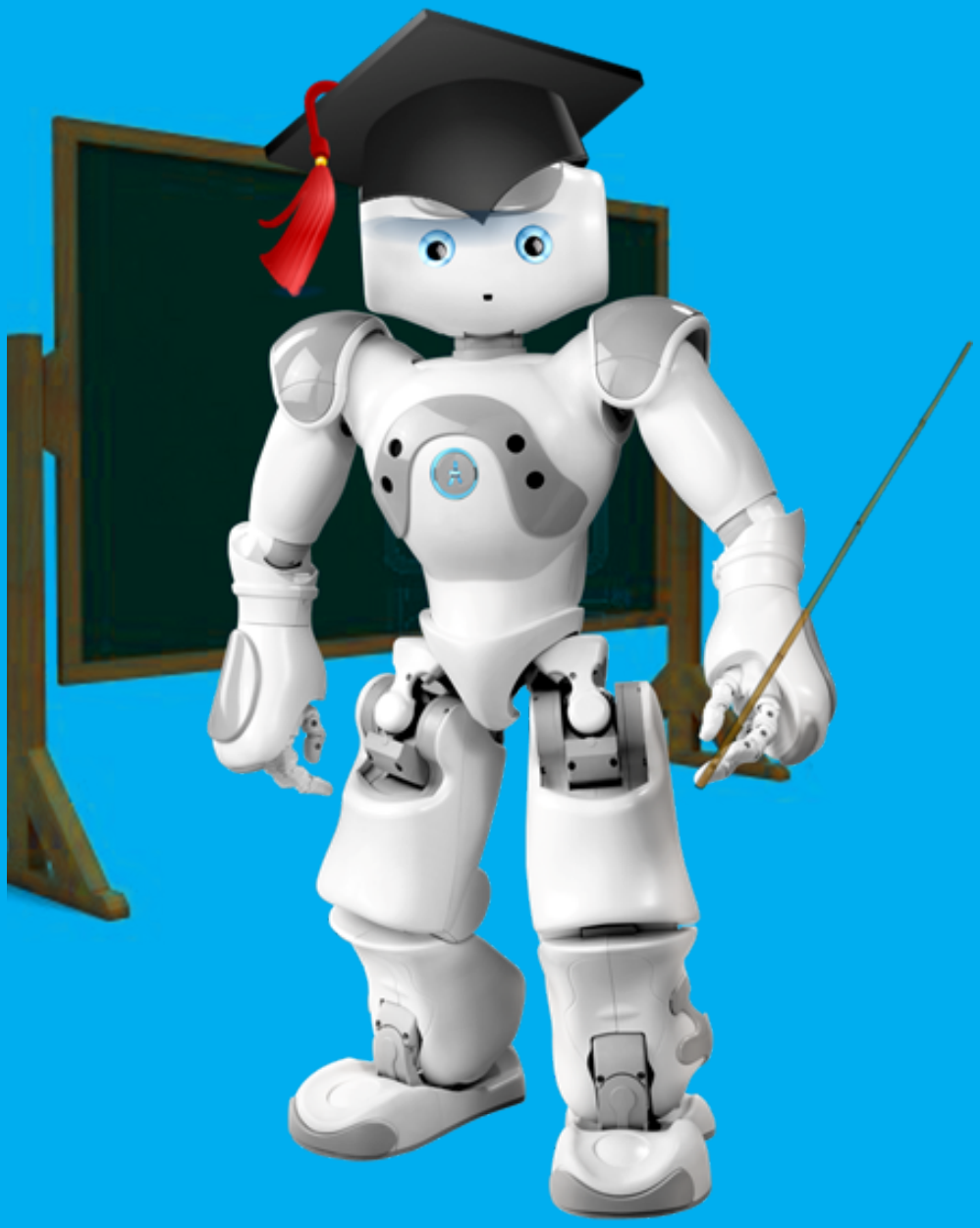
Question answering with the RoboTutor Nao

Selman Ercan



TU Delft

Question answering with the RoboTutor Nao

by

Selman Ercan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 17, 2017 at 10:00 AM.

Student number:	4044568
Project duration:	January 28, 2016 – August 17, 2017
Thesis supervisor:	Assoc. Prof. Dr. K. Hindriks
Thesis committee:	Prof. Dr. M. A. Neerincx,
	Asst. Prof. Dr. D. J. Broekens,
	Asst. Prof. Dr. C. Lofi

This thesis is confidential and cannot be made public until August 17, 2017.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Cover image source: `https://robotutor.weblog.tudelft.nl/files/2015/11/image001.png`

TU Delft

Abstract

This thesis project worked towards extending the interactive capabilities of the RoboTutor Nao, by enabling it to answers natural language questions about topics such as computers, robots and the Nao itself. The main questions we focused on were 1) what a system for answering questions in Dutch and intended for elementary school age children should look like, 2) how the interaction with users should be structured for a smooth interaction and 3) what type of experimental studies and metrics are best for evaluating the quality of the QA system and assess its impact on users.

We chose an existing English-language QA system and adapted it to work with Dutch content. An interaction manager was developed for handling the interaction between the system and users. Finally, the system was configured so that it could be interacted with via the Nao robot.

The system was then evaluated via two experimental studies. First a quantitative evaluation without users was carried out, evaluating the performance of the system using a collection of test questions. Secondly, a field study with users was done. We took the robot to a day care center where children could ask it questions and assessed the effects of these interactive Q&A sessions.

Results from the quantitative evaluation highlighted slow answering times as the main shortcoming, while recall (~70%) and precision (~50%) scores were more competitive. The field study showed that even though the occasional repetitiveness and lack of speed were picked up on by users, this did little to curb their enthusiasm. Interaction was smooth, users were interested in many topics the system supported and they continued interacting with it despite the occasional mishap. Overall, the results indicate that, with some speedup, the delivered QA system would be a suitable choice for answering questions about the mentioned topics.

Preface

Selman Ercan
Delft, August 2017

Before you lies the thesis “Question answering with the RoboTutor Nao” carried out to obtain the degree of Master of Science, at the Interactive Intelligence department of the Electrical Engineering, Mathematics and Computer Science faculty of the Delft University of Technology. The aim of this project was to create a Dutch natural language question answering system that could be interacted with via the Nao robot. I was engaged in research and development work and writing this thesis from February 2016 to August 2017.

The project was part of the RoboTutor project and was carried out under the supervision of Associate Professor Koen Hindriks. I would like to thank my supervisor for his quick and thorough feedback throughout the whole project and for his guidance on practical matters, especially regarding the outlining and detailing of the experimental studies we carried out. I would like to also thank the various members from the Interactive Intelligence department who lended a helping hand at one point or another, whether it was with a useful discussion about the interaction manager or a brainstorm about the field study.

Special thanks to the YodaQA and Alpino developers who helped me in my initial efforts to get up to speed with their software. Finally, thanks also to the day care center where we carried out the field study. Not only because it was an important part of the project, but also because it provided the opportunity to work with children. After having been immersed in technical details for a long time, witnessing their enthusiasm when interacting with the question answering system and the robot was a rewarding experience.

I hope you enjoy your reading.

List of Figures

List of Tables

Contents

Part I

Preliminaries

1 Introduction

Apple's *Siri*, Amazon's *Alexa*, Microsoft's *Cortana*: these are relatively new software systems the majority of computer literate people have at least heard about and maybe even used. Besides these more famous instances there is a large number of similar but less well-known tools, ranging from chatbots (like the social network bot *Tay*) to customer facing systems (providing interactive dialogue and customer service features on webshops).

What do these systems share with each other and what lies behind their increasing appeal? An important aspect is that they can all be interacted with using human language, without having to learn the 'language of the machine' itself. This enlarges the target audience so as to include less tech-savvy users as well. Another reason is that they are systems that perform functions traditionally requiring humans and that are, in principle, available on demand.

Besides the aforementioned commercial applications, there are many academic initiatives in this area as well. One academic institution that also has several ongoing projects aimed at aiding people in different tasks using intelligent systems is the Interactive Intelligence department[1] of the EEMCS faculty at the Delft University of Technology.

Among these is the RoboTutor project with the goal of providing an education assistant in the form of the Nao robot, to aid teachers in classrooms in Dutch elementary schools. An important future milestone is advancing the robot's capabilities to such a degree that it can interact with its audiences by asking them questions and answering their questions, while making use of additional techniques like speech recognition and hand-raising detection to facilitate this interaction. The purpose of this project is to meet one of these sub-goals: adding question answering capabilities to the Nao.

1.1. Question answering

Question answering systems represent one of the newer steps in the continuing line of software systems that enable users to effectively search for information. Among the earlier examples were relational databases (enabling users to store and searching for data in a structured way) and expert systems (for inferring new facts from known data and supporting decisions).

There is an important distinction between QA systems and the former, more 'traditional', software systems where interaction takes place using machine language. In contrast to databases, for example, where users are required to learn the system's language, the type of systems focused on in this project are ones that take the opposite approach and try to understand the users' language. As mentioned, this has the advantage of potentially appealing to a broader audience, as people unfamiliar with technical systems can also make use of these systems.

In the next three subsections we will first discuss QA systems in general, next the parameters that together define a specific QA system, and finally the type of QA system we are looking to create as part of this project.

1.1.1. QA systems

Despite individual differences in design and implementation, natural language question answering systems share a number of important characteristics:

- **Natural language interaction**: users interact with the system using human language. This necessitates the use of *natural language processing (NLP)* tools: software that can analyze natural language input. Examples of such analysis include breaking up a sentence into words, identifying its subject and main verb, and determining which parts of a sentence depend on which other parts.
- **Exact answers**: the system aims to return an answer that fits the expected type of answer. This can be pieces of data like names, numbers or locations. This is in contrast to, for example, web search engines, where a list of documents possibly containing the answer are returned instead of the actual answer itself.
- **Sources**: the system has access to sources it can search through in the process of answering a question. This can take the form of an offline data repository, but searching for information online is an option as well.

Figure 1.1 shows what a generic question answering pipeline looks like. The main tasks are *analyzing the question*, *retrieving information* that could be helpful for formulating an answer, *analyzing the data* that has been retrieved and creating candidate answers from them, *evaluating* the quality of these candidates, and finally generating a *response* from the analyzed candidates and presenting it to the user.

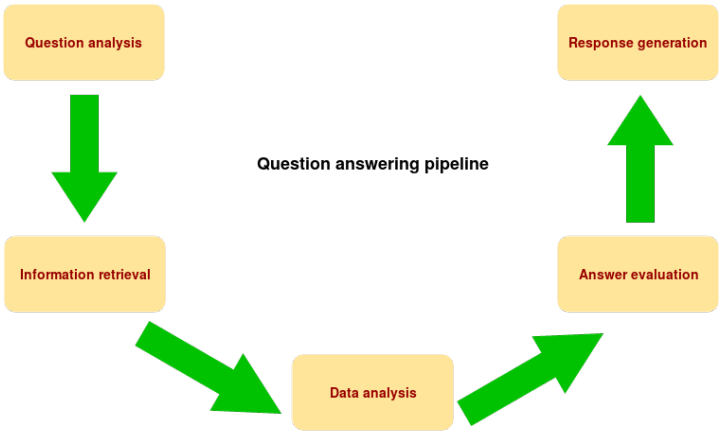


Figure 1.1: Generic question answering pipeline

1.1.2. QA system parameters

A specific QA system, in turn, is defined by a number of parameters. The most important of these are the following:

- **Language**: in which languages can users ask questions and get answers? This has a significant impact on the system as natural language processing tools are typically tailored for one language and for that language only. This means that, in the absence of NLP tools for another language, extending a QA system with support for that language is no trivial task.
- **Domain**: questions about which topics can the system answer? Is there only one subject it is expected to know well, or should it be capable of handling a wide array of subjects?
- **Question types**: can we only ask it about definitions of terms, can it also find bits of data like dates of birth and country capitals, or can we even expect it to come up with reasons behind events or detect relationships between different pieces of information?
- **Data sources**: is the data contained in the sources simply free text (unstructured data), or does it have an explicit structure and is it maybe closer to a database (structured data)?

1.1.3. QA in this project

Building on the previous subsections, we now use the parameters previously introduced to define the QA system we are aiming at as part of this project:

- **Language**: as the intended audience consists of Dutch elementary school children, the QA system should be able to process Dutch questions and return Dutch answers.

- **Domain**: the current focus of the RoboTutor project's educational efforts is on robotics, so we can identify this as the first subject to focus on. In addition, previous experiments in similar settings have shown that the audience is very interested in details about the Nao robot itself - this is therefore another important domain.
- **Question types**: questions about capabilities of the robot (whether it can do this or that), and questions about factoids (like the largest robot ever, or the Nao's weight). To a lesser degree, definitions.
- **Data sources**: Both unstructured and structured sources can be beneficial for the purposes of this project. Wikipedia because lot of information available about robotics. Structured data sources for data that can be easily modeled that way.

1.2. Project plan

Based on the previous discussion, we are now in a better position to visualize this project's main aspects and tasks. Figure 1.2 is a graphical illustration of the project:

Figure 1.2: Project overview
sources: http://samooborona-stena.ru/wp-content/uploads/2017/06/994_5.jpg
https://wp.hum.uu.nl/wp-content/uploads/sites/113/2016/09/robot-colored.png

The fundamental task is shown on the bottom right: the implementation of a QA system for Dutch and about robotics. NLP tools are needed to have it working for Dutch, appropriate data sources must be accessible for answering questions about robotics, and the pipeline of the QA system needs to be adapted to and optimized for the specific requirements of the project.

Going up, the QA system needs to be integrated into the Nao's existing software architecture. Modules for, among others, carrying out a presentation and converting textual input to verbal output already exist: the QA system ultimately needs to become another module in this platform.

Finally, there is the human-computer interaction aspect of the project. The QA system, integrated into the Nao software architecture, will be used within the context of an elementary school classroom.

The following tasks need to be carried to achieve the project goals:

1.3. Research topics and questions

The research questions to be focused on in this project can be derived from the preceding discussion. The questions are formulated below, categorized into the five research topics part of this project.

1.3.1. Question answering

Context: Figure 2.1 showed what a QA system might look like architecturally. From an implementation point of view, the mentioned pipeline of five components from *Question Analysis* to *Response Generation* can be singled out: during execution this will be the subsystem responsible for the bulk of the processing.

Table 1.1: Project tasks and deliverables

Milestone	Tasks	Deliverables
1	Review existing research and work	Literature review report
2	Define context, use cases, requirements	Product specification document
3	Investigate QA systems	Report on trade-offs between alternatives and a choice for a QA system
4	Set up sources	Configured sources ready for use by the QA system
5	Set up NLP pipeline	Capability of processing Dutch questions and answers
6	Integrate QA system into Nao platform	QA system running via the Nao robot
7	Evaluate system	Report on the evaluation and achieved results

Any two question answering systems will differ regarding the way their pipelines have been implemented. It is therefore likely that, for the system we choose, this will have been done in not quite the way we would consider fitting the requirements of this project.
RQ 1 *What adaptations need to be made to the question answering pipeline of the chosen system?*
Approach Investigate workings of existing pipeline in chosen system. Design alternative adaptations where relevant, experimentally test their performance and implement the best one.

1.3.2. Sources

Context: The QA system will be put before the target audiences with a certain type of usage and purpose in mind. This is to serve as a source of information for questions about robots, computers and the Nao itself. This leads to certain requirements for the knowledge sources to be used: these must be of a type fit to store data relevant to these topics.

Also, since the robot will be answering questions in real time, a proper indexing and preprocessing of these sources is crucial to keep down answering times.
RQ 2 *Which sources and source types contain information that supplement the topics the Nao is presenting about and fit the interest of the audience?*
Approach: Investigate relevant data sources than can be mined and processed to provide ready input for the QA system. Provide documentation on how to extend the knowledge base with new data of the same type. Finally, analyze whether and how the knowledge base can be extended with other types of answer sources.

1.3.3. Natural language processing

Context: Natural language processing comes in at the two ends of the question answering pipeline: question analysis and answer analysis. There currently are no readily available open-source QA systems that can work with Dutch questions. This means that the chosen English QA system needs to be adapted by adding functionality for analyzing Dutch questions.

As the intended audience of the system speaks Dutch, Dutch question analysis modules should be used. Because of the time frame associated with this project, the focus will be on simpler types of factoid questions: ones asking for names, dates, locations, verification (yes/no), and lists. More complex questions about, for example, causality and relationships, are best left for future work building on this project.
RQ 3 *Which Dutch natural language processing libraries are fitting tools for processing the natural language content the system will work with?*
Approach: Investigate existing Dutch natural language processing libraries. Assess their suitability for processing the desired question types and the natural language information about the robotics domain it will be configured to use. Provide configuration and usage details.

1.3.4. Interaction design

Context: Implementing the QA system does not yet mean that all is ready for answering questions from users. What is needed is a mechanism that 'wraps around' the system and structures the interaction with the audience.

A number of aspects need to be considered for this. What are good ways of interacting with users during a Q&A session? Should we adopt a strictly turn-based approach, with one user asking one question which the system then tries to answer? How should the system handle questions it cannot find an answer for, or only answers with very low scores?

RQ 4 *What type of robot-audience interaction allows the robot to hold a clear, structured and informative Q&A session?*
Approach: Research usage of robots in education and robotic QA; design interaction according to findings. Implement interaction manager to handle a QA session, striving for maximum autonomy and minimal human intervention.

1.3.5. Evaluation

Context: The performance and impact of the QA system can be measured in various ways. On the one hand we have the technical metrics, like the quality of answers and answering speed. These will be assessed using a quantitative evaluation, carried out internally using a set of questions.

On the other hand there is the perception of the robot by the users during a QA session, which leads to considerations such as how the answers are presented and how the system behaves when no satisfactory answers could be found. This is best evaluated via a field study: taking the QA system, running on the Nao robot, to users and having them interact with it there.
RQ 5 *Which evaluation metrics best serve to assess the effectiveness and efficiency of the QA system, given its domain and type of usage?*
RQ 6 *What type of experimental setting and which measurements allow us to evaluate the effect on the audience of having the RoboTutor hold a Q&A session after a presentation?*
Approach: Investigate literature on evaluating QA systems and the various measurements and scoring functions that are available. Also analyze how actual QA systems that participated in QA challenges were evaluated. Finally, review literature about robots used in education.

1.4. Thesis overview

The next chapter in this part includes a review of literature relevant to this project and takes a look at previous work in the areas of QA, NLP and educational robots. Chapter 3 looks at the specific considerations related to the project and the different technological alternatives available.

In *part two* we start looking at the research and development work done in the project. This part focuses on the selected question answering system and the work done to adapt it to the requirements of the RoboTutor project. We will discuss the design and workings of the selected QA system and explain the work done to adapt its NLP modules and data sources. The interaction manager for the system is laid out here as well. *Part three* follows up on the previous one and aims to make the matter discussed in *part two* more concrete. Here we will briefly go into the details of the QA system, zoom in on the pipeline and execution flow, and follow a question from start to finish to better explain all steps taken.

With the implementation finished, in *part four* we turn towards the evaluations carried out of the QA system. This part discusses the experimental setups used to test the system will be detailed and the results of these studies presented. Both the internal, quantitative evaluation and the field study are discussed here. Finally, in *part five*, an analysis and discussion of the results will be presented. We will then close off with our concluding remarks.

2

Literature review

This chapter reviews existing work in the fields of question answering, natural language processing and the use of robots for educational purposes. During this review we will be guided by the main considerations introduced in the previous chapter: that we are aiming at a Dutch QA system about robotics and the Nao, that is embodied by a robot, and used by elementary school age children.

Section 2.1 provides an overview of the field of question answering, discusses the state-of-the-art in this field and identifies important aspects of QA. Section 2.2 makes this more concrete by looking at the tasks a typical QA system carries out and what its architecture looks like. Various performance metrics for evaluating QA systems and their respective advantages and disadvantages are discussed in section 2.3, where we try to identify which metrics are appropriate in our case. Current issues and challenges in this field, and which of these we should keep in mind for this project, are discussed in section 2.4. Section 2.5 continues with a tour of existing QA systems and describes prominent initiatives in this area, in what contexts they were used and how they performed. This section is the basis from which, later on, we will decide which QA system to adopt as part of this project.

Because natural language QA systems by definition need to handle natural language content, natural language processing (NLP) is a crucial part of question answering. In section 2.7 we investigate important NLP techniques and libraries, and look at which of these could be useful for us. Finally, robots used for the purposes of question answering and education are investigated in section 2.8: besides previous studies, we discuss specific opportunities and challenges in this area.

2.1. Question answering

Hirschman and Gaizauskas[2] provide a characterization of natural language question answering systems (emphasis mine):

> **"systems that allow a user to ask a question in *everyday language* and receive an *answer* quickly and succinctly, with sufficient context to validate the answer"**

The two highlighted parts of this formulation can be used to illustrate the differences with two related types of systems: traditional database management systems (DBMSs) and web search engines.

Firstly, the *everyday language* requirement distinguishes it from DBMSs. DBMSs usually do meet the *answer* requirement: data collection formats and query language syntaxes often enforce adherence to specific data types (like *text* and *number*), which means that the type of the returned result will fit what was asked for in the query. However, queries must be formulated using the DBMS' own language, which is in contrast to QA systems. An important feature of natural language QA systems is therefore their ability to understand the language of users, instead of the users having to learn the system's language.

Secondly, the *answer* requirement distinguishes it from typical web search engines. Even though these engines can be interacted with using natural language and return relevant responses, the level of granularity is usually the document or paragraph containing the answer. This is again in contrast to QA systems: if we

ask a QA system about someone's name we want to get the actual name, not a list (however promising) of documents that might contain the name.

In short, with DBMSs we can get specific bits of information of a certain type but we have to use their language, while with web search engines we can use natural language but have to make do with less precise responses. Natural language question answering systems aim to combine the best of both types of systems.

The importance of natural language question answering is closely related to these two characteristics. As noted by Kolomiyets and Moens[3], available information is growing at very high rates and the information need of users can't be met by information retrieval (IR) techniques or databases alone. Natural language question answering facilitates this by 1) supporting natural language so that non-technical users can also navigate the vast amounts of information and 2) finding relevant pieces of data with precision and presenting them at the desired level of granularity.

We now turn to specific QA system considerations, which influence the design of a particular QA system. The most important of these are which *question types* it supports, questions in which *domains* it can answer, and which types of *sources* it can consult. The following subsections will discuss these considerations one by one.

2.1.1. Question types

The first consideration to note is the types of questions the system is designed to answer. A number of question types are discussed by Kolomiyets and Moens[3] which we present below. For each type, an example question is included for clarity.

