Arttu Heikkilä

NATURAL LANGUAGE PROCESSING TECHNIQUES IN CHATBOT DEVELOPMENT: HOW DOES A CHAT-BOT PROCESS LANGUAGE?

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2020

ABSTRACT

Heikkilä, Arttu
Natural Language Processing In Chatbot Development: How Does a Chatbot Process Language?
Jyväskylä: University of Jyväskylä, 2020, 72 pp.
Information System, Master's Thesis
Supervisor: Kyppö, Jorma

Chatbots are an extremely prominent way to interact with a software system. The need to build maintainable that scalable systems is more present than ever, while the understanding of needed technologies is generally lacking. This is demonstrated by disconnected literature, the popularity of oversimplified building tools, and generally sub-par conversational agents. This provides a need to understand, and educate, how chatbots are built. This narrows down the gap between theory and practice to an applicable format, where a software developer could have a better stance at building maintainable conversational systems.

This thesis studies the underlying techniques and technologies that go into chatbot development. A case study is presented with source code to explore, to understand the somewhat hidden structures that go into understanding the user input. A literature review precedes a detailed view of the technologies in a real-life example.

Contrary to popular perception, this type of artificial intelligence is not complicated. A modern chatbot uses multiple different components to achieve bot scalability and performance. However, a lot of these technologies are fairly easy to understand and debug to a professional in the technical field. A chatbot processes the input text through NLP-techniques and assigns it to a predefined intent through a classifier. Another classifier is then used to determine proper actions, be it a response or custom software. Understanding this type of pipeline can prevent technical overhead when fixing issues built on a black box.

Keywords: Chatbot, Conversational Agent, Natural Language Processing, Natura Language Understanding, Rasa

TIIVISTELMÄ

Heikkilä, Arttu
Luonnollisen kielen prosessointi chatboteissa: Kuinka chatbot prosessoi kieltä?
Jyväskylä: Jyväskylän Yliopisto, 2020, 72 s.
Tietojärjestelmätiede, Pro Gradu-tutkielma
Ohjaaja: Kyppö, Jorma

Chatbotit ovat yleistyvä ratkaisu ihmisen ja tietokoneen väliseen vuorovaikutukseen. Tarve rakentaa ylläpidettäviä ja skaalautuvia keskustelevia ratkaisuja on kasvava, mutta ymmärrys perustavanlaatuisista teknologioista tarpeeseen on vähäistä. Tätä näkökulmaa tukee vähäinen kirjallisuus, yksinkertaistettujen alustaratkaisujen yleisyys, sekä ala-arvioisten chatbottien yleisyys. Tämä luo tarpeen ymmärtää ja kouluttaa, kuinka botit ovat pohjimmiltaan rakennettu. Tämä vie tarvittavan teorian lähemmäksi käytäntöä, joka tukee botin kehittäjää rakentamaan ylläpidettävää ja skaalautuvaa arkkitehtuuria.

Tämä tutkielma tarkastelee niitä perustavanlaatuisia teknologioita ja konsepteja, jotka saavat tietokoneen ymmärtämään ihmistä. Tapaustutkimusta hyödynnetään ymmärtämään yksityiskohtaisesti, kuinka jokseenkin piilotetut tekstiprosessointitekniikat saavat chatbotin ymmärtämään puhetta. Tapaustutkimusta pohjustaa kirjallisuuskatsaus tekstiprosessoinnista ja niiden yhtymisestä moderneihin chatbotteihin.

Yleisestä näkemyksestä tekoälyyn poiketen, chatbotit eivät ole monimutkaisia järjestelmiä. Moderni chatbot käyttää monia eri tekstiprosessointi ja koneälytekniikoita skaalautuvuuden ja suorituskyvyn saavuttamiseksi. Nämä ovat kuitenkin suhteellisen yksinkertaisia, ja tekninen ammattilainen pystyy helposti tekemään korjauksia ja muokkauksia näihin. Chatbot prosessoi käyttäjäsyötteen luonnollisen kielen prosessoinnin tekniikoilla, ja luokittelija luokittelee tuloksen ennalta määriteltyyn 'tarkoitukseen'. Toinen luokittelija antaa jatkotoimenpiteet, oli se sitten vastaus tai jokin muu. Tämänkaltaisin arkkitehtuurin ymmärtäminen tukee bottikehitystyötä, kun ongelmia tai tarpeita uusille ominaisuuksille ilmenee.
Avainsanat: Chatbot, keskusteleva agentti, Luonnollisen kielen prosessointi, Luonnollisen kielen ymmärrys, Rasa

FIGURES

TABLES

TABLE OF CONTENTS

1 Introduction

The general perception of artificial intelligence can be complicated and convoluted, while the idea of imitating human interaction with a computer software can be fairly straight forward. This also applies to the technologies. Various methodologies go into developing systems that are, or appear to be, intelligent. These can range from computationally intensive algorithms to extremely simple programs just used right.

Conversational agents, or chatbots in this thesis, are a form of computer systems designed to communicate with the user in a humanized manner. This is achieved by imitating certain facets of normal communication between people. Simple communication consists of questions asked and responses given both ways, which together form a dialogue between two entities. The most basic idea of a conversational agent therefore is a system that can understand a question and give a sensible answer to said question. This can be imitated in a multitude of different manners, and it proposes various interesting questions in information systems science, and even linguistics.

The aim of this thesis is to explore the question "How does a chatbot process language?". The core interest lies in the area between abstraction layers of naturally produced language and the data held in computer memory to produce dialogue between a user and a software system. What technical actions can or need to be taken to make the user input understandable or calculatable for an inherently deterministic machine.

An oversimplifying answer to the question is natural language processing and machine learning. Language processing to make the input into a readable format, and machine learning to teach the system appropriate responses to selected questions. While this is often, but not always, true in modern chatbots, it leaves a lot to the imagination. What are the exact steps that need to be taken to understand the inner workings of a conversational systems?

Chatbots are starting to appear everywhere. And for every chatbot there is an engineer who build the chatbot. Most often than not, as with any modern software, frameworks and tools are readily available to build them, which partly might explain the surge of simple conversational agents integrated to different systems. These tools enable chatbot development without much, or any, experience in text processing or even software development techniques, and the commercial toolkits pride themselves in being easy to implement. This is enough to a certain degree, but when scalability, optimizations, and fixes become a frequent topic, a deeper knowledge will be beneficial. Does a regular software engineer have the appropriate knowledge on needed topics, especially when commercial chatbot building platforms do not necessarily want to provide information on their technologies? And does a data scientist designing language processing and machine learning algorithms have the needed capabilities to build maintainable and scalable software around it? This issue is very apparent in the literature. There lies a large knowledge gap between technical and theoretical system level research.

Importance of this topic is amplified further by the fact that the techniques and technologies are often not complicated, just hidden. This kind of issue can cause major overhead in development by fixing simple issues with complicated additions. To build better, more maintainable, and more optimized conversational agents, the developer will benefit from understanding the underlying technologies.

This thesis attempts to explore this topic through a case study, and an extensive literature review on the underlying technologies. The case study looks into a real-life example of a domain specific chatbot, with access to source code and appropriate documents. This allows for a detailed look into the technologies and techniques of a chatbot built with modern technology stack.

First this thesis explains the ideas and architecture of chatbots. Certain baseline definitions are given, and simplified architectures presented to gain an initial understanding of the topic at hand.

To narrow down the topic into text processing techniques within chatbots, second part will explain basic natural language processing concepts in depth. Natural language processing is a field of computer science concerned with making naturally produced language to a structured, computer readable form (Chowdhary, 2020). While these techniques were initially designed for use in document processing, the need is very apparent in chatbots as well. Understanding these topics is key to understanding the case study.

Thirdly, the thesis explores the usage of these technologies within chatbots. Core idea of this sections is to learn how these techniques might be implemented into chatbot systems. This means some exploring into intents and entities, as well as dialogue structures. This part also explains some different approaches to chatbots in general. To finalize the literature review and to build a base for a case study, a short exploration into different development suits for chatbots are explored. This explains how they are built based on some of the previous concepts presented in the thesis.

The case study is an effort to explore Laubot, a personal assistant chatbot for financial management team in a municipality's government. Development happens by an external consulting organization Gofore. First the setting is explained in more detail, and right after shifts into the actual review of the system.

The system review is a detailed look into the source code of the system to specifically understand what Laubot does to “understand” the user input. This entails a lot of natural language processing, and additional algorithms do decide the proper actions. An example is presented, and low-level details are extracted about the different steps. This is done along with appropriate documentation related to them.

Finally, an analysis and discussion are presented. This reflects the case study against the literature review and concludes some ways this case study reflects the general architectures and ideas. Short discussion on the topic is presented before conclusion to explore and explain some problems and limitations of this study.

2 Chatbots and QA-systems

Chatbots per se are not a recent technology, but the improvements in usable platforms and advances in natural language processing are making them more prevalent each year (Brandtzaeg & Følstad, 2017). More and more customer service and other simple support tasks are being digitized with digital conversational agents. Use cases, however, are not limited to support tasks. Oftentimes the aim is to increase human-computer interaction by replacing existing systems through a more intuitive user interface. In these cases that means a conversational agent utilizing natural language processing to decipher messages between human and a computer.

2.1 Chatbot

2.1.1 Definition

Conversational agents can take many different forms and functions. These can vary from the earliest conversational systems which were only supposed to achieve communication with natural language between human and computer (Weizenbaum, 1966) to more advanced personal assistants like Apple's Siri (Apple Inc., 2020) and Amazon Alexa (Amazon Inc., 2020). Plethora of different applications and approaches to conversational systems makes chatbots a very general definition. Throughout this thesis the general focus will be on chatbots, conversational agents, and question answering systems. The term chatbot is used to refer to a digital system where a computer holds a conversation between a human user using natural language. Question answering systems are discussed further in chapter 2.2, but simply put they refer to systems designated to answer singular questions, without general attempt to maintain human-like conversation. Conversational agent or conversational system in this thesis are terms used to refer to the general application of conversational interaction with a computer using natural language.

2.1.2 Chatbot architecture

Chatbots generally, but not always, follow a similar architecture to one-another. Jack Cahn (2017) provides a great general explanation of chatbots and their inner workings in their senior thesis. Continuing from his work this thesis aims to present simplified architectural explanation of chatbot functionalities to better reflect on the technical aspects. This is visualized in figure 1. Each layer is presented and briefly explained what their function is. It is important to note that this is a semi-universal representation of the general functionalities and might not reflect actual use cases. Each layer can be designed to fit the purposes of the target system better. This can also be expanded to include more minor parts of a system, as well compressed to match less expansive systems, like question answering systems (Pundge, Khillare & Mahender, 2016).

Table 1: Chatbot general architecture

User interface	The interface the user uses to communicate with the system. This can range from anything from chat software to command line interpreters. Speech-to-text interpreters can be utilized to provide more usable and natural interaction with an artificial intelligence system. Moreover, speech interfaces play a big role in accessibility (Abdul-Kader, & Woods, 2015)
Natural Language Processing and/or intent classification and entity recognition	Natural language processing and understanding is covered more in depth in chapter 2.3. This layer functions as the processing of a given message. There is a multitude of ways to do this, but the aim is to provide machine understandable format of the message, being it an intent or a numerical vector representation of the message. (Shum, He, & Li, 2018; Seneff et al., 1998)
Dialogue management and response generation	Generating conversational responses is key to maintaining human-like interaction with the chatbot. Response generation layer aims to choose an informative and natural response based on the data passed to it. (Xing et al., 2017) This is can be considered the "core" of a dialogue system, as it serves the functionality of conversation. This can be achieved by a multitude of methods, ranging from simple rule-based classifications to extensive machine learning models (Cahn, 2017).
Knowledge management and training	Knowledge base can function as "what the dialogue system knows". Knowledge is stored and can be extended to improve the bot's functionalities. In a domain specific context this can be a combination of database connections and corpora, for example. In a general conversational system this information can be more freely scraped to gather more dialogue options (Huang, Zhou & Yang, 2007).
Dialogue management	This can be done by managing the state of the dialogue, and choosing appropriate actions (end, execute function, response) based on the state of the dialogue. This part also manages the iterative process of conversation. (Bocklisch, Faulkner, Pawlowski & Nichol, 2017)

2.2 Question answering systems

2.2.1 Chatbots or QA-systems?

Question answering system is simpler version of a conversational system, where instead of aiming to hold a conversation, the focus is on answering a question asked in natural language instead of a technical query (Pundge et al., 2016). Chatbots also are often prepared for a multitude of tasks that are not questions necessarily. These virtual assistants work to fulfill the administrative needs of the user instead of just answering questions. While the technology is not deemed mature enough yet, there has also a demand found for emotional support and companionship using chatbots (Zamora, 2017).

While the definite terminology and functionality differs from place to place when it comes to defining conversational systems, some fundamentals are present regardless. Therefore, for this thesis the aim is to maintain general understanding of the usage and technologies in both without distancing them from each other. Moreover, the case study presented later in this thesis lies in the gap between question answering system as well as a functional chatbot, depending how the user decides to interact with it. in short, the definition does not matter in this scenario since the dialogue management is the defining factor. Language processing in larger scale systems usually happens before dialogue management, as it is crucial to "understand" what the user's intention was before taking appropriate actions.

2.2.2 Functions of QA-systems

If the aim is to fetch information from a database utilizing natural language queries or questions, these systems can also be called natural language interface databases, or NLIBD (Androutsopoulos, Ritchie & Thanisch, 1995). These often function in the closed rather than open domain since the data is proprietary and structured. These turn the question asked into a database query and work as an interface for databases. Other question answering systems might gather their data form different sources, such as parsing existing natural language documents and finding question answering paragraphs form there. Oftentimes these function in the open domain by the help of web scraping and existing search engines. (Pundge et al., 2016; Soares, & Parreiras, 2018)

The surge of research and development of QA-systems generally precede the surge of that of chatbots and more sophisticated conversational systems. As with most AI-technologies, the effects of increased computing power might explain some of the shift in focus. Additionally, the interest in commercial applications of chatbots most likely plays a role as well. The research on QA system can be still very useful to gain understanding of certain layers of the presented architecture (Table 1), mainly on the utilization of NLP techniques and NLU-parts of the architecture.

