.

2.2 QA TECHNIQUES AND APPROACHES

In this section, QA systems will be classified according to three different perspectives: techniques for question analysis; techniques for retrieving answers from knowledge repositories; and techniques for composing the final answer.

Other studies of the same genre divide QA systems into different components. We do not state that a QA system is strictly composed of only these three components, but try to divide them into the most generic way that is possible, in order to accommodate all QA systems.

2.2.1 *Question Analysis*

Linguistic approach (Sasikumar and Sindhu, 2014)

A Question Answering system based on methods that integrate Natural Language Processing (NLP), i.e. methods that try to derive meaning from natural language input. NLP is itself a very prominent field of computer science and artificial intelligence that involves techniques like parsing and machine learning. The process consists in converting the user's question into a database query written in a formal language such as SQL and SPARQL. The output of the query will usually be given as an answer.

BASEBALL (Green Jr et al., 1961) and LUNAR are two early QA systems that fit perfectly into this category. BASEBALL answered questions about baseball games played in the American league, and LUNAR answered question about analysis made to rock samples from the Apollo missions. LUNAR was so successful that in a convention in 1971 it was able to answer to 90% of questions posed by geologists (Mollá and Vicedo, 2007). Both of them use a natural language interface to a database,ie. they extract information from a structural database that contains the information needed to answer the question.

START[1] is another example. In operation since December, 1993, it describes itself as "the world's first Web-based question answering system". Contrary to the previous examples it is still operational and maintained. It is also an open-domain QA system while BASEBALL and LUNAR are closed-domain which means that the complexity to derive meaning from a question is augmented since it can not rely on a specific knowledge domain.

Pattern Matching and Tagging

The QA system analyses a question and labels it in order to find a pattern. If the pattern corresponds to the expected pattern for a certain answer, then this answer should be the right one. For example, when posed with the question, **"Who is the President of Portugal?"** the system interprets the question as **"<Person Name>is the <President of Portugal>?"** and expects the answer to be the name of a person who is president of Portugal.

QACID (Ferrández et al., 2009) an ontology-based QA system, applies tagging algorithms in order to extract a query pattern, i.e. a query in natural language labeled with morphological information and ontological concepts. To overcome tagging difficulties, due to the complexity of natural languages and human error, QA systems might rely on, for example, synonyms and algorithms for removing stem words, stop words and vowels.

The AQUA system (Vargas-Vera and Lytras, 2010), among its various steps for processing a question, divides the sentence into subject, verb, propositional phrases, adjectives and objectives. Similar to QACID, it produces a semantic representation of the query that is used by search algorithms when trying to find an answer in the knowledge base.

Ravichandran and Hovy (2002) propose a method that combines patterns with machine learning techniques in an open domain QA system. They use the machine learning technique of bootstrapping to build a tagged corpus from some examples of hand crafted question-answer pairs. These examples are passed to a search engine and from the results of the search, the system extracts patterns. The precision of each pattern is calculated for each type of question. The patterns are then employed in finding answers for new questions.

2.2.2 *Answer Retrieval*

Ontologies and Structured Knowledge Based QA

Systems that rely on structured knowledge sources and ontologies about a specific domain are mostly closed-domain QA systems. The idea is to make queries over a database that contains structured information about the system domain. Meaning that the information was produced before the questions were asked and that, if the knowledge source works as

1 http://start.csail.mit.edu

it should, the effort lays mainly in understanding the question rather than finding the best answer. This is only feasible when the scope of the domain is well defined and restricted, and the knowledge source is relatively small, very well defined and structured.

QACID (Ferrández et al., 2009) uses an ontology populated with information about the cinema domain, provided by LaNetro, a Spanish company that provides information about tourism in Spain. The ontology, stored in the RDF format, is used as a structured databased in which information is obtained by means of SPARQL queries.

The system proposed by Chantrapornchai et al. (2014) uses an ontology to represent knowledge about the domain of Thai Cats. They clearly put great focus on their ontology, using various sources of information and other ontologies as references. SPARQL queries are then used to retrieve information from the ontology.

General Ontologies and Free Text Based QA

In this case the information, from which the system derives answers to the user's questions, is stored in textual documents written in natural language. Ontologies are mostly used to define a language in which documents and questions can be represented and exploited (Mollá and Vicedo, 2007). The most interesting feature of these systems, and what makes them perfect for working with in an open-domain, is the capacity of taking advantage of the ever increasing amount of textual information, available throughout the Internet. Web search engines like Google can be used to find and retrieve these knowledge sources from the Internet. Nonetheless, not all Question Answering systems rely on this technique, since most of the time, there is no guarantee of the correctness of the retrieved information.