- **Definition**
  The definition of a term or concept.
  What is a computer?, What does hacking mean?
- **Factoid**
  Asking for bits of data like a location, name, amount or year.
  When did World War I start?, Where was Bill Gates born?,
  Who founded IBM?, How much does a Boeing 747-400 cost?
- **Confirmation**
  These questions are in a sense the opposite of factoid questions. Instead of asking for a piece of information it is now 'suggested' or 'claimed', and the QA system is used to check whether it is true or false.
  Is Mark Zuckerberg the founder of Facebook?
- **List**
  These ask for a list of things like entities, concepts and facts.
  "How many Dutch newspapers have more than 100 000 readers?", "Which are the 5 largest countries?".
- **Procedural**
  These state a result or state and ask for the steps needed to get there.
  Which steps are needed to compile and execute a Java program?
- **Descriptive**
  Stating an entity or concept and asking the system to describe it and mention some of its important attributes. *What are some characteristics of a smartphone?*
- **Opinion**
  This is different from other categories in that answers don't necessarily need to conform to objective reality. An answer to an *opinion* question is correct if it fits the knowledge and insights available to the robot - if it is indeed what the system should 'think' given the data it has access to.
  What do you think about pair programming?
- **Relationship**
  These inquire into possible relations between multiple entities or phenomena.
  What is the correlation between age and marital status worldwide?
- **Causal**
  This category includes questions that ask about reasons and causes.
  What are the reasons behind the increase in popularity of social media?

This listing is ordered here roughly in ascending order on difficulty: from the simple task of returning the definition of a given term, to the more complex tasks of identifying relationships between two or more specified inputs or searching for the reasons behind a given decision or event.

The first group of three types (**definition**, **factoid** and **confirmation)** are about retrieving a piece of information that is out there. Compared to the next types, this is a relatively easy task: if the question can be properly understood by the system and if the sources contain the answer, then in principle the answer will always be found.
The second group of three (**list**, **procedural** and **descriptive**) is a step further. What is needed is not only the capacity to find data, but finding pieces of data that are relevant but potentially distributed, and presenting them together coherently. Of course this applies only if the list asked for isn't specified explicitly as a single entity in the data source, in which case it would basically be a large factoid.
The final group (**opinion**, **relationship** and **causal**) is yet another step further. Here even piecing together related bits of distributed information does not suffice: the system needs to make inferences from them and use the gained insights to formulate its response.

The first two types of questions, definitions and factoids, are to a certain extent already supported by web search engines. Search queries like *"What is an atom?"* and *"When did the First World War start?"* return references to web pages containing the correct answers in different engines like *Google*, *Bing*, *Yahoo* and *DuckDuckGo*. Some, like Google and Yahoo, go even further and return the actual definition or factoid - in the case of Yahoo, by making use of the extensive question collections of Yahoo Answers.

What are the implications of this analysis for this project? The third group of question types is the hardest and no readily available systems exist for answering them. Supporting the second group is certainly possible, but would require a significant amount of implementation work. It is also out of the project's scope, as it is meant to have a human-computer interaction aspect besides the more technical question answering aspect. The question types supported in this project are therefore the three in the first group. This is further supported by school surveys carried out by and anecdotal evidence collected by researchers in the RoboTutor project, from which we learned that the most frequently asked question types are factoid and confirmation - especially about the capabilities of the robot itself.

2.1.2. Lexical answer types

One goal of the natural language processing applied to a question is to determine what type of an answer is expected. This is known as the *lexical answer type* (LAT). Ferrucci *et.al*[4], in their report on the development of the DeepQA architecture, define a LAT as the word that "...indicates the type of the answer, independent of assigning semantics to that word". A LAT can be explicit in the question or it may be necessary to infer it. For example, the LAT in "Which country has the highest GDP?" is explicit (*country*), but the one in "How many mountain ranges exist?" is implicit (*amount*) and needs to be deduced from the keywords *how many*.

Depending on the specific requirements on the QA system, its domains and supported question types, many LATs can be defined. The LASSO[5] question answering system, for example, features 26 question subclasses distinguished by their different expected answer types. The corresponding LATs include, among others, *person*, *number*, *location*, *definition* and *reason*.

Many of these can be useful within our project. To a certain extent, the variety of factoid questions that can be answered is influenced by how many LATs we distinguish and how well these can be identified. Some LATs are not needed; *reason* is one of them as we will not support causal questions.

2.1.3. Domains

Natural language question answering systems are traditionally divided into *restricted domain question answering* (RDQA) systems and *open domain question answering* (ODQA) systems. The supported domains influence the diversity and size of the sources the system needs to use.

RDQA systems developed with a certain task in mind. An early RDQA system was LUNAR[6], which enabled users to ask questions about rocks collected on the Apollo mission. Naturally the QA system was tailored specifically for this domain and the knowledge sources coming with it.

In their analysis of RDQA systems, Diekerma *et.al.*[7] note that, in contrast to earlier systems that encoded domain knowledge in sources, modern ones work with custom data extraction rules for domain specific collections. In this way the systems are made more flexible and supporting additional sources becomes easier.

RDQA often makes use of handcrafted sources, because the target audience can be assumed to be already familiar with the domain and therefore has specific information needs. ODQA, on the other hand, can make use of large bodies of data that have not been specifically tailored; it is reasonable to assume that users are not experts in a large number of domains simultaneously.

Deciding on the domains to be supported therefore involves a trade-off between depth and breadth. As mentioned, RDQA is expected to demonstrate more insight and deeper knowledge about its one domain, thereby 'justifying' or compensating for its narrow focus. On the other hand, an ODQA system is expected to be able to give decent answers across a broad range of topics. Although high accuracy is of course always desirable, some loss is to be expected and acceptable given its much larger scope compared to RDQA systems.

A challenge that arises, however, is achieving a sharp translation of the question into a search query. Yang *et.al.*[8] highlight this problem of formulating a query that is flexible enough to match relevant data and simultaneously sufficiently restricted to avoid irrelevant passages. While this problem arises in RDQA systems too, it is more pronounced in ODQA. Because many domains are supported, query keywords can match passages in documents from *multiple* topics while only *one* of these can be relevant to the question.

The RoboTutor project currently is mainly focused on robotics and the Nao itself. This means that our QA system will combine characteristics of both RDQA and ODQA. For questions about robots and computers it will have to be more ODQA than RDQA: it is not quite open domain, but it will have access to data from a sufficient number of domain not to be considered RDQA.

On the other hand, it will be more RDQA when it comes to questions about the Nao robot. Need to model information about attributes and capabilities of the Nao explicitly and, indeed, *handcraft* that part of the sources which is dealing with this.

2.2. QA architectures

Natural language question answering involves a number of important and recurring subtasks, which enables us to imagine a generic QA system architecture with components we can expect to be see in different QA systems in some shape or form. Hirschman and Gaizauskas[2] present such an architecture for an end-to-end QA pipeline. They use the visualization shown in figure 2.1 (coloring mine) to illustrate such a pipeline.

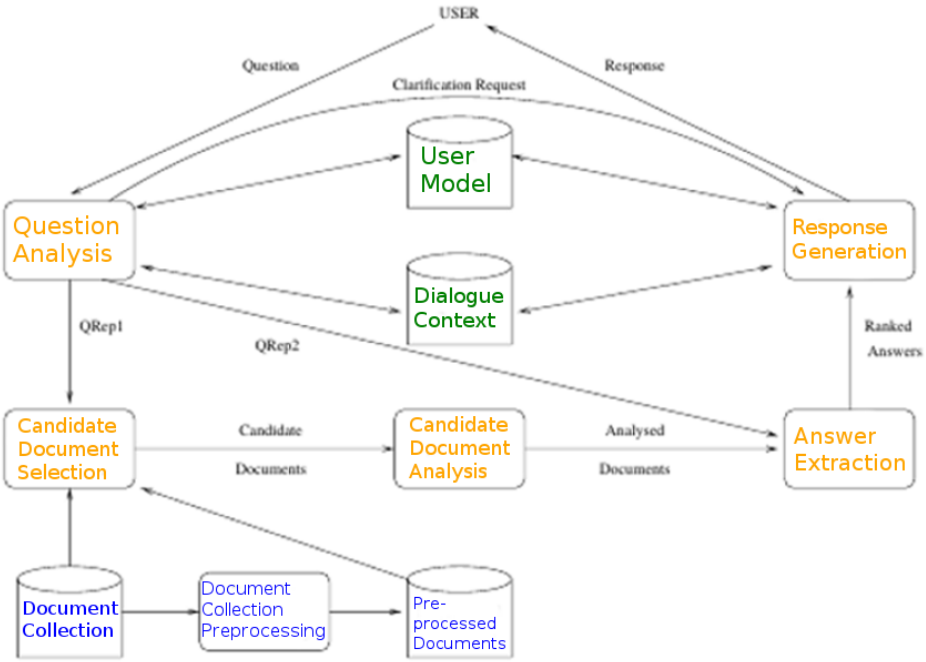


Figure 2.1: Generic question answering system architecture

2.2.1. Pipeline

The five orange components represent the main question answering pipeline.

- **Question analysis** Aimed at, among others, identifying the semantic type of expected entity (name, date, location, etc.) and identifying constraints (key words, required relationships).
- **Candidate document selection** This phase is responsible for the information retrieval part of the pipeline. The retrieval is based on the information and clues acquired in the *question analysis* phase. The output of this stage is a list of entries in the sources considered promising for further analysis.
- **Candidate document analysis** Further processing of the output of previous stage; this is not needed if document preprocessing was done. The goal is to transform the intermediate results into a suitable format for matching with the question.
- **Answer extraction** Here, question representations are matched with representations of documents considered likely to contain answers. Candidate answers are then produced from the retrieved documents and ranked by relevance.
- **Response generation** Once a list of one or more answers has been determined, a response needs to be given to the user. This can be done in different ways. A first consideration is whether to return only the top answer or the top x answers. Another issue is whether a justification for each answer should be included (e.g., document used for answer).

2.2.2. Sources

The three blue components are related to the sources the system can access.

- **Document collection** The collection of data the system is set to search through as part of the process of answering a question. This can include both structured and unstructured sources.
- **Document collection preprocessing** The preprocessing done in this stage enables the QA system to access and process documents more effectively during the *candidate document selection* phase. Practical usage considerations dictate the offline preprocessing of the corpus if this corpus has a significant size. Often, conventional document indexing engines like Apache Solr are used for this purpose.
- **Preprocessed documents** An ideally more machine-friendly representation of the data originally contained in the document collection. Having the documents already processed saves time, as the system won't have to do this while answering a question.

2.2.3. User interaction

The two green components represent the system's model of its users and what the interaction between itself and the system should look like.

- **User model** What knowledge does the system have about its intended audience, and how should it make use of this information?
- **Dialogue context** What is the setting in which the question answering session is held, and according to what kind of protocol?

2.3. Evaluation of QA systems

There are two important ways of assessing a given QA system: we can distinguish between criteria that focus more on the system as a whole and its effects on users, and more technical metrics that quantitatively evaluate the performance of the QA system. This corresponds to the two experimental studies we want to carry out in this project: a 'qualitative test' in the classroom to witness the system in action in front of a live audience, and a 'quantitative test' for a quantitative analysis of its effectiveness and efficiency based on the answers given to a test set of questions.

2.3.1. Qualitative evaluation

The following evaluation criteria, identified by *Burger et al.*[9], serve as a useful starting point:

- **Timeliness** How quickly does the system return to the user?
- **Accuracy** Is the answer correct?

- **Usability** Is the system useful for users within the intended contexts?
- **Completeness** Is relevant information combined if it is distributed across multiple sources?
- **Relevance** Does the answer make sense within the context it is operating in?

In this project, the intended usage setting involves elementary school children who regard the Nao robot more as a conversational partner than as a natural language interface to a database with information about robotics. This places **usability** and **timeliness** at the top of the most important metrics: shortcomings in these areas will damage the smooth and natural interaction experience the most. Next up are **relevance** and **accuracy**: the returned answers don't need to be perfect all the time, but they should at least be relevant to the question and have decent scores on accuracy. Finally, **completeness** seems to be the least significant factor - the audience is unlikely to criticize the system for failing to include every relevant piece of information that is available in its answer.

2.3.2. Quantitative evaluation

Two of the most frequently used metrics in tasks related to information retrieval, classification and pattern recognition are *precision* and *recall*. If A is the number of known answers, C the number of correct, distinct answers returned by the system and N the total number of answers returned by the system, then precision $= C/N$ and recall $= C/A$. In words, precision indicates which proportion of the retrieved answers were correct, and recall indicates which proportion of correct answers were retrieved.

More sophisticated measures exist besides these basic ones. Kolomiyets and Moens discuss a number of them:

- **Mean reciprocal rank** (MRR) looks at how high the system ranked the first correct answer in its list of outputted answers, averaged over a number of questions. One of its shortcomings, however, is that it can't evaluate questions without answers in the used knowledge sources. Another one is that it does not consider recall, as 'unretrieved correct answers' are not a part of the measure.
- **Confidence score** is a variant on the *MRR* score: whereas *MRR* looks at the rank of the first correct answer, the confidence score looks at how many of the first x answers are correct. This score is calculated n times (with n the number of answers) and then averaged.
- **F-score** is a measure defined as a function of the precision and recall scores as introduced above. Precision and recall can be given equal weight in the estimation, or varied by tuning the *F-score*'s parameter. This is used when the specific usage requirements of the system imply a preference for high precision at the cost of recall, or vice versa.
- **Weighted score** is more of a meta-technique than a scoring formula in itself. It is an answer to the problem of evaluating the system's performance on different *types* of questions: presumably, we would get misleading results if the same performance measures were used for, for example, both *factoid* and *list* questions. *Weighted score* allows a more nuanced calculation by providing *weights* that can be attached to the separate scores.

As noted before, we consider relevance and accuracy as more important criteria than completeness. This means that we should emphasize precision over recall. While the returned information should in itself be accurate, it is less of a problem if there are other, also relevant, bits of data in the sources that are left behind. The F-score can be used to implement this emphasis, by tuning its parameter to emphasize precision over recall.

Between MRR and confidence score, the former is more fitting. Not only do we intend to show only one answer (the top one), but multiple answers are also unlikely to turn up for the type of questions the system will support. This implies that values for the *confidence score* would almost always be identical to MRR scores because there is only one correct answer.

Incidentally, another implication is that recall will be either 0 or 1, with no scores in between. This ia again because there is only one correct answer, and the system will either have found it or not.

At this point we can also distinguish between the 'development' mode with full list of candidate answers with confidence scores, and 'deployment' mode with only top answer without confidence score. In the latter case precision takes on a different meaning, as it can be either 0 or 1.

No such distinction applies for confirmation questions, as in both settings only one answer is outputted for this type of question (*yes* or *no*). For the same reason, for confirmation questions MRR is the same as precision.

2.4. Issues and challenges in QA

Natural language question answering is a relatively recent field and therefore has a number of important open issues. Burger *et.al.*[9] identify some of these challenges which will be discussed next.

2.4.1. Context

Context indicates the setting in which the system operates and the audience it interacts with. In case of ambiguous information or multiple promising candidates, knowledge about the system's operational setting can be used to help decide on a fitting course of action. A simple example is a mention of the word *table*: is this to be interpreted in a 'furniture' context or a 'database' context?

This consideration is not an issue within this project. Even though there is a specific context to be operated in (in classrooms, about robotics), this context is known in advance and fixed, and can therefore be represented explicitly within and built into the QA system.

2.4.2. User interaction

The best performing question answering system is useless if users cannot understand how they are supposed to use it and what to make of its responses. *Dialogue models* can help in this regard by defining protocols to guide the interaction. Another technique is *user profiling*: getting to learn the preferences and interests of the users, and possibly having the system come up with its own suggestions. Finally, *repair and fallback strategies* can be important for the graceful handling of errors and questions that cannot be answered with a high degree of confidence.

In contrast to the previous consideration, this is a very important issue, probably more so than the raw quality and accuracy of the answers given to questions. An embodied system that interacts with users smoothly and handles mishaps gracefully will be perceived as more pleasant than one which gives the occasional perfect answer but is more difficult to handle and interact with.

2.4.3. Reasoning and inference

From an information retrieval point of view, we can make a rough distinction between two types of QA systems.

The first type only looks up and returns pieces of information it considers relevant. The information retrieval component of such a system can therefore be considered similar to a traditional database management system, only now with a natural language interface instead of the database's own query language.

The second type can, in addition to locating relevant data, also identify relationships between them and infer new facts from them. For example, if it finds two separate entries "organization *O* was founded in 2000" and "*Bob* founded *O* at age 20", such a system could then deduce that "*Bob*'s date of birth = 1980".

It would clearly be useful to have this second type of functionality in the QA system selected for our project. Especially because the system will be single domain: basic information can be assumed to be already known by the audience, and actual insight and relationships between data that the system could identify would be valuable.

But the technical challenges associated with it are too significant for the scope of this project. To get the inferential aspect of a QA system that does not yet have it up to speed is possible, but would detract too much from the human-computer interaction aspect of the project and transform it into a mostly technical and analytical one. It therefore is not a part of the project.

2.5. QA systems

In this section we provide brief descriptions of existing QA systems. First we will discuss closed source systems and then continue with a number of open source alternatives.

2.5.1. English systems

2.5.1.1. Closed source

- **IBM Watson**
  Probably the most famous question answering system of all time, Watson[4] is best known for its victory in the quiz show *Jeopardy!* in 2011. It emerged out of efforts by IBM to adapt certain existing systems

(PIQUANT, OpenEphyra) for the *Jeopardy!* challenge. A major overhaul was carried out when it became clear that the results fell short of what was needed to be successful in the challenge.

As a result, the DeepQA architecture was created, which is a *massively parallel probabilistic evidence-based architecture*. OAQA, a QA project discussed below, is an effort that spawned from this project.

With the *Jeopardy!* challenge behind, research and development efforts[10] for extending the Watson system and the underlying DeepQA architecture to fields other than *Jeopardy!* are being undertaken. These include business applications, enterprise search and especially medicine, where it can fulfill the role of a clinical decision support system.

- **IBM Watson Developer Cloud**
  The IBM Watson Developer Cloud contains a number of services that can together serve as parts of a QA pipeline. The most important ones are the Natural Language Classifier[11] and Retrieve and Rank[12] services. These services require a paid Bluemix account to run.

  The Natural Language Classifier can determine the sub-topic that a given question most likely belongs to. The Retrieve and Rank service takes a query as input and (1) finds the documents it considers to be relevant (*retrieve*) and (2) sorts them based on their estimated relevance(*rank*).

2.5.1.2. Open source

- **YodaQA**
  YodaQA[13] is an open-source QA system implemented in Java that is based on the Apache UIMA framework. It is accessible via a GitHub repository[14]. It is an end-to-end pipeline: the different components that make up the system (question analysis, information retrieval, answer generation, etc.) are already linked together to form a system where questions can be entered and answers can be retrieved.

  The system searches in an indexed collection of documents taken from a Wikipedia dump (~40GB). However, it is designed in a way that allows it to make use of structured answer sources as well.

  YodaQA's language processing components make use of Apache OpenNLP. This framework doesn't support very many languages, but Dutch NLP components are among the provided ones.

  There have been two attempts to use the system in challenges[15][16]. These papers detail how the original system was adapted for the specific needs of the challenges, and presents recall and accuracy measurements.

- **OAQA**
  OAQA[17] is a project that consists of a number of separate frameworks. In contrast to YodaQA, it is not an end-to-end pipeline: the various OAQA components need to be set up in a particular end-to-end configuration by the user. It is also based on Apache UIMA like YodaQA. The basic QA functionality resides in the BaseQA repository, which contains basic I/O, data processing and evaluation components.

  OAQA has also been used in practice[18]. An OAQA configuration implemented at Carnegie Mellon University was applied to the BioASQ 2015 challenge. This challenge is the same as one of the YodaQA applications mentioned above.

  OAQA comes out on top in this challenge and has the best score for five of the six main performance criteria. This picture is somewhat skewed though, as compared to the YodaQA application it had more developers, used a more thorough approach, spent months on development and ran hundreds of experiments. On the other hand, the YodaQA application knew beforehand of a lack of resources and seems to have focused on just getting an initial, baseline result.

- **OpenEphyra**
  OpenEphyra[19] does information retrieval by querying web search engines (Google, Bing, Yahoo). Its language specific components can be replaced by ones other than English. On the other hand, by default it does not support local knowledge sources. It is also not being maintained and developed actively, as its source was last updated more than two years ago.

- **OpenQA**
  OpenQA[20] is a more recently introduced system. It supports searching by using web search engines, databases and unstructured sources. The system can keep track of user context by storing location, statistics and previous queries. Regarding maintenance it is the same as *OpenEphyra*, its source having been updated two years ago as well.

- **Tequesta**
  Tequesta[21] is the University of Amsterdam's Textual Question Answering System for answering English questions. The TreeTagger[22] part-of-speech tagger was used for the analysis of questions. Various syntactic and semantic characteristics of questions were identified using partial parsing, and rules for specific constructs of interest (names, companies, organizations, etc.) were crafted manually.

2.5.2. Dutch systems

There have not been many publicized initiatives to build question answering systems that can answer Dutch questions. Two of them, both academic, are discussed below.

- **Quartz-d**
  A prominent Dutch QA initiative was Quartz-d[23][24], also developed at the University of Amsterdam. The system is based on the idea of combining different approaches at the same time. In effect, there are five subsystems that all try to answer a given question in their own way.

  One of these subsystems is the aforementioned Tequesta system, to which the English translation of the Dutch question is submitted. Another is a Dutch version of Tequesta, adapted by replacing language components specific to English by Dutch ones.

  It achieved *MRR* scores between 0.335 and 0.428 when evaluated on the CLEF 2003 test collection.

- **Joost**
  Joost[25] is a question answering system that works with both structured data (database of relational information) and unstructured data (text snippets). Its question answering pipeline works by extracting keywords from question, retrieving relevant data, and ranking candidates using generated clues. It makes use of the Alpino parser for parsing questions and has been tested on various CLEF test collections.

Both Quartz-d and Joost participated in international QA challenges about a decade ago. Both systems have since been decommissioned and neither is available for use anymore.

2.6. Sources

Another important question is which sources the system will be made to use. A significant decision to be taken, even before deciding on the sources to use, is the *type* of sources to be used. A fundamental distinction can be made here between structured and unstructured sources.

2.6.1. Structured data

Structured data is data characterized by its organization and by being relatively easy to process and search through by software. Unstructured data is the opposite: entries within an unstructured data source can all look different, lack a common structure and come without any identifications or labels explaining what we are looking at.

Yao and van Durme[26] discuss two challenges with using structured sources for question answering. The first is converting the natural language question into a structured query that can be submitted to the source, which has to conform to the internal structure of the source. The second is matching data from sources with the given natural language question. A certain degree of insight into the used source is required, because a relation or expression used in the question might be represented and formulated differently in the source.

Burger *et al.*[9] discuss a number of structured sources. Examples of these include relational databases, expert system knowledge bases and ontologies. Unstructured sources can be written text (essays, articles, news items), speech, pictures, audio and video. A prominent example is DBpedia[27], which we will now discuss in more detail.

2.6.1.1. DBpedia

DBpedia is a project for extracting structured data from Wikipedia and presenting this in a format that allows users to query it in a way similar to querying a database (hence the 'DB' in 'DBpedia'). Its English version contains more than 4.5 million entries of which more than 90% (persons, places, organizations) have their place as resources in a consistent ontology. Since the extraction from Wikipedia is automated, the growth of the DBpedia knowledge base follows that of Wikipedia. The data model it uses is RDF.