3 Natural language Processing

Natural language processing, also known as NLP, is a research area focusing on processing and structuring natural language to be understandable by computer systems (Chowdhary, 2020). People communicate in natural language, which is called such to make a distinction between other forms of communication within information science. "Language" in itself can refer to multitude of programming and assembler languages in addition to spoken languages. Natural language is still language after being structured but is no longer 'natural' for people to use when talking. Natural language therefore refers to the normal, or natural way, for people to talk to other people.

The aim of conversational systems is to narrow down the interaction gap between human and computer. However, it is important to understand that natural language processing is a much wider paradigm and does not revolve solely around user interaction. For this diversion's sake we will discuss natural language processing completely separately to gain an understanding before exploring natural language understanding and the usage of NLU in chatbots. Moreover, natural language processing is often deemed as artificial intelligence even though majority of applications that merely process natural language to more structured form, can do it on a simple rule-based, deterministic basis.

3.1 Natural language processing research

3.1.1 Early research

Processing natural language is a task that can be done in numerous different ways. Research on different types of processing is not new either, dating back as far as the 1940's. Early work on processing natural language was focused on machine translations from one language to the other. This, however halted for a time due to the difficulty (or rather impossibility) of perfect machine translations. (Liddy, 2001) Few decades later the focus shifted to more theoretical approach, and progress was achieved in representing meaning within natural language and making it computationally traceable. While this was not enough yet to be easily computational, it spawned additional research to represent the syntax and semantics in natural language. (Liddy, 2001) These advances were demonstrated for one with the earliest conversational systems, such as an early chatbot built to replicate conversations with a patient and a psychologist, ELIZA (Weizenbaum, 1966). This early system would just echo the users input with a question formatted from it, like a psychologist would try to reflect on what the patient is telling them. This is done by recognizing keywords and decomposing the input. This would be then recomposed by se reassembly rules. (Weizenbaum, 1966)

In the 1970's the research continued to gain traction on semantic issues, while also dealing with discourse phenomena and communicative goals (Liddy, 2001). Later this work helped to start the research on generative NLP applications. TEXT system presented by McKeown (1982) applied relevancy criteria, discourse structures and focus constraints to generate English text (McKeown, 1982).

These methods and the research in NLP kept gaining more attention later on, due to increase in computational power and the availability of text online. During the end of millennium, real world applications also started to gain ground. (Liddy, 2001)

3.1.2 Different approaches to NLP in the 21st century

While the principles stay the same, the use cases and approaches can vary widely depending on current interests in research, and especially in modern times, in business. Early 2000's and onwards semantic analysis has been in the spotlight due to applicable use cases in business. Sentiment analysis as a business intelligence tool can be applied to a multitude of tasks. Automatically mining text to gain insight on areas like customer satisfaction, competitors, and pricing can be advantageous to utilize. Risk detection different fields, and even election forecasting can be achieved with text analytics. (Sun, Luo, & Chen, 2017) Different social medias in the 21st century has provided previously unimaginable type of data for opinion mining.

3.2 Basic structure of natural language processing pipeline

Most modern applications use similar structure to handle language processing. Be it a chatbot, question answering system or a business intelligence software, the fundamentals stay similar, at least at the beginning of the pipeline. This also represents the definition, or difference, between natural language understanding and natural language processing. A system capable of understanding natural language will most likely *process* the text first similarly to another system, and after preprocessing the *understanding* might happen with completely proprietary method. Natural language understanding is a subtopic in NLP which aims to specify the need to understand the meaning in a text.

One of the most used and referenced pipelines nowadays is the Stanford CoreNLP toolkit (Manning, Surdeanu, Bauer, Finkel, Bethard & McClosky, 2014) which serves as good point of reference for explaining the structure of language processing. This architecture is presented in Figure 1. As the name suggests, this is a toolkit of different techniques of natural language processing. Each does something different from another. Or in most cases, they can be used together to achieve desired results, or a starting point for other analysis. This depends on the use case, and it is crucial to know different technologies and possibilities so one can choose the most beneficial for one's cause. The following segments explain the different methods and how they generally work.

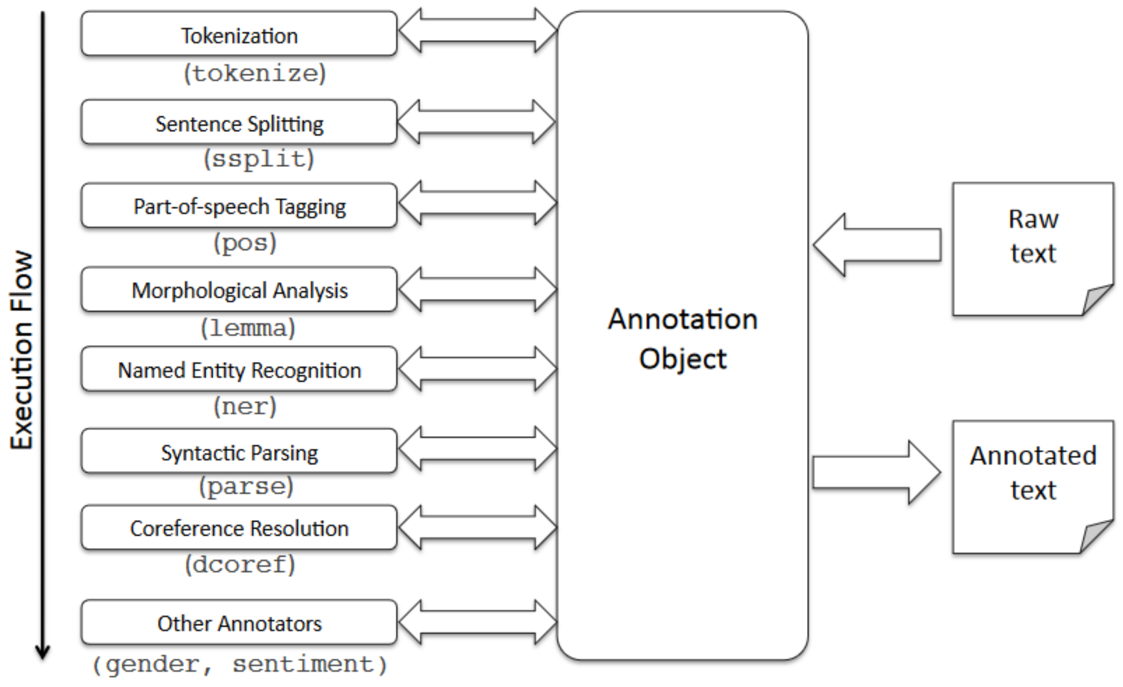


FIGURE 1 System architecture of Stanford CoreNLP pipeline (Manning et al., 2014, p.55)

3.2.1 Tokenization

Tokenization is the 'basis' for most text processing tasks. Tokenizer is a tool which splits the given text into *tokens*. In most language processing tasks, token is often a single word or split from a sentence or a longer text, example shown in Figure 2. What language is being processed determines the tokenization rules. While a language like English can bet tokenized to a degree with just using spaces and punctuations, other languages are not as trivial and require more sophisticated algorithms for tokenization. (Sun, et al., 2017; Straka, Hajic & Straková, 2016) In many cases nowadays, this is done by the help of a treebank, which is a parsed text corpus with different annotations to help structure text. Universal Dependencies (UD) is a project that aims to make treebanks more universal with guidelines and collections on how to represent parsed text. (Nivre et al., 2016). Files built can be used to train neural networks to perform tokenization. Tokenization can also be done to a certain degree with regular expression.

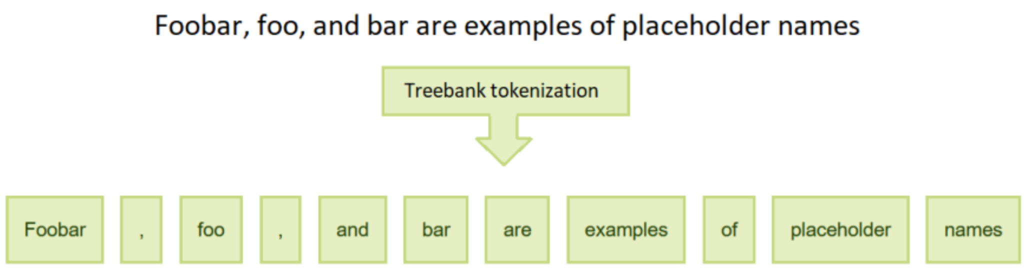


FIGURE 2 Tokenization of an example sentence, where each block represents a single token.

Tokenization does not have an absolute right result. The result can vary a lot between methods used. But it is important to note that tokenization does not provide lemmas or stems, nor does it conjugate verbs.

Sentence splitting is a technique needed in some cases and can be considered type of tokenization. Simply put, it splits individual sentences in a corpus, and results in type of tokens that each represent one sentence. Same principles apply to it as word tokenization, as some languages can be done in simpler ways than others.

3.2.2 Part-of-Speech tagging

Part-of-speech (POS) tagging is the process of categorizing each word or token in a corpus to a certain part of speech, demonstrated in Figure 3. This is an extremely language dependent task since different languages have different parts of speech. Common parts of speeches found in many languages are for example *noun, verb, adjective, pronoun,* and *numeral.* Depending on the language these can extend more above the most common ones.

Part-of-Speech tagging is not as trivial as tokenization. This is because part of speech is ambiguous. This means the part of speech which a single word belongs to can vary depending on the context it is used in. Let us take an example word, *brake.* This can be either a noun or a verb, depending solely on the context. In a sentence *'The car brakes'* the word is used as a verb. However, in a sentence *'The car has brakes'*, while being extremely similar, it is used as a plural noun.

What this means to the methods used is more sophisticated statistical, or machine learning models are used to achieve precise POS tagging. Similarly to tokenization, the UD treebanks aim also to provide data for POS tags.

POS-tagging has also been researched extensively in the past, and different ways of approaching this task has been tried and discovered. Earlier versions from the 1970's approached the issue with rule-based systems with limited results. Later research achieved highly improved results with statistical modelling of the contextual dependencies. More precisely the usage of Markov Model has been prominent in POS-tagging. Later in the research statistical learning, or machine learning models have achieved even better results in POS-tagging. (Schmid, 1994) Further improvements have been found with improved networks (Toutanova, Klein, Manning, & Singer, 2003). Existence of training data has improved and made POS-tagging more available to multiple languages.

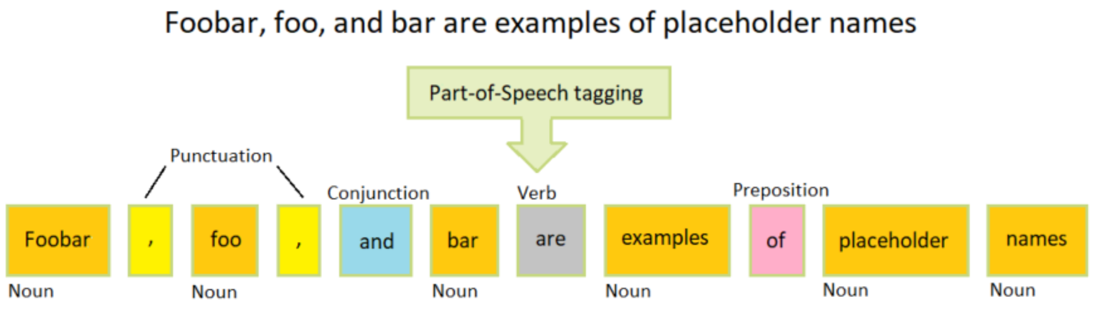


FIGURE 3 Part-of-Speech tagged example sentence

Opposed to tokenization, part-of-speech tagging most often does have a right answer. This means accuracy can be measured more consistently. Current solutions have seen significant accuracy gains in the past two decades, and accuracies of the state-of-the-art POS-taggers hover around 97% accuracy (Plank., Søgaard, & Goldberg, 2016; Toutanova et al., 2003; Schmid, 1994). This however does not mean improvements have not happened, they are just more prominent in efficiency.

3.2.3 Morphological analysis, lemmatization, and stemming

Morphological analysis, or just morphology in linguistics is the study of words and meaning, forming, and relations to other words. In natural language processing this area of language is approached by lemmatization and/or stemming. This is morphological parsing, where individual words are decomposed into *morphemes*. A morpheme is a unit of a word that holds meaning. Most cases it is either an affix (suffix or prefix), or a stem or a lemma. Stem is the part of the word that suffixes and prefixes are attached to. Lemma is form of the word as how it appears in the dictionary, or a 'base form' so to speak. Earlier presented CoreNLP (Manning et al., 2014) produces lemmas, as shown in Figure 1. Lemma and stem are often identical, but not all cases. For example, the word *shave* in the English language would conjugate to forms like *shaves, shaved,* and *shaving*. Therefore, the lemma would be *shave,* while the stem would be *shav* since the affixes are attached to it.

Lemmatization and stemming are forms of data structuring. They normalize word forms, which can be indexed to improve searches and queries. In some languages, for example in a very inflectional language like Finnish, this can make a huge difference in the dimensionalities of structured texts, since a word with multiple different forms can be normalized into a single point of data. (Korenius, Laurikkala, Järvelin, & Juhola 2004). Whether stemming or lemmatization is better for indexing, can depend on the language and the use case. Lemmatization is often ruled based system since languages and grammar tend to work in a rule-based manner in general linguistics.

3.2.4 Named entity recognition

Named entity recognition, or often referred to as NER, is the task of recognizing named entities in a text. This means real world objects like people, places, products, and such. In some definitions named entity recognition also concerns numerical units, like numbers, units, and dates (Nadeau & Sekine, 2007).

Approaches to this problem have varied throughout the years. NER systems have showed a switch from rule-based systems to different machine learning solutions. While rule-based systems get good results, it requires a lot of work on the system, in a problem where domain specific named entities play a huge role. Supervised machine learning methods on the other hand require an annotated corpus as a training data. (Nadeau & Sekine, 2007) In more recent years, bidirectional long short-term memory neural networks have been utilized to gain better results with less training data required (Lample, Ballesteros, Subramanian, Kawakami, & Dyer, 2016).