Dumais et al. (2002) describes a Question Answering system that uses the large amount of data online as a knowledge source. Instead of focusing on complex linguistic techniques, as most systems do, it relies mainly on the redundancy of large corpora. Based on experimental results, they claim that question answering accuracy improves by increasing the amount of data used for learning. The greater the redundancy, the more likely that the answer occurs in a simple relation to the question. Therefore, there is no need to map questions to answers using complex lexical, syntactic, or semantic relations. In short, the systems rewrites the answer in order to transform it in the form of a probable answer and then passes this new sentence to a search engine. Various filters are then applied to the results retrieved by the search engine and multiple answers are presented to the users in a certain order, accordingly to the higher probability of being the right answer.

MULDER (Kwok et al., 2001) is an open-domain, fully-automated Question Answering system that, rather than treating queries as a set of keywords, parses the queries in order to determine its syntactic structure. This allows the system to transform the question into multiple queries that are used to find the answer. The system starts by constructing a tree to the structure of the question phrasing. A classifier analyses the tree and determines the type of answer that the system should expect. On the next step, queries are formulated using the tree. The queries are sent in parallel to the search engine (Google), increasing performance. From the results of the search engine, the system extracts parts of texts and generates possible answers. This probable answers are then classified and presented to the user. The fact that the system uses Google as a tool to find documents on the web, shows how a Question Answering system is more than a simple search engine and what conventional web services such as Google or Bing may be lacking.

2.2.3 *Answer Formulation*

Fragments of Texts / Text Highlighting

Some Question Answering systems are text based, meaning that the answer to the question posed by the user, will be a fragment of a text. By using this approach, a system becomes intricately related to other information access techniques such as "document retrieval", in which entire documents are retrieved, and "passage retrieval" in which chunks of text are returned (Lin et al., 2003) as answers. Even though, this does not offer much more than a web search engine such as Google, Yahoo, etc. it might be enough to answer a question, since it relies on the intelligence of the user to extract the exact meaning from the text.

In Attardi et al. (2001), PiQASso(Pisa Question Answering System) is proposed starting with a reference to an expression by Pablo Picasso: "Computers are useless: They can only give answers". This sentence might be a hint for the advantages of using such a system. PiQASso uses a paragraph search engine, where a collection of documents is stored, to retrieve paragraphs that are likely to contain the right answer. These paragraphs are ranked and presented as answers. If none is sufficiently likely to be an answer, the process is repeated using further levels of query expansion.

In Kaisser (2008) is explained how QuALIM (Kaisser and Becker, 2004) answers can be supplemented with paragraphs from the Wikipedia[2]. QuALIM is an open-domain Question Answering system which uses linguistic analysis of questions and candidate answers when attempting to find an answer. The system presents a succinct answer to the user questions, thus it cannot fit this methodology. Nonetheless, if the answer is retrieved along with some paragraphs or text from the Wikipedia, complementing the answer, then the system could be classified as belonging to both methodologies presented in this section. The idea is to return passages that can give context to the answer. Wikipedia is perfect as a source of information because it aims at covering all areas of knowledge; is free; is frequently updated; and most important when finding an answer, it is an encyclopedia, meaning that the information is very well structured.

2 https://www.wikipedia.org

Succinct Answers

Contrary to the last approach, this technique tries to give a succinct answer that directly satisfies the user information needs. Besides giving a direct answer in a natural language, some of these systems will also provide additional informations related to the topic. For example the question "Who is Cavaco Silva?" would be answered with "President of Portugal" and followed with some information like "Aníbal António Cavaco Silva; born 15 July 1939, is the 19th President of Portugal, in office since 9 March 2006".

Evi[3], formerly known as True Knowledge is an open-domain Question Answering platform that uses a base of the world's knowledge in structured form, combining common sense, factual and lexical knowledge (Tunstall-Pedoe, 2010). The system translates the user's questions into a language independent query that is executed using the knowledge base and an inference system. The result of the question is a direct answer to the user's question.

3 https://www.evi.com

3

A PYTHON QA SYSTEM: PROPOSAL

All the research that was done, provided sufficient background to theorize about the most important techniques, that could be used by the QA system proposed by this master's thesis.