RDF
RDF (Resource Description Framework) is the format in which graph data models like DBpedia are formulated. Data is expressed in RDF as *subject-predicate-object* triples. RDF supports the definition of namespaces, which serve to group related concepts together.

An example ontology is shown below. One way of organizing this ontology would be to put the concepts on each line in different namespaces, ending up with a *Car brand* namespace, a *Color* namespace, and so on.

- **Subject**
  Car brand: *Opel*, *Toyota*, *Nissan*, ...
  Fruit: *apple*, *orange*, *banana*, ...
- **Predicate**
  Measurement: *mass*, *height*, *volume*, ...
  Relation: *parent of*, *cousin of*, *sibling of*, ...
- **Object**
  Color: *red*, *green*, *blue*, ...
  Measurement: *100 grams*, *20 cm*, *1000 kg*, ...

Statements can be made by combining elements from these categories. To indicate the mass of a banana, for example, one could use the triple (*banana - mass - 100 grams*).

2.6.2. Unstructured data
Unstructured data sources exist in larger numbers and are more readily available. There exist many types of unstructured sources: papers, articles, essays, and collections and web sites that contain them. The lack of structure means that less effort is required to create such sources. This is in contrast to structured data sources, where the very structure of the source implies that some work has gone into packaging the data in an ordered way.

Because the available amount of unstructured data is increasing continuously, and thanks to tools for structuring and extracting insights from this data, the amount of structured sources is growing too. A major advantage of being able to use a structured source for QA is that, as mentioned above, the data can be queried and processed in a more machine-friendly way.

2.6.3. Structured vs. unstructured sources
Both types of sources are useful for our project. Unstructured sources are useful because they include lots of existing information; in our case, this would allow us to directly include Wikipedia articles and papers about robotics. Structured sources are useful for data that has an obvious internal structure and organization: this organization will then enable the system to process it more easily. An example could be the attributes of the Nao, which could be easily formulated in a table form.

Some data are better fit for one format than for the other, as not everything can be expressed in RDF triples with the same ease. This format is best for data that can be modeled with clear entities, relations and attributes; on the other hand, information in which such structure is more difficult to identify can be put into natural language articles. Consider the following two examples:

1: The robot was introduced in 2015, has a 2 GHz processor and costs $5000.
2: Because of the consistently decreasing profits from *A*'s sales, the company's focus gradually shifted towards its successor, *B*.

The first sentence can be expressed simply in three RDF triples (for *introduction date*, *processor* and *price*). Doing this for the second sentence requires more effort. Within the knowledge base, a number of objects and relations would first need to be specified within the ontology: at the very least, *company*, *product*, *profit* objects and *company-product*, *predecessor-successor* and *cause-effect* relations. Next, within the QA system, some sort of a 'reasoner' needs to be in place that can, given all the triples that together express the information in this sentence, work its way back to the whole picture, understand the expressed information, and use it to answer questions.

As can be seen from the first sentence, data that can be modeled in an object-oriented way is a particularly good fit for placement in structured sources. The *field,value* pairs of such objects can easily be modeled in such sources.

2.7. Natural language processing

Given that natural language interaction is such a crucial aspect of question answering and that, in this project, the language (Dutch) is one for which relatively few tools exist, a significant amount of research and implementation efforts in the area of natural language processing is likely to be necessary.

Natural language processing is relevant for question answering mainly for two tasks: analyzing a given question and analyzing candidate answers. In the first case, the system needs to understand what the question is about and what type of answer it is expected to provide. In the second case, sentences within candidate passages need to be analyzed to be able to assess their relevance and determine their suitability as answers.

The next subsections describe some of the important NLP tasks, which are *parsing, named entity recognition* and the use of *word nets*.

2.7.1. Parsing

Covington[28] identifies two different types of natural language parsing: constituency parsing and dependency parsing. Analysis using the former technique involves breaking sentences into successively smaller subunits, while the latter works by identifying links (dependencies) between individual words.

The parsing process serves to annotate a given natural language input with various NLP features. Some important ones are the following:

- **Part-of-speech**: In a 1957 paper, Brown[29] uses the term 'class of formal equivalents' to define part-of-speech (POS). A part of speech is a word category like *noun*, *verb* and *adjective*.

  Consider the incomplete sentence "I . . . like this." as an illustration. We see that there are many *formally equivalent* words that can fit in the empty space. 'Really', 'definitely', 'absolutely' are some examples, and they are all words that would be assigned the part-of-speech tag *adverb*.
- **Sub-phrases**: token groups representing semantic subunits of phrases. The subject of a sentence, for example, might be composed of multiple words which together represent a sub-phrase.
- **Lexical categories**: semantic function served by sub-phrases. Important ones are *subject*, *main verb* and *object*. Note that, in contrast to POS tags, lexical categories can be attached to multiple tokens.
- **Dependencies**: specifications of relations between sub-phrases. Example use cases are determining which sub-phrase is the object of a given verb, or which adverbs are associated with which verbs in a sentence.

The first three features can be identified using constituency parsing, but dependencies can only be identified using dependency parsing. Parsers are trained using training corpuses ('treebanks'), containing sentences annotated with NLP features like the ones mentioned above. The training results in a parser model, which can then be applied on new input.

Which features do we need for this project? We will discuss them one by one and see in which situations they become necessary.

POS tags can be quite sufficient to handle simple questions about structured data. For example, a question with only one noun and only one verb can be used to look for an RDF triple, using the noun as the subject and the verb as the predicate. The object of the matching triple can then be used as the answer.
Example:
Question: "Wat kost de Nao?"
Query: $(Nao - kost - ?)$
Matching triple: $(Nao - kost - \$1000)$
Answer: $1000

Next, consider a more complex case with multiple noun phrases. Without lexical categories, we cannot determine which of these is the subject. Moreover, if these can consist of multiple words, we need to be able to identify sub-phrases as well.

Example:
Sentence: “De Nao heeft een microfoon."
In this case we need sub-phrases and lexical categories to recognize that “de Nao", and not “een microfoon", is the subject of the sentence.

Finally, we could encounter sentences that contain multiple sub-phrases serving a similar role. In this case we need dependency annotations to resolve the ambiguity.
Example:
Question: “Wat kost de Nao?"
Sentence: “Pepper kost $2000 en de Nao kost $1000."
In this case, just looking for the first sub-phrase matching the given predicate (“kost") would give the wrong answer. We need dependency annotations to see that it is the second “kost" token that is linked to the question subject (“de Nao"), and then use the dependency relation between the “kost” and “$1000" tokens to arrive at the correct answer.

All of these features are needed for the QA system to be used in this project. None of the examples represents a rare or extreme case; on the contrary, they are sentence types we can definitely expect to encounter.

There are a number of parsers for English; these include the Apache OpenNLP parser and the Stanford parser. They are both sophisticated tools, but none of them supports Dutch. For Dutch, there is the Alpino dependency parser. It is useful because it can identify a variety of dependency relations.

2.7.2. Named entity recognition

The concept of ‘named entities’ was first formulated twenty years ago for the *Sixth Message Understanding Conference*, as reported on by Grishman and Sundheim [30]. The task of named entity recognition and classification (NERC), as conceived then, was considered an important information extraction task and had as its goal the identification of 1) names of people, organizations and geographic locations and 2) numeric expressions like dates, times and money within a given text.

Nadeau and Sekine[31] present an overview of research carried out up to 2006 within this field. As they note, NERC systems have since transitioned from relying on rule-based algorithms to making use of machine learning techniques. While the bulk of the work has been for English, a number of other languages have also been well-studied: the *CONLL-2002* and *CONLL-2003* conferences stimulated efforts for German, Spanish and Dutch.

Named entities can be an important source of clues when answering a question, as they can often be used as effective filters when searching for relevant information. It is a reasonable strategy, for example, to direct searches towards entities called x if x is a named entity occurring as the subject or object of a question. The availability of Dutch NERC tools can therefore be put to good use within this project. An example is the Apache OpenNLP Dutch named entity recognizer.

2.7.3. Word nets

George A. Miller’s paper *Dictionaries of the Mind*[32] from 1986 is among the first introductions of word nets, called *semantic fields* by him. It is formulated as a solution for the problem of digitally organizing lexical information. The starting point is the assumption that words can be grouped together in a semantic field based on a shared semantic concept.

The most prominent example in this area is the English WordNet[33], the ideas behind which are outlined by Fellbaum[34]. This lexical database is in fact the brainchild of the aforementioned George A. Miller.

Word nets are data structures consisting of synonym sets and the relations between them. Their use can provide a QA system with an additional degree of flexibility when analyzing sentences, as this allows it to also consider synonyms of the words within these sentences.

To illustrate this with an example, suppose that a user inquires about the “height of x”. Without word nets, the sources would need to contain the required information in the exact given formulation (using the word ‘height’) for the system to consider it relevant to the question. With word nets, more formulations can become relevant: besides “the height of x is ...”, “x is ... tall” and “x is ... high” will now also be considered because of the high similarity scores between *tall* and *high* on the one hand and *height* on the other.

For Dutch, there is the Open Dutch WordNet[36]. It is derived from the Cornetto database, the Princeton WordNet and a number of other external resources. It is available on GitHub[37], together with a Python module for using the word net.

2.8. Robots in education

The QA system implemented for this project will be an *embodied system*: it will be integrated into and tested using a Nao robot. To familiarize ourselves with this type of usage, we will in this section look into previous studies that have experimented with using robots in similar ways.

There is an increase in the usage of robots for educational purposes. Chang *et.al.*[38], in their review of studies with educational robots, explain this development using three factors:

- **Unique learning experience**: in contrast to a teacher from flesh and blood, a robot represents a package of advanced technology and allows an interactive session with direct feedback.
- **Cost**: while earlier robots were costly and not intended for widespread use, advances in technology have reduced costs and made robots affordable to a wider range of people and institutions.
- **Plug-and-play feel**: to allow a wider public to make use of robots, their interfaces have to be as accessible as possible. Improvements in this regard have made robots easier to configure and use, as a result of which they are not restricted anymore to tech-savvy users.

Attributes of robots that can facilitate teaching and improve the educational experience:

- **Repeatability**: robots can consistently present the same content in the same way for different groups of students, avoiding subtle changes caused by varying circumstances.
- **Digitization**: robots can be integrated into existing large bodies of digital data. Users then have the opportunity to interact with an agent that can access and handle much larger quantities of data than would be possible for a human.
- **Body movement**: the interaction can be enhanced with the use of non-verbal communication by the robot.
- **Interaction**: enables back-and-forth between student and teacher.
- **Anthropomorphism**: on the one hand a humanoid robot sufficiently resembles a human tends to be treated like a real person by children. On the other hand the children are still aware that it is not a real person, so there is no fear of being ridiculed by it or disappointing it.

Within education, robots have been used mostly as learning materials and learning companions and less as tutors or teaching assistants. Regarding robots serving as tutors, it is emphasized that their key purpose can be considered to be assisting instructors in presenting teaching material.

Chang *et.al.*[38] carried out a study with robots in this latter category and studied the impact of having a robot assist teachers in classrooms during language teaching. This was done in five different modes and one of these was a question-and-answer mode, in which the robot randomly chose a student and asked him or her a question. They found that this benefited low-achieving students in particular, because of the reduced anxiety of disappointing the teacher or getting mocked. This is closely related to the **anthropomorphism** factor mentioned above.

Among the important suggestions received from the teachers involved in the studies were to make the robot come across less cold and more engaging, by varying its voice and expressing emotion. We cannot vary the voice of the Nao much, but we can make it come across as more engaged using body language.

A study carried out by Fagin and Merkle[39] has an interesting result: introductory computer science students making use of robots were *less* encouraged to continue study in this field than other students taking the course in a more traditional way. Although the setting and purpose was different than in our case, in that it was a part of a year-long teaching program, we can still benefit from this negative result. The authors believe the main reason to be the time and effort required to get the robots up and running and solve problems - if this then is not compensated by any benefits gained from using these robots, interest will naturally fade.

The lesson to be learned for our project is therefore to shield users as much as possible from low-level, implementation and execution related ideas and to provide an effective and smooth experience. Interacting with robots should not become associated with having to solve complex problems just to be able to get to use it.

2.9. Conclusion

Natural language question answering is a field of increasing relevance due to the increased information need of users. Important subtasks in QA include analyzing questions and candidate answers, searching for and filtering information, and returning fitting response to users.

There exist a variety of QA systems, from open to close source and from academic to commercial initiatives. Contemporary challenges within QA are about better understanding and interacting with the audience and within the intended context, and demonstrating insight into retrieved information and handling it in 'smarter' ways.

Question answering depends heavily on NLP techniques for gaining insight into natural language content. These include parsing to identify sentence structures and the use of word nets to gain flexibility when matching questions with data from sources. Promising tools for our project are the Alpino parser, the Open Dutch WordNet, and Apache OpenNLP components for Dutch.

Finally, we should be mindful of the unique advantages the use of a robot in an educational setting can provide. This mainly involves being able to integrate a QA system coming with a large volume of digital data, and sharing this with users in interactive sessions. It is important to shield users from the low-level details as much as possible; this is just as, if not more, important than the purely technical performance of the robot and the QA system running on it.

3

Technology choices

This chapter considers the advantages and disadvantages of the different technologies available for use within this project. We first look at the available QA systems, assess how well they fit our requirements and explain why we settle on an open-source English-language system. Next, we concentrate on the specific problem of handling Dutch content. The main decision to be taken here is whether to translate content back and forth between Dutch and English, or to use tools for the specific purpose of processing and analyzing Dutch. Finally, we turn to the human-computer interaction aspect. After all, there needs to be some sort of a wrapper around the QA system that handles and structures the interaction between it and users. What tools and techniques can we use for this purpose?

3.1. QA system

We will decide on the QA system to use in two steps. First, we need to determine whether to use a Dutch or English system. Next, we decide on the actual system to be used in the chosen language.

We reviewed the prominent initiatives in the area of Dutch question answering in chapter 2. As mentioned there, no public, readily available Dutch QA system could be found. Since the two largest initiatives, Quartz-d and Joost, have already been decommissioned and cannot be used anymore, we have to take the route of adopting an existing *English QA system* and adapting it to our purposes.

Next, we assess the suitability of the introduced English QA systems.

OpenEphyra

OpenEphyra only supports web search and no local sources. Significant implementation efforts would be needed to extend it with functionality to interact with and query custom, local data sources.

OpenQA

OpenQA is a relatively recently created system and in the research phase. It is mainly geared towards making use of structured data sources.

OAQA

The advantages of OAQA are its modular approach (many repositories with various components) and the good results it has achieved in international question answering challenges.

On the other hand, there is no comprehensive tutorial and questions in the project repository about how to set up a basic configuration are unanswered. Within the scope of this project, we can therefore consider OAQA too complex to get up to speed with based on the level of available support.

YodaQA

YodaQA is inspired by Watson and the DeepQA architecture behind it. It has interfaces for searching through both structured and unstructured data sources. A framework for the parsing of natural language content is present as well.

It can be extended to languages other than English: a team consisting of three YodaQA developers created a basic fork in Czech in a one-day hackathon. The project repository contains resources both on supporting different languages as well as on making use of different knowledge sources.

IBM Watson

This is the most complete product and the one with the most man-hours devoted to it. Choosing it would

allow us to 'assume' a working QA system, and build on top of it and focus solely on interaction design and the human-computer interaction aspect of the project.

Conclusion
Table 3.1 presents the insights gained from the preceding analyis. We can discard OpenEphyra because it does not work with local sources or structured data. OpenQA has the opposite issue, as there is no framework in place that can make use of unstructured data in the process of answering a question. Components provided by OAQA are very comprehensive, but they are complex and not easily accessible for outsiders.

The IBM Watson system is the only commercial option in this list and the one that has actually been used in practice by clients. The first issue with it, however, is that it runs as a cloud service. This means that there will always be additional network overhead for communication with a third-party server. Secondly, there is the obvious closed-source disadvantage of being unable to modify any parts of the system. Thirdly, the data that can be entered for it is only free text, meaning that we can only make it work with unstructured data. The last and most important problem is the absence of support for Dutch: extending it to Dutch is being worked on, but will not be available within the duration of this thesis project.

The remaining option is YodaQA. It works with local sources and has interfaces to handle both unstructured and structured data sources. Additionally, there already exists a resource[40] within the YodaQA repository with information about how to go about adding support for a language other than English. For all these reasons, YodaQA is the system we choose for this project.

Table 3.1: Comparison of QA systems

QA system	Local sources	Structured data	Unstructured data	Accessibility
OpenEphyra	-	-	+	+
OpenQA	+	+	-	+
OAQA	+	+	+	-
YodaQA	+	+	+	+
Watson	+	-	+	+

3.2. Processing Dutch

After deciding to use an originally English QA system for Dutch question answering, the next question arises: do we use translation, or do we employ Dutch NLP tools?

Using translation would look as follows. On receiving a Dutch question, it is translated to English and passed on to the system. When, at the end of the pipeline, we acquire English candidate answers, these are translated to Dutch and used to formulate the response to the user. The main reason for doing this is to avoid having to put into place a Dutch NLP pipeline, as we would simply use the existing English NLP pipeline of the chosen QA system.

Translating English questions and answers from and to Dutch means limiting ourselves at the start regarding the maximum level of question complexity we will be able to handle. For the purpose of question answering, it is important to understand the question sentence as a whole and not just what each of its words mean individually - it is precisely the former insight that is lost more easily in translation. Moreover, since not only the question but also the answers have to be translated, such loss of information will potentially occur at both ends of the question answering pipeline.

Finally, there is the additional computational load on the system associated with these translation operations. Note that using an English system and taking the translation route does not free us from having to use NLP tools, since the English content still needs to be parsed and analyzed. Translating only adds more work on top of the natural language processing that needs to take place anyway.

The use of Dutch natural language processing tools is therefore the better and more appropriate approach. It requires more work initially to get working, but will lay the basis for further improvements later on and pay its dividends in this way.

3.3. Interaction manager

For the interaction managing component there are various options. The first is the Dialog service[41] from the IBM Watson Developer Cloud. This is, for our purposes, a relatively restrictive option: it mainly supports a rule-based interaction script, that is created by training it using conversation/question-answer specifications in XML format.

Another option is the dialog component that is being developed as part of the PAL project. However, this component is still under development and not ready for use.

Finally, a custom application can be developed that runs separately from the QA system, communicates with it and serves as the wrapper around it. Given the lack of viable alternatives, this is the approach to be recommended.

3.4. Conclusion

The QA system we choose for this project is YodaQA. As Dutch QA systems are not available and closed source options lack support for Dutch, YodaQA is one of the best suitable open source systems for this project.

For handling Dutch content we will use Dutch NLP tools. Translation is a suboptimal approach: it prevents gaining good insights into complex questions and means an unnecessary computational load on the system. Using Dutch NLP tools requires more work initially but will pay off further down the road.

Finally, we will design and implement a small application to serve as the interaction manager around the QA system. This system will structure the interaction between users and the QA system.

Part II

Research and development

4

YodaQA

The next chapters in this part will look at implementation details and discuss the concrete steps taken to create a Dutch version of YodaQA. Before going into these low level details, this chapter will first describe the system on a conceptual level. Section 4.1 gives a component level overview of YodaQA and section 4.2 describes the main technologies it uses.

At the end of this chapter we will be in a better position to answer our first research question:
RQ 1: *What adaptations need to be made to the question answering pipeline of the chosen system?*
From our analysis it will become clear what parts and components would need to be added, changed or removed to make the system capable of handling Dutch content.

4.1. System architecture

YodaQA[13] is a question answering system inspired by the Watson[4] system and its architecture. Its fundamental characteristic is the gradual accumulation of question and answer features along the question answering pipeline.

The main processing tasks are divided across distinct stages within the pipeline. As YodaQA is an end-to-end pipeline, the different stages are integrated to form a complete question answering system. This is in contrast to certain other open-source systems that include standalone components for parts of the QA pipeline, but don't represent an end-to-end pipeline that can receive a question and return an answer.

The main components of the YodaQA question answering pipeline, shown in figure 4.1, are described below.

1. **Question Analysis:** annotates questions using NLP tools to extract linguistic features (like *subject*) and identifies important question answering features (like *lexical answer type [LAT]*).
2. **Answer Production:** carries out searches in the configured data sources, based on the clues acquired during the previous step. Retrieved entries are further narrowed down by *passage extraction*, during which only the parts of the entries that are considered relevant to the question are kept.
3. **Answer Analysis:** extracts pieces of information from the 'strongest' candidate passages that best fit the expected answer types. If, for example, the *question analysis* phase has detected that a country name is being asked for, *answer analysis* will look for bits of information that meet this requirement.
4. **Answer Merging:** removes duplicate answers (which can arise because of the different data sources being used) and consolidates the collection of candidates into a single final list.
5. **Answer Scoring:** evaluates the quality of candidate answers using machine learned classifiers and adds a calculated confidence score to each candidate.

From a theoretical perspective, we can see that the YodaQA components closely match the components of a generic question answering system suggested in [2]. To recap, those were *Question Analysis*, *Document Collection Processing*, *Candidate Document Analysis*, *Answer Extraction* and *Response Generation*.
The first stage, *Question Analysis*, is matched exactly in YodaQA: a separate stage that deals with identifying the type of the question and extracting clues from it.

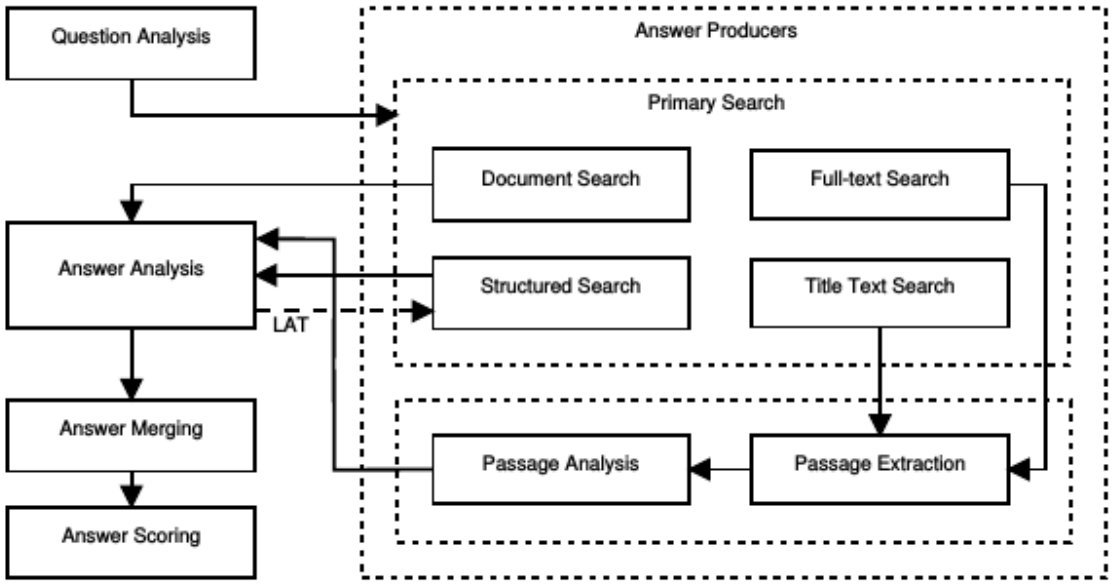


Figure 4.1: YodaQA architecture
source: Baudiš[13]

The work of the second stage, *Document Collection Processing*, is done separately before any querying takes place. This is because of the large sizes associated with the data sources, and indeed the authors also suggest this as a potentially better option for large document collections.
Next, the paper presents the *Candidate Document Analysis* stage as a stage with the purpose of further analyzing candidate documents, in case the preprocessing results of the second stage are insufficient for question answering purposes. In YodaQA the *Passage Analysis* module, which is a part of the *AnswerProduction* stage, is responsible for this task.
Finally, the *Answer Extraction* and *Response Generation* stages correspond more or less exactly to YodaQA's *Answer Analysis* and *Answer Writer* stages.
While all this does not say anything definitive about the quality of the system, we can nevertheless conclude that attention has been paid during design and development to the main subtasks involved in building a natural language question answering system.

