Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare

LUCRARE DE DIPLOMĂ

Agent conversațional pentru interacțiunea cu personalități istorice

Conducător Științific:
Ș.l. Dr. ing. Traian REBEDEA

Autor:
Adrian BOGATU

București, 2015

University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

Bachelor Thesis

Conversational Agent for Interacting with Historical Figures

Scientific Adviser:
Dr. Traian REBEDEA

Author:
Adrian BOGATU

Bucharest, 2015

Abstract

This paper describes a conversational agent that answers questions pertaining to a given historical figure. The answers are selected from a set of sentences extracted from the biographical text of that personality using syntactic and semantic analysis on both the question and the sentences at hand. This is accomplished with a top-down approach, starting from the entire biographical text and working our way down, in several steps, to the

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Designing and building a conversational agent that impersonates a historical personality, a public figure or a know celebrity and can answer user input questions about that person regarding the life and work of that person is a challenging task implying various methods like information retrieval, natural language processing (NLP) and question answering (QA). These methods are broad sub-fields and integral parts of the domain of Artificial Intelligence (AI).

These sub-fields and the difficulties accompanying them are described, in short, in the next three paragraphs.

Information Retrieval. Information retrieval is a method of gathering relevant information pertaining to a given problem from a collection of resources. The problems with this approach stretch from knowing how people use and process information and how the knowledge can be represented, to the text indexing methods and relevantly scoring parts of the information extracted from such texts.

Natural Language Processing. Natural language processing is a field of AI that explores how the natural language can be analyzed, understood and manipulated by a computer. The research in NLP attempts to find out how humans express and understand language so that appropriate tools and techniques can be developed to make computers perform the desired tasks [6]. NLP is an essential part of creating a conversational agent that interacts "naturally" and "coherently" with a human through language. The main difficulties that arise in NLP are of semantic nature: understanding of the context in which an analyzed sentence appears and overcoming the ambiguities that appear in natural human language. The problem pertaining to the conversational agent in this thesis is that it is needed to understand the human language from both the biographical corpus exctracted for a given personality and the question posed by the user.

Question Answering Systems. QA systems are programs that attempt to find the most relevant and accurate answer for a given question. The main issue that appears in QA is the need for taxonomies depending on the set of conceptual categories that covers the wanted domain for the potential questions. Other issues are: question rephrasing, answer extraction and answer formulation [5].

All the aforementioned difficulties add up to make the task of creating a conversational agent challenging.

The attempts of programming machines to interact with humans via natural language and "behave" as if they are human started when Alan Turing raised the question "Can machines think?" [28] more than 60 years ago. Since then, a considerable part of the computer science community has dedicated its time and resources to create computer programs that can convince people that the answer to this question is "Yes".

Over the years, notable advancements have been made towards this goal. The testament to that are various programs, from simple agents, like ELIZA [31], A.L.I.C.E. [30] or Freudbot [15], to more complex and "smarter" agents, like Cleverbot[1] and IBM Watson [9]. The mentioned agents are summarily described in the Related Work chapter.

[1] Cleverbot Webpage, http://www.cleverbot.com/

Chapter 2

Project Overview

The conversational agent for interacting with historical figures is an already existing project and it is designed as a rule-based system, with rules generated from information extracted from DBpedia and Wikipedia. The solution proposed in this thesis is a complement to the initial implementation of the agent and it is used as a fallback method in case the first approach cannot deliver an answer to a user question.

The following sections present why this project is relevant, what are the goals it attempts to achieve and a short description on the steps taken to achieve these goals.

2.1 Project Motivation

In an age of computerization and advancement of technology, natural interaction with devices for both socialization and education has become more and more important. This started since the dawn of the Internet, when information has become wildly accessible.

2.2 Project Objectives

The main objective of this project is to build a question answering system in the form of a conversational agent that can be used as a historical e-learning method for children and a virtual tour guide in museums. Therefore, the program must have a high correctness rate and also, even if the conversational agent has a set of possible answers, but the answer are scored poorly because the probability of them answering the question is small, these answers should be ignored.

The main focus of this project is to make the conversational agent be able to correctly answer questions about different moments in their life, like important dates ("When did you win your first war?", "When did you fight in the battle of ...?" etc.), important places ("Where were you born?", "Where did you die?" etc.), important people ("Who was your wife?", "Who are your children?" etc.) or other miscellaneous facts ("What was your debut album?", "What instrument did you play?" etc.). As one can see, the relation between the question words and the category of the answer is not coincidental.

More details about the use of the type of the question (*when, where, who, what etc.*) can be found in chapter 5.

To accomplish all stated above means that the conversational agent must do two things. First, the agent has to understand the question, in order to create relevant queries to interrogate its knowledgebase. Second, the program must store the information about a given person in a structured manner so that the retrieval of relevant information and analysis and manipulation of content is easy to achieve.

In addition to the accuracy of the information in the answer, the program should also achieve good results in both speed performance and memory usage. The methods of doing that are presented at the beginning of subsection 5.2.2.

2.3 Project Description

The conversational agent that is the subject of this thesis is implemented as a dialog system that is intended to impersonate a given historical figure or, in general, a known person that has accessible biographical texts. This approach implies that the chat-bot can be "aware" of the historical facts and details in the life of the person it impersonates by extracting relevant information from the biographical corpus of that person. This means that the conversational agent is able to find the information that is semantically closest to the question and, therefore, answer the question correctly.

As stated before, the project is made up of two components. The first one is a pre-existing part of this project and it implies using a set of generated rules that help the agent retrieve the answer based on the input question. The second one is the component described in this thesis and it employs the answer sentence selection method and sequentially filters the sentences from the biography until the correct (or, at least, most probable) answer remains.

These two components are introduced in the next two paragraphs and their implementation is explained in depth in chapter 5.

Rule-based system. A rule-based system is a system that attempts to match a given input on a set of rules in order to produce an output. The initial implementation uses ChatScript, a chat-bot engine that interprets scripts containing rules of the form of question-answer pairs. ChatScript is presented in section 4.2. The rules are generated starting from DBpedia properties and matching them with information extracted from Wikipedia. Details on the implementations can be found in subsection 5.2.1.

Answer sentence selection. Answer sentence selection is the task of finding a sentence that most likely answers the given question starting from a set of sentences where the answer might be. In the case of the conversational agent being discussed, the set of possible answers is the set of all the sentences in the Wikipedia article of the historical figure being impersonated. So, given a question, the task at hand is to select the right sentence (or to filter out the wrong ones) that has the highest probability of being the correct answer. The mentioned probability is a function that depends on different methods of scoring, which are explained later on, in the thesis.

The second approach brings improvements to the original method. Arguments to support this statement are presented in subsection 5.2.2.

Chapter 3

Related Work

The following sections in this chapter present advancements and similar work in the areas of NLP, question answering, and conversational agents.

The projects in each one of the two sections are presented in a chronological order, in order to put emphasis on the continuous advancements in their particular field.

3.1 Open Domain Question Answering

[13], [14]

[17]

[7]

[26]

[22]

The highlight of this section is the well known IBM Watson [9].

[25]

[33]

3.2 Conversational Agents

E.L.I.Z.A. [31]

* Cleverbot

Freudbot [15]

[16]

Chapter 4

Tools

The next sections present the most important programming tools used to build the conversational agent. The first two sections describe tools that are part of the ontological approach to the implementation of the program. The next ones discuss tools used in both the initial method and the answer sentence selection approach.