Thus, this chapter proposes a system, or rather, the essential components of the system and the most critical decisions that were taken.

3.1 PROBLEM SPECIFIC DOMAIN

The objective of this master's work, as said in the Introduction (chapter 1), is to develop a closed-domain Question and Answering (QA) system.

The complexity of constructing an open-domain system that could rival with current systems would be considerable vast. Moreover, even if a system of this kind could be built in the short time assigned for this master's thesis, most certainly, it would not add many or any advantages and novelties to the already well explored field of open-domain question and answer systems.

Thus, a closed-domain system is more adequate when wanting to built something different from what is available today. Since it deals with a smaller scope of information, it is possible to analyse the problem in greater detail, to create a system that could solve a real problem of a certain domain.

When one is trying to learn a programming language, especially when we have no programming experience, it is very common for a person to have multiple questions regarding the syntax or the semantics of the language. The process of going to a search engine such as Google to find answers is a morose and tedious solution that consumes time and effort.

We propose a system that tries to solve this problem. Using a programming language as the domain of a QA system allows the beginner to progress faster by simplifying the process of finding answers to very concrete questions. The experienced programmer will also benefit from the tool, since the goal is to include various levels of complexity, some of them being far advanced for the inexperienced programmer.

3.1.1 *Chosen Domain: Python*

The Domain of the proposed QA system is the programming language Python[1].

Python has been capturing attention in recent years and as mentioned in Chapter 1, its popularity among both beginners and experienced programmers is rapidly increasing.

People want to learn Python mostly by themselves, resorting on large amounts of scattered information available throughout the Internet. The QA system being proposed could help programmers by reducing the effort necessary to find useful data among all this information.

Python is a widely used programming language, especially in industry and scientific research, and has the following characteristics:

- It is a general-purpose language;
- Supports multiple programming paradigms such as object-oriented, imperative and functional programming;
- Provides dynamic typing;
- Offers high-level data structures;
- Offers automatic memory management;
- Gives great emphasis to readability and provides means for programmers to write few lines of code in comparison to what would be needed with Java or C++.
- Provides a free extensive standard library available in source or binary form;
- Comes with a free interpreter, available to all major platforms.

These characteristics explain the reason behind the popularity of Python. Nonetheless, other languages such as Java, C++ or Ruby have some or all of these functionalities and could as well serve as the domain of the proposed system.

This aspect is not overlooked. Even though the system is built for the domain of the Python language, it should be capable of being used in the domain of another programming language without the need for structural changes.

3.2 KNOWLEDGE SOURCE

Large amounts of knowledge (data) can be found in a multitude of places, such as the Internet or in more classic sources, such as books or someone's individual knowledge. With

1 `https://www.python.org`

this in mind, and knowing that Phyton is a widely used modern programming language, it easy to infer that collecting information from the Internet is the most easy way to find information, regarding the use and definition of Python.

Although, accessing and retrieving information from the Internet can be extremely easy, we do not always have the means to guarantee the accuracy of this information. The question and answering system proposed by this thesis, as most closed-domain systems, tries to be as accurate as it is possible. Thus, the system does not search the web for information, regardless of the source. Knowledge is collected from reliable sources and used to populate the database. By using a database that is populated, prior to any questions made to the system, it is possible to treat and structure the information in the best way that suits the system's intentions.

As a developing methodology, the starting point is to build a system that is able to answer simple and generic questions and only when there is sufficient evidence of the system's robustness, more data is added to its database.

On a first phase, the Python Frequently Asked Questions (FAQ)[2], will be used as a knowledge source. Since the goal is to create a system capable of answering question in natural language about Python, the fastest way to create a functional prototype is to populate its knowledge base with answers and questions that the Python Software Foundation classifies as "frequently asked".

On a second phase, more data is added to the database in order to enhance the system capabilities. Even though the FAQ might answer to the most common questions, it does not contain all the information that, for example, an experienced programmer might need. To overcome this problem, the system resorts once more to the Python Documentation[3]. Besides the FAQ, the Python Documentation contains information, such as, library references, descriptions of the syntax and language elements, documentation about the Python/C API, etc.

3.3 COMPONENTS DESCRIPTION

Figure 1 depicts the proposed system's architecture. The system starts by accepting a question from the user. Then it parses the question in order to produce a query that will be used to retrieve information from the database. This information will then be analyzed and presented to the user.