4.2. Technologies

The basis of YodaQA can be regarded as a big data infrastructure, as it involves various modules for the processing of large volumes of data in consecutive stages. As such, the efficient integration of the different technologies has a prominent role in the design and implementation of the system. Below is a list of the most important technologies used by YodaQA, described in further detail in subsections 4.2.1 to 4.2.7:

- **Apache UIMA**[42]
- **Apache Solr**[43]
- **Apache Jena**[44]
- **Apache Fuseki**[45]
- **Apache OpenNLP**[46]
- **Stanford NLP**[47]
- **DKPro**[48]

4.2.1. Apache UIMA

Apache UIMA is a framework for *unstructured information management applications*, characterised by the need to process large amounts of unstructured data. It is the framework YodaQA is built on and represents

the backbone of the system.
The advantage its use provides is that applications developed using this framework can be set up as modular components, with the details of the exchange of data between them handled by UIMA. Different components can be defined for the different analytical tasks to be carried out on the incoming data. These can be used to build up a pipeline, with the individual components serving to transform, process, annotate, analyze and/or synthesize the original data and the data generated within the pipeline.

Below is an overview of important UIMA concepts that will resurface regularly in the coming chapters:
Analysis engine
Analysis engines (*AE*s) are modules that analyze artifacts (documents, audio, video, etc.) and infer information from them. Analysis engines that contain a single annotator are called *primitive* AEs and ones with multiple annotators are *aggregate* AEs.
Annotator
Analysis engines are made up of annotators. They contain the custom logic for the specific analytical tasks the application needs to carry out. The analysis results they produce are available for use by succeeding annotators.
CAS, JCas
From an information flow point of view, the Common Analysis Structure (*CAS*) has a central place in UIMA. These are data structures that serve as the containers used to store objects, properties and values. The communication between UIMA components takes place through CASes.
The CAS is merely an interface specification and so is not tied to a particular implementation. As YodaQA is written in Java, it would be easiest to interact with CASes in this language. This is done via *JCas*: the Java interface for CAS bundled with the UIMA SDK that facilitates programmatic handling of CASes within Java applications.
Sofa (subject of analysis)
A CAS can contain multiple *subjects of analysis*, which represent different views on the artifact being analyzed. In case of a news article, for example, there could be different views for its syntactic structure, for information about the authors, and for other, potentially related, articles.

Figure 4.2 provides a visualization of this: the pipeline represents different analysis engines, and on top are the different views within a CAS.

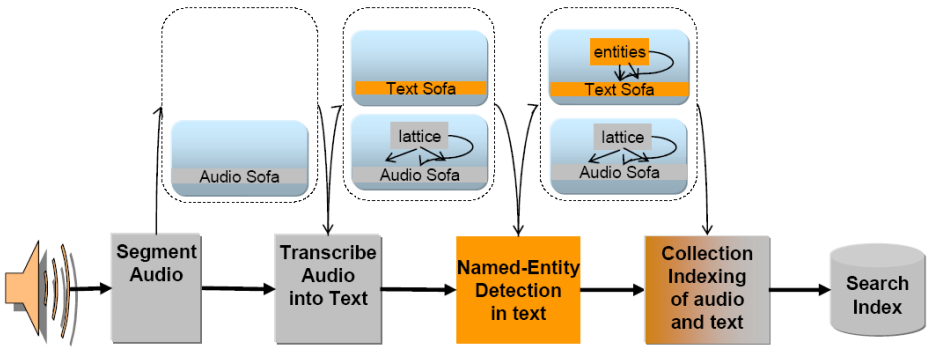


Figure 4.2: UIMA Analysis engines and sofas

source: https://uima.apache.org/d/uimaj-2.4.0/images/overview-and-setup/conceptual_overview_files/image014.png

FeatureStructure, Feature
Annotators produce results within CASes in the form of *FeatureStructures*: data structures with a type and a set of *(attribute, value)* pairs. An example is a *Person* feature structure containing a *name* feature with value *Bob*. A feature structure therefore serves a purpose analogous to a Java class, and features can be thought of as similar to fields within a class.
Type system
A type system includes the types living within a given CAS. UIMA comes with a number of built-in types and provides the option to extend this collection with user-defined ones. Continuing the Java class analogy, a type

system can be considered a collection of classes that together model a certain part of the conceptual space associated with the task at hand.

jCasGen

Tool that 'autogenerates' Java types based on type system descriptions specified in XML. This makes them directly usable by UIMA within CASes. The reason why the developer doesn't specify the types directly in Java is that *jCasGen* injects low level logic into the types to make them usable within UIMA. In other words, the XML specification represents an abstraction layer shielding developers from such 'boilerplate code'.

4.2.2. Apache Solr

Apache Solr is a search server providing data storage and querying capabilities. It exposes a REST API that can be used to query data within Solr instances using web requests.

Solr supports advanced text searching functionality and works well with large volumes of data. This makes it a fitting choice for a question answering system, where the ability to access and efficiently search within a comprehensive knowledge base is crucial.

4.2.3. Apache Jena

Apache Jena is a Java framework for semantic web and linked data applications. It is geared towards RDF data and contains a number of subprojects for storing, querying and processing such data. Jena comes with a number of APIs: for representing RDF data, for querying using the SPARQL language, and for Apache Jena Fuseki, a SPARQL server.

Jena is used within YodaQA for interacting with structured knowledge bases. Because an increasing number of such bases are becoming available for different domains, and because they are more 'machine-friendly' in contrast to unstructured sources, effectively handling them can potentially be very beneficial to the performance of a QA system.

4.2.4. Apache Jena Fuseki

Apache Jena Fuseki is part of the Apache Jena project. Fuseki is a server with a purpose comparable to Solr: whereas Solr stores documents that can be queried using the Lucene query syntax, Fuseki stores RDF triples that can be queried using SPARQL, also using web requests.

4.2.5. Apache OpenNLP

OpenNLP is a library containing NLP tools for various purposes, like *named entity recognition*, *segmentation*, *tokenization* and *part-of-speech tagging*. YodaQA uses it chiefly for the first two purposes.

4.2.6. Stanford NLP

Similar to Apache OpenNLP, Stanford NLP is another library providing various NLP tools. From YodaQA's perspective, the *Stanford parser* is the most important one among these: it is the tool used to parse all text in the system's default (English) version.

4.2.7. DKPro

DKPro is a framework for the integration of the various NLP libraries used by YodaQA so that they can all be interfaced with in a similar way within the codebase. The question of integration arises due to the structure of YodaQA. Because it is based on UIMA, all different annotator modules need to conform to the interface requirements imposed by UIMA. DKPro meets this challenge via components that make various existing NLP tools (parsers, lemmatizers, tokenizers, etc.) usable within a UIMA application like YodaQA - mainly by providing Java wrappers for them.

4.3. Conclusion

This chapter provided an overview of YodaQA and the main technologies it depends on. As can be seen in the list of technologies at the start of section 4.2, six of the seven most important libraries and frameworks are related to natural language processing and information retrieval; while being a very fitting choice, UIMA isn't designed specifically for either of these purposes.

This is no surprise for a question answering system: we need to handle natural language in the form of both questions and candidate answers, and we need comprehensive data sources for finding candidate

answers in the first place. It follows that, during the process of adapting the question answering pipeline to its new language and domain, the focus will mainly be on these two areas. The next two chapters will therefore detail the work done to adapt the information retrieval and natural language processing stages of YodaQA.

Chapter 5 discusses the replacement of the English sources in place with ones containing Dutch content. The adaptation of the natural language processing pipeline is then discussed in chapter 6.

5

Sources

This chapter describes the technical steps that were required to set up Dutch data sources for YodaQA. A useful feature of YodaQA is its ability to work with and combine information from multiple data sources. The default technologies used are Apache Solr for unstructured data and Apache Jena Fuseki for structured data.

The research question this chapter bears on is the second:

RQ 2: *Which sources and source types contain information that supplement the topics the Nao is presenting about and fit the interest of the audience?*

Any installation and configuration steps are discussed only briefly here. Refer to appendix B for links to the full instructions.

5.1. Structured and unstructured sources

Within Solr and Fuseki, Wikipedia and DBpedia are the two main sources of information used. Wikipedia contains articles expressed in natural language ('free text'). DBpedia contains data in machine-readable format, expressed in the form of RDF triples (*subject-predicate-object*).

The sources that can used within YodaQA are currently limited to these two sources: Solr and Fuseki. Because the answer production stage (as shown in figure 4.1) comes with interfaces only for these, substituting different technologies cannot be done readily. This requires extending YodaQA with custom interfaces for the new technologies and is not a part of this project. On the other hand we are *not* limited to Wikipedia and DBpedia; in Solr we can place any data that can be modeled with a *title* and *text* attribute and in Fuseki any data that can be modeled using RDF.

As discussed in 2.6.3, within this project we can make good use of both types of sources. In DBpedia we can store data that can be modeled as an object or concept with well-defined attributes, such as the Nao robot and its characteristics. In Wikipedia we can store articles about the topics we are supporting (robots, computers and technology in general) that cannot be easily transformed into a structured format.

5.2. Data locality

When choosing between online and offline datasets, a tradeoff needs to be made between being up-to-date and speed. On the one hand, if the system is set to go online to search for answers, the most up-to-date information can (in principle) be found, but at the cost of network overhead. On the other hand, using local copies allows for faster lookup, disk latency generally still being smaller than network latency. Since real time operation is a crucial part of this project, having the data sources offline is the selected approach.

Of course in this case update mechanisms need to be in place, to allow maintainers to periodically update the local data with new information as it becomes available. Sections 5.3 and 5.4 describe the setup and maintenance of the unstructured and structured data sources, respectively.

5.3. Unstructured data

For unstructured data, entries containing free text (called 'documents') are hosted by a Solr instance. For the English version of the system, a Solr instance containing an English Wikipedia dump on the original author's machine is used.

Using Wikipedia is not strictly required: any data that can be expressed as an entry with *id*, *title* and *text* attributes (news articles, encyclopedia entries, essays, etc.) can be inserted into Solr via its user interface and then made use of by YodaQA. The reliance on Wikipedia by YodaQA as the default unstructured data source is primarily because the aim of the system is answering factoid questions, and Wikipedia is a rich source in this respect.

5.3.1. Setup and maintenance

The following steps were taken to set up a Solr core containing Dutch Wikipedia articles:

1. **Retrieval:** downloading a Wikipedia data dump.
2. **Preprocessing:** transforming the articles for importing. The Solr interface provides functionality to import entries from XML files. For the Wikipedia dump, a custom extraction tool was used, which transformed the dump into a single XML file ready for importing.
3. **Importing:** inserting the individual documents specified in the data file into a Solr instance. Two modes are offered by Solr for imports: *full-import* simply imports everything in the file and creates a new collection index, while *delta-import* only imports entries present in the file but not in the Solr core and updates the existing index.
4. **Filtering:** filtering out articles not related to the desired domain. Solr provides querying functionality within its interface: filtering can be done by specifying text which should (or should not) be in the document titles and bodies.
   For this project, all documents within the Dutch Wikipedia dump that did not contain one of the following keywords were filtered out (both partial and exact matches sufficed):
   robot, *computer*, *kunstmatige intelligentie*, *nao*, *aldebaran*, *sensor*, *actuator*, *programmeren*.
   This resulted in a dataset of about 6600 documents.
5. **Customization:** extending the collection with custom articles, by filling in the *title* and *text* attributes of the new documents.

Maintenance

Individual documents can be added to the database via the command line or using the Solr graphical user interface. For bulk inserts, the individual documents should be combined into an XML file, after which this file can be imported from within the Solr interface as described above. For new document collections the *full-import* mode can be used; otherwise, *delta-import* is more efficient. For all documents, whether entered individually or in bulk, *title* and *text* attributes should be specified, and optionally an *id*.

5.4. Structured data

For structured data, a Jena database containing RDF triples is used. This database is hosted by a Fuseki server, which can serve RDF data over HTTP in response to query requests. By default, an instance containing an English DBpedia[49] dump on the original author's machine is used. The use of DBpedia is a logical choice after settling for Wikipedia as the unstructured source, as DBpedia tracks it and extracts data from it.

5.4.1. Structured data in YodaQA

Searching within structured data in YodaQA is a two-step process.
First, based on the subject and other important clues identified in the question, a primary query is executed to determine whether any of these tokens match the labels of any resources in the Jena database. This step is carried out by a separate component called the *label lookup* service[50].
If this step returns a positive result, all attributes associated with the matching resource(s) are retrieved. With the use of Wordnet[34], these attributes are then scored based on their similarity to one or more of the identified clues in the question.

5.4.2. Setup and maintenance

The following steps were taken to set up an RDF database containing Dutch DBpedia entries:

1. **Retrieval:** downloading a Dutch DBpedia dump.
2. **Importing:** creating a Jena database and importing the dumps into it.

3. **Customization:** extending the database by creating a *Nao* resource and modeling Nao attributes (like *name*, *weight*, etc.) as RDF triples.
4. **Indexing:** creating a list of the labels (a *label index*) of all resources in the database (including the new one for the Nao), to be used by the label lookup service.

Maintenance
The YodaQA DBpedia interface looks specifically for DBpedia resources. Resources and properties live within the `http://nl.dbpedia.org/resource/` and `http://nl.dbpedia.org/property/` namespaces, so these namespaces are also the ones in which new resources and properties should be placed.

The sample query below creates a *Nao* resource if it doesn't exist and updates it with the given *label* and *naam* attributes. The prefixes indicate the namespaces to be used within the query.

```
PREFIX nldbr:    <http://nl.dbpedia.org/resource/>
PREFIX nldbp:    <http://nl.dbpedia.org/property/>
PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema#>

INSERT DATA
{
  nldbr:Nao rdfs:label "Nao (robot)" .
  nldbr:Nao nldbp:naam "Robin" .
}
```

When adding/deleting resources, attention should be paid to also add/delete the name of the resource in the *label index*, to allow it to find this new resource/not look for it anymore. When updating existing resources, no changes need to be made to this file (unless of course it is the name of the resource that is being updated).

5.5. Conclusion

In this chapter we took a closer look at the sources used by YodaQA. We saw identified a need to be able to use both unstructured and structured sources. The system currently works with unstructured data in a Solr instance and structured data in a Fuseki instance.

The Solr instance originally containing English Wikipedia dump was replaced with a Dutch Wikipedia dump. A similar process was carried out for Fuseki by importing a Dutch DBpedia dump.

6

Natural language processing

This chapter describes the technical steps that were required to adapt the question and answer analysis pipelines to work with Dutch input. The research question that corresponds to this chapter is:
RQ 3: *Which Dutch natural language processing libraries are fitting tools for processing the natural language content the system will work with?*

The two main NLP libraries used by YodaQA are the Stanford parser and the Apache OpenNLP parser. The former has a more prominent role in the system and is the default tool for processing questions. Another important resource is the English WordNet, used to estimate the similarity between two words. Within YodaQA this is an important source, used to assess the relevance of DBpedia properties to concepts identified in a question.

Unfortunately for our purposes, neither library supports parsing Dutch by default and there also were no Dutch parsing models available for either parser. The English WordNet also, of course, only supported English. This meant that these components had to be replaced with others that do support Dutch.

Section 6.1 describes this process for the Dutch word net. The remaining sections explain this for parsing. Section 6.2 discusses the initial setup with TreeTagger, section 6.3 the more advanced setup using the Alpino parser and section 6.4 the technical details involved in running this parser.

6.1. Open Dutch Wordnet

As mentioned in the section about structured data, the English version of YodaQA makes use of Wordnet for word similarity calculation. This is used to have more flexibility when matching questions to candidate answers: if similar words can also count (instead of requiring exact matches) when evaluating the relevance of candidate answers, the likelihood of identifying suitable answers increases.

Two libraries were used to get this working for Dutch. The first is the Open Dutch Wordnet[51], containing word nets (synonym sets and relations between them) for Dutch. The second is the WordnetTools[52] library which, given a word net, can calculate the similarity between a pair of words. These come into action when DBpedia properties are found for a concept identified in the question: for each property, its similarity to each token in the question is calculated. High scoring properties are then taken along as strong candidate answers.

6.2. TreeTagger

As a first step towards analysing Dutch sentences, a part-of-speech (POS) tagger called *TreeTagger*[22] was integrated into the question analysis pipeline. This is a tool that can identify the lexical categories (*noun, verb, adjective, ...*) to which words in a sentence belong.

There were a couple of reasons for starting in this way. First off, it helped in getting familiar with the codebase, its structure and how to go about refactoring and extending it. Additionally, it has already been used successfully in projects with a similar focus[21].

The listing below shows the POS annotations for “Wat is een robot?" (“What is a robot?"). The left column

contains the tokens, the middle one contains the POS tags assigned to them by the tagger, and the right one contains the lemmas. The tags shown here are *question pronoun*, *present tense singular verb*, *determiner/article*, *singular noun* and *punctuation mark*.

```
Wat     pronquest       wat
is      verbpressg      zijn
een     det__art        een
robot   nounsg          robot
?       $.              ?
```

This approach already led to decent results when the answers were present in structured data. Since the data was structured, it was possible to get away with a relatively shallow understanding of the question, without identifying sub-phrases and dependencies.

Technical details Before the tagger output can be used within the QA system, a mapper needs to be in place which can transform the POS annotations into Java types. The library used by YodaQA for integrating the various NLP components, DKPro, already provided a wrapper for the tagger. However, this didn't work out-of-the-box, because the input format passed to the tagger didn't match what was expected by *TreeTagger* and so had to be refactored.

6.3. Dependency parsing

The shortcomings of the 'part-of-speech only' approach became apparent when processing questions for which the answers were present as natural language text. Since in this case the data and corresponding properties aren't explicitly modeled, questions need to be analyzed deeper.

To understand this better, consider the *Nao* resource in the structured data source. All attributes of this resource will have been formulated explicitly in the form of RDF triples. In the unstructured data source, on the other hand, this data will be present within natural language sentences, which can vary greatly in their structure. Whereas we know to look at the *object* elements for candidate answers in RDF triples, in free text we first have to understand how a given sentence is constructed and where exactly the answers might be located.

This is the point at which the use of a parser that can identify structure and dependencies becomes relevant. Besides identifying concepts like nouns and verbs, it becomes useful to identify (among others) subjects and expected answer types (name, location, number, ...), so that we can detect possible answers more easily.

6.3.1. Alpino parser

For this purpose, the Alpino dependency parser[53] was integrated into the question analysis component of the system. This tool runs as a server process and by creating a client within YodaQA, question sentences can be passed off to it to receive responses containing the parsing output. The parser returns a parse tree in XML format, which the client can then process recursively to create annotations about tokens, sub-phrases and dependencies.
To make the parser output usable within the QA system, UIMA[42] type systems had to be created within YodaQA for the Alpino lexical category, dependency and part-of-speech annotations.

The listing below shows the XML parsing output for "Wat is een robot?". Nodes can cover multiple tokens (as indicated by the *begin* and *end* tags). Nodes can have many more attributes (*id*, *case*, *tense*, etc.), which have been left out here for brevity. Lexical categories are indicated with the *cat* attribute, dependency types with *rel* and POS tags with *pos*.

Listing 6.1: Parse tree

```
<?xml version="1.0" encoding="UTF-8"?>
<alpino_ds version="1.5">
  <node begin="0" cat="top" end="5" rel="top">
    <node begin="0" cat="whq" end="4" rel="--">
      <node begin="0" end="1" lcat="np" lemma="wat" pos="pron" rel="whd" word="Wat"
          />
      <node begin="0" cat="sv1" end="4" rel="body">
        <node begin="0" end="1" index="1" rel="predc"/>
```

```
        <node begin="1" end="2" lcat="sv1" lemma="zijn" pos="verb" rel="hd" root="
            ben" word="is"/>
        <node begin="2" cat="np" end="4" rel="su">
          <node begin="2" end="3" lcat="detp" lemma="een" pos="det" rel="det" root="
              een" word="een"/>
          <node begin="3" end="4" lcat="np" lemma="robot" num="sg" pos="noun" rel="
              hd" word="robot"/>
        </node>
      </node>
    </node>
    <node begin="4" end="5" lcat="punct" lemma="?" pos="punct" pt="let" rel="--"
        root="?" word="?"/>
  </node>
  <sentence sentid="127.0.0.1">Wat is een robot ?</sentence>
</alpino_ds>
```

Dependency relations for the same question, as identified by the Alpino *dependency triples* hook, are shown below. The left column contains the *head* words of the dependency relation, the middle column specifies the *relations*, and the right column contains the *dependent* words. The final line, for example, indicates that the "een" token acts as a "determiner" for the "robot" token.

Listing 6.2: Dependency triples

```
top/top          |top/hd|        wat/[0,1]
wat/[0,1]        |whd/body|      ben/[1,2]
ben/[1,2]        |hd/predc|      wat/[0,1]
ben/[1,2]        |hd/su|         robot/[3,4]
robot/[3,4]      |hd/det|        een/[2,3]
```

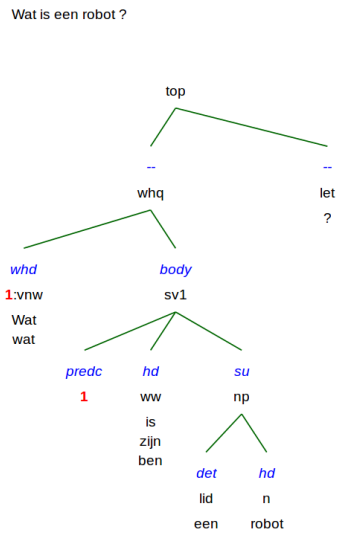


Figure 6.1: Alpino parse tree for "Wat is een robot?".