Named entity recognition has many use cases in domain specific applications, localized searches, and general structuring and annotation. This can further improve the normalization of structured data mentioned in chapter 2.2.3.

3.2.5 Syntactic parsing, or dependency parsing

While the previously presented techniques aim to gather information on singular words out of their context, syntactic parsing is the process of obtaining sentence-level structural information and parsing it to reflect the syntactic structures and dependencies of language. Example of a parsed and annotated sentence is presented in Figure 4. This method aims to gather information of a sentence beyond the content of the words within it. This is a central for applications such as semantic analysis. Sentences containing semantically similar words can mean wildly different things when structured differently within a sentence. (Gómez-Rodríguez, Alonso-Alonso, & Vilares, 2019) Other NLP-applications benefitting from syntactical parsing are for example machine translations and grammar checkers. Syntactic parsing is also more often called dependency parsing, based on the way sentences are parsed. Constituency parsing is another way of parsing syntactic information (Gómez-Rodríguez et al., 2019). Oftentimes syntactic parsing and dependency parsing are used interchangeably.

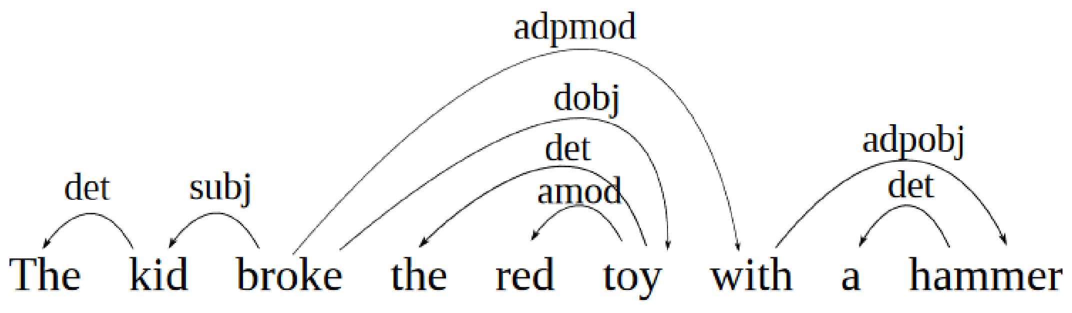


FIGURE 4 Dependency parsing example of a sentence (Gómez-Rodríguez, Alonso-Alonso, & Vilares, 2019, p.4)

This figure shows the grammatical structure of the sentence done by an algorithm. It annotates and gives relations in the form of detonator (*det*), subject (*subj*), adpositional modifier (*adpmod*), adpositional object (*adpobj*), adjectival modifier (*amod*), and direct object (*dobj*). The format these are presented in follow the CoNLL-U format of universal dependencies, mentioned in chapter 2.2.1 (Nivre et al., 2016).

3.2.6 Coreference resolution

In natural language, people often use ambiguous references to entities within the discourse rather than repeating the explicit terminology. When two or more expressions refer to the same entity, they form a coreference. Coreference resolution is the task of recognizing finding these expressions (Lee et al., 2013). Lee et al., (2013) implemented a deterministic algorithm that is present in the CoreNLP toolkit which architecture is presented in Figure 1 (Manning et al., 2014).

While deterministic approach gained state of the art results, this step of natural language processing can be approached also with machine learning models. In recent years, the models utilizing neural networks have gained performance increases over deterministic models (Lee, He, Lewis, & Zettlemoyer, 2017).

Coreference resolution can be very beneficial for tasks aiming for some level of natural language understanding. Lee et al. (2013) mentions tasks as summarization, question answering and information extraction.

3.2.7 Other annotations

CoreNLP introduces couple more steps in their basic pipeline, for example sentiment analysis. Sentiment analysis is the process of extracting subjective information in a corpus. This can be emotional, personal view or attitude present within a text. This can be very useful technique on its own or utilized in certain chatbots for example.

Gender annotation mentioned in the toolkit adds annotations to names which tells if the name is considered a female or a male name (Manning et al., 2014)

3.3 Usage of NLP

What does it mean to "use NLP"? The term natural language processing is often used, especially in the business worlds, very broadly. As with any complex topic, simplicity means understandability. Moreover, it is often considered a subset of artificial intelligence in a sense. While oftentimes NLP tools are utilized in applications that are perceived as AI, it is important to note that many features of text processing are (or at least can be) deterministic algorithms created by smart people instead of smart machines.

It is important to understand the difference between natural language processing and understanding. Processing is the "treatment" of the natural language to produce structured, annotated representation of a given text. It is an automation task that can be done by a human, but much, much slower. And in many cases, this needs to be a professional linguist to achieve reasonable results in a task like dependency parsing.

3.4 Natural language understanding

Natural language understanding is a subtopic within natural language processing with the purpose of finding and understanding or interpreting meaning within a text. While it can be considered a subtask in natural language processing, NLU often extends on the structuring work of NLP to gain understanding of text. Defining language understanding in general is not an easy feat, since natural language hold multitudes of different types of meaning within it. A system can be described as soon as it connects given language commands to a predefined task. Ergo, understands the request and acts accordingly. This creates a large variance between natural language understanding systems, since in a simple closed domain system a single keyword recognition is able to understand what the input means.

As mentioned, language holds different aspects that can be understood. Semantic analysis shown in Figure 1 is often considered an NLU task, since it aims to find semantic meaning in a corpus instead of predefined grammatical structures.

3.4.1 Aspects of meaning in general natural language understanding

To better understand these structures of meaning, some NLU system performance metrics are a good approach to find what they measure. Wang et al. (2018) presented GLUE, or *General Language Understanding Evaluation,* benchmark. NLU systems often being heavily domain constrained, GLUE aims to facilitate research going towards more general understanding models (Wang et al., 2018). We can use this work to take a look at what factors should be expected from a state-of-the-art general natural language understanding system, and utilize that knowledge to find what aspects of meaning are these systems trying to and/or capable of deciphering.

3.4.2 GLUE benchmark

GLUE provides a pre-defined diagnostic dataset to analyze and evaluate NLU systems. This is part of the evaluation benchmark they developed to capture general linguistic phenomena rather than application specific understanding of internal meaning. Diagnostics dataset contains sentence pairs that have been manually annotated to reflect a multitude of categories. (Wang et al., 2018) These categories are a good reference point to get a general understanding of the different aspects that can be used to measure general capabilities of a natural language understanding system. Wang et al. (2018) present four coarse categories divided into smaller phenomena. These are shown in Table 2.

Table 2: Categories of linguistic phenomena annotated in the pre-defined diagnostic dataset of the GLUE benchmark. (Wang et al., 2018, p.4)

Coarse-Grained Categories	Fine-Grained Categories
Lexical Semantics	Lexical Entailment, Morphological Negation, Factivity, Symmetry/Collectivity, Redundancy, Named Entities, Quantifiers
Predicate-Argument Structure	Core Arguments, Prepositional Phrases, Ellipsis/Implicits, Anaphora/Coreference Active/Passive, Nominalization, Genitives/Partitives, Datives, Relative Clauses, Coordination Scope, Intersectivity, Restrictivity
Logic	Negation, Double Negation, Intervals/Numbers, Conjunction, Disjunction, Conditionals, Universal, Existential, Temporal, Upward Monotone, Downward Monotone, Non-Monotone
Knowledge	Common Sense, World Knowledge

Lexical Semantics is the analysis of word-level meaning. Lexical semantics is the phenomena of how words relate to each other in terms on negation, synonymity, redundancy and such. Additionally, word entailment show relations where higher-level meaning is interpreted within a word. For example, "a car" is a vehicle, while "a plane" is also a vehicle, but a vehicle can not be both at the same time. It also covers things like named entities (opened in chapter 2.2.4) as well as quantifiers, where difference between "many cars" and "no cars" is wildly different. (Wang et al., 2018)

Phenomena within **Predicate-Argument Structure** are more involved with the sentence-level meaning and how the lexical units together form meaning. This entails finding and defining core-arguments, possible alternations of sentences and generally the syntactic semantics. (Wang et al., 2018) Simply put, this can be thought as the part where we aim to find semantics from the structure of the sentence or language syntax so to speak. Some of this play closely with coreference parsing (2.2.6).

Logic can be interpreted almost as mathematical logic, where logical operations such as conditionals and negations play a major role in the meaning of a sentence. Very minor subexpressions change the logical meaning of the sentence greatly. All subcategories in this section are factors that influence the logical meaning of the sentence in some way. (Wang et al., 2018) For example let's look at quantifications with a similar example as presented by Wang et al. (2018): "Some phones have a camera" does not entail "My phone has a camera", while "All phones have cameras" does. The logical semantics change by applying quantifiers such as "some" or "all" (Wang et al., 2018).

One interesting aspect from a computational perspective is the **knowledge**. World knowledge and common sense can be reflected in sentences where certain logical conclusions can be made based on common knowledge of the world, culture, social structures et cetera. (Wang et al., 2018) This means that certain structures can be ambiguous, since people tend to drop some syntactically crucial information from natural language based purely on the pretense that the listener understands the context surrounding the sentence and the world. Considering the sentence “Things are heating up in the parliament” and what it means. Objects in the parliament building are not literally getting hotter, but rather “matters currently discussed in the parliament are getting more disputed”.

These examples give a reasonable understanding of the factors that come into play when discussing what understanding natural language actually means. Processing is a more deterministic process with right answers, and NLU is an extension of that. Developing for example a chatbot, a lot of nuances in language need to be considered if the aim is to create a natural conversational agent. While a plethora of tasks can be achieved with conversational agents with relying mostly on structured natural language, more advanced artificial conversational agents need to understand these semantic differences, and much more.

4 Language processing in chatbots

While language processing is a vast and complicated study, chatbots can oftentimes be much simpler in how they understand the user inputs. Since a lot of the NLP and NLU concepts are aimed towards more general processing of language, which is often not needed in simpler question answering tasks. This chapter takes a look on how prevalent these language processing techniques are in different chatbot building tools.

Language processing in chatbots is not as simple as adding previously mentioned processing tasks into a dialogue system. Text processing architecture of a conversational agent can, or even has, to do different kinds of information extractions in order to keep the conversation active. Earlier shown in table 1, intents and entities are mentioned as a core concept for natural language processing tasks in chatbots. This chapter will introduce intents and entities better, to understand how they tie up language processing and chatbots together.

The field of chatbots and conversational systems has also changed a lot during its lifetime. Chapter 1 on chatbots dives briefly into the technological history of earlier chatbots. While advancements have happened from pattern and/or keyword matching chatterbots towards more generative natural language understanding ones, even in modern field the distinction is not that clear. Natural language processing has always been a separate research from chatbots, and merely serves as a foundation for dealing with natural language within computer systems. Conversational systems have taken many forms in the processing of the user input. This chapter contains review of the different methods, and further down takes a better look towards the newer ones, which utilize more advanced natural language processing.

Firstly, this chapter covers reviews on different technologies, and gives introduction on how they work. After that the focus shifts towards dialogue structuring and elements of dialogue. Furthermore, we will investigate the machine learning approaches, and focus more on the modern versions of conversational systems.

4.1 Review on different technologies in multiple conversational systems

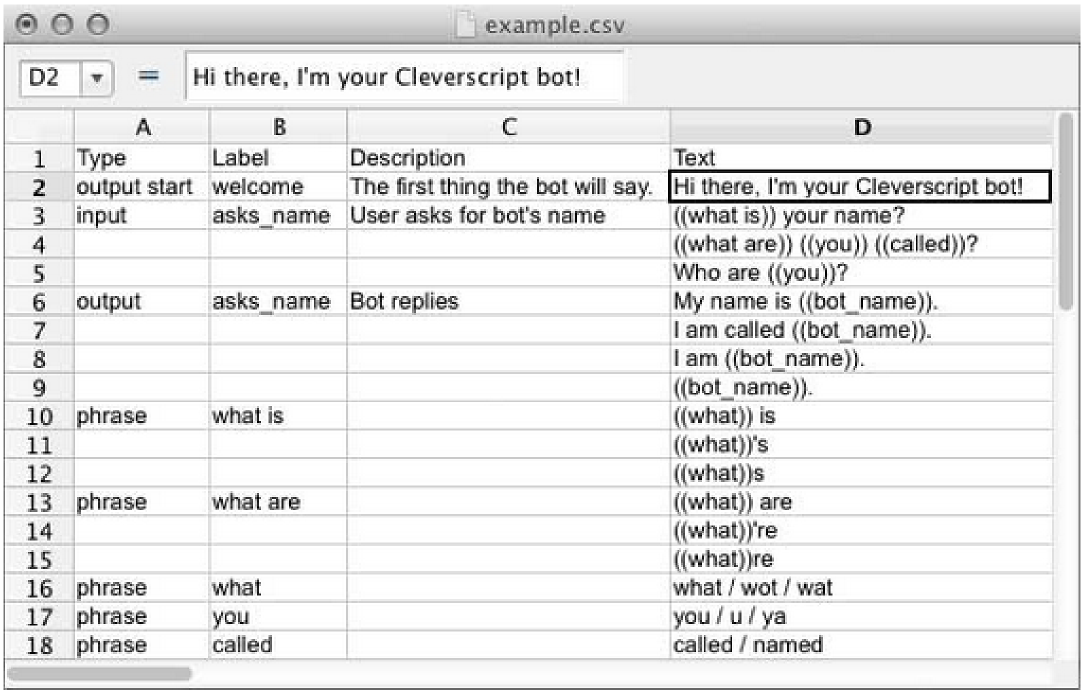
In some parts the subjects presented differ greatly from the natural language processing techniques presented in chapter 2. It is important to note the generality of natural language processing, as it is not a technique designed solely to support conversational agents, but to structure natural language.