All procedures are explained below.

2 `https://docs.python.org/faq/`
3 `https://docs.python.org/`

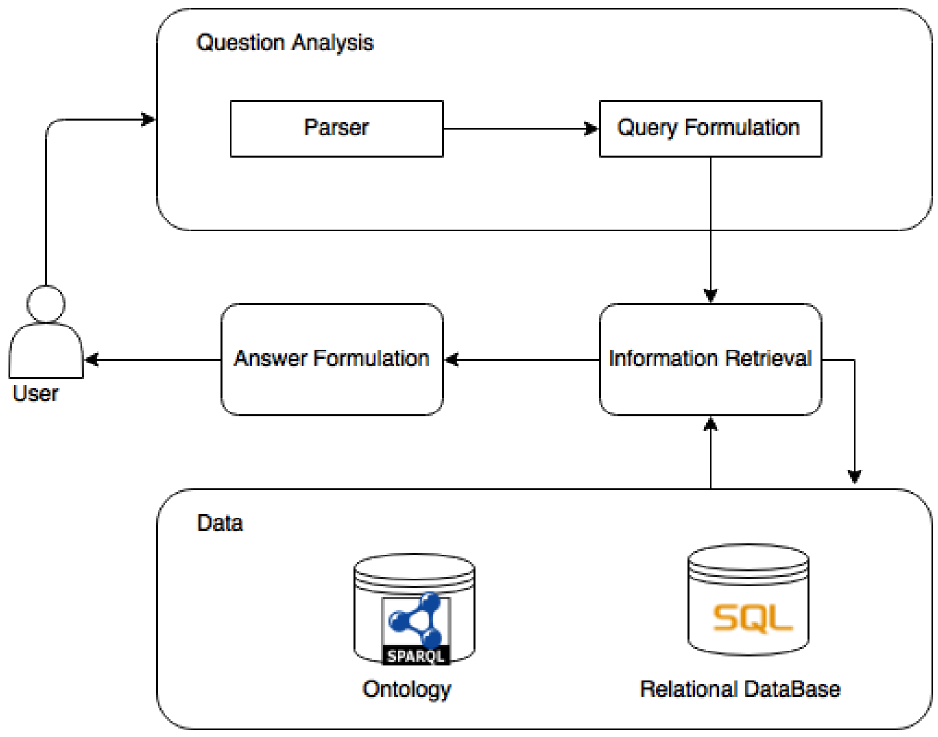

Figure 1: Proposed System's Architecture

3.3.1 *Question Analysis*

Every QA system starts by receiving a user's question. Understanding the meaning of the question is vital for the process of retrieving the correct information.

Some QA systems resort to complex NLP techniques, occasionally using semantic trees or converting the user's question into various queries in order to obtain multiple results. This master's thesis proposes a simpler approach to the question analysis problem. The knowledge source is very restricted and without redundancy, thus using techniques intended to produce a large number of possible answers, would not add any advantages.

The proposed system parses the question in order to identify keywords. Any other words are discarded leaving only the extracted keywords and preserving the order that they had in the user's question. This 'meta-question' is then used to construct a query in a language created specifically for this project. The query could then be translated to SPARQL or SQL to retrieve data from an ontology or a relational database. Further information about ontologies and relational databases is provided below.

3.3.2 *Data Storage*

The knowledge data should be stored in a way that offers stability, safety and fast access to data.

Since the system uses data from web sources but does not access these sources in real time, the web is not considered as a data storage, or to be more accurate, a place from were data can be retrieved. Systems that take advantage of the web, using for example, search engines or specific third party documents, are relying on others to provide a critical resource of any question and answer system.

The fact that the proposed system is closed-domain, means that the scope of information and consequently the amount of data that will be stored, is not very substantial, especially when compared with an open-domain system.

With all this in mind, it was concluded that the most adequate methodology is to use some type of structured knowledge base. Being this knowledge base an ontology, a relational database, or both.

The Oxford English On-line Dictionary[4] defines the word "Ontology" as:

1. *"The branch of metaphysics dealing with the nature of being"*;

2. *"A set of concepts and categories in a subject area or domain that shows their properties and the relations between them"*.

In computer science, an ontology is a practical application of a philosophical ontology. It can be viewed as a description or definition of concepts and relationships that exist for entities of a particular domain of discourse (Gruber, 1993).

A relational database is a database system whose data is managed using a structure. Data is represented in tuples and grouped into relations, allowing information to be stored in a high degree of organization.