6.4. Alpino integration

6.4.1. Running Alpino

The Alpino parser can be run in two ways: as a system process and as a server. The system process processes all inputs sequentially, regardless of whether they were passed one by one or in bulk. Consequently, this approach provides no scaling benefits when multiple sentences need to be parsed.

The Alpino server, on the other hand, does have parallel processing capabilities and can parse multiple inputs simultaneously by creating separate 'workers' for each input. The server can also be deployed on

a machine other than the one on which the QA system is running. These considerations are why running Alpino as a server is the approach taken in this project.

6.4.2. Extracting dependencies

While the Alpino parse tree (see listing 6.1) implicitly indicates the dependency triples within the sentence it has parsed, it is less complete than the output received from the `triples` hook. As listing 6.2 shows, the output from this hook explicitly indicates *which* token has *what* relation to *which* other token, together with their indices.

We would not want to have to submit two requests to Alpino per sentence, one for the parse tree and one for the dependency triples. Fortunately, the Alpino library comes with an *XQuery* script that can process the parse tree and extract these triples from it. With small modifications (to include the index of the tokens in the output), the script was refactored into giving the same output as the one received from the `triples` hook.

The last step was to be able to carry out this extraction within the QA system itself, from a parse tree while answering a question. For this, the *Saxon XSLT and XQuery Processor*[54] was used. This library provides functionality to load XQuery scripts and apply queries specified in them to XML documents.

The way this is used in YodaQA is as follows: the Saxon query evaluator is built up once during program initialization (to avoid having to repeatedly load the same query file from disk), using the mentioned XQuery file bundled with Alpino. Then, within the pipeline, on receiving the parse tree for a sentence, the evaluator is applied to the tree and the results are used to generate dependency annotations.

6.5. Conclusion

Adapting the natural language processing pipeline of YodaQA to work with Dutch content required the use of new software tools. Two tasks presented themselves: integrating a Dutch word net and integrating a Dutch parser.

For the first task we configured the Open Dutch WordNet. It was integrated into YodaQA using the WordNetTools library. This library allows programmatic access to the word net and can be used to determine word pair similarity during the process of answering a question.

For the second task we first used the TreeTagger POS tagger as an initial setup. We soon reached the limit of this tool as only a limited number of question types can be handled if only POS annotations are used. Therefore, we turned to the Alpino dependency parser and configured it for YodaQA. We use Alpino by running it as a server and communicate with it using web requests via a client within the YodaQA. Finally, we use the Saxon XSLT library to extract dependencies from Alpino parse trees and acquire our dependency relations in this way.

7

Answer matching

After searching for relevant data on the basis of the analysis carried out on a question, the QA system gets to the task of matching the question representation with representations of candidate answers. This has the purpose of determining which parts of which candidates are most relevant for the given question. This chapter explains how we do this in this project, for data from both structured and unstructured sources.

This chapter, like chapter 4 about YodaQA, is related to our first research question:
RQ 1: *What adaptations need to be made to the question answering pipeline of the chosen system?*
Here we will investigate what changes to the answer matching logic were needed to make it work for Dutch candidate answers.

7.1. Structured data

Compared to matching answers from Wikipedia, matching ones from DBpedia takes place via a simpler way. The question representation includes annotations like *subject*, *main verb* and *object*. These are used to retrieve relevant DBpedia resources. Finally, their $(property, value)$ pairs are assessed based on how similar the *property* entry in these pairs is to one of these tokens. The similarity score is calculated using the Open Dutch WordNet.

Example:
Question: Wat is de prijs van de Nao?
Retrieved resource: Nao
Highest scoring $(property, value)$ **entry**: $(prijs, €5000)$
Answer: $5000

7.2. Unstructured data

Matching candidate answer representations coming from unstructured sources is more challenging, because of the lack of structure and annotations in these representations.

There are two important strategies that can be used to match candidate answer sentences with questions. The first is to take the question predicate, follow it in the candidate answer sentence, and use its object as an answer. The second approach works in the other direction: we start by looking for certain grammatical structures in the candidate answer sentence and assess their relevance to the question.

We explain both approaches using an example:
Question: "Wie maakte y?"
Sentences: "x maakte y" and "x ontwikkelde y"

7.2.1. Approach 1: Following the predicate

The first approach starts with the predicate in the question; in this case, this is the verb phrase "maakte" linked to the subject ("x"). As this predicate does occur in the first sentence we 'follow' it along to its object, leading us to identify y as a candidate answer.

The predicate does not occur in the second sentence and this approach therefore does not consider it further.

7.2.2. Approach 2: Searching for structures

The second approach uses the expected answer type as a starting point. As will be detailed below, we determine that the question expects a *person/name* and that the grammatical structure to look for is a *predicate complement (predc)*.

Next, we use this insight to look for structures of type *predc* in the given sentences. The "*y*" sub-phrases in both sentences fit this requirement, so we mark both as candidate answers. Finally, to assess their relevance, we determine the similarity of the question predicate to both predicates in the sentences ("maakte" and "ontwikkelde"). The first one obviously gets the highest score. The second also receives a significant score (more than half of the maximum); as it should, since both sentences come down roughly to the same.

We use both approaches when matching question and candidate answer representations. The first approach allows us to cut the search short if we find an explicit formulation in the sources that closely matches the question formulation. The advantage of the second approach is its added flexibility. It enables us to also consider formulations that are not identical but similar to the question, as long as it contains a grammatical structure that fits the *expected answer type* constraint inferred from the question.

Table 7.1 gives an overview of which grammatical structures to look for for each type of expected answer. The subsections afterwards explain these using examples.

Table 7.1: Expected answer types and corresponding grammatical structures

Expected answer type	**Target grammatical structure(s)**
Location	locative or directional complement (*ld*)
Date/time	measure phrase complement (*me*)
	modifier (*mod*)
Person/name	predicative complement (*predc*)
Amount	direct object (*obj1*)
Default	predicative complement (*predc*)
	prepositional complement (*pc*)
	direct object (*obj1*)
	secondary or indirect object (*obj2*)

7.2.2.1. Location

Question: "Waar is Microsoft gevestigd?"
Expected answer type: "Waar" → location.

Candidate passage: "Microsoft is gevestigd **in de Verenigde Staten.**"
Matching answer: The highlighted sub-phrase matches because it is of type *ld*.

7.2.2.2. Date/time

Question: "Wanneer is Bill Gates geboren?"
Expected answer type: "Wanneer" → date/time.

Candidate passage: "Bill Gates is geboren **op 28 oktober 1955**."
Matching answer: The highlighted sub-phrase matches because it is of type *mod*.

7.2.2.3. Person/name

Question: "Wie is Steve Jobs?"
Expected answer type: "Wie" → person/name.

Candidate passage: "Steve Jobs is **de medeoprichter en topman van Apple**."
Matching answer: The highlighted sub-phrase matches because it is of type *predc*.

7.2.2.4. Amount

Question: “Hoeveel verdien je?"
Expected answer type: “Hoeveel" → amount.

Candidate passage: “Ik verdien **1000 euro per maand**."
Matching answer: The highlighted sub-phrase matches because it is of type *obj1*.

7.2.2.5. Other/default

The structures in this category are used when the expected answer types is not one of the four mentioned. Of all the remaining Alpino types, these four are the ones that can serve as the object of a verb in a sentence.

Question: “Wat is een computer?"
Candidate passage: “Een computer is **een apparaat waarmee gegevens volgens formele procedures (algoritmen) kunnen worden verwerkt**."
Matching answer: The highlighted sub-phrase matches because it is of type *predc*.

Question: “Waarvoor wordt een computer in het onderwijs gebruikt?"
Candidate passage: “In het onderwijs wordt de computer gebruikt **voor het opzoeken van informatie**."
Matching answer: The highlighted sub-phrase matches because it is of type *pc*.

7.3. Conclusion

In this chapter we discussed the third significant change to the YodaQA pipeline: that of adapting the answer matching logic. Doing this for structured data was not a big change but rather of configuring the Dutch word net and rewriting queries accordingly. For unstructured content, however, more changes were needed.

A matching algorithm was implemented that combines two approaches: following the question predicate in the candidate answer, and looking for a semantic structure fitting the expected answer type of the question. This was done by inferring the question type from keywords in the question and making use of Alpino annotations to look for matching structures in the candidates.

8

Interaction manager

All chapters up to now in this part about implementation have been about the question answering system itself. Just as important, however, is the part of the system facing users. This chapter describes the interaction manager: a wrapper around the QA system that handles the inputs from and outputs to the audience.

The research question relevant to this chapter is:
RQ 4: *What type of robot-audience interaction allows the robot to hold a clear, structured and informative Q&A session?*
Our main goals are to develop a manager that provides a smooth experience, shields users from low-level details and presents the QA system and the Nao robot as a conversational partner.

The standard input mechanism of the system is the command line: a bare-bones prompt where a question can be typed. This is fine for internal usage, but lacking for the average user that might not even have used a command line before.

The interaction manager has a number of features for enhancing the user experience. One is the usage of a pool of 'prompts' that it uses for asking for a question. Another one is a pool of negative responses (in case no answer could be found or only ones with low scores).

The manager also repeats questions verbally to the user after passing them to the QA module. This serves both to give the user a confirmation that the question was received properly, and to fill up a part of the answering time. If the answering process for a question takes a long time, we also fill the time with various progress updates. Finally, all verbal output of the robot is accompanied by various gestures and movements.

8.1. Algorithm

Listing 8.1 describes the interaction management algorithm.

Listing 8.1: Interaction manager

```
P = collection of prompts
U = collection of progress updates
R = collection of negative responses

Introduce session
   Ask for the name of the user and greet him/her with own name
While exit command is not received:
   Ask for question
      Output randomly selected entry from P
   Submit question to QA module
   Repeat question verbally
   While QA module is busy:
      Periodically output randomly selected entry from U
```

```
Return response
    If no answer or low scoring answer, output randomly selected entry
    from R
```

P contains prompts like *Is there something you'd like to ask?* and *Do you have another question?*.
U has updates like *Let me see . . .* and *I have to think about that.*.
The negative responses in *R* are along the lines of *I don't know that.* and *I couldn't find an answer to that question.*.

After the introduction, a loop is entered in which a question is received, submitted to the QA system, and the answer to it passed on to the user. If the system is still busy after some time, a progress update is given. If the returned answer is not certain enough or there is no answer at all, it indicates that no answer could be found.

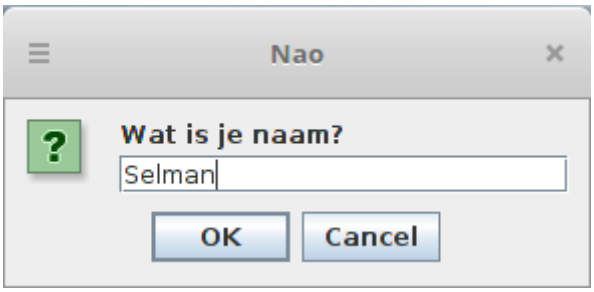
8.2. Implementation

We created a separate Java application for the interaction manager which implements the aforementioned algorithm. This application communicates with YodaQA over sockets. For this work, a new mode was added to YodaQA that works with sockets; this involves starting up a server within YodaQA and listening for connections. The client within the interaction management application can then connect to this server and submit questions to it in this way.

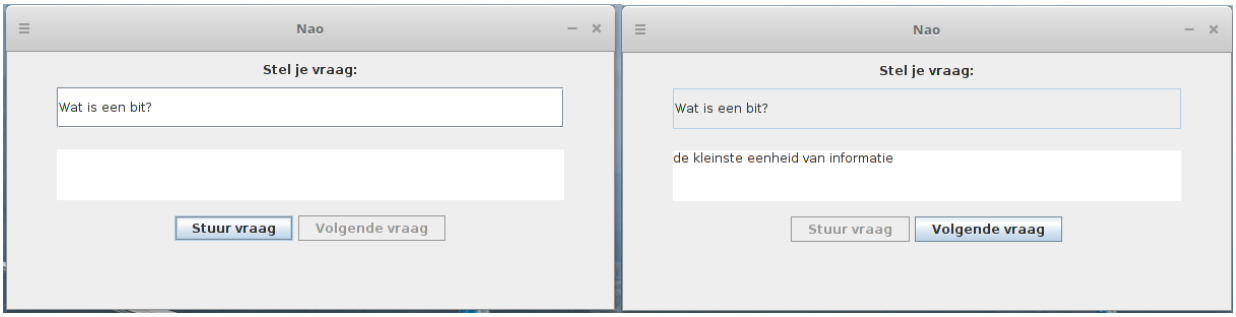
As mentioned somewhere in literature review, we want to provide a smooth experience and shield users from low-level details. For this reason, we wanted to avoid having the children enter their questions into the command line and a rudimentary graphical user interface was added to the program. Questions can be entered into this interface and answers returned by the QA system are displayed here as well.

A special `START` command was included in the application that, if entered, makes the manager introduce a new session by asking for the user's name. In this way, the program does not have to be restarted to switch to a new user.

Figure 8.1 shows some screenshots from the interaction manager. (a) is the screen in which users can enter their name, (b) shows a question being entered, and (c) shows the answer being displayed.



(a)



(b) (c)

Figure 8.1: The interaction manager graphical interface

8.3. Conclusion

The interaction manager for the question answering system should be such as to make the system comes across as a conversational partner and provide a smooth experience. For this purpose we implemented an interaction management algorithm that passes questions and answers to and from the QA system. Besides this, it has features like introducing itself to users, asking for questions and giving progress updates. To further enhance the user experience a graphical interface was added to the manager in which questions can be entered and answers displayed.

Part III

Zooming in

9

Question answering pipeline

In the last few chapters we covered a number of topics like the QA pipeline, the natural language processing modules, and the usage of the configured data sources. The two chapters of this part serve to make this all more concrete. This chapter describes the question answering pipeline in more detail. The next one will illustrate its workings by traveling along this pipeline, starting with a question and ending with the system's response.

As discussed in 4.2.1 about Apache UIMA, the YodaQA pipeline is composed of analysis engines. Data within this pipeline is represented in the form of *Common Analysis Structure (CAS)* objects. Each engine annotates the CASes it receives from its predecessor and might also create new CASes itself.

The YodaQA pipeline has five main stages: analyzing the question, searching for information, analyzing retrieved data, consolidating everything into a single collection, and scoring it. Each of these stages is described in detail in the next sections. For each stage we describe what input it takes, the annotations it carries out and the output of the stage. For clarity, generic terms have been used here instead of the low-level YodaQA names and terminology. From the second stage on, inputs are not stated as these are equal to the outputs of the previous stage.

9.1. Question analysis

This engine contains annotators processing the question. This engine contains annotators that process the input question. The goal is to identify important tokens and sub-phrases and determine what type of answer we are looking for. Annotations and clues generated in this stage will be needed later on, especially once we start trying to match candidate answers.

Stage input: a question sentence
Tasks:

- *Parsing*: parse the question sentence using the Alpino dependency parser. The output of the parser contains a *parse tree*, *lexical categories*, *dependency annotations*, *sub-phrases* and *part-of-speech* tags. This output forms the basis for all subsequent tasks in this stage, as every coming task makes use of some part of the Alpino output.
- *Focus identification*: identify the *focus* of the sentence. This can roughly be defined as that part of the question that can be replaced with the answer.

  Example:
  Question: "What is the mass of the Nao?"
  Focus: "the mass", because this is the part of the sentence that can be replaced with the answer.

  The *focus* annotation serves to locate answers in relevant candidate answer sentences. In the example below we note that the question focus does occur in the candidate and follow the associated predicate ("is") to arrive at the answer.

  Candidate answer sentence: "The mass of the Nao is 4.3kg."
  Answer: 4.3kg

- *Subject generation*: identify the sub-phrase serving the role of subject in the sentence, which can be inferred directly from the Alpino *subject* annotation. *Subject* annotations are used for querying the sources, based on the assumption that if *some* token of the question sentence has an entry dedicated to it within the sources, it is probably the subject.
- *Main verb selection*: determine the main verb of the question to focus on. This is deduced by identifying the verb that depends on the *subject* of the sentence.
- *LAT generation*: determine the *lexical answer type* (number, name, location, etc.) expected by the question. This is the semantic type to which candidate answers must conform to be considered relevant. This comes in handy during the later answer analysis and scoring stages, where it is used to assess how well a candidate answer fits what is expected by the question.
- *Verb-initial sentence detection*: check whether the question starts with a verb. This is to determine whether the question is a confirmation (yes/no) question; if it is, the execution flow will change slightly, as will be explained in the next sections and illustrated in the next chapter.

  As an aside, this step can also be used to detect imperative sentences. Since we don't do anything with such input, though, it is irrelevant for our purposes.

Stage output: a *Question* CAS with values for all mentioned annotations.

9.2. Information retrieval

This stage is responsible for searching for data within the configured sources, based on the clues generated during question analysis. It consists of three parts: for DBpedia, Solr and Freebase. We do not use the one for Freebase as Freebase is an English-language source only.

9.2.1. DBpedia

Tasks:
A search query is formulated based on the clues generated during question analysis. This is basically a disjunction of all significant tokens and sub-phrases identified during this phase. A 'fuzzy' lookup is used to compensate for small spelling errors like incorrect case or punctuation mark usage.

The retrieved DBpedia resources are then filtered to keep only those that match at least one clue.

9.2.2. Wikipedia

Tasks:
This engine carries out full-text searches through Solr and looks for potentially relevant passages within Solr documents.

Retrieved passages are first parsed by Alpino, in the same way as in the question analysis stage. Then the degree of matching between the passage text and question clues is determined. Afterwards, noun phrases and named entities within the passage are extracted, again similar to question analysis.

Stage output: DBpedia resources and passages from Wikipedia articles.

9.3. Answer analysis

The tasks in the *information retrieval* stage typically result in the creation of a number of CASes, one for each candidate answer. It is the task of the annotators within the *Answer analysis* stage to annotate these candidates with NLP and QA features.

This is analogous to the tasks carried out in the question analysis stage. The purpose is to annotate the candidate answers to a sufficient degree to enable the upcoming *Answer scoring* stage to evaluate them. At the end of this stage we will have annotations for the question on one hand and for the candidate answers on the other hand, which will allow us to see how well they match each other.

Tasks:
Determine the overlap between the candidate answer and the clues identified in the question analysis phase. The remaining tasks are similar to the question analysis stage and involve identifying the focus of candidate answer sentences and generating *LAT* annotations for them.

Stage output: candidate answers together with annotations.

9.4. Answer merging

This engine acts as a 'sink' for all answers that have been found. On detecting that all candidate answers have passed through the *Answer analysis* stage, it consolidates them into a single list and removes any duplicates. The reason why duplicates can arise is because searches are carried out in different data sources and these might contain the same piece of information.

Tasks:
Merge candidates into one collection.

Stage output: a single list containing all unique candidate answers and their annotations.

9.5. Answer scoring

This stage evaluates the quality of the answers. The original (English) version did this by applying two classifiers. The first is a model-free rule-based classifier and the second is a trained decision forest classifier.

For the Dutch version only the first one was kept. Its rules were adapted by building on the existing algorithm, so as to consider DBpedia similarity scores and matching Wikipedia sentences in a way that better fits the requirements of this project.

This approach led to good results, so no effort was dedicated to training a decision forest or another machine learning classifier. However, investigating the effects of using this type of classifier can definitely be considered for future improvements.

Tasks:
Score all candidates.

Stage output: candidate answers annotated with confidence scores, ordered in descending order on these scores.

10

From question to answer

This chapter is a follow-up to the previous one where we outlined the structure of the pipeline. In this chapter we will fill in the pipeline by following the path of a question from start to finish. We will see this pipeline in action by feeding it a question and looking at the actions and output this input triggers throughout all stages.

For each of the five stages of the pipeline the main tasks it carries out and its output will be described. We will do this twice: once for a factoid question and once for a confirmation question.

10.1. Factoid question

Question: “Wat is een computer?"

10.1.1. Question analysis

- The sentence is submitted to the Alpino parser, which determines that “een computer" is the *subject* and that it depends on the verb “is".
- “computer" is identified as the *focus* and “is" as the *selective verb*.
- “computer" and “een computer" are given *subject* annotations, as they are the narrowest and widest sub-phrases serving the function of subject in the sentence.
- “computer" is also annotated as an *LAT* (lexical answer type); the more a candidate answer is similar to its type, the more relevant it will be considered.
- *Clues* are created for “computer" (because it is a *LAT*, a *subject* and the *focus*) and for “een computer" (because it too is a *subject*). Normally a clue is also created based on the selective verb but not in this particular case, as common verbs like “to be" and “to have" are excluded because of their relatively small added value.

Stage output: the aforementioned NLP and QA features.

10.1.2. Information retrieval

Unstructured data

- The generated clues (“computer" and “een computer") are used to formulate a query and submit it to Solr. In this case the query takes the form of `‘computer’ OR ‘een computer’`.
- The result contains documents with titles like: “Computer", “Accumulator (computer)", “Personal computer" and “Schijfloze computer".
- Sentences in each article are ordered by how well they match clues, and the first couple of best scoring ones are kept.

Structured data

- A lookup within DBpedia using the search term “Computer" is carried out, which returns three resources (`Computer`, `Simputer` and `CompuServe`) with decreasing relevance scores.
- Only resources exactly matching a clue are kept, which in this case only leaves `Computer`.

Stage output: Best matching sentences from Wikipedia articles related to computers, and the DBpedia `Computer` resource. For this specific question, one of the sentences to keep an eye on is “Een computer is een apparaat waarmee gegevens volgens formele procedures (algoritmen) kunnen worden verwerkt.", as it will be a prominent candidate further on.

10.1.3. Answer analysis

Unstructured data

- Every candidate answer sentence is submitted to the Alpino parser, similar to the question analysis stage.
  - For each sentence, promising sub-phrases are looked for by checking whether their semantic function matches the expected answer type and whether they are related to the subject or other clues identified in the question.
    - ◇ A prominent candidate answer identified in this way is part of the sentence mentioned in the previous subsection: “een apparaat waarmee gegevens volgens formele procedures (algoritmen) kunnen worden verwerkt". This is because it fits together with the question (“Wat is een computer?", “Een computer is een ...") and because the subject (“computer") occurs in the sentence.
- For each candidate answer analysis results are included - examples of these are which clues it matched, whether it refers to the question subject, etc..

Structured data

- All properties, in the form of `(property name, property value)` pairs, of the `Computer` resource are iterated over but none are selected, since no property name is similar to any of the question clues.
- Similar to candidates from unstructured sources these ones are also annotated, according to criteria like their WordNet scores.

Stage output: A list of annotated candidate answers.

10.1.4. Answer merging

- The main task of this step is to remove duplicates and produce a single list of candidate answers.
- There are no duplicates in this case because 1) on the ‘structured’ side no DBpedia property was selected and 2) on the ‘unstructured’ side every kept sentence was unique.

Stage output: A list of unique annotated candidate answers.

10.1.5. Answer scoring

- First a simple rule-based scorer is applied to the list, based on features like word net scores and whether the answer was a named entity identified in the question.
- The answers are then scored using a random decision forest classifier, pre-trained on a question collection.