Masche and Le (2017) did a comprehensive review on 59 different conversational systems in the past 50 years, starting as early as the ELIZA (Weizenbaum, 1966), deemed as the earliest chatbot. This can be used as a comprehensive view to multitude of systems, and as a starting point towards deeper dive into specific ones. While Masche and Le categorize conversational systems into chatbots and dialog systems, they do also discuss how this is not a clear distinction, but rather to support their review by classifying distinct systems. The classification is done based on the typical components, where chatbots tend to be based on pattern matching algorithms and dialog systems utilize natural language understanding concepts (Masche & Le, 2017). Since the aim is to understand the underlying technologies, these are classified in the following chapters as pattern matching and NLP-based systems. However, this is neither a clear definition between any of the technologies, but rather a way to structure a plethora of knowledge into more concise form.

4.1.1 Pattern matching

From the simplest keyword matching, ELIZA paved the way for later iterations. It matches keywords in the input to a predefined dictionary and applies associated rule for response. The responses would imitate certain psychotherapists in a simple manner, by asking reconstructed questions based on the user input. (Weizenbaum, 1966) This type of matching has been used in multiple chatbots throughout the ages (Masche & Le, 2017).

Similar paradigms were used later on, in of the more popular general use chatbots of the 21st century, Cleverbot. This chatterbot was originally released as *Jabberwacke.* After rebranding to Cleverbot a chatbot building library Cleverscript (Cleverscript, 2020) was also published. While being amusing, and extremely successful in the popular culture as well as being scientifically recognized, Cleverscript is still uses a pattern matching paradigm to generate responses. (Masche & Le, 2017). Both possible inputs and outputs are stored in separate lines in the spreadsheet. It can store words and phrases in a spreadsheet with types of 'variables' that can take a place in a sentence. This is best understood with an example since it is deceivingly simple. Picture is presented in Figure 5 to illustrate this.


example.csv

D2 = Hi there, I'm your Cleverscript bot!

	A	B	C	D
1	Type	Label	Description	Text
2	output start	welcome	The first thing the bot will say.	Hi there, I'm your Cleverscript bot!
3	input	asks_name	User asks for bot's name	((what is)) your name?
4				((what are)) ((you)) ((called))?
5				Who are ((you))?
6	output	asks_name	Bot replies	My name is ((bot_name)).
7				I am called ((bot_name)).
8				I am ((bot_name)).
9				((bot_name)).
10	phrase	what is		((what)) is
11				((what))'s
12				((what))s
13	phrase	what are		((what)) are
14				((what))'re
15				((what))re
16	phrase	what		what / wot / wat
17	phrase	you		you / u / ya
18	phrase	called		called / named

FIGURE 5 Example of a spreadsheet working as the brain of a bot built with Cleverscript. (Fetched from https://www.cleverscript.com/about/ on 27.8.2020)

The basis is that Cleverscript matches the input text to any predefined input phrase, whilst allowing modification to the individual variables within that phrase. This allows more robust use of language to match to the same outcome in a very simple way. Learning happens in solution utilizing Cleverscript by adding more and more data to the spreadsheet, often from the user inputs. This causes the bot to have more possible input-output variations, and more variations to say those things.

Chatscript works in a similar manner to Cleverscript. It has internal files for transforming inputted words into substitutions to achieve the same as Cleverscript, in a sense that it groups different ways of saying certain things to one centralized input, and the generates output. Chatscript files also include general abbreviations and misspellings and such to act as a "cleanup" of words or phrases. (Arsovski, Cheok, & MuniruIdris, 2017) Chatscripts approach to language processing is built around rules that create a script. These rules roughly simulate some concepts presented in chapter 3.2.

In addition to Chatscript and Cleverscript, AIML (Artificial Intelligence Markup Language) is one of the more popular languages to build a conversational agent. It is also a pattern matching algorithm with rule scripts, based on XML (Extensible Markup Language). (Masche & Le, 2017) The ease of use and the simple format and implementation has made AIML one of the most popular languages to build chatbots (Arsovski, Cheok, & MuniruIdris, 2017).

All of these are examples of pattern matching algorithms to match inputted natural language text into a predefined 'category'. This category then has rules for the output (e.g. response or action) or it is passed to a dialogue engine.

While these are simpler than natural language processing in chapter 3.2, they effectively do the same thing, structure natural language into a structure. This is an effective way to build conversational systems, especially for more simple general use. Many of these solutions have also won the Loebner prize even in the recent years (Masche & Le, 2017). Loebner prize is awarded annually to a computer program that is considered by the judges to be most human-like. It is based on the work of Alan Turing and the Turing test. (Mauldin, 1994)

4.1.2 NLU-based systems

While it has been mentioned previously, natural language understanding is not a clear distinction towards a certain technology. This terminology is used in this case to differentiate deterministic pattern matching systems from generally non-deterministic systems, where the outcome is governed by mathematical machine learning algorithms.

Preprocessing is the task of processing the text into a structured format, or in other words, natural language processing. This is oftentimes done in conversational systems as a mean to formatting the input before feeding it to a statistical algorithm. *Generally* chatbots do not utilize every part of the pipeline presented in chapter 3.2, but are found to utilize mostly tokenization, Part-of-speech tagging, sentence detection (a form of tokenization), and named entity recognition (Masche & Le, 2017).

Masche & Le (2017) present **natural language understanding** as the next step in their review for these types of systems. Latent Semantic Analysis based on a Vector Space Model was used in multiple conversational systems. Term Frequency-inverse document frequency techniques (TF-IDF) was utilized in one. (Masche & Le, 2017). These models and methods will be presented in more detail later on.

4.1.3 Language tricks and other features

According to Masche & Le (2017) many of said chatbots used different language tricks to better succeed in said tests. Canned responses are used to hide the fact that user asks questions not anticipated in the knowledge base, by giving open ended answers. Creators have also given the bots personal history, which enriches the social background of a bot, while being completely fictional. Open-ended statements that do not really mean anything, and occasional misspellings and simulated keystrokes aim to further humanize the chatbot. (Masche & Le, 2017)

Additionally, some special features can make the bot more human. Some systems are built to learn facts about the user from the conversation and use it later. Some systems also recognize gibberish or nonsense and can respond appropriately. Animations, voice, and facial expressions have also been utilized to make the agent appear smarter than it might actually be. (Masche & Le, 2017).

4.2 Dialogue management

Understanding the user utterance is one thing, while deciding how to react to it is another. Dialogue management refers to the process of deciding how to proceed with the conversation after the user. This part can be referred to as dialogue engine.

Role of a dialogue manager in a conversational system architecture is to guide to conversational to natural and productive path (Goddeau, Meng, Polifroni, Seneff, & Busayapongchai, 1996). This can mean various different things in different scenarios. More common actions the dialogue manager should try and to is to prompt or guide the user to keep within the boundaries of the domain and capabilities (Goddeau et al., 1996). This can mean suggesting certain actions, keeping the information and goal in the dialogue, or to further ask for clarifications or corrections from the user when needed.

Figure 6 shows a general structure of dialogue management, presented by Williams, Raux & Henderson (2016). While it is built around speech dialog systems, chatbots function in a similar manner. For example Rasa builds around similar architecture, where dialogue states determine the dialogue policies (Bocklisch, Faulkner, Pawlowski, & Nichol 2017), pictured also in Figure 7.

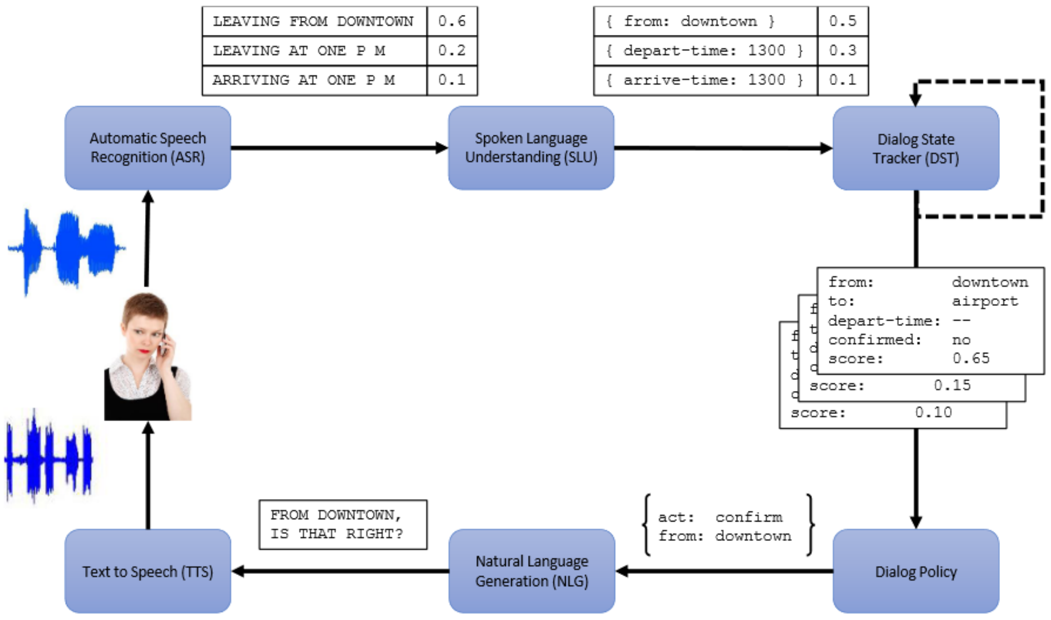

LEAVING FROM DOWNTOWN 0.6
LEAVING AT ONE P M 0.2
ARRIVING AT ONE P M 0.1
{ from: downtown } 0.5
{ depart-time: 1300 } 0.3
{ arrive-time: 1300 } 0.1
Automatic Speech Recognition (ASR)
Spoken Language Understanding (SLU)
Dialog State Tracker (DST)
from: downtown
to: airport
depart-time: --
confirmed: no
score: 0.65
score: 0.15
score: 0.10
Text to Speech (TTS)
FROM DOWNTOWN, IS THAT RIGHT?
Natural Language Generation (NLG)
act: confirm
from: downtown
Dialog Policy

FIGURE 6 Example of a speech dialogue system showing the Dialogue management function (Williams, Raux & Henderson, 2016, p. 5)

4.2.1 Dialogue State

Quite often dialogue management is achieved by **Dialogue State Tracking.** Dialogue state tracking is used to estimate the current goal or state of the conversation in relation to the discourse that has happened already. Dialogue state tracker therefore is a function of the inputs and information passed on by the NLU-engine, where the output is the estimate of the dialogue state within the conversation. (Williams, Raux & Henderson, 2016) This is the passed on to dialogue policy and response generation. Dialogue state tracker combined with dialogue policy can be considered a **dialogue manager**.

Dialogue state manager achieves done properly achieves more natural conversation following an understandable pattern. Two prominent ways to achieve dialogue state tracking can be seen in finite state-based systems and frame-based systems (Masche & Le, 2017). Finite based systems follow a pre-defined with certain actions prompting transitions to a different path in the dialogue-graph. Frame-based approaches utilize a frame like state, where a frame is a form containing slots. These slots are filled by asking more information from the user. A state is recognized, and by prompting the user to give out more information needed to achieve the goal, the conversation flow is not fixed like in the finite-state approach. (Masche & Le, 2017)

As with the user input interpretation in chapter 4.1, dialogue management can be done in a generative (statistical learning) way, or by hand-crafted rules (Williams, Raux & Henderson, 2016). Hand-crafted rules gained the best performance early on, while generative models came more prominent with the increase of computing power. Typically dialogue states can be modeled as a Bayesian network, where dialogue state is in relation to system action, user action, and preprocessed input result. The idea here is to apply a distribution to all possible to possible dialogue states, which is then iteratively updated when new actions are produced. (Williams, Raux & Henderson, 2016)

This, and most other systems aim to train a classifier that encodes dialogue history into the training to teach the classifier. This is to maintain the conversation in the same topic and to gain more information on it to achieve the user goal. In addition to Bayesian approach, Markov models, Conditional Random Fields models, and recurrent neural networks have been applied to dialogue state tracking with success (Williams, Raux & Henderson, 2016).

4.2.2 Dialogue policy

Dialogue policy is a part of the architecture that determines the next action. Dialogue state tracker provides the dialogue policy with a state containing current knowledge gathered from the conversation and the situation, and dialogue policy decides further action based on the information passed in the state. If the state needs more information, the dialogue policy can decide to ask more information from the user. If enough information is passed to reach the goal, the dialogue policy can decide to go on and represent information from a knowledge base, and 'finalize' the conversation.

The definitions between dialogue state trackers, dialogue policies and dialogue managers vary significantly between different systems and different literature. It is important to note that not all systems follow this kind of structure, and they may utilize elements of a certain paradigm and not the others (Masche & Le, 2017).

4.3 Response generation

One part of the presented modular architecture in this thesis is the response generation. This part is responsible for choosing the response given to the user. This chapter investigates the technologies of response generation. Since oftentimes conversational agents are built in a task-oriented way, the agent's function is to respond to the users' questions and needs. Therefore, just responses alone define most of the conversational agents' interactive "language" and present a big role in the believability and the experience of using the conversational system.

As is the case in many of the parts in conversational systems, response generation can be a scripted information retrieval system or a smarter generative model. Both serve a function in conversational systems. More task-oriented question answering systems may find the best outcome with a retrieval models, whereas generative models are usually used to achieve more natural, human-like conversation in a general domain.

