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

More than 60 years ago, Alan Turing raised the question "Can machines think?" [1]. Can people believe that a machine thinks?

Chapter 2

Project Overview

2.1 Project Motivation

2.2 Project Objective

2.3 Project Description

Chapter 3

Related Work

The following sections in this chapter present the background in the areas of NLP, question answering,

3.1 Open Domain Question Answering

* IBM Watson

3.2 Conversational Agents

* Freudbot

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.

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 [3].

[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." [4]

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 [5, 6] 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 [7].

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 [6].

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 [5].

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 [8]. 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 [9] 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 [9] 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 [10]. Lucene achieves high performance due to the inverted index [11] 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 [10].

[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 [10].

Searching. The implementation of Lucene's querying supports several types of searching mechanisms, like: wildcards, fuzzy search, proximity search, binary operators and others [10]. 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 [12]. 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

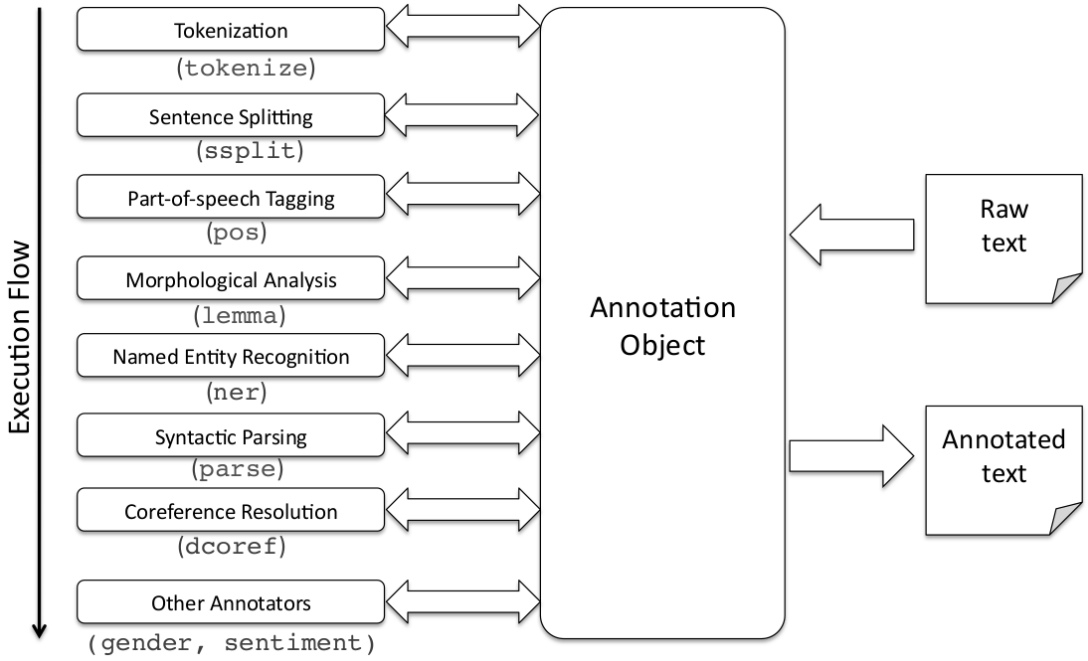

Figure 4.1: Stanford CoreNLP execution pipeline [12]

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) [13].

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 [12]. The English POS tagger uses the Penn Treebank tagset described in [14]. 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 [15].

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) [16]. 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 [12]. 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 [17].

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 through 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 large 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.

5.1.2 Back End

5.2 Conversational Agent

5.3 Testing

Chapter 6

Results

Chapter 7

Conclusions and Future Work

7.1 Summary

7.2 Future Work

Bibliography

[1] A. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. 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)*, vol. 4825 LNCS, pp. 722–735, 2007.

[3] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mende, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia – A Large-scale , Multilingual Knowledge Base Extracted from Wikipedia," *Semantic Web*, vol. 1, pp. 1–5, 2012.

[4] B. Wilcox, "Beyond Façade: Pattern Matching for Natural Language Applications," *Gamasutra*, pp. 1–5, 2011.

[5] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to wordnet: An on-line lexical database," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, 1990.

[6] G. a. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[7] C. Fellbaum, *WordNet: An Electronic Lexical Database*, vol. 71. 1998.

[8] M. a. Finlayson, "MIT Java Wordnet Interface (JWI) User's Guide," pp. 1–10, 2011.

[9] M. A. Finlayson, "Code for Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation," *the 7th Global Wordnet Conference*, 2013.

[10] A. Białecki, R. Muri, and G. Ingersoll, "Apache Lucene 4," *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*, pp. 17–24, 2012.

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

[12] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014.

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

[14] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[15] K. Toutanova, D. Klein, and C. 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),*, pp. 252–259, 2003.

[16] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," *in Acl*, no. 1995, pp. 363 – 370, 2005.

[17] H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky, "Deterministic coreference resolution based on entity-centric, precision-ranked rules," *Computational Linguistics*, vol. 39, no. 4, pp. 1–54, 2013.