Stage output: ordered list of candidate answers with their final confidence scores. For our example, the highest score is assigned to the candidate mentioned in subsection 10.1.3, as it is part of a sentence containing the question subject *and* matches the expected semantic structure.

10.1.6. Response generation

Output answers together with their confidence scores, ordered descendingly on confidence score.

Stage output: List of answers and their confidence scores. The highest scoring answer is indicated below.

Answer: “een apparaat waarmee gegevens volgens formele procedures (algoritmen) kunnen worden verwerkt"

10.2. Confirmation question

Question: “Kan de Nao dansen?"

10.2.1. Question analysis

Confirmation questions are detected by looking for Alpino's *sv1* annotation. This stands for *verb initial sentence* and is the annotation Alpino attaches to confirmation questions and imperative sentences.

In this case, *sv1* is present among the annotations and we mark the question as a confirmation question. Next come two specific checks mandated by the project scope: confirmation questions are only supported if they are about a capability of the Nao. We therefore first check whether “Nao" is the subject of the question, and then whether the lemma of the main verb is “can".

Stage output: All output that was also created in the case of the factoid question, plus an annotation indicating that it is a confirmation question.

10.2.2. Information retrieval

Confirmation questions are only supported if they are about a Nao capability., and we store Nao capabilities in RDF format in a structured source. This means that, in case of a valid confirmation question, we can skip searches in unstructured sources.

Stage output: No results from unstructured sources, so we continue only with DBpedia resources and properties. Querying DBpedia takes place in the same way as for factoid questions.

10.2.3. Answer analysis

In contrast to factoid questions, here we try to match with property *values* instead of property *names*. Too see why, consider the following example:

Factoid question: “Wat is de prijs van de Nao?"
Strategy: Look for a property with name *prijs*, and use its *value*.

Confirmation question: Kan de Nao dansen?
Strategy: Look for a property with value *dansen*, and check whether its *name* is relevant.

With factoid questions we don't know the target piece of information and need to find it. Here, with confirmation questions, we already have that information and need to check whether it's actually true.

Stage output: There does exist a property with value *dansen* and its *name* (“capability"), being a perfect match with the question verb, is relevant. The output therefore contains this property and has a high score attached to it.

10.2.4. Answer merging

The tasks in this stage proceed in the same way as for the factoid question.

Stage output: A list of unique annotated candidate answers.

10.2.5. Answer scoring

The tasks in this stage proceed in the same way as for the factoid question.

Stage output: An ordered list of candidate answers with their final confidence scores

10.2.6. Response generation

We use a cutoff score for formulating the answer. If the highest scoring property has a score higher than 2 we return 'Ja', and 'Nee' otherwise. This threshold is chosen based on the range of similarity scores: WordNet scores are in a range of about $[1, 3.5]$, so a score of 2 can be seen as representing a relevance of more than 50%.

Stage output: 'Ja'.

Answer: "Ja"

Part IV

Evaluation

11

Quantitative evaluation

In this chapter we present the results of the quantitative evaluation of the QA system. We will be answering our fifth research question:
RQ 5: *Which evaluation metrics best serve to assess the effectiveness and efficiency of the QA system, given its domain and type of usage?*

The aspects we focused on during this evaluation were whether the system can find correct answers, how high it ranks them and how quickly it returns with responses.

To better position our system in the space of existing QA systems, we compare it to two alternatives: the original (English) version of YodaQA and Google. We chose the first to assess the impact of the porting effort to Dutch, and the second because it is the most prominent of available web search engines.

The next sections describe the sources and questions used for the evaluation, which performance measures we used, and the experimental setup and results. Throughout this chapter *YodaQA* refers to our system and *YodaQA (English)* to the original version.

11.1. Research questions

The experiment is focused on evaluating the performance of the system, looking for any emerging patterns, and comparing it to alternative systems. We define the following research questions to guide us during the evaluation, together with the method of data collection to be used for each one:

1. *How does the QA system perform in terms of finding correct answers, ranking them highly, and doing this quickly?*
   Measurement: running system on test questions and evaluating its performance using various metrics.
2. *Are there topics or question types the QA system is good at or struggles with?*
   Measurement: looking for results that deviate, either from other categories of from expectations.
3. *How does the performance of the QA system compare to the original (English) version?*
   Measurement: comparing the collected results for the different systems.

11.2. Sources

The data sources used for the experiment are the following:

- A Solr instance with a Dutch Wikipedia dump containing `6879` documents. This collection originated from a full Wikipedia dump with almost 2 million documents but was filtered to only keep articles related to topics like computers, robots and computer science. This was done by using a set of keywords and removing articles not containing any of these keywords.
- A Fuseki instance with a Dutch DBpedia dump containing `80 million` RDF triples. This dataset was not filtered, because filtering it is not as straightforward as for the Wikipedia articles.

Using the same approach as for Solr would likely delete many relevant resources, as DBpedia resources typically contain much less text and sentences than Wikipedia articles. It is therefore less suited to simple text filtering and the ontology it is a part of would need to be considered as well. Moreover, their relevance to the topics we are interested in might even be indirect (it might extend some matching resources but not be it itself), so filtering them is more challenging.

11.3. Questions

The contents of the collection of test questions used for the evaluation was influenced by a number of factors:

School survey: One inspiration was the results of a survey taken at a Dutch secondary school early on in (but not as part of) this project, where students were asked about the questions they would like to ask the Nao. The survey showed that the students were primarily interested in attributes and capabilities of the robot.
QA system purpose: Another factor is the goal we had in mind when building the system. The goal was not to have a QA system that can answer any and all types of questions about any topic. Instead, we developed a system that is more technology oriented and knows about robotics, computers and related topics - the sources the system is configured to use reflect this.

Since we carry out the evaluation to assess how well the system can formulate responses based on the data it has access to, the test questions should, given these sources, be answerable. This means that (most of) the questions should be ones for which answers are present in these sources. It is possible to deliberately include some unanswerable questions to see how the system handles such cases. However, the focus remains on whether the system can find relevant data, recognize it as such and use it in its response.

The collection that was used contains 136 questions spread over four question categories: *definition* and *factoid* questions about general topics, and *factoid* and *confirmation* questions about the Nao robot specifically. This collection is included in appendix F.

11.4. Performance measures

We analyze the system's performance using four metrics: *recall*, *MRR (mean reciprocal rank)*, *precision* and *answering time*. For a single question, these have the following meaning:

- **Recall**: 1 if the correct answer occurs anywhere in the returned list and 0 otherwise. It is important to note that, for virtually all used questions, there is only one correct answer that can be found; this is the reason why we only consider two values for this metric.

  Its *recall* score can be seen as the first hurdle for the system: can it locate the correct answer in the first place?
- **MRR**: the rank of the correct answer in the list of returned answers.

  This represents the second step the system needs to take: is it able to recognize that the candidate is a fitting answer and rank it highly?
- **Precision**: 1 if the correct answer was in first place and 0 otherwise. This is especially useful to investigate with the deployment setting in mind, when the system will output only the top answer from its list of answers.

  A high score for this metric means that the system was able to take the third and final step: identify the candidate as *the* correct answer and assign it a top ranking.
- **Answering time**: this is the number of seconds the system took to return with a response after submitting a question.

11.5. Experimental setup

11.5.1. System specifications

The experiment was run on an Asus N551JQ-CN045H laptop computer, with an Intel Core i7-4710HQ (2.5GHz, 6M Cache) processor and 8GB of DDR3 RAM. The operating system of the computer was Ubuntu 16.04 LTS.

11.5.2. YodaQA

For the experiment the output format of the QA system was adapted to also include the time taken for each question. At the start of the evaluation, first a random question was submitted as part of the startup process.

We do not evaluate the system's performance for first questions, because the system needs to load certain resources on its first run and therefore takes longer. Afterwards we proceeded with the actual evaluation by inputting the questions one by one, and noting the returned response and time taken.

11.5.3. YodaQA (English)

The same approach as for *YodaQA* was used for the English version, with the additional step of translating all questions to English before submitting them.

11.5.4. Google

There are some differences to take into account when comparing YodaQA to Google.

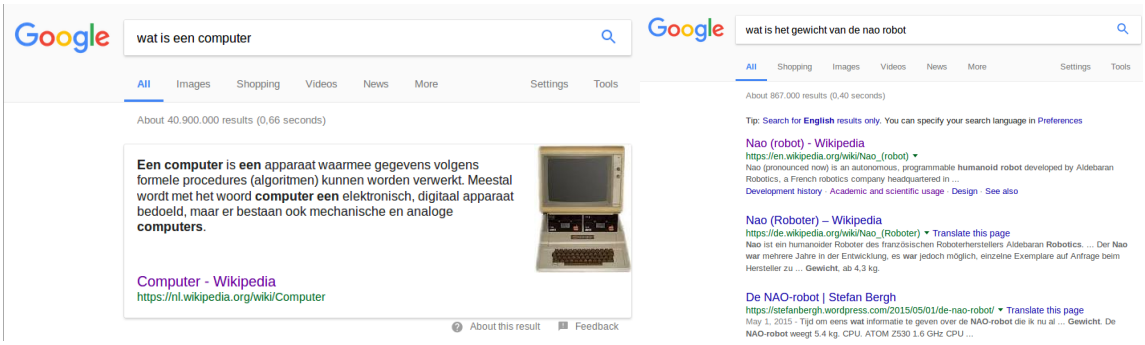
First of all, measuring answering time is not very informative as this is almost always less than a second. Given the huge storage and computational capacity available to it, together with the indexing/preprocessing work carried out, this is an area where YodaQA cannot compete against Google.

The second is that we can distinguish between three different types of responses from Google:

1. The answer is present in a separate box on top. This means that the relevant information was found *and* recognized as the result best fitting the search query.
2. The answer is present among the returned results. This means that the relevant information was found but not explicitly recognized as the answer.
3. The answer was not found.

Figure 11.1 illustrates the difference between the first two cases. On the left we see the box containing the answer; on the right, the answer can be seen at the bottom of the image as part of the third result.

This difference is significant as it touches upon one of the important distinctions between information retrieval systems and question answering systems. QA systems go further than just returning a relevant passage and narrow it down to the exact desired bit of data: the *answer* to the given question. As part of our experiment, we will therefore also note for what percentage of questions *case 1* applied.



(a) (b)

Figure 11.1: Case 1 and case 2

The metrics we introduced above for YodaQA keep the same meaning for the case of Google:

- $Recall$ = 1 if case 1 or 2; 0 otherwise. As long as the answer is *somewhere* in the response, it counts.
- MRR = rank of the result containing the answer.
- $Precision$ = 1 if case 1 *or* case 2 with the answer included in the top result; 0 otherwise. For the *precision* score we do not distinguish between cases 1 and 2; just like for YodaQA, $precision$ = 1 if the answer is part of the first result.

11.6. Hypotheses

Before carrying out the evaluation we can formulate certain hypotheses based on the insights we have about the systems:

1. Low answering times for confirmation questions about the Nao, because in this case Wikipedia searches are skipped and only DBpedia is queried.
2. Lower answering times for factoid questions about the Nao than for general questions, because the occurrence of the *Nao* token acts as a filter and reduces the amount of content to analyze.
3. For YodaQA, definition questions should have higher MRR and precision scores, because definitions can often be found explicitly and at the beginning of Wikipedia articles.
4. Better scores than Google for questions about the Nao, because part of our sources have been tailored specifically for this purpose.
5. Poor performance for YodaQA (English) for questions about the Nao, as this is a very specific topic for which its sources have not been optimized.

11.7. Results

The scores given in table 11.1 are averages over the given number of questions, as specified in column two. A precision score of 48%, for example, means that for 48% of the questions the correct answer was put in first place.

Table 11.1: Averaged scores for YodaQA

Question type	#questions	Recall	MRR	Precision	Answering time (seconds)
General, definition	50	78%	3.72	46%	17.33
General, factoid	40	70%	3.11	47.50%	18.43
Nao, factoid	22	100%	4.77	59.09%	9.74
Nao, confirmation	24	86.96%	1	86.96%	1.14

Table 11.2: Averaged scores for YodaQA (English)

Question type	#questions	Recall	MRR	Precision	Answering time (seconds)
General, definition	50	16,00%	13.63	0%	9.07
General, factoid	40	62,50%	2.64	35,00%	8.82
Nao, factoid	22	0%	N/A	0%	6.99
Nao, confirmation	24	N/A	N/A	N/A	N/A

Table 11.3: Averaged scores for Google

Question type	#questions	Recall	MRR	Precision	Answering time (seconds)
General, definition	50	100%	1.30	80%	< 1
General, factoid	40	76,92%	1.32	56,41%	< 1
Nao, factoid	17	25.29%	3.33	5.88%	< 1
Nao, confirmation	24	87.50%	1.48	66.67%	< 1

Note: performance for *Nao, confirmation* questions was not measured for *YodaQA (English)* because it does not support this type of questions. Also, because *recall* was zero for *Nao, factoid* questions, *MRR* is undefined for this category.
Note: answers from Google were evaluated leniently. For example, in response to the question whether the Nao could dance, the top result (which we counted as an answer) was a video where a Nao could be seen and the title included 'dancing'. We did not specifically require explicit formulations of answers.

The first result to note is that, in contrast to *hypothesis 3*, the average *MRR* score for definition questions is not higher than the one for factoid questions. This seems to be caused by a suboptimal ranking of the candidate answers, because the recall is significantly higher for definition questions. Answers are found more often than for factoid questions, but not always ranked very highly.

We can get a better idea of the distribution of *MRR* and *answering time* scores by plotting them in histograms. Figures 11.2 and 11.3 show these histograms for *MRR* and *answering time*, respectively.

In figure 11.2 we can see that, for each question type, in the majority of cases the correct answer is ranked at or close to the top. Figure 11.3 shows that, for 'general' questions, the system typically takes between 5-25 seconds. Questions about the Nao are answered faster, taking around 10 seconds. This is in line with

hypothesis 2 made above. Finally, confirmation questions are very fast, not taking more than a few seconds. This also is in agreement with *hypothesis 1*.

Figure 11.4 visualizes the results presented in the tables above, by showing comparisons between the three tested systems for all question categories. A separate plot is used for each performance measure. Higher bars are better for *recall* and *precision* and lower bars are better for *MRR* and *answering time*.

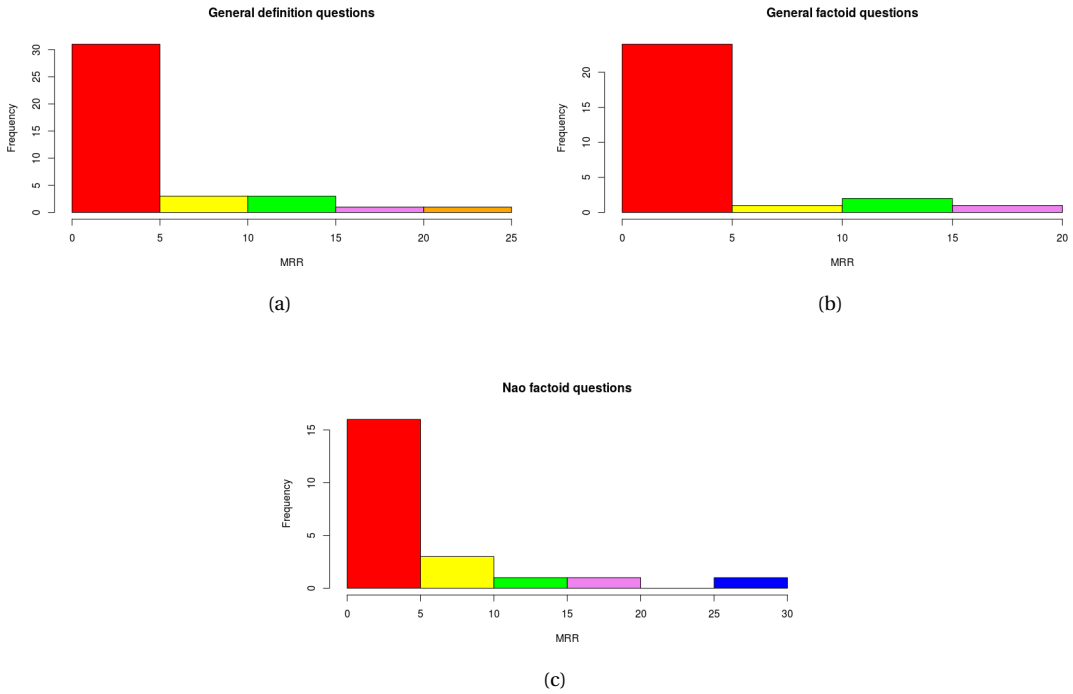


Figure 11.2: Histograms for YodaQA MRR scores

11.8. Conclusion

In this chapter we presented the results of the quantitative evaluation of our QA system. We created a collection of test questions and used it to evaluate the performance of three systems across various measures.

Returning to our first research question about the performance of the system, we can conclude that in the majority of cases (at least 69% on average) YodaQA could find the correct answer and, on average, rank it in the top 4 of its returned answers. As expected, recognizing it as *the* correct answer represented a bigger challenge and it could manage this in slightly less than half the time.

But perhaps its most important shortcoming, especially when compared to the tested alternatives, was its running time. For general questions the average response time was close to 20 seconds; for questions about the Nao this dropped to almost half of that, but was still 10 seconds.

Our second question was about particular topics or question types that stand out in the results. The *Nao, confirmation* stands out positively in this regard with high scores for all metrics. To an important extent this is caused by this being a quite restricted category, as it included only questions about the Nao and then only about capabilities of the Nao. Data required to answer it was placed only in DBpedia as well, allowing us to skip Wikipedia searches for these questions and achieve a speedup in this way.

On the other hand, the *General, definition* category stands out negatively. Not because the scores are particularly bad in themselves, but because we would expect them to be higher given that they are easier than factoid questions. After all, definitions are often present explicitly in Wikipedia articles and are often even the first sentence of the article. The *recall* score is decent but suboptimal ranking of the retrieved answer occurs in a high proportion (32%) of cases.

Comparing our system with the two tested alternatives to answer our final question, we see that it performed

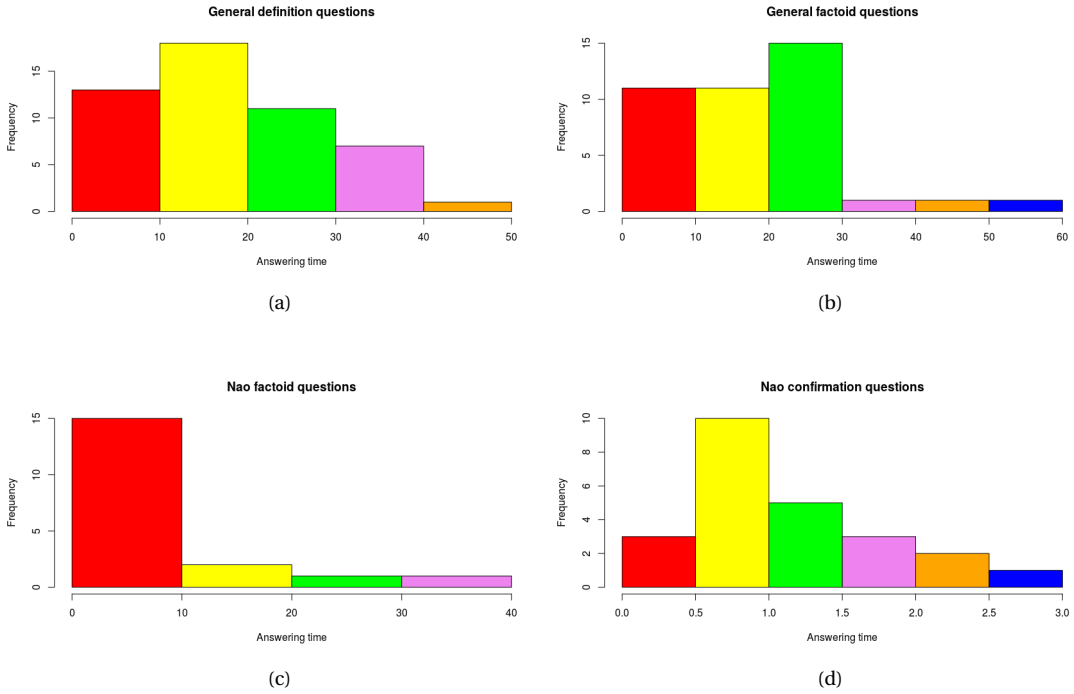


Figure 11.3: Histograms for YodaQA answering times

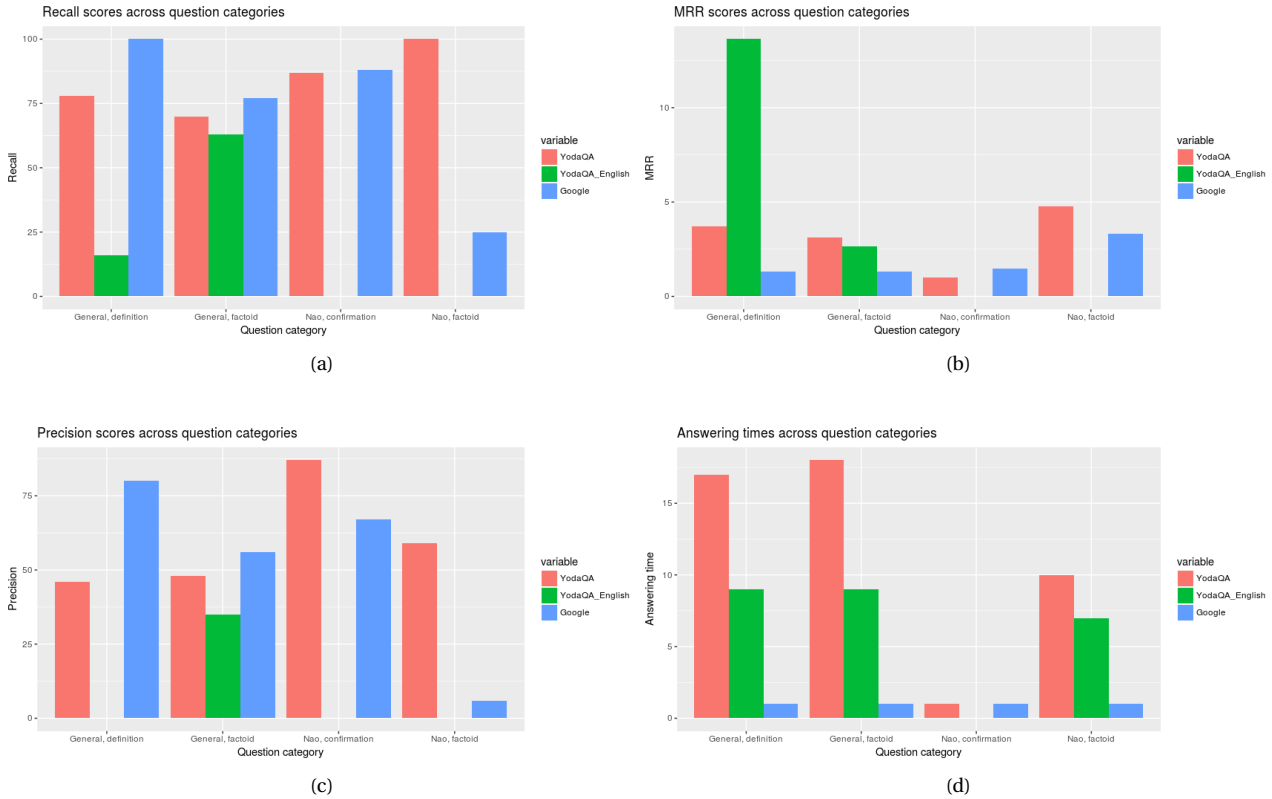


Figure 11.4: Comparison of all systems across all metrics

better than both for questions about the Nao. This is because the sources have been customized for this purpose and is among the main advantages of our system: the sources can be adapted as desired to accommodate new information.