4.3.1 Retrieval models

Information retrieval-based models aim to find the most relevant answer in a predefined repository (Yang et al., 2019). Repository could either be a historic repository of conversations or a predefined knowledge base. Retrieval model does not always mean pattern or keyword matching but can be done by neural rankings as well (Yang et al., 2019). The definition is rather a question about what response in a pre-defined corpus is given with these sets of inputs. The inputs depending on the system can mean an unprocessed input sentence or a finely processed vector representing information gathered by parsing with NLP, as well as contextual information from the dialogue engine. Main difference how retrieval models differ from generative models is the pre-defined corpus of response candidates. Thus, retrieval-based models lack flexibility of responses due to being predefined to an extent, while gaining advantages in diversity and information richness (Yang et al., 2019). Even the best matching response may not be a good response, if the question is out of the domain of the repository. (Song et al., 2018)

4.3.2 Generative models

Generation-based models aim to find the ability to generate coherent new responses within the context. Generative response generation systems tend to be used in more chatter-focused chatbots, due to their ability to create more diverse and coherent responses while lacking in ability to include contextual information. (Yang et al., 2019; Song et al., 2018)

According to Song et al. (2018) a typical way to achieve generative response generation is to use *seq2seq* model. This Sequence-to-Sequence model is constructed by using to recurrent neural networks as the encoder and the decoder. "The encoder is to capture the semantics of the query with one or a few distributed and real-valued vectors (also known as embeddings); the decoder aims at decoding the query embeddings to a reply." (Song et al., 2018, p. 1)

4.3.3 Hybrid

To achieve the pros of both approaches and to limit the cons, recent years have spawned efforts to create systems that utilize elements from both types of modules. (Yang et al., 2019; Song et al., 2018). Song et al. (2018) present a model which first uses retrieval system to choose the candidate responses from a repository. For those candidates, each one is integrated into the generation module to enrich the meaning of generated responses. These enriched generated responses are re-ranked and evaluated with additional retrieved candidates, and then given as the response. In this case, the response could either be enriched generated on or a purely retrieved one. (Song et al., 2018)

Yang et al., (2019) propose a similar model. But instead of enriching generated data, they apply more care into both models, retrieval and generation. Both models produce candidate responses which are then re-evaluated, and the best response is the final system output. (Yang et al., 2019). Both hybrid models presented tend to outperform non-hybrid approaches (Yang et al., 2019; Song et al., 2018).

4.4 Levels of dialogue

As stated previously, the aim of natural language processing (and oftentimes natural language understanding), is to structure unstructured language into a usable format. User input in modern chatbots are generally structured into intents and entities (Braun, Mendez, Matthes, & Langen, 2017). However, these are not the only usable and classifiable information within dialogue. Especially older systems, and nowadays more complex general use chatbots might classify parts of the conversation with dialogue acts and goals. Both of these represent a higher level of classification. Figure 6 represents examples of some of the possible classifications. While being an image of a certain program, it highlights some parts that are present in most modern chatbots, such as intents. Most of the prominent chatbot development platforms use intent and entity classification (Braun et al., 2017)

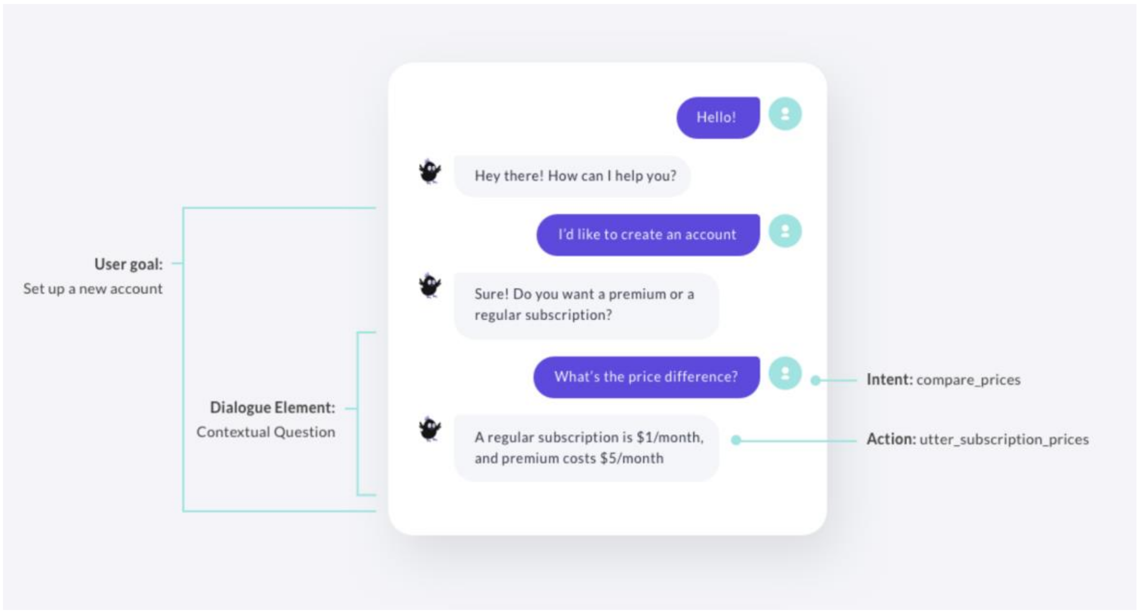

Hello!
Hey there! How can I help you?
I'd like to create an account
Sure! Do you want a premium or a regular subscription?
What's the price difference?
A regular subscription is $1/month, and premium costs $5/month
User goal: Set up a new account
Dialogue Element: Contextual Question
Intent: compare_prices
Action: utter_subscription_prices

FIGURE 7 Dialogue elements represented in Rasa (Fetched from https://rasa.com/docs/rasa/dialogue-elements/dialogue-elements/ on 25.8.2020)

It is important to note while discussing classification, that not all classification is necessarily computational. Meaning some of the concepts we look at are to better understand the aspects of dialogue withing chatbots, rather than finding ways to optimize the understanding. However, most things within this chapter focus on the computational ones.

4.4.1 Intent

While talking about chatbots and conversational agents, intent is the keyword one will face first when developing their chatbot. Intent is the goal, meaning or aim of the input. Intent is the classification given to an input through the chatbots engine that often determines the action or response the chatbot is programmed to give. These are often programmed manually, and they determine the bot's capabilities. As seen in figure 5, the user input is classified as *compare_prices*. This intent with some additional information is passed on to a dialogue engine and appropriate action is taken. In this case, a comparison of prices.

Intent classification is something to be discussed later, since in many cases it can be considered the 'brain' of the system. Especially when looking at machine learning solutions within chatbots, intent classification is the core problem faced in optimizing the bot.

4.4.2 Entity

Entities can be considered modifiers to the intent. These give additional information to the response generation or dialogue engine. Entities can be almost anything within the input that give more information to give the right answer or to continue within the right action. Figure 6 gives an example input with more than one entity the better understand the function.

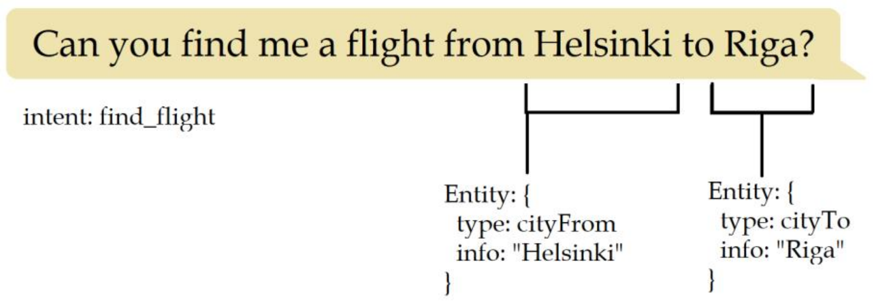

Can you find me a flight from Helsinki to Riga?
intent: find_flight
Entity: { type: cityFrom info: "Helsinki" }
Entity: { type: cityTo info: "Riga" }

FIGURE 8 Example of entities within user input

In this example the user asks for information about flights. While the intent in this example could be classified as for example *find_flight*, the entities give additional information about the departure destination and the arrive destination. If this interface were connected to a flight search engine, it could fill those needed fields and complete a search. It could either then proceed to show results or ask for additional information. This is determined by the response generation. If the user were to give a timeframe for the flight, it could be classified also as entity, since it gives additional information to the intent *find_flight*.

As with intents, entity classification is a text processing task, and the actual process will be discussed further in this thesis through examples.

4.4.3 Dialogue act

One prominent way of giving natural language some form of meaning within the context of conversation, is to assign a given input (phrase, sentence) into a *dialogue act.*

Dialogue act is a sentence-level unit of what the function of the sentence or utterance is. This classification can include categories like *question, statement, hesitation,* and *greeting*. Therefore, instead of context or intents, dialogue act is a classification for the discourse structure. (Stolcke et al., 2000) This classification can be used to keep to conversation flow concise. Chatbots could utilize dialogue act classification to narrow down the possible discourse options, or to add human-like features into the conversations. Classifying whether an input sentence is acknowledging something or asking something and acting accordingly can determine a lot how the chatbot is being perceived.

In Figure 5, dialogue act is referred to as *Dialogue element*. While Rasa pictured in the figure does not classify dialogue acts to the dialogue engine to process[1] the figure shows example of this, more to the context of understanding and communicating dialogue in general. To further understand dialogue elements in the context of common discourse, figure 9 shows an example by Stocke et al. (2000). This shows some common ways to classify dialogue acts. However, they are not the only way, and dialogue acts can be determined to suit the applications needs (Stocke et al., 2000).

Noteworthy is also the inclusion of acts such as *abandoned* and *appreciation.* These do not necessarily add any semantic meaning to the conversation but may be beneficial to achieving more human-like conversation, especially in chatbots that utilize voice recognition instead of text as the main source of user input.

Speaker	Dialogue Act	Utterance
A	Yes-No-Question	So do you go to college right now?
A	Abandoned	Are yo-,
B	Yes-Answer	*Yeah,*
B	Statement	*it's my last year [laughter].*
A	Declarative-Question	You're a, so you're a senior now.
B	Yes-Answer	*Yeah,*
B	Statement	*I'm working on my projects trying to graduate [laughter].*
A	Appreciation	Oh, good for you.
B	Backchannel	*Yeah.*
A	Appreciation	That's great,
A	Yes-No-Question	um, is, is N C University is that, uh, State,
B	Statement	*N C State.*
A	Signal-Non-Understanding	What did you say?
B	Statement	*N C State.*

FIGURE 9 Dialogue act classification in a casual spoken conversation (Stocke et al., 2000, p. 340).

4.5 Different languages in conversational systems

So far natural language has been discussed in a very general form in this thesis. But as anyone can understand, people communicate most naturally with their

[1] Rasa passes dialogue act type classifications as *intents*. https://rasa.com/docs/rasa/dialogue-elements/dialogue-elements/

native language. How does this affect the language processing in conversational systems? Different languages in natural language processing were briefly mentioned in chapter 3.2, but it does not really answer this question. Why is this factor often not discussed in the literature?

For natural language processing pipelines, the language absolutely matters. This is exactly what universal treebanks and dependency parser are created for, discussed briefly in chapter 3.2.1. Good thing about the pipeline presented is that proper tokenization and part-of-speech tagging makes the rest of the tasks more trivial. The general way how languages differ are already solved in the early steps, which makes the consequent tasks rely on already structured information.

As discussed in chapter 4, chatbots and conversational systems often steer away from the later steps, and sometimes do not apply natural language processing tasks at all. For example, the pattern matching algorithms or generative end-to-end models in chapter 4.3 do not really care about the language in any way. Additionally, the architectures used in the most popular modern build tools, such as Rasa do not rely heavily on the language or are built with prepackaged dependencies, or other type of support. This makes sense in commercial products.

Case study presented in this thesis is a conversational system built on the Finnish language. This example provides more insight into this issue. There we will look at how this specific instance handles a different, fairly marginal language.

5 Chatbot development tools

As for any general application, there is a multitude of development tools to achieve the goals. These tools include names such as Rasa, Googles Dialog Flow and Wit.ai, Jarvis et cetera. These vary between how they are structured and utilized. Some are designed to be simple building tools with graphical user interfaces for people with no development background, while some are more technical software libraries to give more control.

This chapter takes a deeper dive into the technologies and software tools based around building a chatbot. Chapter 2 presented a general flow of a natural language processing pipeline, and chapter reviews different methodologies to achieving language processing within chatbots. This chapter aims to combine the two and tries to find the connection of NLP and NLU techniques in environments currently used in chatbot development. This is to gain an idea on what is the situation within commercially used conversational agents, and how they apply the previous knowledge in chatbots.

5.1 Review on the performance of modern development tools

Braun, Mendez, Matthes, & Langen (2017) present a very usable research on evaluating the different technologies within a select group of chatbot building toolkits. We can use this as a basis for seeing how they differ in their usages, and after that take a deeper dive into a select few. Braun et al. (2017) take a similar attitude towards selecting the evaluated NLU services; the most popular ones to use. In their research they chose LUIS, IBM Watson Conversation, API.ai, wit.ai, Amazon Lex and Rasa. Most of them follow the same structure of classifying input text to intents and entities. (Braun et al., 2017) Table 1 in chapter 1 shows the general structure and architecture. While Braun et al. (2017) present a slightly varying generalized structure (Figure 10), the same principles apply.

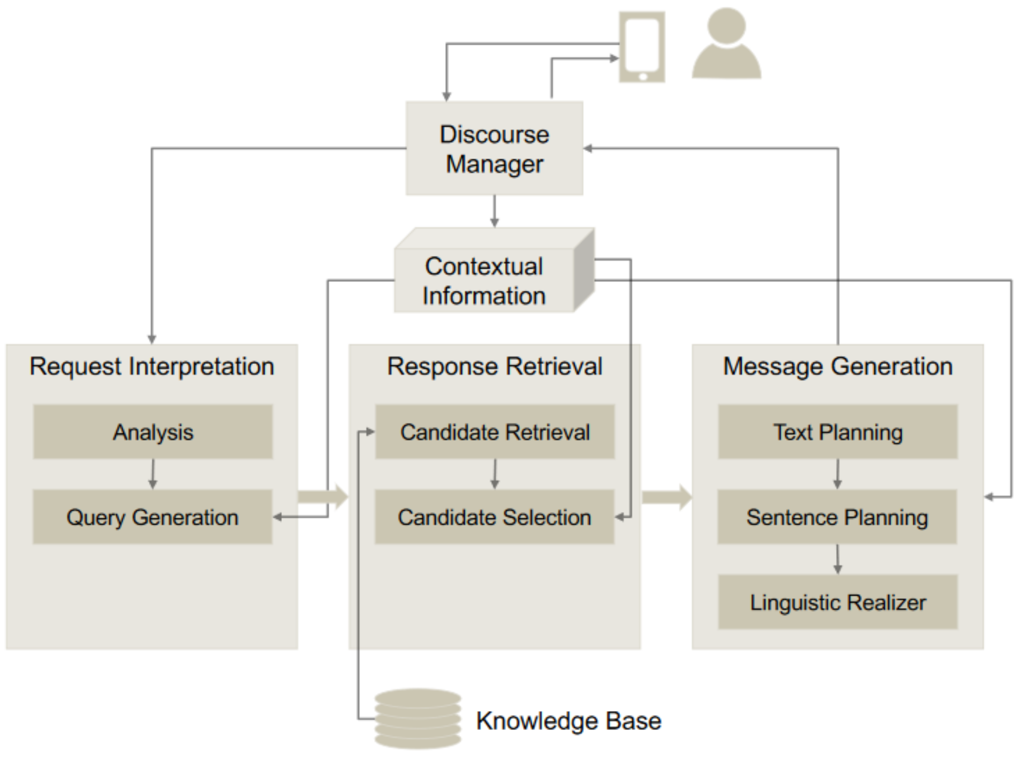


FIGURE 10 Generalized architecture of select chatbots (Braun et al., 2017, p. 176)

While a multitude of different architectures and techniques have been presented in this thesis so far, one can start to see similarities in the generalizations. While the terminology differs from study to study, similarities and connections are very present in architectures. Figure 10 shows dialogue management as discourse management, response generation as a combination of response retrieval and message generations, and NLU as request interpretation and analysis. Query generation in Figure 10 can be thought of as generating the intent-entity structure based on the analyzed input text.