4.1 DBpedia

DBpedia[1] is a project that aims to convert content extracted from Wikipedia into a structured dataset, that is subsequently made available on the World Wide Web. The final purpose of this project is to provide an interface so that the users can query Wikipedia in a structured manner using Semantic Web techniques. The structured information is published using the RDF[2] specifications. In addition, DBpedia links its dataset with other published open datasets, in total reaching 2 billion RDF triples (in 2007) [2].

The DBpedia datasets can be accessed in three ways:

- through the Linked Data interface, where the DBpedia resources (published as RDF data) can be accessed using URIs.
- using the SPARQL Endpoint that supports specific SPARQL queries.
- downloading RDF dumps containing larger parts of the DBpedia dataset.

The important aspect of DBpedia for building our conversational agent is that DBpedia has a separate, independent *Persons* dataset that has more than half a million RDF triples containing information (extracted from Wikipedia articles written in English) for around 760,000 people, as of 2012 [19].

[1] DBpedia, dbpedia.org
[2] Resource Description Framework, http://www.w3.org/RDF/

4.2 ChatScript

ChatScript[1] is an engine for building conversational agents. ChatScript is a rule-based system relying on its own scripting language (similar to AIML[2]) to model the conversational behavior of an agent. Its purpose is to "pattern-match on general meaning" by using "sets of words and canonical representation." [32]

A chat-bot is modeled through a set of script files that contain rules. A rule is formed from a pattern and a response. The response represents the output that a ChatScript bot will provide if the input matches the pattern. Two ChatScript rules are presented in Listing 4.1. The elements in the parentheses constitute the pattern and the sentence after the pattern represents the answer returned if the user input matches the pattern. The $*$ (star) symbol is a wildcard that can match none, one or more words.

```
u: ( Where * you * born ) In the capital.
u: ( When * born ) This century.
```

Listing 4.1: ChatScript rules example

4.3 WordNet

WordNet[3] is an online lexical reference database [24, 23] containing: words, lexical relations and sematic relations.

Word. A word is defined as a pair (*f*, *m*) between a form *f* (the string representation of the word) and a meaning *m*. If a word has more than one meaning (it is polysemous), WordNet differentiates between the meanings and keeps a pair (*f*, *m*) for each meaning. The polysemy of a word can be extended to its part of speech, i.e. a word can have different meanings depending on its part of speech; e.g. *die* can be a polysemous noun meaning either *dice* or is "a device used for shaping metal", or a verb meaning *decease*. In the WordNet database only "open-class words" are present. The database includes about 155,000 nouns, verbs, adverbs and adjectives [8].

Synonym set. The words are grouped into synonym sets, also known as *synsets*, which are the core element in WordNet. A synset is used to represent the meaning (or sense) of a word [23].

Lexical relations. A lexical relation between two words is a relation between the form of the words. For example, synonymy and antonymy is stored in the database as a lexical relation [24].

Semantic relations. A semantic relation is a relation between word meanings and is stored in the WordNet database as a pointer between synsets. The semantic rela-

[1] ChatScript, http://chatscript.sourceforge.net/

[2] AIML: Artificial Intelligence Markup Language, http://www.alicebot.org/aiml.html

[3] Princeton University "About WordNet." WordNet. Princeton University. 2010., http://wordnet.princeton.edu

tions between the synsets are useful to model concepts like: hyponymy, hypernymy, meronymy etc.

4.3.1 MIT Java WordNet Interface

The Java WordNet Interface (JWI) is an Application Programming Interface (API) written for the Java programming language that is used to access and query the WordNet database files. JWI features calls to retrieve index words and synsets and calls that allow following lexical and semantic pointers [11]. Mainly, it can be used to access synonyms, antonyms and hypernyms/hyponyms of a given word. Three most important advantages to use JWI as presented in [12] are:

- JWI provides both file-based and in-memory dictionary implementations, allowing you to trade off speed and memory consumption
- JWI sets no limit on the number of dictionaries that may be instantiated in each JVM
- JWI is high-performance, with top-ranked speeds on various retrieval metrics and in-memory dictionary load time

It can be seen in the benchmarks from [12] that JWI is one of the fastest libraries for WordNet. Average access time for retrieval of an entry and for iterating through entries in the database are shown in Table 4.1.

Object	Retrieval time (μs)	Iteration time (μs)
Index Word	12.3	296
Synset	7.1	798
Word-by-Sense-Key	17.2	141
Exception Entry	16.1	4
Synsets by Index Word	-	1.8s

Table 4.1: Average time of access of WordNet database

4.4 Apache Lucene

Apache Lucene[1] is a text-search library written in Java. It provides an API for performing common search tasks like text indexing, querying, highlighting results and others [3]. Lucene achieves high performance due to the inverted index [29] approach. In addition, the search speed can be increased by placing the text in memory, using the RAMDirectory class.

Lucene's three main features, presented below, are: analysis of incoming content and queries, indexing and storage and searching [3].

[1] Apache Lucene Core, https://lucene.apache.org/core/

Language Analysis. The texts to be searched and the queries are stored internally in a modified form for faster access. The transformations made are: character filtering, tokenization, stemming, lemmatization, stopwords removal and others.

Indexing and storage. Lucene supports inverted indexes on more than one field per document, useful for annotating the text with different information (e.g. ISBN). All this data can be stored on a persistent device (for large sets of documents) or, as previously stated, in the RAM memory for faster access (for reduced size documents). In addition, Lucene can be configured to use different kinds of query scoring, based on total word frequency, unique word count and total document frequency of all words [3].

Searching. The implementation of Lucene's querying supports several types of searching mechanisms, like: wildcards, fuzzy search, proximity search, binary operators and others [3]. Querying can be optimized by providing a number of top matches after which the search stops.

4.5 Stanford CoreNLP

Stanford CoreNLP is one of the most exhaustive tools for natural language analysis. It is an open source project written in Java and has a comprehensive API that is easy to use. Architecturally, Stanford CoreNLP is a pipeline that annotates the input text with relevant information, like the part of speech of the words. It also generates graph-like structures containing links between words, representing relations like: syntactic dependencies and co-references. The execution flow of the annotator is represented in Figure 4.1.

The provided annotators that can be included in the processing are: "tokenize", "cleanxml", "ssplit", "truecase", "pos", "lemma", "gender", "ner", "regexner", "parse", "sentiment" and "dcoref". Most of the annotators are built as separate modules that are integrated afterwards in the core. Some of these main modules are presented in the following sections.

4.5.1 Stanford Tokenizer

Although the tokenizer is not an independent part of the Stanford NLP project, it appears in multiple modules of the project. The role of the tokenizer is to split the raw text into a sequence of individual tokens [20]. The tokenizer uses Penn Treebank style tokenization[1]. The functionality of the tokenizer is implemented in the *PTBTokenizer* Java class.

During the tokenization process, besides the actual list of tokens, the *PTBTokenizer* also generates a text annotation for each token in order to retrieve the original form of the text when needed. This is mostly useful when the lemmatization process, described below, is applied to the text.