In conclusion, the proposed system wants to take advantage of the relational and structural properties of relational databases and ontologies and the capacity of ontologies to define a domain of knowledge. All this complexity and care regarding the storage and organization of information is necessary to assess the assertiveness of the system and ease the process of distinguish and find information.

3.3.3 *Information Retrieval*

Since ontologies and relational databases are used to store information, a simple interpretation of the user's question and a somewhat conventional find matching operation, is not the best method for finding information.

4 http://www.oxforddictionaries.com

To manage both ontology and database this master's thesis proposes the conversion of the user's question into a query. This query is written and structured in a language created only for this purpose. The queries should contain the nuclear information of the user's question, while maintaining a certain degree of abstraction, making it possible to easily convert the queries to SPARQL and SQL.

SPARQL Protocol and RDF Query Language (SPARQL) is a semantic query language for databases where data is stored using the Resource Description Framework (RDF) format. SPARQL provides operations, similar to those of SQL, such as, SELECT, JOIN, SORT, which allow data to be retrieved from ontologies, written accordingly with the RDF data model.

Structured Query Language (SQL) is a very well known programming language based on relational algebra and tuple relational calculus, which is used to retrieve and manipulate data stored in relational databases.

Both SPARQL and SQL are the state of the art in their own fields of computer science, and therefore, perfect for the purpose of retrieving information from the data storage structures (ontology and relational database, respectively) used by the proposed question and answer system.

3.3.4 *Answer Formulation*

When providing an answer to a user's question, we are usually confronted with a dilemma. Should the answer contain just enough information to answer the question succinctly or should the system present additional information about the context of the answer?

We approach the problem by establishing a middle term between two methodologies. If a user's question makes sense within the domain of the system, then a succinct answer should be presented to the user. At the same time extracts of text or web links containing information about the topics of the question are also shown as a complement.

Other systems, as some presented in chapter 2, offer other functionality, such as multiple answers to the same question and answer ranking. These are great features, but for this project it was decided to go in a different direction. Since the system is dealing with a closed-domain, accuracy is put in front of redundancy. Meaning that to take benefit from a question and answer system of this kind, over a conventional search engine, the answer should be short and straight-forward. The additional textual information included in the answer should serve only as context to enrich the answer.

When the system can not find a satisfiable answer, a message notifying the user of this fact appears instead of the desired result. Obviously, this cannot be avoided. The main idea is that if the question was well formulated, accordingly to the system's domain, then an answer should be found. If it was not, then the user has to reformulate the question.

4 CONCLUSION

Before any development process, one should always explore what has already been theorized or done before, not only to get acquainted with the necessary knowledge for the development process, but also to know were a contribution could be most needed. The research work that was already performed for the writing of this document, will certainly prove to be a good basis for the successful conclusion of this master's thesis. For the time being, the research work has already proved the initial suppositions that a QA system about the Python domain is something new, and has the potential of offering great assistance to Python developers.

4.1 WORK SCHEDULE

The duration of this master's project is estimated to be one year. It was agreed that the work would be split into four phases according to the following schedule:

MONTH 1ST **TO** 2ND: The first two months period was used for the search of the basic bibliography and its revision.

MONTH 3RD **TO** 4TH: This second phase will be used for search, discuss and select all the developing methods that will be employed. Research information about Python and ways to store, access and represent it are also part of this phase.

MONTH 5TH **TO** 10TH: The development of the tool will happen during this phase.

MONTH 11TH **TO** 12TH: The last two months will be used to evaluate, test and draw conclusions relating the outcomes of the developed tool and the studies done in the project.

The thesis report, or dissertation, will be written along all the four phases above.

The first phase is entirely encompassed by what has already been done, and its conclusions are explained in this document, particularly in chapter 2.

The second phase is currently in progress. Some conclusions, regarding the architecture of the proposed QA system were already achieved and were explained in chapter 3. This phase is extremely important, since it provides the basis for the construction of the system. In order to maximize results and minimize wasted time and software bugs it is essential to know in some detail the specifications of a QA system and choose the best tools and methods that will be used to build the proposed QA system.

In the current phase and the next, the focus will always be, how to build and implement the proposed QA system. The first phase, proved to be important in comprehending a QA system and it's various types and approaches. While not forgetting the research made in the first phase the next phases should directly aim at the methodologies that are necessary to build the proposed Python QA System.