Braun et al. (2017) present three hypotheses on the evaluation of the NLU services. Their performance varies between services, commercial ones perform better, and the quality of the labels are influenced by the domain. These were tested with two separate corpora of questions annotated with entities and intents. First hypothesis was supported, and great variance was found in the evaluation. However, due to Rasa performing extremely well made the second hypothesis (commercial services being better) unsupported. Regarding the third hypothesis the answer was not as clear: “although the domain influences the results, it is not clear whether or not it should also influence the decision which service should be used” (Braun et al., 2017, p. 180)

This type of research gives a good baseline for more detailed research. As Braun et al. (2017) mention in the paper, the producers tend to be secretive about their technologies. Therefore, it is beneficial for the research to look more deeply into the ones that do, mainly open source ones. In this case, this means

Rasa. And as can be seen later, Rasa tends to closely follow a very generalized architecture found in many studies and can be programmed to utilize multitude of different techniques presented in this thesis. While complexity does not always include better results, in the context of research on technologies, this is the best baseline for modern development tools.

5.2 Popular modern chatbot frameworks

Actual statistics were not found to determine the most popular frameworks used nowadays. A simple web-browsing was conducted to get an idea on what are the prominent ones. Additionally, the aim is to present the ones that provide the users enough information about the technologies behind the framework. Since chatbots have become a commercial success, the technology giants have taken it under their wing to provide solutions to the market. Problem with this for research is the secrecy of the backends, since a lot of them work as a cloud-based solution. These are often inclusive development suits, where graphical user interfaces are provided for non-technical people to develop chatbots. These kind of general functionality platforms are not in the interest of this thesis, since the technologies and techniques matter, not the functionalities.

All this considered, it is important to understand the business needs in relation to technologies. While certain algorithms mentioned previously might achieve better results on their own, that does not make them suitable for development in many cases. The general end-to-end model *seq2seq* has achieved outstanding results in input-based response generation. This might not necessarily suit the end goal of a development process. Chatbots tend to be built for domain specific applications with very specific outcomes or answers from an existing knowledge base. Modularity and modifiability get and advantage over an accurate end-to-end model.

While being apparently popular choices in the business world, not much literature is based on actually verifying the performance of different tools. In addition to Braun et al., (2017), in 2018, Canonico and De Russis (2018) conducted a critical review on the most prominent chatbot building tools provided by the big technology companies. This review included six of the biggest ones; Google's *Dialogflow*, Facebook's *Wit.ai*, Microsoft's *LUIS*, IBM *Watson Conversation, Amazon Lex*, and *Recast.ai*. (Canonico & De Russis, 2018) These are all similar cloud-based platforms utilizing machine learning algorithms in their NLU tasks. They also provide a multitude of integration and development tools to choose from. These platforms also rely on *intents* and *entities*[2] (Canonico & De Russis, 2018; Braun et al., 2017). The results are varying, partly dependent on the compared tools, but also on the methodologies. Canonico & De Russis (2018) base their comparison mostly on the functionalities and intent recogni-

[2] Amazon Lex does use intents, but entities are presented in a different manner and with different terminology *slots* (Braun et al., 2018; https://aws.amazon.com/lex/)

tion, Braun et al. (2017) composed a more detailed comparison with different entities and corpora.

It is inconclusive on what is the best tool to build chatbots, and multiple factors affect this. To better get an idea of the "shared functionalities" within the modern tools, some are presented here. Dialogflow gives a good documentation on their functionalities, while Amazon Lex uses a differing structure form the norm. Later Rasa is explained in greater detail to support the case study, and it being the most reflective of different methodologies possible.

5.3 Dialogflow by Google

Dialogflow appears to be one of the most prominent search results when looking at adequate and popular chatbot development platforms. Built around Google cloud platform, it provides a platform for building different types of conversational agents. Additional functionalities include visual builder, cross-platform support and collaborations in the Google Cloud environment (https://cloud.google.com/dialogflow). Dialogflow is divided into Dialogflow ES and Dialogflow CX, where CX is the more advanced tool.

Dialogflow names the NLU core as an "agent". Not much is told in the documentation about the natural language understating of Dialogflow. It uses machine learning algorithms to train a classifier to classify the end-user input into an intent. Some additional settings are included such as spell correction and classification threshold, as well as the ability to train multiple agents to a single application.

Dialogue management is done as "flows" in Dialogflow. Flows are predefined graph-like structures determining the conversation path. In combination with *Pages* which correspond to states in the general literature, Dialogflow handles the discourse structure. *State handlers* change this state by looking at factors within *a page* and giving responses that might change the state to another.

Dialogflow uses intents and entities in a similar manner as presented in chapter 4.4. Intent is the main way to categorize or structure the end users input. When an input is given, the agent compares the input to the training phrases and gives out the best match. It gives intents a confidence score between 0 and 1, and if the threshold provided in the agent settings is met, a match is provided. The developer can train the agent to match intents by providing training phrases, which can also be annotated with certain words to match entities. Dialogflows machine learning automatically extends the intent classification set based on provided training phrases. Dialogflow ES uses both pattern matching as well as machine learning algorithm to match inputs to intents.

Dialogflow provides custom and predetermined entities. Predetermined entities represent the most common types, such as dates and times. Regular expression entities are supported for cases where specific term recognition is not suitable. Additionally, *fuzzy entities* provide better matching for multi-word entities to match same entity with varying words representing it.

5.4 Amazon Lex

Using the same engine as the popular personal assistant Alexa, Amazon Lex utilizes at least seemingly simpler structure compared to the traditional intent-entity structure, present for example in Googles Dialogflow. As Braun et al. (2017) found, Lex veers away from this structure compared to other solutions by large providers, by not utilizing entities. This is supported at the time of writing this, according to Amazon (https://aws.amazon.com/lex/).

While this is true, Amazon still uses intents as their main way of classifying user utterances. And entities are not forgotten, but rather replaced by *slots.* Slots work similarly to entities and states combined, where slots present a place in an intent to fill. Additional features include integrations with Amazon AWS Lambdas, and intent chaining. Intent chaining with slot fulfillment appears to achieve similar outcome as entities and state management.

5.5 Rasa

Rasa is an open source natural language understanding and chatbot stack/toolkit. It includes Rasa Core and Rasa NLU. Rasa Core functions as the dialogue engine, while Rasa NLU is the engine behind the natural language understanding. This division is done to enable non-academic developers with little experience in textual analytics to build chatbots with Rasa without concerning themselves with the NLU engine. (Bocklisch, Faulkner, Pawlowski & Nichol, 2017)

5.5.1 Rasa architecture

Rasa uses a modular architecture to ease the integration of separate systems. This means the Rasa Core and the NLU motor can be accessed separately through exposed HTTP APIs. (Bocklisch et al., 2017) High level architecture of a Rasa system is pictured in Figure 1. In this figure the Interpreter works as the NLU engine, and it extracts structured information from natural language message. Tracker maintains the conversation state. Policy receives state of the conversation form tracker and decides what action to take. Action is executed and logged to the tracker to keep state. Action can be a message to the user, or something else like ‘listen’. This is iterative process keeps repeating throughout the conversation with the bot.

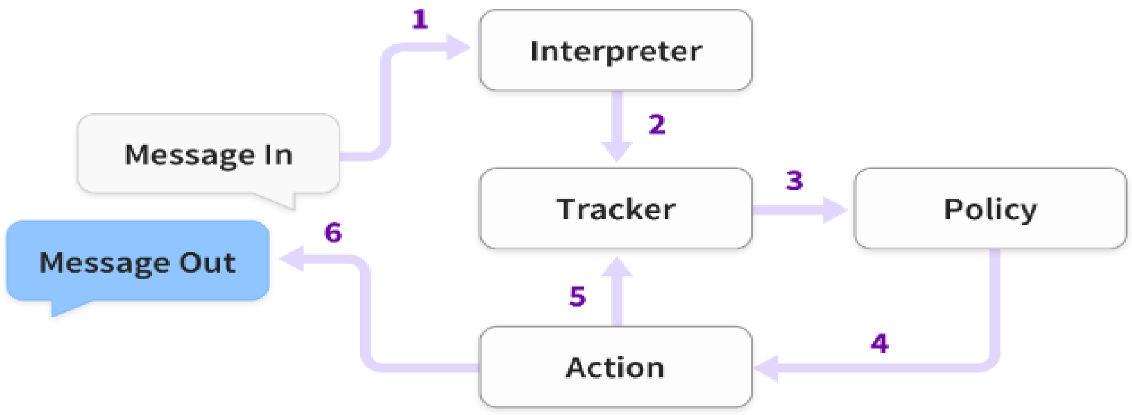


FIGURE 11: Rasa architecture

In addition, Rasa includes a paid enterprise version called Rasa Platform that provides graphical user interfaces for developing, analyzing, and shipping chatbots through Rasa.

5.5.2 Rasa NLU

The natural language understanding module in Rasa, the Rasa NLU, is a collection of machine learning and natural language processing API:s and components. These coupled together form *pipelines* that can hold different steps of the NLP process. Part of developing and training the chatbot is to choose an appropriate pipeline and components for the need. Rasa documentation helps with this and recommends certain ones for general purpose chatbots.

Rasa NLU building follows roughly the general structure presented in Table 3. We can compare these to the concepts presented in section 2 and see how natural language processing techniques are used to develop modern chatbots.

Table 3: Parts of the Rasa NLU pipeline

Pipeline Section	Explanation
Word Vector Sources	These components contain pre-trained models to use
Tokenizers	Similar as explained in section 3.2.1. Gives the tokens of the input to the next part of the pipeline.
Entity Extractors	Entity extractors extract entities, such as person names or locations, from the user message. Similar to NER presented in section 3.2.4
Text Featurizers	Returns the feature vectors of the input for the classifiers in the pipeline. Feature vector is a numerical representation of the information that can be fed to the classifier.
Intent Classifiers	This part of the pipeline is a machine learning model to apply a predefined intent(s) to the given input/feature vector
Policy-pipeline	Rasa Core. Separate pipeline to manage the dialogue to

the appropriate response or action.

Rasa is a development suite for chatbots. This means it utilizes multiple separate libraries to complete the language processing. For example the pipeline can comprise of multiple different providers for each of the NLP steps described in chapter 3.2. Rasa holds many different tokenizers for start. These are provided partly by existing NLP libraries or models, like SpaCy and ConveRT. It also includes different tokenizers out of the box, like white space tokenization for general use, or Jieba tokenizer for the Chinese language. (Rasa Technologies, 2020)

While we can see some concepts presented in chapter 3 appear also in the Rasa NLU pipeline, most of them do not necessarily add any value to a natural language understanding task and are therefore not supported by default. One can still develop and add custom components to the pipeline. For example, sentiment analysis. In most cases, knowing the sentiment does not add any significant value to a general use chatbot, but in certain cases it can be programmed to lead into a different dialogue path depending on what the user is "feeling".

6 Case study: Methodology, organizations, and the project

Literature review so far has taken a deep dive into the inner workings of conversational systems. Going all the way from text processing to different approaches to achieve natural language understanding in chatbots, it can be deduced that the field of conversational systems is far and wide. Original aim was to gain a solid view of an architecture and a best practice for building chatbots. However, this turned out as an impractical task due to the sheer amount of different views and approaches. As it often stands out, the research world takes a different approach to certain concepts from is present in the business world. This can be seen clearly in the basic architectures, where commercial systems designed to be built around the customer's needs are heavily modifiable, easy to understand architectures. While research aimed to create the most accurate, human like conversational agents are starting to use end-to-end architectures or are relying on copious amounts of crowd-sourced data.

A case study is conducted for better understanding how conversational agents can be developed to process the input into a usable form. This chapter presents the case study, participating organizations, and the research methods in a brief overview. Chapter 6 presents the study and findings in more detail.

6.1 Research methods

This study is conducted as a case study. This study is conducted in two interconnected parts: A document review and a theme interview based around it, and a code analysis. Document review is conducted around all documentation present in the project, including source codes and project-related documents such as contracts and guidelines. Based on the findings of the document review, an interview is conducted to get additional insight on design choices and unclear concepts or factors within the development process. This gives a baseline to read and understand the code base. With the help of debugging tools in Visual Studio Code and Rasa, more insight can be gained on the text manipulation that happens within the system.

According to Runeson & Höst (2009), a case study is an empirical research method that aims to "investigate contemporary phenomena in their context" (Runeon & Höst, 2009, p. 134). This definition within its context is the guideline used for this thesis. Same research provides more defining aspects and guidelines for a case study conducted in software development. This study also raises other factors that make a good point for a case study, which include multiple sources of evidence as background as well as unclear boundary between phenomenon and context. (Runeson & Höst, 2009)

Case study often utilizes elements of other research methods. Each element has its purpose and fit a different scenario or research question. The purpose of this study is to find out what is happening and gain new insight, as well as portraying a phenomenon in a situational context. This makes this case study exploratory and descriptive. (Runeon & Höst, 2009)

As the literature review shows, no simple answer is found and a general lack of standards and definitions make it difficult, if not impossible to gain a generalizable answer based on the question "How do digital conversational agents process language and how do they understand it?". Therefore, a natural context around the development is suited to help the study of this question.