[1]Penn Treebank Tokenization Specifications, https://www.cis.upenn.edu/~treebank/tokenization.html

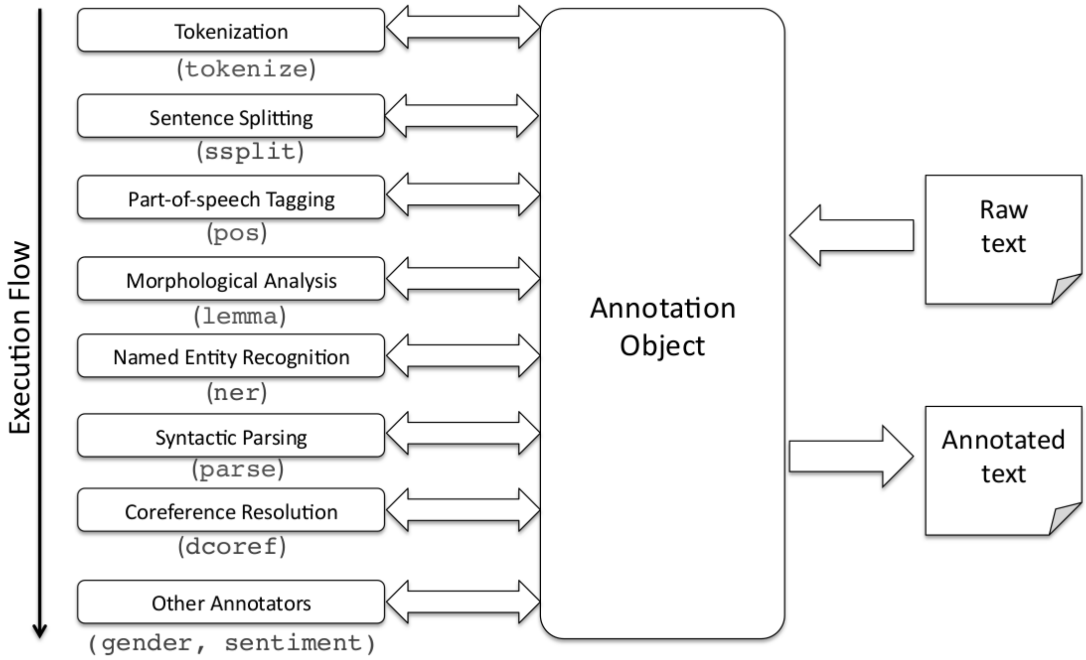

Figure 4.1: Stanford CoreNLP execution pipeline [20]

Another use for the tokenizer is to accomplish sentence splitting (added with the "ssplit" annotator option as mentioned before). This is done by tokenization after one of the sentence-ending characters (**.**, **!** and **?**) if they are not grouped with other characters into a token (such as for an abbreviation or number) [1].

After the tokenization has been performed, the individual words can be lemmatized (using the "lemma" annotator option), meaning that a word is annotated with its dictionary form (or base form). For example: *has*, *had* and *having* become *have*; *children*, *child's* and *children's* become *child*.

4.5.2 Stanford Log-linear Part-Of-Speech Tagger

The purpose of the Part-Of-Speech (POS) tagger is to label words with their corresponding POS tag, using a maximum entropy POS tagger [20]. The English POS tagger uses the Penn Treebank tagset described in [21]. This type of tagging is particularly useful for syntactic analysis in NLP applications, for example it is important to determine a word's part of speech when finding the appropriate synonym for the word if the word is polysemous.

The POS tagger uses a cyclic dependency network model trained using lexical features of words extracted from the Penn Treebank dataset. The network is then used as a classifier fed with the given text as input. The architecture of the dependency network used for feature-rich POS tagging is described in depth in [27].

4.5.3 Stanford Named Entity Recognizer

The Stanford Named Entity Recognizer (NER) is an annotation tool that labels words (or sequence of words) that are likely to stand for names of things (like people and places). Stanford NER uses a statistical model trained on a collection of Reuters articles annotated with four entity types: person (PER), location (LOC), organization (ORG), and miscellaneous (MISC) [10]. The Stanford NER dataset contains statically assigned types for a set of entities, but it can also infer the type from the context, as seen in Table 4.2. Table 4.2 shows that *Bucharest* is a known entity with its type previously assigned as opposed to *Ploiești* which is wrongly considered a person in the sentence *"I like Ploiești"*. Despite this fact, the NER successfully identifies *Ploiești* as a location in the *"I live in Ploiești"* sentence.

Stanford NER is useful for the Stanford Co-Reference System, described in subsection 4.5.4, where a relation needs to be established between a pronoun (her, his, it etc.) and a name of a person.

Sentence	Token	NER
I like Bucharest	Bucharest	Location
I live in Bucharest	Bucharest	Location
I like Ploiești	Ploiești	Person
I live in Ploiești	Ploiești	Location

Table 4.2: Inference of types using the Named Entity Recognizer

4.5.4 Stanford Deterministic Coreference Resolution System

Stanford's Coreference Resolution System implements mention detection and both pronominal and nominal co-reference resolution [20]. This tool is used to link pronouns to the entities they refer to. In order to achieve that, the architecture of the system applies several deterministic co-reference models, one after the other. The first, and most important step of this algorithm is the *mention detection*, where the nominal and pronominal mentions are identified using an algorithm that selects all noun phrases, pronouns and named entity mentions. After that, the co-reference models are applied from highest to lowest precision. All the steps of the algorithm are described in [18].

An example of this system's functionality is presented in Figure 4.2, where the pronoun *he* is linked to the word *Lennon*, the name of the person the pronoun refers to.

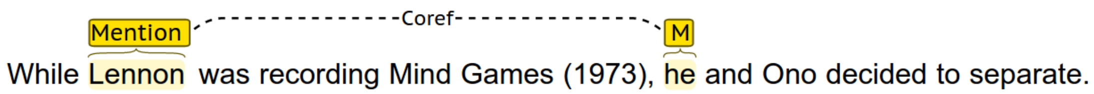

Figure 4.2: A co-reference example where the mention of the pronoun *he* is connected by the *coref*-type relation with the mention of the enitity *Lennon*

Chapter 5

Implementation

The architecture of the application contains two main modules: the user interface (UI) in the form of a Web Application, presented in section 5.1 and the actual implementation of the conversational agent, detailed in section 5.2.

In the next sections reference to the chat client, chat server and chat-bot signify the front end user interface, the back end of the web application and the conversational agent program respectively.

5.1 Web Application

The web application is the interface with the help of which the user interacts with the conversational agent. It implements a chat client, a program where the user can input its questions and see the answers processed by the conversational agent and returned by the chat server. It also implements a chat server that connects to the chat-bot's endpoint. After the connection succeeds, the chat server passes the query received from the client to the chat-bot, waits for a reply and then sends the reply back to the client.

The chat client is described in subsection 5.1.1 and the chat server is described in subsection 5.1.2. The chat-bot's endpoint of the aforementioned connection between him and the chat server is explained at length in section 5.2.

5.1.1 Front End

The front end is written in HTML, CSS and JavaScript and it is composed of two main screens: the first page where the user can input the name of the personality he wishes to speak to; the second page, the actual chat box, where the conversation is displayed, and the input box, where the user can write the question and submit it.

The first screen is shown in Figure 5.1. Clicking the *Enter* button would send a request to the chat server that would later on initialize the chat-bot with the inputted personality.

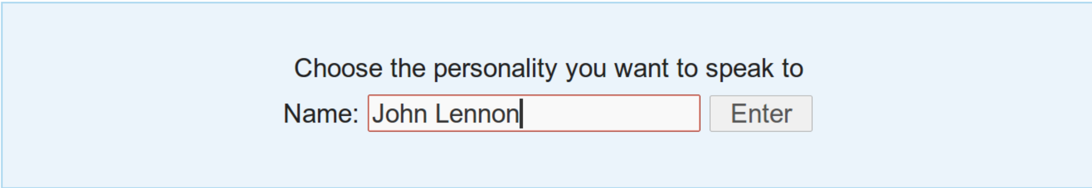

Figure 5.1: The first page of the Web App where the user can input the name of the personality he wants to talk with

In Figure 5.2, the chat box containing a few questions and answers can be seen. At the bottom of the chat box there is the input text box where the user can write its questions and submit them.