The performance of *YodaQA (English)* was close to *YodaQA* regarding *general, factoid* questions. As is to be expected, neither system can compete with Google regarding to response time due to the large computational and storage resources available to it.

12

Field study

All chapters from chapter 3 onwards have focused on the technical aspects of the project, where we have looked at the underlying technology and implementation. The last chapter presented the results of the quantitative evaluation, carried out internally. With all of this behind us, it is time to return to the original purpose of the project and the more 'human' side of it all: using the QA system in front of an audience of children, deployed on the Nao robot. This chapter discusses the field study that was carried out to evaluate the system in such a setting. It reports on the work carried out to answer our sixth and final research question:
RQ 6: *What type of experimental setting and which measurements allow us to evaluate the effect on the audience of having the RoboTutor hold a Q&A session after a presentation?*

Our focus during this evaluation was on the way children approach such a system, what they focus on when using it, and on the effect the robot has on them when it is interacting with them and answering their questions. For this purpose, we planned an evaluation at a day care center and allowed children to have Q&A sessions with the Nao. The next sections describe the experiment and its results in detail.

12.1. Research questions

This evaluation serves to answer our final research question about evaluating the effect of the QA system on users and its performance in a live session with users.

1. *Which question types do users prefer?*
   Same considerations apply as discussed in previous chapter about the contents of the test collection. On the one hand the choice of questions will be constrained by the system's capabilities. As only questions about the topics it supports will have any chance of success, the choice will be restricted to the topics and questions the QA system can handle.

   Still, the system supports more than one topic and question type and so there is a choice. Users can choose between the four question types introduced in the previous chapter: *general-definition*, *general-factoid*, *nao-factoid* and *nao-confirmation*.
   Measurement: logging the questions asked for each category and determining the frequency for each category.
2. *How are the QA system's response times perceived?*
   Do the intermediate progress updates given by the robot serve to fill up this time and influence this perception? Does it matter if the system is noticeably slow or fast?
   Measurement: asking about users' perception in a survey.
3. *Are the QA system's responses understood?*
   This is related to the age of the participants, which is in the range of 8-13 years. The system can return perfect answers, but they might be understandable only to users with more knowledge about computers.
   Measurement: asking about users' perception in a survey.
4. *Do users feel that the QA system found fitting answers?*
   Note that this is different from the question of whether the QA system *actually* finds fitting answers,

which was part of the focus of the previous chapter. This question is about the users' *perception* of the system's accuracy. This might be influenced by factors such as the robot's way of presenting information.
Measurement: asking about users' perception in a survey.

5. *What is the effect on users if the QA system cannot find an answer or gives a wrong answer?*
Do users lose interest? Do they care, or do they continue as if nothing happened?
Measurement: observation of users and taking notes by the research leader (the author of this thesis).
6. *What effect do details like users' age/gender/tech exposure have on their perception of the system and the questions they ask?*
Are there any significant differences between measurements related to the previous questions that correlate with differences in user attributes like age and gender?
Measurement: correlating survey answers and observations with the acquired demographic data.

12.2. Measurements

Different techniques were used to collect the data needed to answer our research questions. The three methods we used were the following:

- **Logs**: Logging functionality was added to the interaction manager to record the questions asked, the answers given and the response times.
- **Survey**: A survey, included in appendix G, was given to participants after finishing their Q&A sessions with the robot. The intention behind it is to determine users' perception of the system and its performance; it includes questions related to the system's speed, clarity and accuracy.
- **Observation**: The research leader was present during the individual Q&A sessions to record data not captured by other tools, using observation and written notes. This includes body language and any comments users might have during the session, but also any input by the participants before and after the Q&A sessions.

12.3. Preparations

Administrative

The experimental setup was first submitted for review to the TU Delft Human Research Ethics Committee. The application was approved and the study could go ahead. We next contacted three day care centers to introduce the project and ask whether they would be interested to participate in an evaluation. The evaluation was planned at the first center that replied.

About two weeks prior to the evaluation, the concrete setup to be used was discussed in person at the location with the manager of the center. We handed the center the informed consent forms to be sent to the parents of the children wanting to participate. The manager also requested that the session not be photographed or filmed because of privacy requirements.

Experimental

A sample list of questions was created, intended to be given to the participants before the individual Q&A sessions. Children could select from these questions and come up with their own if they wanted.

We knew from previous studies involving presentations given at schools that this audience is especially interested in the robot itself and tends to ask questions about its attributes and capabilities; this is why such questions are well represented in the sample list of questions to be presented to the participants in the study.

We also created a survey to be handed out to the children after finishing their Q&A session with the robot. We paid attention to set up these up in such a way as to stimulate children to think and make distinguished choices, rather than, for example, choose 'agree completely' all the time. Answers to multiple-choice questions were shuffled and they were textual rather than on a linear scale.

The setup for Q&A sessions we settled on was individual rather than in groups. This was so as to focus solely on the effect on the user and his/her approach to it. In this way we could avoid any potential group effects, disorder or different behavior due to being seen and heard by peers.

Technical

Next, the interaction manager described in chapter 8 was implemented. It was extended with logging func-

tionality, and given the capability to introduce itself to each user by asking for his/her name. A graphical user interface was added to it to make it more user-friendly.

12.4. The experiment

Introduction

Nine children were present at the start and would be participating in the whole study. The research leader kicked off the session by explain the contents of the session. The audience was once again informed that participation is voluntary and that everyone could withdraw at any point, without having to provide a reason and without any further consequences.

Nao demonstration

The Nao started with a brief speech, introducing itself and the session. It continued with a short dance; this is what it is best known for and is always a hit with children. The robot then concluded by explaining the setup for the Q&A session itself: that it would be inviting the participants one by one and that they could then ask their chosen questions.

Preparation for Q&A sessions

After the Nao's demonstration, a list of sample questions was distributed to give the audience an idea about which question types and topics the system can handle. The questions were drawn from the same four categories introduced in chapter 11: definition and factoid questions about general topics, and factoid and confirmation questions about the Nao.

The children were instructed to select a maximum of ten questions from this list, which they would then ask the Nao. They were also told that, if they wanted, they could come up with their own questions as well.

During this time the Nao was set up in a separate room for the Q&A session. It was connected to the laptop on which the QA system was running. This was the same laptop as the one used for the quantitative evaluation.

Q&A session

The participants were invited one by one to the separate room. Beside the child and the robot, there was also an employee from the center present for guidance and the research leader for technical assistance.

The Nao first asked for the child's name, which was used to greet the child and occasionally include it in some of the upcoming verbal progress updates. After that a loop was entered in which the participant submitted questions to the QA system by typing them in, and the system then returned with an answer, which was output both verbally and on the laptop screen.

After a participant was finished with the questions it was given the survey. After filling in this survey, the participant was finished with his/her part of the evaluation.

12.5. Results

12.5.1. Logs

In total, the participants asked the Nao 70 questions of which 5 were invalid due to typos. Of the remaining 65, 24 were questions that had already been asked by an earlier participant. 32 of the 41 valid and unique questions were answered correctly, giving a *precision* score of 78.05%.

The category from which questions were chosen most often was *general-definition* (as introduced in chapter 11): 25 of the 70 questions were of this type. This was followed by *general-factoid*, *nao-factoid* and *nao-confirmation*, respectively.

The lengths of the individual Q&A session were almost all in the range of 4.5-6.5 minutes, with one outlier of 8 minutes. Answering times ranged between 1-39 seconds. Confirmation questions were all answered in less than 2 seconds, just as in the quantitative evaluation. The average response time was 11.34 seconds.

Figure 12.1 shows these results graphically: *a*) shows the distribution of session lengths, in *b*) we see how many questions from each category were asked and *c* is a histogram of answering times.

12.5.2. Survey results

The survey we used is included in appendix G. We collected seven valid filled in surveys. The possible answers to the multiple-choice questions in the survey did actually follow a linear scale, even though this was

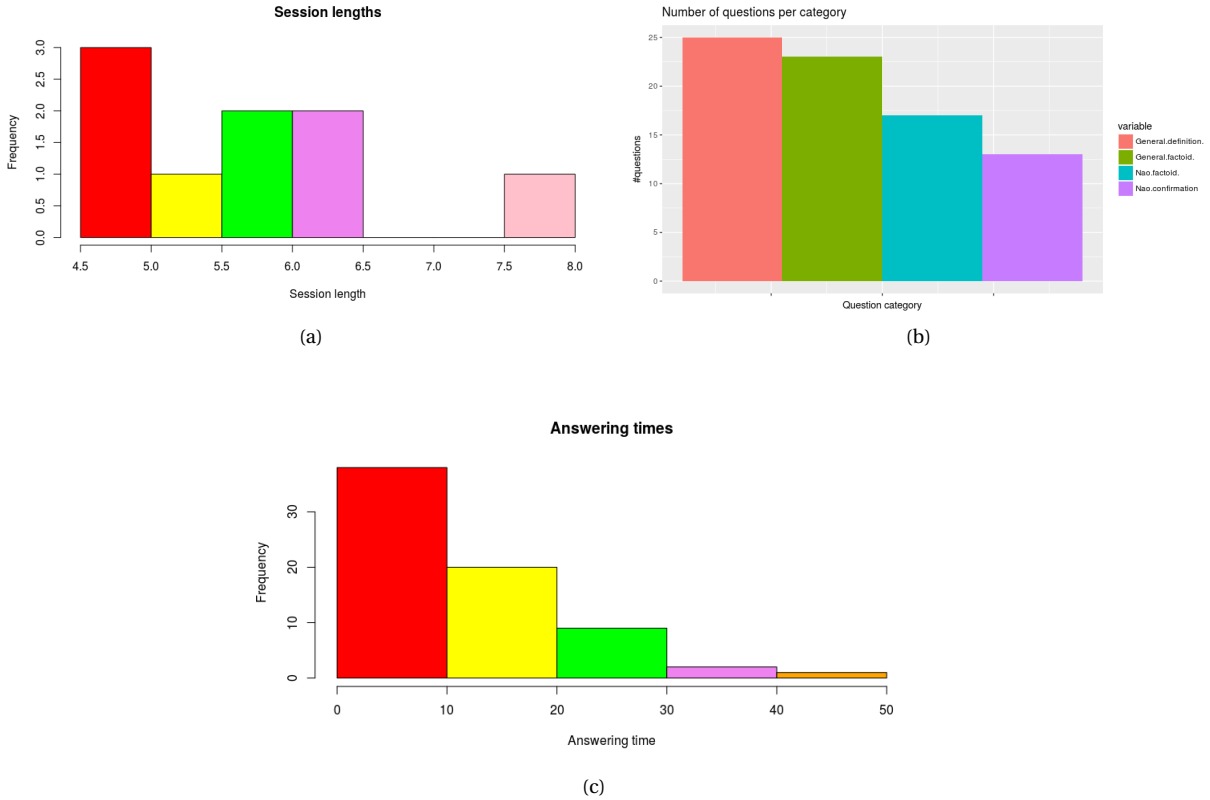


Figure 12.1: *a*) **Q&A session lengths,** *b*) **questions asked per category and** *c*) **distribution of answering times**

deliberately obscured. In the discussion below, '1' means 'disagree completely' (for example, 'No, the Nao was not fast at all') and '4' means 'agree completely' (for example, 'Yes, the Nao was very fast').

Answering speed received an average score of **2.00**. It was perceived as decent most of the time. One participant chose confirmation questions for more than half of his questions, and this was the only participant that gave a 1 to the Nao's speed. Confirmation questions were all answered in a few seconds and this factor is likely to have motivated this assessment.

Also interesting to note is that the participant whose questions took the longest, accurately assessed the robot's speed with a 3. This was the only 3 received for speed.

Answer clarity received an average score of **1.67**. Clarity was ranked highly most often and this was the question which received the highest number of 1's. Only once was a 3 given and three times a 1.

Answer quality was ranked least favorably and received an average score of **2.33**. It received the highest number of 3's out of all three criteria. The times the robot gave a clearly wrong answer or failed to find an answer clearly did not go unnoticed.There was one outlier worth mentioning, however: 6 of the 7 questions of one participant were answered correctly, but this didn't prevent **answer quality** from being given a 3.

The Nao did not receive a 4 on any criterion by any participant. Perceptions of answer quality and correctness seemed to vary the most. On the one hand, some participants did assess the system accurately and gave it a high score if it managed to return proper responses often. On the other hand, some participants agreed with the statement that 'the Nao had trouble with quite a number of questions', even if it had managed to answer the majority of their questions.

Two participants had had previous exposure to technology: they had created games before. Neither of them asked a single question about a capability of the Nao. However, it is noteworthy that the only participant who remembered an answer given by the robot during the Q&A session, was one of these two participants with previous technology experience.

To summarize: the participants thought that the Nao's answers were understandable most of the time. They were a bit more critical about its speed, especially those children who had not asked any confirmation questions. The correctness and relevance of the answers was assessed worse than the other two criteria.

12.5.3. Observations

What was noticeable during the study was that the answers and their correctness was not cared about very much. The participants mainly wanted to continue with the next questions and keep the interaction with the Nao going. Even if some of the answers given by the robot were clearly wrong, this was not perceived as an issue. This is further supported by the fact that, after the sessions, almost no one could remember an answer by the robot anymore.

Long answering times were noticed, especially when the (randomly selected) progress updates happened to be repeated successively. We can mark this as a suggestion for improvement: only choose those progress updates which haven't been used yet.

The audience had a definite interest in the capabilities of the Nao. This was clear already at the very beginning of the whole session, even before the introduction by the Nao, from the questions directed at the research leader. Given this, it is remarkable that this was the question category used the least for the individual Q&A sessions. Definition and factoid questions related to computers and robots were asked more frequently than questions about the capabilities, or even attributes, of the Nao.

12.5.4. Correlation with demographic data

If we look for correlations between the results and the demographic data about participants, we see that the results are quite balanced. There were three female and six male participants, all in the age range of 7-10 years. Both female and male participants asked questions from all categories. Exactly one female and one participant gave the Nao more than one 1's. The only result that could be noteworthy is that none of the girls gave the Nao a 3 on any criterion, in contrast to three boys that did give at least one 3.

12.6. Conclusion

In this chapter we reported on the field study of the QA system, running on the Nao robot, that we carried out in a day care center together with nine elementary school age Dutch-speaking children. The study consisted of an introduction given by the Nao, after which the participants could ask it questions they had selected in individual sessions with the robot.

For the children, the Q&A session were more about the interaction with the robot than about the information it was providing. Although they felt they could follow the Nao and perceived it as relatively fast, they did think (correctly) that it had trouble with certain questions. This did not detract from the overall experience, but long answering times and the occasional repetitive output were seen as minor stumbling blocks.

Regarding users' preference for question categories, we see that questions about general topics were more frequent than questions about the Nao. The participants were interested in all topics the Nao knew about. Besides the consistent interest of the target audience in the capabilities of the Nao, they also showed interest in general topics like robots, computers and technology in general by asking many questions from these categories.

Response times were decent on average and some, especially in case of confirmation questions, were quite fast. Its speed is one of the most obvious aspects of the Nao's performance and very easy to assess and notice.

Answer clarity was rated highest and answer quality the lowest. On occasion, clearly wrong answers were given and these of course did not go unnoticed. Still, the users were quite forgiving and mistakes did not detract much and users still wanted to interact with the robot.

Correlation with demographic data did not reveal any special results. In any case, the users were quite similar to each other in terms of age and previous technology exposure.

Part V

Closing

13 Discussion

13.1. Quantitative evaluation

The answering time of our QA system was the highest of all tested systems. The main bottleneck responsible for this is the wait for responses to the requests sent to Alpino. There is a possibility for improving this in the future: a planned Alpino enhancement involves adding the possibility to submit all inputs at once instead of one by one.

Even without this improvement, the QA system and all required tools would benefit from being deployed on a cloud computing service. It would need one with at least 8, preferably more, gigabytes of RAM. However, only 'micro' instances with about 1-2GB of memory are free of charge; these are of course far below even the minimum, let alone recommended, requirements for our QA system.

The results show a definite improvement over YodaQA (English) for the topics and question types we tested our system with, especially in the area of answering definition questions using Wikipedia. This can be seen primarily in the scores for *general, definition* questions.

In a sense, this is closely linked to the above point of long answering times. We use Alpino and NLP techniques to match candidate answers from Wikipedia, and we pay for this in the form of longer answering times.

YodaQA (English) does come close for *general, factoid* questions. This is because structured data sources are a better fit for factoid questions than for definition questions, and the English DBpedia is much more comprehensive than the Dutch version. YodaQA (English) is therefore able to make good use of the English DBpedia for answering questions of this type.

There is a 25-35% difference between *recall* and *precision* scores. This means that in 25-35% of cases the system finds the correct answer but fails to give it a top ranking. It is not that big of an issue as the average MRR score is around 3-4, so correct answers are still ranked fairly highly.

A possible way forward in this area is improving the matching and scoring algorithms. Another is to investigate training and applying a model-based classifier, like the decision forest used in the English version.

In histograms we see that MRR scores are heavily concentrated on the lower end. Answering times, on the other hand, show quite some variance, especially for general questions. There is an inverse relationship present here: if the question is a simpler one about which a lot can be found, the system will take longer because of this abundance.

The same pattern is not visible for questions about the Nao. When asking questions of this type it is mostly the same resources that will be retrieved: the ones about the Nao. Since it is the information retrieval and analysis phases that dominate running time, most questions about the Nao have similar answering times.

It is clear that our system cannot compete with Google when it comes to speed. On the other hand, a crucial feature of our system is that it returns exact answers instead of relevant passages within documents. This admittedly is not that difficult an addition to make, but it still counts as an extra feature of our system. After all, there are a number of relatively easy additions/improvements our system could make use of as well.

The main advantage of the system we have built and evaluated is that it is a Dutch QA system with customizable data sources. While data used by web search engines is out of our control, we can customize the sources for our QA system as required. Additionally, the way is clear for extending it with more capabilities: adding support for list questions, answering confirmation questions for questions other than about Nao capabilities, using machine learning classifiers detecting question types and scoring candidate answers and much more.

13.2. Field study

The number of questions asked was highest for the category *general-definition*. This category was followed by *general-factoid*, *nao-factoid* and *nao-confirmation*, respectively. There might be some influence because of the way the questions were ordered on the sheet given to the participants, which matched this ordering exactly. Then again, there was sufficient variation between individual participants: a number of participants did ask more confirmation questions than other types of questions, for example.

What is interesting to note is that the only participant who both remembered an answer and had previous non-trivial technology exposure (creating a game), was also the participant who had the longest session and asked the most questions. It is logical to assume a relation between tech exposure and an interest in and desire to interact with a robot, but the number of participants is still too low for this observation to be significant statistically.

Answering speed was important to the users and significant differences in speed were in fact noticed. Whereas on criteria like clarity and quality participants at times tended to give 'wrong' assessments, their assessments of answering speed consistently matched the system's actual performance. Very fast sessions were given high scores and slower sessions were evaluated accordingly.

An explanation is that it is the easiest metric of the three to evaluate: it could be that you don't know whether the answer is correct or not, that you don't know whether it could be formulated better, but you will know whether it took a short or long time.

The interaction with the robot had a higher priority for the users than focusing on the answers it provided and trying to assess its quality. Most of the time, participants wanted to move on without pause to the next question, regardless of whether it was a fitting answers or clearly incorrect. This means that it is important to be aware of this 'wow-effect' of the Nao, and see to it that it does not obscure the (lack of) quality of its answers and detract from its learning function and educational purpose.

14

Conclusion

The aim of this project was to extend the interactive capabilities of the RoboTutor Nao by enabling it to answer natural language questions in Dutch. To this end, we investigated various QA systems and settled on an open-source QA system for English: YodaQA.

Our **first three research questions** were about adapting the question answering pipeline of the chosen system to work with Dutch content, and the data sources and natural language processing tools required for this. To this purpose, we first adapted its natural language processing framework by integrating Dutch NLP modules in place of the English ones, like the Alpino dependency parser and Apache OpenNLP components for Dutch. We next configured sources for it: Wikipedia as the unstructured source and DBpedia as the structured source.

Additionally, we gave the system extra flexibility by using the Open Dutch WordNet to determine word pair similarities. We also implemented mechanisms for matching answers from unstructured sources using two approaches: by looking for passages semantically fitting in with the question, and by looking for grammatical structures relevant for the inferred question type.

With the implementation finished, we turned towards the two evaluations we wanted to carry out to assess its performance: an internal, quantitative evaluation and a field study. Our **fifth research question** focused on the quantitative evaluation and on how to assess the performance of the QA system. We created a set of test questions, selected a number of performance measures, and used these to evaluate the performance of both our system and two others for comparison. The measures were selected to provide insights into whether the system was able to find answers, rank them highly, recognize them as the best answer and do this quickly.

For the questions we evaluated with, correct answers were found in at least *two-thirds* of the cases, they were ranked in the *top five* of returned answers, and they were ranked first in about *half* of the cases. Answering time was the area with the most room for improvement: this was about 18 seconds on average for general questions and 10 for questions about the Nao.

Our **fourth and sixth research questions** focused on the field study and the interaction manager needed to structure the interaction with users. We integrated the QA system to run via the Nao. An interaction manager was implemented that interacted with users according to a turn-based protocol and used prompts and updates for a smoother interactive session.

Next, we contacted and planned an evaluation at a day care center, where nine elementary school age children would be participating. We then took the Nao to the center and gave the participants the opportunity to ask the Nao questions in individual Q&A sessions.

About 60% of the questions the children asked were answered correctly. They were interested in all question categories the Nao supported and all four categories were well represented in the questions they chose to ask.

One of the main deliverables of this project is a public, open-source, Dutch QA system capable of using both unstructured and structured sources for answering natural language questions about various topics, including robots, computers and the Nao robot.

The second deliverable is a framework for running the QA system via the Nao using an interaction manager. This is a tool that runs separately from the system and serves as a wrapper around it, handling the interaction with users. It uses prompts to ask users for questions, confirms their input, fills the waiting time with various updates and returns with the output of the QA system.