6.2 Participating organizations

6.2.1 Gofore Inc.

Gofore is a Finnish consultancy company, specializing in the digital transformation process of clients across different industries. It provides experts in different technical fields to conduct development projects. Gofore provides different information system solutions, as well as multiple different services around them, such as design, project management, service- and systems architecture. Both private sector and public sector are represented in the client base. Gofore is a publicly listed company with currently over 600 employees, mainly in Finland.

For this case, the focus is on an outsourced design and development project in the public sector. Gofore provides 3 consultants for this project in the span of several months. At the time of research, the project is ongoing, and development is happening all the time. A snapshot of a current version is used for consistency's sake throughout the code-analysis

6.2.2 Municipality of Laukaa

Laukaa is a municipality in central Finland, with a population of close to 19 000 in 2018. Municipality is run by a governmental organization consisting of the municipal council, municipal government, and the mayor. Under these governmental powers, different service sectors. Service sectors prepare proposals on rulings, while the governmental organization implements them. (Laukaa, 2020)

These service sectors represent different sectors within the municipality, out of which one part consists of the financial management. Financial management supports the municipal government by providing different services to different parts of the governmental organizations. These services are related to things like accounting, transactions, and procurements.

6.3 Project summary

6.3.1 Planned functionalities in short

This project aims to provide the municipality of Laukaa's financial management a digital assistant to support their information management. This conversational bot aims to facilitate the search and utilization of existing financial information. Additionally, the project provides a system independent and reproduceable model for similar development for other municipalities in Finland.

The function of the bot being developed is to provide a wide range of domain specific information to the personnel of the financial management team. This information includes information about cost centers, their financial situation in the form of reports, assets, predictions, and comparisons. Different financial plans of the cost centers are provided to support more detailed usage.

Invoices and purchases form another big part of the provided information. Data is provided about different providers and their collaborations with the municipality, as well as invoices that are connected to them.

Certain alerts are designed to further improve the idea of a "digital assistant" that helps the employees of financial management to lead and make decisions based on information. Alerts are made for example about significant disparities in certain comparisons, or if there are invoices to check.

The digital assistant is integrated to Microsoft Teams communications software. This will be the main platform for utilizing the chatbot.

6.3.2 Project development in short

The project is completed throughout the year 2020, in close collaboration with the financial team and the development team. The small development team consists of 3 experts in different areas, such as project management, infrastructure, data analytics and software development. Close collaboration with the financial management team is crucial to garner the domain knowledge needed to build a digital assistant that answer the real needs of the end-user.

Project is managed as an agile software development project, following the SCRUM-framework (Schwaber & Beedle, 2002). Assistant is developed in an iterative process to periodically provide the customer with testable part of the software.

The chatbot is built with Rasa framework and is written in Python programming language. The software is deployed to a proprietary server of the customer and is ran in containers utilizing Docker. Source code version control is handled with Git, and the remote is hosted in Giltab. Gitlab also functions as the project management and documentation platform for the development process. The financial data is managed by Pro Economica software provided by an external organization. From there it is exported to a separate database to be utilized by the chatbot. For this research, a development name "Laubot" is used throughout to refer to the system which is being developed.

7 Case Study: How does Laubot process language?

As mentioned in the previous chapter, this study is conducted in roughly two parts, with the aim to answer the question "How does a chatbot process language?". First part will be conducted as a review on the existing documentation, which mostly consists of the source code of the software being developed. This part also includes a non-structured interview to better understand and gain insight on the documentation. The second part is the in-depth code review including local running and debugging the code to gain access to the memory slots used by the software.

The source code review will then follow a structure through examples. An example phrase is presented, and through code references and debugging tools we will look what manipulation or processing is done to the text through what means, and how it affects the outcome. In addition to the source code, this study heavily relies on the documentation of Rasa found on www.rasa.com/docs/ (Rasa Technologies, 2020). This chapter also presents a detailed view of the code-level architecture to understand what parts of previously presented NLU-elements connect to this context.

Access to the code repository allowed this to be completed without extensive interviews. One Non-structured interview was deemed adequate to complete the code review. The interviewee provided tips and tools on debugging the system and reading the Rasa documentation.

Certain parts of the system are not focused extensively within this thesis, as the aim is to understand the language processing and not the general system architecture. Things like deployment, shipping, hosting, UI, and database are not in the core interest, and are presented more broadly if they support the natural language understanding pipeline.

7.1 Conversational system with Rasa

Rasa tools and their architecture were presented in chapter 4.3. Rasa is divided into *Rasa Core* and *Rasa NLU* (Rasa Technologies, 2020). Rasa NLU is the natural language understanding engine, which handles the user input. It takes a user input string and returns a JSON representation of the input in structured format of intents and entities. Rasa Core is the "Dialogue manager", which handles response generations and/or actions based on the conversation history. The following chapters consist of explanations on different parts of the Laubot built with Rasa, and how they connect to the concepts explained in previous chapters. This aims to follow a logical structure of the input text, similar to the architecture of Rasa shown in Figure 11.

7.1.1 Structure of the codebase

As this is a technical review of the software, understanding the codebase is important for the following parts. Since the system is built from ground up to meet customer needs, the codebase holds a lot of "unnecessary" code. This means different shell scripts to launch environments, SQL-code to connect to database, integration code to control the user interface software. While all this is necessary for an end-to-end solution that works outside of the local machine, our interests lie in the functionalities of Rasa. The structure of relevant files or directories is presented in Table 4 with brief explanations of their function within the system.

Table 4: Relevant directory structure of Laubot

File/Directory	Function
custom_actions/	Holds Python-files for custom actions the bot can do based on the intents and entities it receives
custom_functions/	Custom functions of internal business logic, not directly related to Rasa functionalities
data/	Data directory holds several files of training data for intents, entities, stories, and actions. The bot is trained on this data.
models/	After training, the Rasa framework creates the model files in this location
sql/	Holds files consisting of sql queries. This is the layer for knowledge base connection

templates/	Different templates for formatting information for different purposes
actions.py	Routes all files in the custom_actions directory to a single end-point
config.yml	Holds Rasa-configurations. This file determines the pipelines and dialogue policies
domain.yml	Master data for possible intents, entities, actions, and default responses

These files and directories hold interesting information for this research. These files are referenced in the relevant chapters, so this is just for an overview to understand the semi-complex structure of the software. File structure has been modified from the Rasa "baseline" structure to accommodate more custom functions and actions.

7.1.2 Starting the bot

Laubot is installed into a proprietary server of the client. Software is deployed in docker containers, running the Rasa software, database as well as integrations.

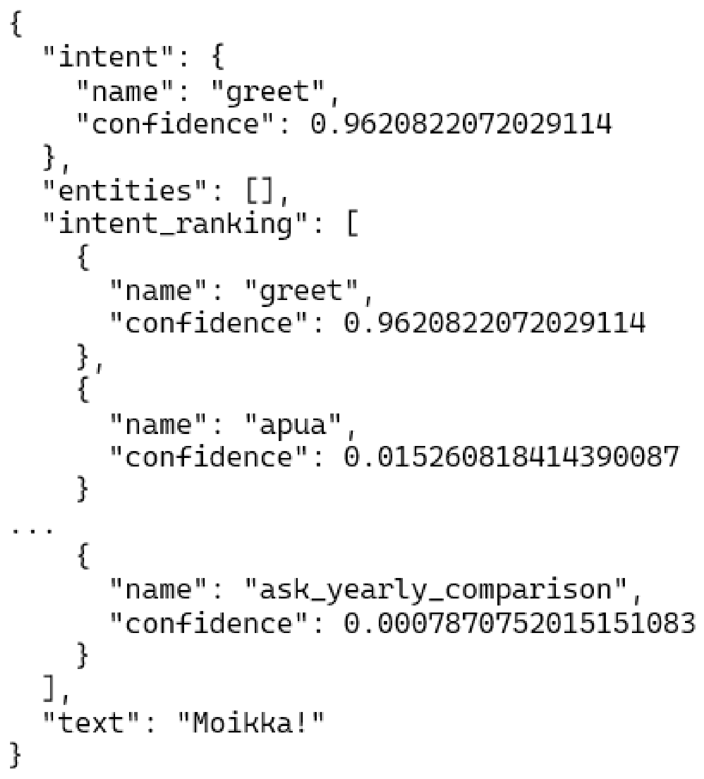
To run Laubot, or another conversational agent built on Rasa, certain conditions must be met. Training data, actions and configuration files need to be present. `rasa init` command takes care of these when starting development, which has already been done in Laubot. The developed bot can be launched with `rasa shell` command if the user wants to chat with it in the command line, or `rasa run` to start it in server mode.

7.1.3 From input to intent

One of the first problems this study is interested in, is how an input string is formed into structured data. As discussed more deeply in chapter 3, a multitude of commercial use chatbot tools utilize a structure of intents and entities to present the input to the following parts of the pipeline. This is also true for Rasa (Bocklisch et al., 2017; Rasa Technologies, 2020).

After running the program, we can communicate with it through a shell prompt. For security purposes, we cannot use the program with a database connection. This can be circumvented by utilizing command `rasa shell -nlu`. This runs the system in a shell-mode to communicate with it. But instead of continuing to the dialogue management parts, it only utilized the NLU part. Instead of a response it gives us a formatted answer of the structured input in *json* (JavaScript Object Notation).

Through a simple example of greeting the bot with a phrase "Moikka! /*(Hello!)*", we get a json representation of the found intents and their confidence scores. This is shown partly in Figure 12.



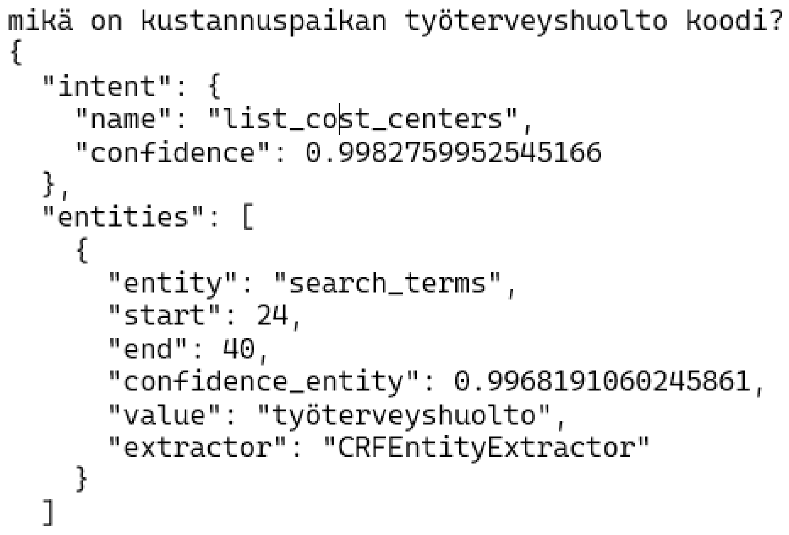
```
{
  "intent": {
    "name": "greet",
    "confidence": 0.9620822072029114
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "greet",
      "confidence": 0.9620822072029114
    },
    {
      "name": "apua",
      "confidence": 0.015260818414390087
    }
...
    {
      "name": "ask_yearly_comparison",
      "confidence": 0.0007870752015151083
    }
  ],
  "text": "Moikka!"
}
```

FIGURE 12 a structured represantation of a greeting in Laubot

So in this case, the bot recognized the given input as the intent "greet", which in our perspective is correct. It gave it a confidence score of ~0.96 which can be interpreted as the bot telling "I am 96% sure this input was meant to connect to this intent". It had the highest score, and it surpassed the NLU threshold set in the file *config.yml* file (shown in Table 4), so therefore this would be the intent passed to the core. In addition to the selected intent, we can see the subsequent intents listed based on the confidence score they got. This shortened version shows that a custom intent "*ask_yearly_comparison*" got only a confidence score of ~0.0008.

Entities are also listed for the recognized intent. But our greeting didn't hold any recognizable entities, it is listed as none. Both intents and entities are listed in the *domain.yml* file, and training data is provided for them in the *nlu/* directory. If we give a more representative example phrase, we get a different result (Figure 13). Assume the phrase *"Mikä on kustannuspaikan työterveys koodi? / (What is the code of the cost center occupational health care?)"*. This shows a populated list of entities. It recognized the entity *cost_centre* with a confidence score of ~0.996. Moreover, it lists the start and end points of the entity text within the phrase, as well as the value ("*työterveyshuolto/occupational health care*") and the extractor used to extract the entity. Different extractors are supported within Rasa.

CRFEntityExtractor (Conditional Random Field) is a Markov chain of different conditions to achieve named entity recognition (Chapter 2.2.4). It looks at multiple factors such as digits, lower- and upper-case characters to recognize the entity. The factors are configurable in Rasa. (Rasa Technologies, 2020)



```
mikä on kustannuspaikan työterveyshuolto koodi?
{
  "intent": {
    "name": "list_cost_centers",
    "confidence": 0.9982759952545166
  },
  "entities": [
    {
      "entity": "search_terms",
      "start": 24,
      "end": 40,
      "confidence_entity": 0.9968191060245861,
      "value": "työterveyshuolto",
      "extractor": "CRFEntityExtractor"
    }
  ]
```

FIGURE 13 Structured representation of an actual input query in Laubot

These show an example of "What Rasa sees" when receiving an intent. Before the bot can do this, it needs to be trained and configured, and this is an example of a pre-trained already built software. To keep in the research question on "how does a chatbot understand language?" a lot of the steps needed to achieve this are disregarded. But the most important parts for this are presented when discussing training and configuring in the later chapters.