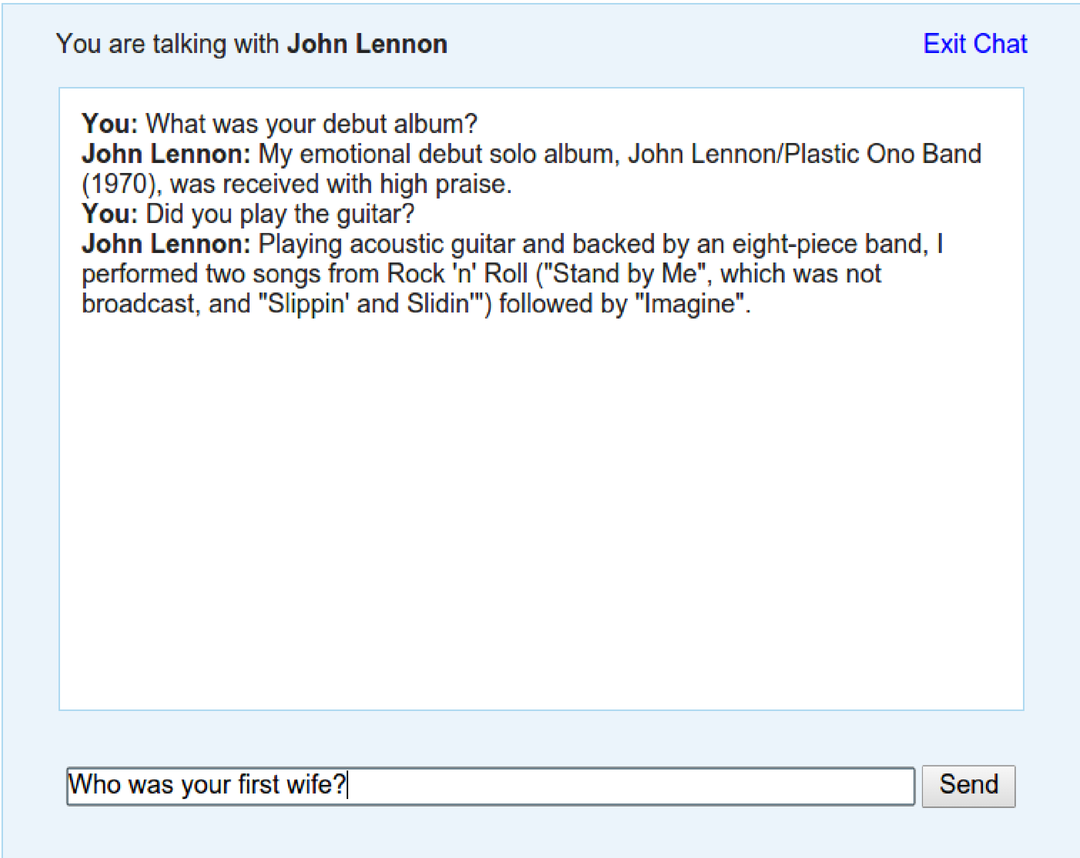

Figure 5.2: The page where the user can submit questions and see the replies

The conversation is saved on the client-side, in memory, using JavaScript, and is persistent until the page is refreshed or closed.

The questions are submitted by sending a synchronous HTTP GET request to the server. The request contains the question. The request is sent using AJAX (Asynchronous JavaScript + XML) and contains a random generated number in the URL so that the browser doesn't serve a cached page. For example, http://<hostname>/chatbot.

php?question=What+was+your+debut+album%3F&t=0.023473239736631513 sends the question *"What was your debut album?"* to the chat server.

5.1.2 Back End

The server side of the web application is made up of XAMPP[1,2] and the PHP scripts that implement the connection with the chat-bot.

Functionally, the communication is accomplished by using a local socket to socket connection between the web server and the conversational agent server. A snippet of the code the shows the principles of communication between the two servers is presented in Listing 5.1. The main aspects of the program are highlighted through comments in the code. The flow of the communication is shown in Figure 5.3.

```
/* Establish connection */
        $sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
        if ($sock == FALSE) {
                echo "error creating socket";
                exit("error");
        }
        $result = socket_connect($sock, $address, $port);
        if ($result == FALSE) {
                echo "error connecting";
                exit("error");
        }

/* Send the question to the conversational agent */
        $question = $_GET["question"] . "\n";

        $err = socket_write($sock, $question, strlen($question));

/* Wait for the answer */
        $answer = socket_read($sock, 12000);
        echo "$answer";

        socket_close($sock);
```

Listing 5.1: Socket communication between the web server and the chat-bot

[1]XAMPP Websitehttps://www.apachefriends.org/index.html

[2]XAMPP is a bundle containing the Apache HTTP server, a MySQL server and a PHP and Perl interpreter

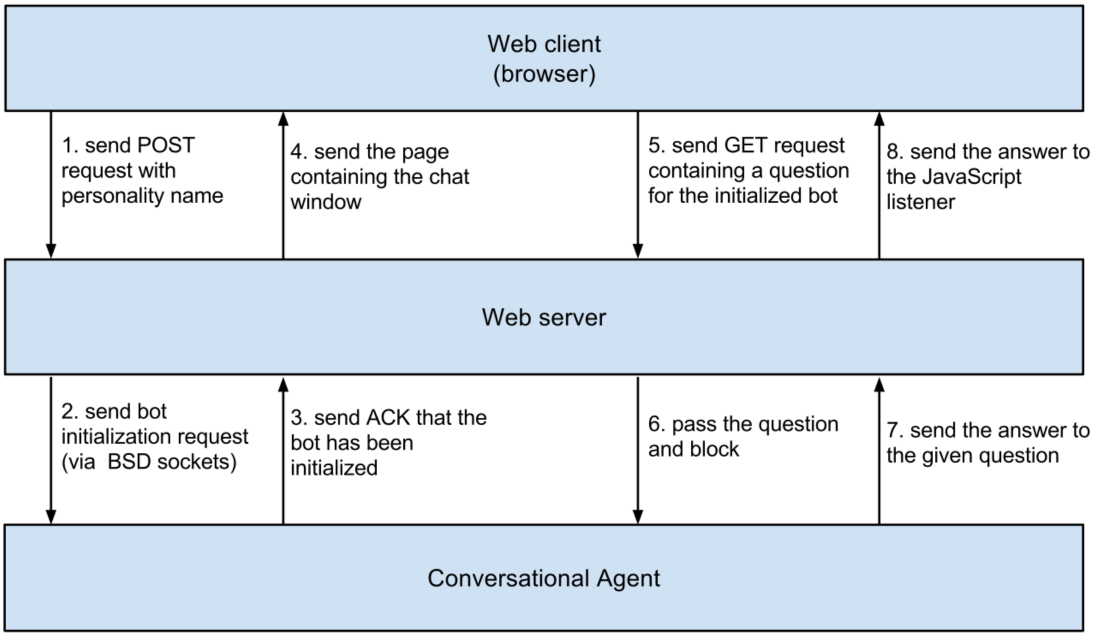

Figure 5.3: The communication flow between the client, web server and conversational agent server

5.2 Conversational Agent

This section contains the architectural decisions and the implementation details of the conversational agent. First, in subsection 5.2.1, the rule-based system approach using the DBpedia ontology and the ChatScript engine is described in short. Next, the approach to implementation employing the *answer sentence selection* method is presented at length.