Finally, we have also reported on our experimental study into the effects of having the Nao robot hold individual Q&A sessions with elementary school age children. Just like in the quantitative evaluation, slow answering times did represent a bit of a hindrance. Still, this study showed that this audience is eager about interacting with the Nao and interested in its attributes and capabilities, as also reflected in the questions they selected for their Q&A sessions. Just as in the quantitative evaluation, however, slow answering times did detract somewhat from the overall experience of the participants. Many questions were answered quickly, but the slow ones did indeed stick out and influence the assessments of the users after the sessions.

Nevertheless, interaction was smooth, users were interested in many topics the system supported and they continued interacting with it despite the occasional mishap. Overall, the results indicate that, with some speedup, the delivered QA system would be a suitable choice for answering questions about the mentioned topics.

14.1. Future work

There is a lot of room for further enhancements to the systems developed as part of this project. Below we present some ideas that can improve the performance of the QA system, make the interaction manager better or allow us to gain a better understanding of their effects on users:

- Extending support for confirmation questions to topics other than the Nao and its capabilities.
- Investigating upcoming versions of the Alpino server, which promise a potential performance improvement by allowing bulk submission of all input.
- Investigating machine learning classifiers for answer scoring to see how they compare to the currently used rule-based scorer and whether they bring an advantage.
- Extending the Dutch word net with custom entries to stimulate higher scoring of word pairs we know are closely related but which currently receive low scores. This problem currently causes correct answers to be ranked too low, especially in the case of DBpedia properties.
- More powerful and flexible answer matching algorithms. One idea is to also match sentences if the question subject does not occur explicitly but implicitly in it - for example, the whole paragraph or article might be about the subject.
- Adding a command processing functionality to the interaction manager. This is to be able to give a ‘demo’ where applicable in response to questions like “Can you do this?" to the Nao, instead of just responding “Yes" and then doing nothing.
- Enhancing the interaction manager by having it give more relevant progress updates that actually have a bearing on the progress so far, instead of providing generic update messages.
- Carrying out a field study in a group setting in addition to the individual evaluations we carried out. In this project we first wanted to focus on the effects of the system on users without any interferences from others. Now that this is done it would be useful to evaluate it with multiple users simultaneously, especially because this is the way the system would be used in a role of classroom assistant.

A

Product specification

A.1. Purpose

The aim of the delivered system is to provide a supporting role in education by assisting teachers in presenting course contents and other relevant material. The system also enables interaction with audiences after presentations, where they can ask questions about the discussed material.

A.2. Audience

The primary audience of the system consists of Dutch-speaking children of elementary school age. The size of the target audience is around 20 and consists of children in the age range of 8-12. One or more teachers will also be present when using the system.

A.3. Scope

System description: A natural language question answering system embodied in the Nao robot.
Existing systems: The Nao robot has a number of modules, among which are modules for presentation, text-to-speech and movement. It can currently carry out presentations autonomously by following a predefined script.
Deliverables: The QA system is the main deliverable, running separately but communicating with the Nao platform.

The second important deliverable is the interaction manager: a 'wrapper' around the QA system that structures the interaction with it. A very simple example could be a script that requests a question, passes it to the QA system, returns its answer to the user and request a new one.

A.4. Requirements

A.4.1. Interface

1. **Input**

   Language The language of the questions should be Dutch; questions in other languages will not be supported.

   Format While asking questions to the Nao using speech is a desired feature, it is not yet implemented and outside the scope of this project. Questions will therefore be entered into the system textually.

   Types The types of questions supported by the system are *definition*, *factoid* and *confirmation* (yes/no) questions. The latter are supported only if they are specifically about capabilities of the Nao robot.

2. **Output**

   Language The system will return answers retrieved from Dutch sources; these will therefore be mostly Dutch. However, any terms and passages in other languages within these sources can still be part of an answer, if they are assessed as relevant by the QA system.

Format Answers should be given to users using speech. The existing text-to-speech module of the Nao can be used for this purpose, as Dutch is one of its supported languages.

A.4.2. Sources

1. **Language**
   The QA system will be making use of knowledge sources containing data in Dutch. Sources in other languages will not be consulted.
2. **Types**

   Natural language Sources of this type are important as they enable as to include free form data that can be collected from various sources (Wikipedia, news sites, research papers, etc.).

   Structured Sources of this type are useful for data that can be modeled more easily in a structured form.
3. **Domain**
   Data contained within the sources should be about the Nao, robotics and other, closely related topics like computers, software, hardware and technology.
4. **Availability**
   Since timeliness is important the sources should be available for offline access. In this way, the system will avoid delays incurred by navigating the web for every question.
5. **Update mechanisms**
   Mechanisms should be in place that enable *CRUD* (create, read, update and delete) operations on the sources. This is especially important because the sources are stored offline, and are therefore not by default up-to-date with the latest available data.

A.4.3. Technical

1. **Performance**
   The system will be operating in real time in front of an audience. Relatively short response times are therefore crucial for a smooth user experience. If necessary a certain degree of completeness and accuracy can be sacrificed, if in return we can make the system significantly faster while maintaining decent answer quality.

   In concrete terms, the aim should be to return with a response in less than 30 seconds.
2. **Robustness**
   Exceptions and crashes must be anticipated in advance: mechanisms should be in place that handle such situations gracefully and without alerting users to them.

A.4.4. Hardware

1. **Connectivity**
   Because of resource constraints the QA system will not be running on the Nao itself. The robot therefore needs to be connected to the system for submitting questions to and retrieving answers from the system, which will be running on a different machine.
2. **Screen**
   The Nao is connected to a screen on which it can display its answers.

B

External resources

- YodaQA repository (`https://github.com/brmson/yodaqa`)
  The original, English version of YodaQA.
- Dutch YodaQA repository in RoboTutor space (`https://github.com/RoboTutor/yodaqa`)
  The Dutch version of YodaQA created as part of this project.
- Wiki with information and installation instructions (`https://github.com/RoboTutor/yodaqa/wiki`)
  Contains information about installing and configuring all components required to run the Dutch YodaQA.
- Interaction manager (`https://github.com/S-Ercan/InteractionManager`)
  The interaction manager developed for the QA system, as described in chapter 8.
- Nao demo (`https://github.com/S-Ercan/NaoDemo`) The Nao demo project used during the field study.

C

Project timeline

This appendix gives an overview of the main tasks carried out during each month of the project. The mentioned tasks were not the only tasks in the given month, as there were also meetings and the writing of parts of this report. Many tasks also extended into the next month, or had to be picked up again at a later point in the project.

2016	
February	Start of RoboTutor team meetings
March	Project plan, project proposal
April	Literature review
	Investigation of available technologies
May	Deciding on YodaQA; installing, configuring and testing
June	Setting up Dutch Wikipedia data
July	Integrating TreeTagger
	Refactoring QA logic for Dutch
August	Setting up Dutch DBpedia data
September	Integrating Open Dutch Wordnet
	Colloquium presentation
October	Integrating the Alpino parser (question analysis)
November	Integrating the Alpino parser (answer analysis)
December	Implementing dependency extraction from parse trees
2017	
January	Further review of relevant literature and previous studies
February	Implementing answer matching for unstructured sources
March	Adding support for confirmation questions
April	Adapting answer scoring and response generation
May	Quantitative evaluation of QA system
	Integrating QA system as Nao module
June	Implementing interaction manager
	Field study
July	Analysis of results, discussion and conclusion
August	Processing feedback and finalizing report
	Thesis defense

D

Contributions

D.1. Contributions

D.1.1. An open-source Dutch QA system

The main contribution of this project is the creation of a public, open-source Dutch QA system. As noted in chapter 2 there have been some earlier initiatives in this area, but they have all been decommissioned and are not available anymore. No publicly available open-source natural language question answering system for Dutch could be found while investigating the available options; the work done during this project can therefore be seen as filling this gap.

Concretely, the following was achieved in the process of porting YodaQA to Dutch:

- Natural language processing
  - Integrating the Alpino dependency parser as an NLP tool for generating dependency, lexical category and part-of-speech annotations
  - Adding support for detecting and processing confirmation questions
- Question-answer matching
  - Two parallel approaches for matching questions with candidate answers from unstructured sources, making use of the *expected answer type* requirement
- Open Dutch WordNet
  - Estimating token similarity while matching tokens from both unstructured and structured sources
- Sources
  - Integrating Dutch Wikipedia as an unstructured data source
  - Integrating Dutch DBpedia as a structured data sources

D.1.2. Evaluation

The field study featured a unique experimental setting involving hosting a QA session about robotics by the Nao robot, with elementary school age children in a day care center interacting with the robot.

E

Tasks

E.1. Sources

Wikipedia

- Set up Apache Solr server
- Import Dutch Wikipedia dump into Solr server
- Filter out articles not related to QA system domains
- Extend with article about Nao

DBpedia

- Set up Apache Jena database
- Import Dutch DBpedia dump into Jena database
- Set up Apache Fuseki server hosting the Jena database
- Add resource about Nao, fill in with attributes and capabilities

Other

- Set up local DBpedia label lookup service
- Extend with Nao label
- Set up Open Dutch WordNet

E.1.1. Natural language processing

TreeTagger

- Integrate TreeTagger into question analysis phase
- Refactor question analysis logic to work with TreeTagger

Alpino

- Create UIMA type systems for Alpino *POS*, *category* and *dependency* annotations
- Configure Alpino to run as server
- Create client within QA system to communicate with Alpino server
- Integrate Alpino into question analysis and answer analysis stages, making them work with Alpino output
- Integrate Saxon library for extracting dependency relations from Alpino parse trees

E.1.2. QA logic

- Refactor answer matching logic for data from unstructured sources
- Implement two parallel approaches
  - Start with question predicate and follow it in the answer
  - Start with grammatical structure implied by expected answer type and assess its relevancy
- Add support for confirmation questions about Nao capabilities
  - Detect confirmation questions using Alpino annotation
  - Extend pipeline to handle confirmation questions
- Adapt answer scoring algorithm

F

Test questions

Table F.1: *General, definition* and *general, factoid* questions

Definition

Wat is een computer?
Wat is een robot?
Wat is programmeren?
Wat is een database?
Wat is een besturingssysteem?
Wat is een harde schijf?
Wat is werkgeheugen?
Wat is een sensor?
Wat is een personal computer?
Wat is software?
Wat is hardware?
Wat is een printer?
Wat is een scanner?
Wat is PHP?
Wat is RAM?
Wat is ROM?
Wat is een CD-rom?
Wat is een CPU?
Wat is een processor?
Wat betekent LCD?
Wat is een actuator?
Wat is een bit?
Wat is een byte?
Wat is een programmeertaal?
Wat betekent SQL?
Wat is een computervirus?
Wat betekent USB?
Wat is een modem?
Wat is een router?
Wat betekent LAN?
Wat is een diskette?
Wat betekent hacken?
Wat is een tekstverwerker?
Wat is de informatica?
Wat betekent WAN?
Wat is een grafische kaart?
Wat betekent kunstmatige intelligentie?
Wat is een neuraal netwerk?
Wat is een mainframe?
Wat is de Turingtest?
Wat is een chatbot?
Wat is een expertsysteem?
Wat is een zoekalgoritme?
Wat betekent broncode?
Wat is een debugger?
Wat is software engineering?
Wie is de ontwikkelaar van PHP?
Wie is de oprichter van Microsoft?
Wie is de ontwikkelaar van Windows?
Wat is de geboortedatum van Bill Gates?
Wat is de geboortedatum van Steve Jobs?
Waar is Microsoft gevestigd?
Waar is Apple gevestigd?
Hoeveel gebruikers heeft Facebook?
Wie is de eigenaar van Facebook?
Wie bouwde de eerste computer?
Hoeveel werknemers heeft IBM?
Wie is de eigenaar van Google?
Wanneer is Google opgericht?
Wat is de rekencapaciteit van de K Computer?
Wanneer werden de eerste supercomputers gebouwd?
Wie is de bekendste bouwer van mainframes?
Hoe heet een regel van een assemblyprogramma?
Wat is het voordeel van programmeren in assembly?
Wie is de ontwikkelaar van Java?
Wanneer werd de eerste versie van JavaScript ontwikkeld?
Hoe heette de eerste browser?
Wat was het eerste spreadsheetprogramma?
Wanneer kwam versie 1.0 van Word uit?
Wanneer werd Windows 2.0 gelanceerd?
Wanneer werd Windows Phone 8 gelanceerd?
Hoe worden smartphones vaak bediend?
Hoe groot is de virtuele adresruimte van een moderne microprocessor?
Wat is het type kernel van Linux?
Wat is de website van IBM?
Hoeveel werknemers heeft Twitter?
Wat is de slogan van Twitter?
Wat is de status van Hyves?
Wie is de ontwikelaar van Internet Explorer?
Wat is de website van Microsoft Edge?
Wat is de typesystem van PHP?

Table F.2: *Nao, factoid* and *Nao, confirmation* questions

Factoid	**Confirmation**
Wat is het gewicht van de Nao?	Kan je op je handen staan?
Hoe zwaar is de robot?	Kan je schelden?
Hoeveel kost de robot?	Kan je presenteren?
Wat is de prijs van de Nao?	Kan je dansen?
Wie heeft de Nao ontwikkeld?	Kan je zingen?
Wie is de ontwikkelaar van de Nao?	Kan je spreken?
Waar ben je van gemaakt?	
Wat is de gebruiksduur van de Nao?	
Hoe heet jij?	
Hoeveel weeg je?	
Wat is jouw naam?	
Hoe oud ben jij?	
Wat is de CPU van de Nao?	
Welk besturingssysteem gebruikt de Nao?	
Hoe lang ben je?	
Wat is de lengte van de Nao?	
Hoe hoog is de Nao?	
Wat is de hoogte van de Nao?	
Waar woont de Nao?	
Wat is de woonplaats van de Nao?	
Van welk materiaal ben je gemaakt?	

G

Survey

Tot slot, een paar vraagjes...

1. **Ik heet...**

2. **Heb je eerder met computers of robots gewerkt? Zo ja, wat heb je er bijvoorbeeld mee gedaan?**

3. **Wat herinner je je nog van de Nao's antwoorden? Kan je een paar dingen noemen?**

4. **Hoe snel was de Nao?**

- ◯ Het had best een stukje sneller gekund.
- ◯ Af en toe moest de Nao wel even denken, maar snel was hij wel.
- ◯ Ik viel haast in slaap, zo traag was het.
- ◯ De Nao was erg vlot met zijn antwoorden.
- ◯ Anders, namelijk:

5. **Was de Nao duidelijk met zijn antwoorden?**

- ◯ Alles wat de Nao zei kon ik wel volgen.
- ◯ Hij gebruikte veel woorden die ik niet ken.
- ◯ Het meeste van wat de Nao vertelde was te begrijpen.
- ◯ Anders, namelijk:

6. **Wist de Nao veel?**

- ◯ De robot kon niets beantwoorden.
- ◯ De meeste vragen lukten wel.
- ◯ Er waren toch best wat vragen waar hij moeite mee had.
- ◯ Op elke vraag had de Nao wel een antwoord.
- ◯ Anders, namelijk:

Bedankt!

Bibliography

[1] Interactive Intelligence Group | Interactive Intelligence . http://ii.tudelft.nl/, 2016. [Online; accessed 16-12-2016].

[2] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: the view from here. *natural language engineering*, 7(4):275–300, 2001.

[3] Oleksandr Kolomiyets and Marie-Francine Moens. A survey on question answering technology from an information retrieval perspective. *Information Sciences*, 181(24):5412–5434, 2011.

[4] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[5] Dan I Moldovan, Sanda M Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Girju, and Vasile Rus. Lasso: A tool for surfing the answer net. In *TREC*, volume 8, pages 65–73, 1999.

[6] William A Woods and R Kaplan. Lunar rocks in natural english: Explorations in natural language question answering. *Linguistic structures processing*, 5:521–569, 1977.

[7] Anne R Diekerma, Ozgur Yilmazel, and Elizabeth D Liddy. Evaluation of restricted domain question-answering systems. 2004.

[8] Hui Yang, Tat-Seng Chua, Shuguang Wang, and Chun-Keat Koh. Structured use of external knowledge for event-based open domain question answering. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 33–40. ACM, 2003.

[9] John Burger, Claire Cardie, Vinay Chaudhri, Robert Gaizauskas, Sanda Harabagiu, David Israel, Christian Jacquemin, Chin-Yew Lin, Steve Maiorano, George Miller, et al. Issues, tasks and program structures to roadmap research in question & answering (q&a). In *Document Understanding Conferences Roadmapping Documents*, pages 1–35, 2001.

[10] David Ferrucci, Anthony Levas, Sugato Bagchi, David Gondek, and Erik T Mueller. Watson: beyond jeopardy! *Artificial Intelligence*, 199:93–105, 2013.

[11] Natural Language Classifier | IBM Watson Developer Cloud. http://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/nl-classifier.html, 2016. [Online; accessed 28-03-2016].

[12] Retrieve and Rank | IBM Watson Developer Cloud. http://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/retrieve-rank.html, 2016. [Online; accessed 28-03-2016].

[13] Petr Baudiš. Yodaqa: a modular question answering system pipeline. In *POSTER 2015-19th International Student Conference on Electrical Engineering*, pages 1156–1165, 2015.

[14] brmson/yodaqa: A Question Answering system built on top of the Apache UIMA framework. https://github.com/brmson/yodaqa, 2016. [Online; accessed 22-03-2016].

[15] Petr Baudiš and Jan Šedivỳ. Biomedical question answering using the YodaQA system: Prototype notes. 2015.

[16] Petr Baudiš and Jan Šedivỳ. Qald challenge and the YodaQA system: Prototype notes.

[17] Open Advancement of Question Answering System by OAQA. https://oaqa.github.io/, 2016. [Online; accessed 22-03-2016].

[18] Zi Yang, Niloy Gupta, Xiangyu Sun, Di Xu, Chi Zhang, and Eric Nyberg. Learning to answer biomedical factoid and list questions OAQA at BioASQ 3B. In *Working Notes for the Conference and Labs of the Evaluation Forum (CLEF), Toulouse, France*, 2015.

[19] Nico Schlaefer, Petra Gieselmann, Thomas Schaaf, and Alex Waibel. A pattern learning approach to question answering within the ephyra framework. In *Text, speech and dialogue*, pages 687–694. Springer, 2006.

[20] Edgard Marx, Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Conrad Hoffner, Jens Lehmann, and Soren Auer. Towards an open question answering architecture. 2014.

[21] Christof Monz and Maarten de Rijke. Tequesta: The university of amsterdam's textual question answering system. In *TREC*, 2001.

[22] Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the international conference on new methods in language processing*, volume 12, pages 44–49. Citeseer, 1994.

[23] Valentin Jijkoun, Gilad Mishne, and Maarten de Rijke. Preprocessing documents to answer dutch questions. In *Proceedings of the 15th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'03)*, volume 434, 2003.

[24] Valentin Jijkoun, Gilad Mishne, and Maarten de Rijke. Building infrastructure for dutch question answering. *DRIPROCEEDINGS*, page 22, 2003.

[25] Gosse Bouma, Jori Mur, Gertjan Van Noord, Lonneke Van Der Plas, and Jörg Tiedemann. Question answering for dutch using dependency relations. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 370–379. Springer, 2005.

[26] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *ACL (1)*, pages 956–966, 2014.

[27] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007.

[28] Michael A Covington. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer, 2001.

[29] Roger W Brown. Linguistic determinism and the part of speech. *The Journal of Abnormal and Social Psychology*, 55(1):1, 1957.

[30] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, pages 466–471, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[31] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[32] George A Miller. Dictionaries in the mind. *Language and cognitive processes*, 1(3):171–185, 1986.

[33] About WordNet - WordNet - About WordNet. `https://wordnet.princeton.edu/`, 2017. [Online; accessed 23-01-2017].

[34] Christiane Fellbaum. A semantic network of english: the mother of all wordnets. In *EuroWordNet: A multilingual database with lexical semantic networks*, pages 137–148. Springer, 1998.

[35] Francis Bond and Kyonghee Paik. A survey of wordnets and their licenses. *Small*, 8(4):5, 2012.

[36] Marten Postma, Emiel van Miltenburg, Roxane Segers, Anneleen Schoen, and Piek Vossen. Open dutch wordnet. In *Proceedings of the Eight Global Wordnet Conference, Bucharest, Romania*, 2016.

[37] GitHub - cltl/OpenDutchWordnet: This repo provides a python module to work with Open Dutch WordNet. It was created using python 3.4. `https://github.com/cltl/OpenDutchWordnet`, 2017. [Online; accessed 09-08-2017].

[38] Chih-Wei Chang, Jih-Hsien Lee, Po-Yao Chao, Chin-Yeh Wang, Gwo-Dong Chen, et al. Exploring the possibility of using humanoid robots as instructional tools for teaching a second language in primary school. *Educational Technology & Society*, 13(2):13–24, 2010.

[39] Barry Fagin and Laurence Merkle. Measuring the effectiveness of robots in teaching computer science. In *ACM SIGCSE Bulletin*, volume 35, pages 307–311. ACM, 2003.

[40] YodaQA in non-English languages · Issue #45 · brmson/yodaqa. `https://github.com/brmson/yodaqa/issues/45`, 2016. [Online; accessed 27-04-2016].

[41] Dialog | About Dialog | IBM Watson Developer Cloud. `https://www.ibm.com/watson/developercloud/doc/dialog/index.html`, 2017. [Online; accessed 09-08-2017].

[42] Apache UIMA - Apache UIMA. `https://uima.apache.org/`, 2016. [Online; accessed 04-10-2016].

[43] Apache Solr. `http://lucene.apache.org/solr/`, 2016. [Online; accessed 04-10-2016].

[44] Apache Jena - Home. `https://jena.apache.org/`, 2016. [Online; accessed 04-10-2016].

[45] Apache Jena - Apache Jena Fuseki. `https://jena.apache.org/documentation/fuseki2/`, 2016. [Online; accessed 16-12-2016].

[46] Apache OpenNLP - Welcome to Apache OpenNLP. `https://opennlp.apache.org/`, 2016. [Online; accessed 04-10-2016].

[47] The Stanford Natural Language Processing Group. `http://nlp.stanford.edu/`, 2016. [Online; accessed 04-10-2016].

[48] Welcome to DKPro. `https://dkpro.github.io/`, 2016. [Online; accessed 16-12-2016].

[49] DBpedia. `http://wiki.dbpedia.org/`, 2016. [Online; accessed 04-10-2016].

[50] brmson/label-lookup: Stand-alone service for fuzzy lookup of string labels of resources. `https://github.com/brmson/label-lookup`, 2016. [Online; accessed 04-10-2016].

[51] Open Dutch WordNet. `http://wordpress.let.vupr.nl/odwn/`, 2016. [Online; accessed 04-10-2016].

[52] WordnetTools | CLTL. `http://www.cltl.nl/results/software/wordnettools/`, 2016. [Online; accessed 04-10-2016].

[53] Alpino. `http://www.let.rug.nl/vannoord/alp/Alpino/`, 2016. [Online; accessed 04-10-2016].

[54] The SAXON XSLT and XQuery Processor. `http://saxon.sourceforge.net/`, 2016. [Online; accessed 08-12-2016].