The development process will follow the next steps:

- Build a front-end where the user could place questions and answers are presented by the system.
- Enrich the front-end with a back-end capable of processing the users questions.
- Store information about the Python language that will be used to extract and formulate answers.
- Connect system and data base in order to produce a working QA system capable of answering exact questions.
- Create variants to questions and enrich the matching mechanism, in order to supplement the system with the capability of understanding questions formulated in various ways.
- Add more data to the database, to increase the systems "knowledge".
- Evaluate the system by running extensive tests, and correct any possible bugs.

BIBLIOGRAPHY

T. Andreasen, R.R. Yager, H. Bulskov, H. Christiansen, and H.L. Larsen. *Flexible Query Answering Systems: 8th International Conference, FQAS 2009, Roskilde, Denmark, October 26-28, 2009, Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009. ISBN 9783642049569. URL https://books.google.pt/books?id=gqDf__ULmR8C.

Giuseppe Attardi, Antonio Cisternino, Francesco Formica, Maria Simi, Alessandro Tommasi, and Cesare Zavattari. Piqasso: Pisa question answering system. In *TREC*, 2001.

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009. ISBN 9780596555719. URL https://books.google.pt/books?id=KGIbfiiP1i4C.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. *KR*, 6:260–270, 2006.

Chantana Chantrapornchai, Archara Sangchan, Atitaya Kantankul, Chidchanok Choksuchat, and Suchitra Adulkasem. Ontology-based question answering system development for case study of thai cats. *International Journal of Multimedia & Ubiquitous Engineering*, 9(3), 2014.

Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, and Andrew Ng. Web question answering: Is more always better? In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 291–298. ACM, 2002.

Oscar Ferrández, Rubén Izquierdo, Sergio Ferrández, and José Luis Vicedo. Addressing ontology-based question answering with collections of user queries. *Information Processing & Management*, 45(2):175–188, 2009.

Lance Fortnow and Steve Homer. A short history of computational complexity. volume 80, pages 95–133, 2003. URL http://www.cs.bu.edu/techreports/pdf/2002-028-computational-complexity-history.pdf.

Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224. ACM, 1961.

Bibliography

Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

W.J. Hutchins and H.L. Somers. *An introduction to machine translation*. Academic Press, 1992. ISBN 9780123628305. URL https://books.google.pt/books?id=NkpQAAAAMAAJ.

Michael Kaisser. The qualim question answering demo: Supplementing answers with paragraphs drawn from wikipedia. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Demo Session*, pages 32–35. Association for Computational Linguistics, 2008.

Michael Kaisser and Tilman Becker. Question answering by searching large corpora with linguistic methods. In *TREC*, 2004.

Cody Kwok, Oren Etzioni, and Daniel S Weld. Scaling question answering to the web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001.

Jimmy Lin, Dennis Quan, Vineet Sinha, Karun Bakshi, David Huynh, Boris Katz, and David R Karger. What makes a good answer? the role of context in question answering. In *Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2003)*, pages 25–32, 2003.

M.F. McTear and T.V. Raman. *Spoken Dialogue Technology: Toward the Conversational User Interface*. Springer London, 2011. ISBN 9780857294142. URL https://books.google.pt/books?id=7B7jBwAAQBAJ.

Diego Mollá and José Luis Vicedo. Question answering in restricted domains: An overview. *Computational Linguistics*, 33(1):41–61, 2007.

John M Prager. Open-domain question-answering. *Foundations and trends in information retrieval*, 1(2):91–231, 2006.

Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 41–47. Association for Computational Linguistics, 2002.

Unmesh Sasikumar and L Sindhu. A survey of natural language question answering system. *International Journal of Computer Applications*, 108(15), 2014.

Clive Thompson. What is ibm's watson. *New York Times Magazine (June 2011). http://www. nytimes. com/2010/06/20/magazine/20Computer-t. html*, 2010.

William Tunstall-Pedoe. True knowledge: Open-domain question answering using structured knowledge and inference. *AI Magazine*, 31(3):80–92, 2010.

Maria Vargas-Vera and Miltiadis D Lytras. Aqua: A closed-domain question answering system. *Information Systems Management*, 27(3):217–225, 2010.

John M. Zelle. *Python Programming: An Introduction to Computer Science*. Franklin Beedle Series. Franklin, Beedle, 2004. ISBN 9781887902991. URL `https://books.google.pt/books?id=aJQILlLxRmAC`.