5.2.1 Rule-based System Approach

This approach relies on the chat-bot engine ChatScript (see section 4.2) and its scripting language. For the chat-bot to answer question, it must match them against the rules in the ChatScript files. Because multiple personalities can be instantiated and no two people share the same biography, it is imperative that a set of ChatScript files containing rules must be generated for each personality the user wants to talk to.

To generate the pattern inside a ChatScript rule and also the relevant response to a certain pattern, a solution is needed for expressing different properties and for finding their value in a Wikipedia article respectively. These two methods are described in the next couple of sections.

5.2.1.1 Expressing a Property

Expressing a property implies finding a statistical model that maps properties extracted from DBpedia to a set of word groups that appear in Wikipedia and that relate to the value of said property. In order to find the best way a property is expressed, a large number of properties from a lot of DBpedia pages are needed. After filtering out properties that are not necessary for building the conversational agent, the only task remaining is to parse the Wikipedia corpus to find the value of that property in the sentences where the person which each article is about appears in [4].

The best way to identify the way a property might be expressed is to analyze the parse tree generated using Stanford CoreNLP and observe how the value of a property influences the structure (topology) of this parse tree. After interpreting numerous examples of sentences that include the value of a property and theorizing the way a property might be expressed, the conclusion is that a good way to express a property is using the words on the path from the property (excluding the value) to the root verb.

Having the desired sentence identified, we annotated it using Stanford CoreNLP and obtained a syntactic parse tree. Analyzing this tree, we determined that the root is the verb directly connected with the subject. Having the parse tree, we considered that the best way to express the property is the path from that property to the root verb.

Some examples of entries in the knowledge base are presented in Table 5.1. The property is extracted from DBpedia and the lexicalization represents an enumeration of ways the given property appears to be expressed in the Wikipedia articles.

DBpedia Property	**Lexicalizations**
birthDate	born, born in
almaMater	receive in, graduate as
award	award, receive
college	graduate from, attend
deathPlace	die in
profession	serve in, become
spouse	marry, marry to

Table 5.1: Examples of how specific DBpedia properties are most often expressed

Two examples of syntactic parse trees can be seen in Figure 5.6. The parse tree in Figure 5.4 is for the sentence "Albert Einstein was born in Ulm." It can be noticed that the DBpedia property that connects "Albert Einstein" and "Ulm" and for which we want to identify a lexicalization is the "birthPlace" property and the lexicalization is "born in". In the second example, Figure 5.5, the method still stands, and having the "award" property and the "Nobel Prize" associated value, it can be determined that the manner of expressing this property is with the "receive" lexicalization.

5.2.1.2 Generating the ChatScript Files

After building a knowledge base of common word choices for a particular DBpedia property, the next step is to build the ChatScript files that are the base to the ChatScript

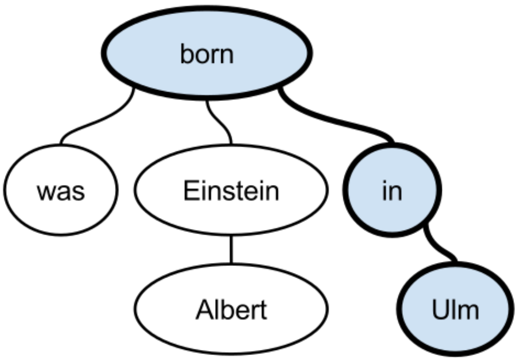

Figure 5.4: Parse tree for "Albert Einstein was born in Ulm." with path expressing a DBpedia property being highlighted

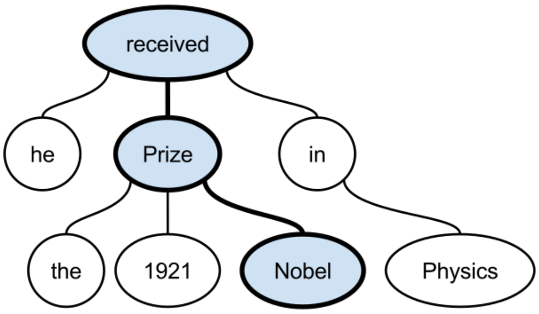

Figure 5.5: Parse tree for "He received the 1921 Nobel Prize in Physics" with path expressing a DBpedia property being highlighted

Figure 5.6: 2 parse trees

engine. In order to generate these files for every desired person, a few steps are needed:

1. fetch that person's Wikipedia page
2. split it into phrases and keep only those that have the person as a subject
3. select only those that express a property from DBpedia matching the expression against the generated property-expression database
4. create a rule-answer entry to add to the ChatScript files

Continuing from the last point, the rule is represented as an expression of a property that appears both in the analyzed sentence and the knowledge base. The answer is the analyzed sentence from Wikipedia which is converted to be expressed in the first person. The conversion from the third person to the first person of the sentence is accomplished with Stanford's Part-of-Speech Tagger and CoreNLP. All these patterns are written in ChatScript's file hierarchy. For a fast and easier way to find the answer, we arranged ChatScript's files by the properties of the person. [4]

5.2.2 Answer Sentence Selection Approach

Because it is impossible to build an exhaustive rule-based system, a secondary approach to this problem has to be taken into consideration in order to give a good answer. In addition, ChatScript has its own limitations coming from the fact that it ignores a rule after it first matches it. Therefore a fallback option is needed in case the former approach fails to provide an answer.

Considering the fact that the former approach gives better answers the simpler and more common questions are asked, we observe that either it fails to match questions that are more complex or it has to many matches for a question that uses a common verb (like "to be" or "to have") and the results will be inaccurate or noisy. The solution to avoid this is to try and find the answer directly from the source of the previously described knowledge base with an ad hoc approach considering every sentence from that respective source.

Following the goal of having to answer a question for a certain historical figure, the set of possible answers is reduced to a set of sentences from that person's biography.

This leaves us with the task of identifying a sentence from a biography that has the highest probability of correctly answering the question at hand. Considering what was previously stated, that this approach tries to find the answer to a more complex question, we can assume, at least for now, that there is a great deal of semantic information embedded in the form of the question (lexically and semantically) so that the chances are a part of the answer textually lies in the question. Therefore, what we can do is actually search for the question (or paraphrases of the initial question) in the reference text.

The design of this approach is built with three main goals in mind: performance regarding speed, small memory footprint and high answer accuracy.

Speed. Good speed efficiency is achieved by using a pipeline architecture, where the set of potentially correct answers is iteratively reduced as the methods for sentence selection increase in complexity. This pipeline can be viewed as a series of sieves that sequentially filter the set of input sentences. The pipeline architecture is discussed in depth in subsubsection 5.2.2.1. In addition, the use of Lucene's in-RAM storage (described in section 4.4) saves a lot of time from accessing persistent storage devices (HDD, SSD etc.). More than that, all tools used are highly optimized and no extra features of these tools are integrated in the program if not necessary.

Memory. By not using a rule-based system, there is no need for huge databases stored on the hard-drive. Also, if the program has internet access, the pages needed for information retrieval can be fetched immediately from the Web. Although, if this is not possible, the biographical corpus used by the agent can be scaled according to the desired set of people the program could potentially instantiate. Regarding the RAM memory, the efforts to minimize the memory footprint are backed by:

- the optimized indexing of Apache Lucene
- the control of Stanford CoreNLP annotations
- the Java language references support (by not duplicating paragraphs or sentences)
- using hard-drive stored dictionaries for WordNet

Answer accuracy. The "correctness" of the answers is achieved through the scoring method of the selected sentences, using both Lucene's internal scoring system and own formulas for scoring sentences according to the score of the paragraphs and WordNet's frequency count for the synonyms considered in the alternative formulations of the input question.

The class diagrams for the program can be seen in Appendix A.

5.2.2.1 Pipeline

The pipeline for the answer selection is made up of two main sieves: a lexical sieve and a syntactic one.

The purpose of the lexical sieve is to filter those sentences that are lexically closest to the input question. The lexical pipeline is made out of the paragraph selector and the sentence selector. This step is made at the lexical level in the sense that the filtering of the sentences is done by matching words against these sentences and seeing which of them is the most similar, in form, with the question.

To get the best results out of all these steps, a little processing was made on the corpus and on the questions. The next paragraphs describe the transformation made.

The Wikipedia corpus from which the biography is extracted is stripped of the irrelevant content, like the "See Also" section, the "Reference" section and other additional, unneeded sections. The remaining text is split into paragraphs, and the paragraphs into sentences, for easy manipulation when filtering sentences. In addition, the words from the corpus are lemmatized, i.e. brought to their dictionary form, in order to increase the likelihood of matching future questions against the text. All this steps are made when the chat-bot is initialized, after the Wikipedia page is fetched. The program blocks until these transformations are made.

Processing is also done for each question submitted to the agent. Each question is stripped of unimportant words, like the interrogative word and stop words, which resulted in the "kernel" of the question (the most important words in a question). After this, alternative formulations of the question were generated using synonyms retrieved from the WordNet dictionaries. Essentially, each word in the kernel question is replaced with every one of its synonyms, practically generating all the possible permutations of synonyms of the words in the question. Details about this process are presented in subsubsection 5.2.2.2.

This lexical pipeline is represented in Figure 5.7. It can be easily seen that the question posed by the user at one point is fed to both the paragraph and the sentence selectors. The main role of these so called "selectors" is to score every paragraph and sentence respectively, based on the matching score (between a query and the searchable text) given by Apache Lucene. After the scoring is done, the selector returns a set of top scored linguistic units (paragraph or sentences) depending on the type of the selector.

In the case of the paragraph selector, the paragraphs are assigned a score solely based on the Lucene matching score. Based on this score, the first N paragraphs with the highest score are returned. N is a value that can be controlled at the initialization of the agent. This value may affect the accuracy and it is important that it N is chosen appropriately.

After the selection, the top paragraphs are split into sentences (using Stanford CoreNLP's "ssplit" annotator) and are added to the set that will be given to the sentence selector. The main difference of the two selectors is that the sentence selector scores the sentences not only based on the matching score between the query and a given sentence, but also takes into consideration the score of the paragraph that contains the respective sentence. The score is computed using the function in Equation 5.1, where S is the

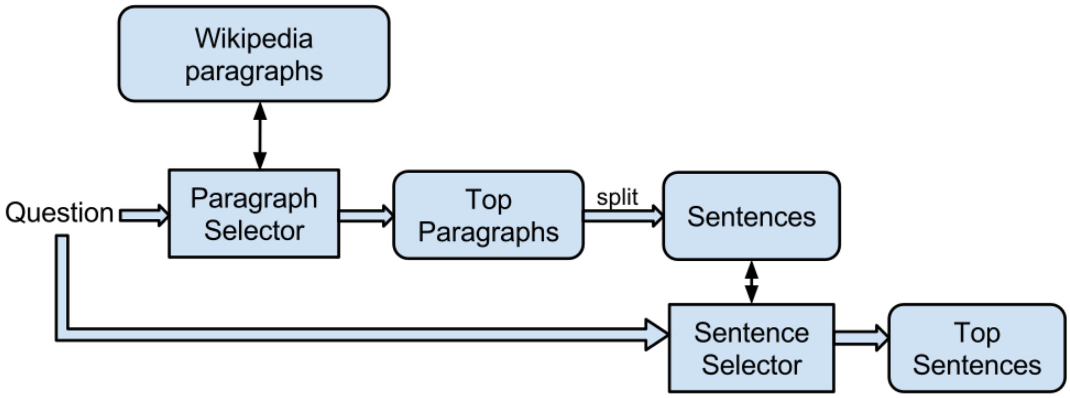

Figure 5.7: The lexical filtering part of the pipeline

score function in relation to a query, s is a given sentence, p is a paragraph (in which the sentence s must appear), α_p is a weight signifying the importance of the paragraph for a given sentence, α_s is the weight of the sentence (can be taken as $1 - \alpha_p$) and S_{Lucene} is the score given by Apache Lucene.

$$S(s,q) = \alpha_p * S(p,q) + \alpha_s * S_{Lucene}(s,q) \tag{5.1}$$

After the lexical pipeline returns the most likely sentences to answer the question, they are passed to the syntactic sieve. The role of the syntactical part of the pipeline is to re-score the sentences based on the possibility that they can answer the question. This syntactical filtering is done only considering the subject and predicate of the verb, therefore, if a sentence considered for selection has the same verb (or a synonym) with the verb representing the predicate in the question, then it is implied that the subject must match.

Having the question "When were you born?", addressed to John Lennon (the character the conversational agent was instantiated with), and two sentences, retrieved by the lexical pipeline, presented in 5.2 and 5.3, we can apply the syntactic selection. It can be easily seen that the verb of the question is "born" and the subject is "you", which, in this case, refers to John Lennon. This means that the answers that have as a predicate the verb "to bear" must be linked to a subject that also refers to John Lennon, in this case. Using Stanford CoreNLP, the differentiation can be made.

Lennon was on tour when his first son, Julian, was born in April. (5.2)

Lennon was born in war-time England, on 9 October 1940 at Liverpool Maternity Hospital to Julia (née Stanley) and Alfred Lennon, a merchant seaman of Irish descent, who was away at the time of his son's birth. (5.3)

In this example, in the first sentence (5.2), the predicate "born" is obviously in relation with "first son", which is the subject and, therefore, must be excluded or, at least, have the score of the sentence decreased. On the other hand, the second statement (5.3) can be preserved in this syntactic analyzation step because the verb "born" is in relation with "Lennon" exactly like in the question.

5.2.2.2 Question rephrasing

Because a phrase can be expressed in numerous ways by changing the order of the words and even the words (by replacing them with synonyms), the lexical form of a word in not enough to use it as a matching method between the question and the potential answers. To overcome that, the selectors in the pipeline that use the given question as input use methods of generating alternatives for that question.

Question rephrasing is done with the help of the synsets in WordNet. Using WordNet, for every word from the kernel question, a list of synonyms (including the initial word) is selected. The list is kept ordered as the JWI API returns it, indicating the importance (quantified by the *frequency count*) of the word. After this, all permutations of the synonyms are generated to get all the reformulations of the given query. The permutations are obtained using a simple implementation of the backtracking algorithm.

5.3 Testing

The testing of the conversational was done by hand by fetching a Wikipedia article of a random person and randomly selecting 10 to 20 sentences, attempting to formulate a question that could be answered by these sentences and then submitting the question to the conversational agent. If the conversational agent would return the selected sentence it means that it correctly answered the question. Of course, the questions can be formulated in a lot of ways, but the question rephrasing system in the implementation should take care of that.

In addition to these haphazardly selected sentences, the testing was also done with general questions like: "Where were you born?", "When did you die?", "Who was your husband/wife?" etc.

Chapter 6

Results

As stated in section 5.3, the testing was done by hand, therefore no exact information about the accuracy of the answers are known. What was observed during the testing is that in the majority of the cases, the answer was somewhere in the top 10 sentences outputted by the lexical pipeline.

Some examples of questions that were answered by the rule-based method, using ChatScript follow (the personality is Alber Einstein):

Question: *Where were you born?*
Answer: I was born in Ulm, in the Kingdom of Wurttemberg in the German Empire on 14 March 1879.

Question: *When did you die?*
Answer: I died in Princeton Hospital early the next morning at the age of 76, having continued to work until near the end.

Question: *Who were you married with?*
Answer: I married Elsa Löwenthal on 2 June 1919, after having had a relationship with me since 1912.

Some example of questions posed to Albert Einstein, and answered using the second method (the answer sentence selection method):

Question: *What awards did you receive?*
Answer: I received numerous awards and honors, including the Nobel Prize in Physics.

Question: *What field did you study in?*
Answer: There have been claims that Marić collaborated with I on my celebrated 1905 papers, but historians of physics who have studied the issue find no evidence that I made any substantive contributions.

As it can be seen in the previous example, the answer is not always given directly, but sometimes it is just hinted.

Here are some examples of answers given by John Lennon, using answer sentence selection.

Question: *Who was your wife?*
Answer: Cynthia Lennon (1st wife)

Question: *When did you die?*
Answer: December 8, 1980: The Day I Died.

Question: *Who shot you?*
Answer: At around 10:50 pm on 8 December 1980, as I and Ono returned to our New York apartment in the Dakota, Mark David Chapman shot me in the back four times at the entrance to the building.

Question: *What instrument did you play?*
Answer: I would continue to play harmonica, often using the instrument during the Beatles' Hamburg years, and it became a signature sound in the group's early recordings.

As seen in the previous example, even that the answer is correct, it is not the perfect answer, John Lennon being mainly known for playing the guitar.

Question: *What was your debut album?*
Answer: They recorded their debut album, Please Please Me, in under 10 hours on 11 February 1963, a day when I was suffering the effects of a cold, which is evident in the vocal on the last song to be recorded that day, "Twist and Shout".

In the last example the answer is, again, partially correct. The phrase refers to The Beatles' first album, not John Lennon's album. Fortunately, disambiguation was obtained with the following question.

Question: *What was your debut solo album?*
Answer: My emotional debut solo album, John Lennon/Plastic Ono Band (1970), was received with high praise.

Because the time it takes to select the answer depends on the number of queries (which is equal to the number of rephrased question), which, in turn, depends on the length of the user question and the number of the synonyms of the words in the question, it is very hard to establish a speed performance benchmark. From the tests done, it is noticeable that all the answers took less than 700 ms on a Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz.

Chapter 7

Conclusions and Future Work

This chapter presents the general conclusions drawn from the efforts made on working on a complex conversational agent. In addition, some new ideas and some improvements on old ones are suggested for future work in section 7.2.

7.1 Summary

In this thesis are presented at length the concept, design and implementation (that uses answer sentence selection) of a conversational agent that can impersonate a historical figure, or any other personality that has a Wikipedia article with their name. The method described is, in fact, a fallback option if the initial approach, based on rules interpreted by ChatScript and matched against user posed questions, does not succeed in answering a particular question.

The approach that is at the center of this thesis is inspired by many advancements made in the world of conversational agents and open domain question answering fields. This approach uses state of the art tools to offer speed, small memory footprint and accuracy to the user, in order to improve the experience of talking with, and learning from, a dialog system that responds as if it were a known personality.

The implementation adds to the performance and also to configurability to provide a final software that is intended to be used for learning (as an e-tutor) or as a virtual guide in museum, because it can provide information about the life and work of any known person.

7.2 Future Work

In order to better improve the program as a whole it is essential to improve both speed and answer accuracy.

For the first matter, that of performance, the simple approach of parallelism can be taken. Because of the pipeline architecture of the program, parallelization cannot be applied on the entire program, but separate modules that form the pipeline can be multithreaded. For example, the text search can be done in parallel, by splitting the paragraph into sets or use the thread pool problem and get each thread to process a set of alternative queries generated from the initial question.

For the second matter, many interesting ideas can be pursued. One of these ideas is to use genetic algorithms to determine what is the best formula (what are the best weights used) for computing the score for a sentence. Another variable that influences the outcome and should be accurately determined is the number of top paragraphs and/or top sentences selected after searching the query.

Another goal is to extend the taxonomy of questions, in order for the conversational agent to correctly answer more ambiguous questions like the ones starting with "why" or the ones starting with "did you" or "would you".

Appendix A

Class Diagrams

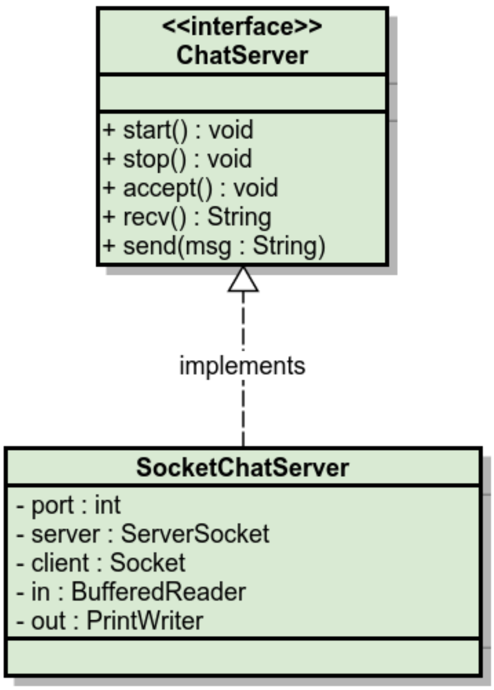

Figure A.1: Class diagram for the chat-bot server

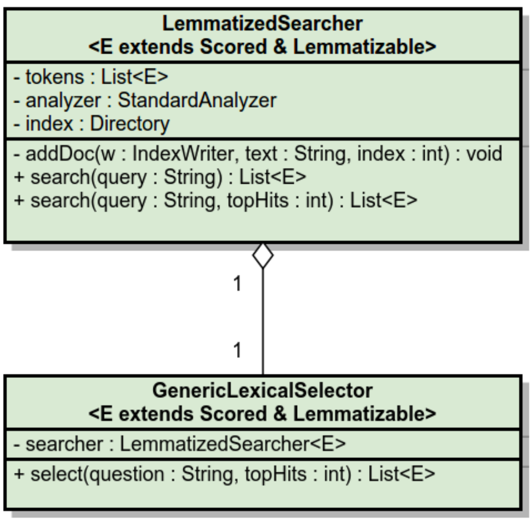

Figure A.2: Class diagram for the text searcher

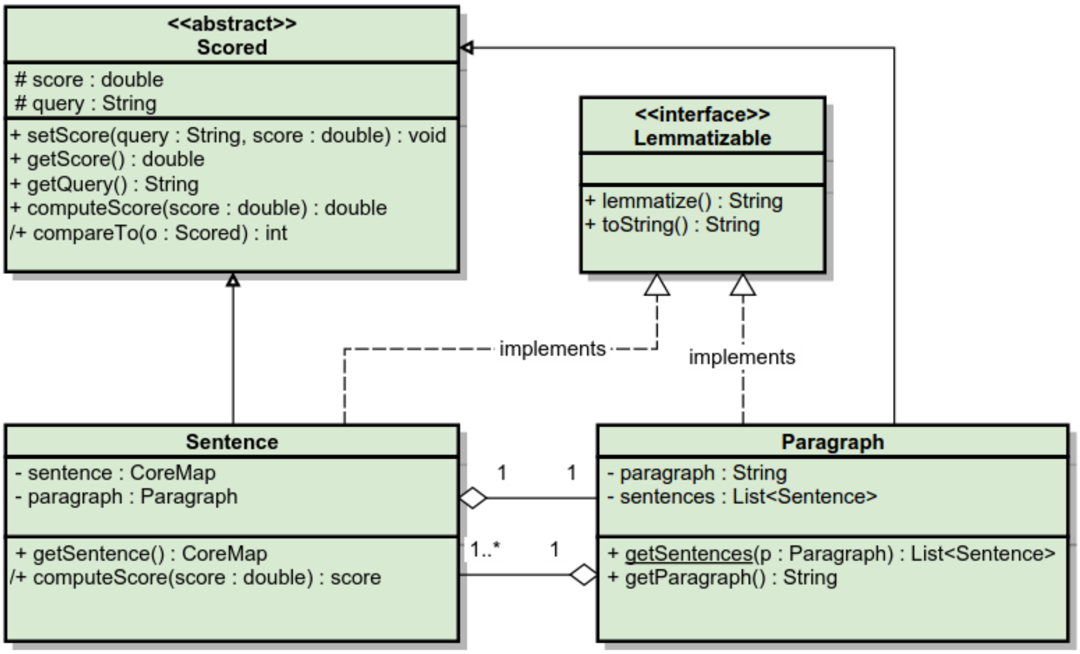

Figure A.3: Class diagram for the objects (paragraph, sentence) being manipulated

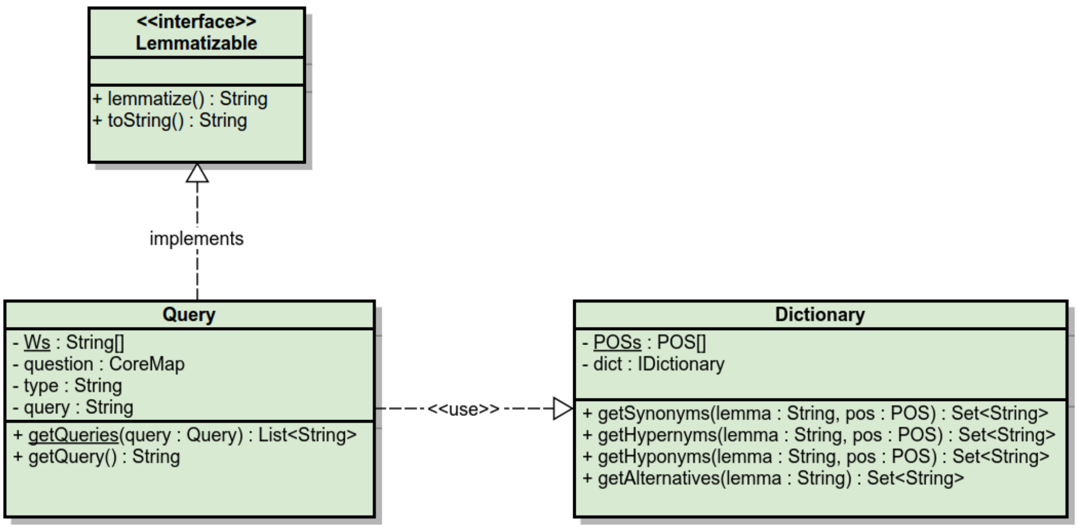

Figure A.4: Class diagram for the query used to interrogate the knowledge base

Bibliography

[1] Stanford Tokenizer Documentation. http://nlp.stanford.edu/software/tokenizer.shtml. Accessed: 2015-06-26.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a Web of open data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4825 LNCS, pages 722–735, 2007.

[3] Andrzej Białecki, Robert Muri, and Grant Ingersoll. Apache Lucene 4. *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*, pages 17–24, 2012.

[4] Adrian Bogatu, Dorin Rotărescu, and Traian Rebedea. Conversational Agent that Models a Historical Personality. *In press.*

[5] John Burger, Claire Cardie, Vinay Chaudhri, Robert Gaizauskas, Sanda Harabagiu, David Israel, Christian Jacquemin, C Y Lin, Steve Maiorano, George Miller, and Et Al. Issues , Tasks and Program Structures to Roadmap Research in Question & Answering (Q & A). *New York*, pages 1–35, 2001.

[6] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003.

[7] A. Echihabi, U. Hermjakob, E. Hovy, D. Marcu, E. Melz, and D. Ravichandran. How to Select an Answer String? *Advances in open domain question answering*, pages 383–406, 2006.

[8] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*, volume 71. 1998.

[9] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An Overview of the DeepQA Project, 2010.

[10] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. *in Acl*, (1995):363 – 370, 2005.

[11] Mark a. Finlayson. MIT Java Wordnet Interface (JWI) User's Guide. pages 1–10, 2011.

[12] Mark Alan Finlayson. Code for Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation. *the 7th Global Wordnet Conference*, 2013.

[13] Fausto Giunchiglia and Pavel Shvaiko. Semantic Matching. *Knowledge Engineering Review journal*, 18(3):265–280, 2004.

[14] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, 2007.

[15] Bob Heller, Mike Procter, and Dean Mah. Freudbot: An investigation of chatbot technology in distance education. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pages 3913–3918, 2005.

[16] Alice Kerry, Richard Ellis, and Susan Bull. Conversational agents in e-learning. *Applications and Innovations in Intelligent Systems XVI - Proceedings of AI 2008, the 28th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 169–182, 2009.

[17] Stefan Kopp, Lars Gesellensetter, Nicole C. Krämer, and Ipke Wachsmuth. A conversational agent as museum guide - Design and evaluation of a real-world application. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3661 LNAI:329–343, 2005.

[18] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):1–54, 2013.

[19] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mende, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Soren Auer, and Christian Bizer. DBpedia – A Large-scale , Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web*, 1:1–5, 2012.

[20] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[21] Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[22] Manish Mehta and Andrea Corradini. Developing a Conversational Agent Using Ontologies. *HCI'07 Proceedings of the 12th international conference on Human-computer interaction: intelligent multimodal interaction environments*, pages 154–164, 2007.

[23] George a. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[24] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[25] Karl Moritz, Hermann Phil, and Blunsom Stephen. Deep Learning for Answer Sentence Selection. pages 1–9.

[26] Kate Taylor and Simon Moore. Adding question answering to an e-tutor for programming languages. In *Applications and Innovations in Intelligent Systems XIV*, pages 193–206. 2007.

[27] Kristina Toutanova, Dan Klein, and Christopher D Manning. Feature-rich part-of-speech tagging with a cyclic dependency network. *In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1 (NAACL '03),*, pages 252–259, 2003.

[28] Alan Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[29] C J Van Rijsbergen. *Information Retrieval, 2nd edition.* 1979.

[30] Richard S. Wallace. The anatomy of A.L.I.C.E. In *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, pages 181–210. 2009.

[31] Joseph Weizenbaum. ELIZA — A Computer Program For the Study of Natural Language Communication Between Man And Machine. *Communications of the ACM*, 9(1):36–45, 1966.

[32] Bruce Wilcox. Beyond Façade: Pattern Matching for Natural Language Applications. *Gamasutra*, pages 1–5, 2011.

[33] Wen-tau Yih, Ming-Wei Chang, Christopher Meek, and Andrzej Pastusiak. Question Answering Using Enhanced Lexical Semantic Models. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1744–1753, 2013.