7.1.4 From intent to action

After having the intent and entities separated through the NLU capabilities, next step is to form an "action" based on the received information. In Rasa, an action can be a response, form, default action or custom action (Rasa technologies, 2020). Custom actions are custom code that can do quite literally anything. Most commonly these are used to make API calls or database queries.

Laubot utilizes custom actions for completing query tasks, and some responses in simpler cases. In this case response is a predefined textual response, which is defined in the *domain.yml* file under the *responses* key. If an intent is matched to these, no custom code is executed, and the response is read straight from the *domain.yml* file. This would have been the case in Figure 12. In Figure 13, since the system needs to complete a database query to find information for the user, a custom action defined in the fold *custom_actions/* would have been completed.

What action to take is governed by ***policies.*** Policies are internally defined algorithms to predict what action is selected based on the given information. For each Rasa project, a set of different policies can be defined in the *config.yml* file. Each policy defined makes the same prediction, and assigns a number of actions a confidence score, similarly to the NLU-engine. The action with the highest confidence score is completed. For Laubot, the configuration file lists the policies shown in Table 5.

Table 5 Rasa action Policies of Laubot

Policy	**Explanation**
MemoizationPolicy	If *stories* are defined in the training data, memorization policy remembers the conversation that has happened beforehand. Threshold can be set to determine how many inputs of conversation are remembered. If the conversation matches a story, a confidence of 1 is given, which makes the bot to follow the story.
KerasPolicy	Keras is a software framework built on Tensorflow to build neural networks. KerasPolicy is a LSTM (Long-Short-Term Memory) neural network built internally with Keras to predict the suitable action(s).
MappingPolicy	This policy is used to determine predefined triggers within intents to map straight to predefined action. If intent defined in the *domain.yml* file is given a trigger such as *greet: {triggers: utter_goodbye},* the intent will automatically match to the given action, in this case *utter_goodbye,* which is a response action.
FallbackPolicy	This policy defines an action to take, if the intent recognition does not give any intent with a confidence above a selected threshold, or if none of the other policies predict a suitable action above the predetermined core threshold.

So for Laubot, each of these policies are considered and compared before action is taken, with the exception of *FallbackPolicy* which is only considered if none of the other ones do not produce a high enough confidence score. Most of the time *MemoizationPolicy* is the determining policy since actions are often connected to stories.

7.1.5 From action to response

For most cases, Laubot follows a *story*. Story if a markup language representation of *intents, entities,* and *actions* to form a skeleton for a conversation for the bot to follow. When a bot recognizes an intent, (in addition to other methods) it can connect to an action to execute through stories. (Rasa Technologies, 2020)

When Laubot figures out an action, either response is given, or custom code is executed. Custom action is written in Laubot by extending an *Action* class from the Rasa software development kit. In addition to importin the Action-class, a *tracker* (See Figure 11) is also imported. Tracker is a storing mechanism for a single user and the conversation. Trackers can be used to access the bot's memory in custom actions (Rasa Technologies, 2020).

Laubot uses tracker to access *slots*, which are key-value pairs to pass information to the custom actions. In addition to many things mentioned before, slots are defined in the *domain.yml* file. Slots are used as a storing place for variable data, such as *search_terms, timespan, receipttype*. While not being the sole purpose of slots, they can be considered as a way to make data gathered from entities into variables to use. This data can be then used in a search query to a knowledge base. Laubot uses slots for specific query data, such as the ones mentioned before, to create specific database queries.

SQL-folder contains database queries, which are then programmatically executed based on the search terms. This is the main function of custom actions in Laubot. Since this is an information retrieval bot, the main function is to query data. On the contrary, the most prominent use of commercial chatbots or conversational agents tend to be of assisting nature. They usually give directions to the place to find information, whereas Laubot aims to provide the information if possible.

Response given is determined by the information found. Most cases a response holds data from the database, either to represent or to perform a calculation on. This response is then given as a *json*-format as the bot's user interface is the chat software Microsoft Teams. *Templates* directory in the codebase contains templates for this. When "leaving the Rasa architecture", most of the times data is not in a representable form, as visualization or representation is handled by another software. This gives additional capabilities of a fully featured chat program to make augmentations.

7.1.6 Domain and configuration

Domain.yml and *config.yml* are a common occurrence when explaining the structure of the system. These files define the settings and the brains for the system being built. This is the starting point if the aim is to understand a bot built with Rasa.

Domain is a concept touched several times throughout this thesis. It serves as the universe where the bot functions. As established, conversational agents are often built around a closed domain. They have 'knowledge' on certain topics instead of the general world around them. This is a way to limit the functionalities needed for the bot to appear intelligent, which subsequently makes the development easier. Laubot does not need to know about birds, it needs to know about the financial information of the municipality.

The domain file holds a list of intents, entities, actions, responses, and slots, as well as configuration part for sessions. (Rasa Technologies, 2020) Laubot lists intents such as *list_cost_centers, ask_invoice_pdf,* and *ask_accounts.* These custom intents must have training data to learn how user might convey the same information. for *list_cost_centers* intent, a training file *nlu_list_cost_centers.md* is provided. It holds over a hundred different training phrases with varying entities. When the bot is trained, it trains the NLU engine with these phrases, and learns to match similar phrases. Rasa core looks at the list of the actions listed here when deciding which action or response to take.

Configuration holds information that defines the inner mechanics of the bot's intelligent features. Mainly it consists of two parts: the NLU engine configuration, and the Core configuration. Core configuration was presented in chapter 6.1.4. The NLU configuration is very short for Laubot, as it only defines the language and the pipeline.

7.1.7 Training

Laubot uses *supervised_embeddings* pipeline to train the NLU engine. The pipeline is configurable, as well as customizable. Custom components can be added to pipelines. This means the Rasa NLU system could be built on partly-, or entirely custom natural language processing engine. Rasa provides certain pipelines out-of-the-box and has different capabilities for different languages. Since Laubot is developed to work entirely in Finnish, the options are unfortunately limited compared to more major languages. Chapter 4.3 discussed briefly about built-in components of Rasa.

Laubot's supervised embeddings mean that it does not use any preexisting model to train the NLU-engine. It learns everything it can from the training data provided to it. This has two advantages why it is used in this case:

- It is language independent, as it learns 'meaning' from the context of training data.
- It suits a domain-specific need, and Laubot is *extremely* domain specific

Supervised_embeddings is a template pipeline. This means it includes components which load automatically without being specified in the *config.yml* file. This can be seen in action in Figure 13, which shows an entity extractor, while none is present in the configuration file. This pipeline template automatically loads it, and several others presented in Table 6. This is the core structure of what Laubot does to a text input when classifying it into an intent before passing the task onto the dialogue management.

Table 6 Laubot NLU model pipeline

Component	**Explanation**
WhitespaceTokenizer	Splits input input sentence/phrase into tokens by using whitespace as a delimiter. Tokenization explained in chapter 2.2.1.
CRFEntityExtractor	Acts as named entity recognition based on the entities present in the training data. NER explained in chapter 2.2.4.
EntitySynonymMapper	Maps entity synonyms which can be described in *nlu.md* file. This helps to deduce similar representations of the same entity into the same one
CountVectorsFeaturizer	Creates a bag-of-words representation based on the previous components. Produces a numerical representation matrix of tokens, intent, and responses to be used in a classifier. This matrix also includes a representation of the count of distinct words of training data appears in the given input.
EmbeddingIntentClassifier	Neural network classifier that is trained against vector representation of the intent labels, which calculates distance between the vectors from previous component against them.

When Laubot is trained, it trains this pipeline. More specifically, when the NLU engine is trained it trains the intent classifier. This trained network is then used to predict intent for each user input by feeding and input vector and network calculating a confidence score for each possible intent.

Since the pipelines are customizable and extendable, they can get increasingly complicated. More commonly spoken languages contain several pre-built pipelines and components to extend on. Rasa uses many functions of the SpaCy library. SpaCy is a natural language processing library, built to be a production ready open-source NLP library. (Explosion/Spacy, 2020) Most notably the ability to utilize pre-trained word embeddings would give word vectorized words predefined feature vectors, that could include data about grammatical and semantical features of a word. But for many cases, simpler pipelines can achieve adequate results without them.

7.2 Actual scenario

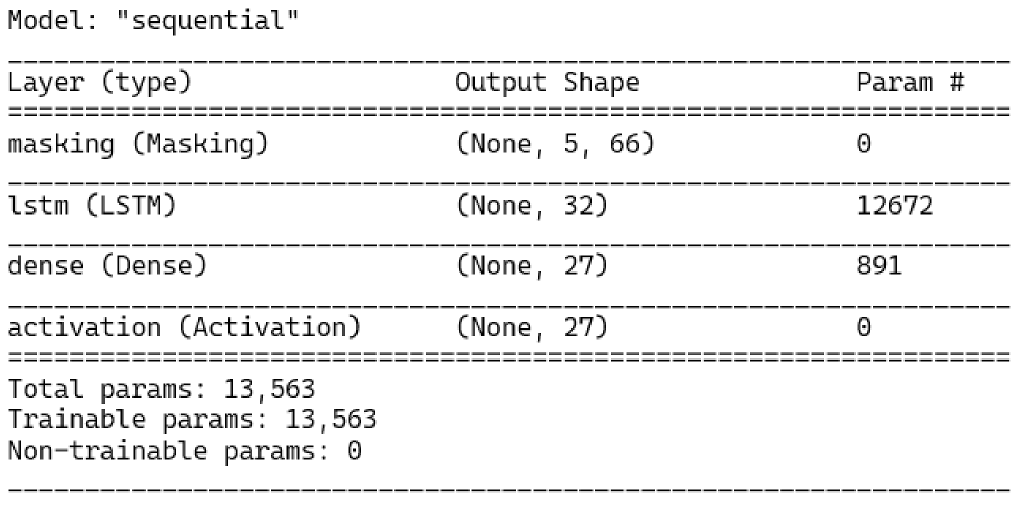
This chapter investigates an example scenario with Laubot. This is somewhat generalized to give an overlook on how this, or a similarly built conversational agent would work in an actual scenario. We can get additional information not directly present in the documentation or source code by utilizing certain debugging parameters that Rasa provides in the command line interface. This chapter takes more chronological order compared to chapter 6.1.

7.2.1 Training

Continuing on chapter 7.1.7, training is the process of calculating *weights* to neural networks present in the architecture. To visualize parts of this, `rasa train –debug` can be used. This prints additional statements about the training pipeline. We can use snippets of the logs to investigate the training process further.

First the core-model (dialogue management, response generation) is trained. The interesting part of this process is the training of the pipeline used to connect intents to actions, presented in Table 5. The "intelligent" part of this pipeline is the Keras Policy, which trains a sequential (meaning a network where each layer gets one input tensor and produces one output tensor) LSTM model, shown in Figure 14. This shows that the training phase teaches a neural network with four layers; masking layer, LSTM-layer, dense layer, and activation layer. Shape presented in the figure reflects the amount of 'neurons' in each layer. The most important part about this structure is to note the number of neurons in the activation layer. When counting all possible actions (including responses) defined in the *domain.yml* file, it comes out as 27. This layer takes an input tensor, and through activation functions give out a confidence score representation for each possible action.

Why the architecture of the network is how it is, is not defined. Building optimal neural networks is a highly time-consuming task that requires deep understanding in statistical mathematics. Rasa provides no information on the architecture of built-in networks.



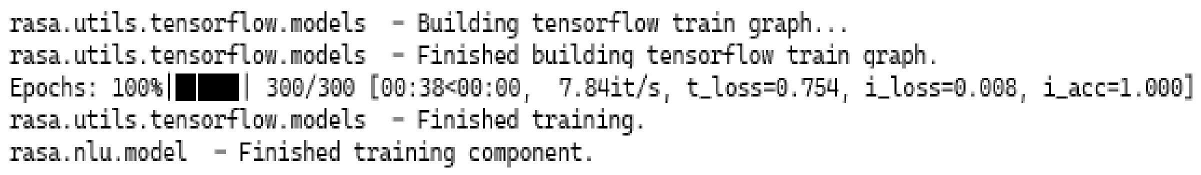
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
masking (Masking)            (None, 5, 66)             0
_________________________________________________________________
lstm (LSTM)                  (None, 32)                12672
_________________________________________________________________
dense (Dense)                (None, 27)                891
_________________________________________________________________
activation (Activation)      (None, 27)                0
=================================================================
Total params: 13,563
Trainable params: 13,563
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 14 Network architecture for Keras Policy in Laubot

Following the core model training, the NLU model is trained. As shown in table 6, Laubot trains an *embeddingIntentClassifier*. We learned that this network receives and input from the previous part of the pipeline, which is *CountVectorsFeaturizer* in this case. This produces a vectorized bag-of-words representation of the input and feeds it to the network. This network is trained at this phase, and produces information presented in Figure 15. Log is different from Figure 14, since this part of the pipeline is built on Tensorflow, while the core policy was built on Keras. These frameworks give out information differently.

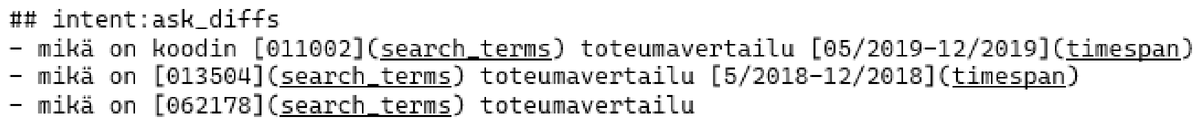
So this part trained the final policy, intent classifier, in the NLU models pipeline in Laubot. It achieved a 100% accuracy on the given training data from the *nlu/* directory. This is not that surprising, since the training data is not that vast and very domain specific. Since the pipeline is supervised, it is trained with the examples given in the directory against the intents presented in each file. Example of how this is structured is presented in Figure 16. Training data is categorized under each intent as their 'label'. The amount of training data is the number of examples given.

The architecture of this network is not readily available, due to it being deprecated from the Rasa stack. Legacy documentation does not list any default values.



```
rasa.utils.tensorflow.models  - Building tensorflow train graph...
rasa.utils.tensorflow.models  - Finished building tensorflow train graph.
Epochs: 100%|████| 300/300 [00:38<00:00,  7.84it/s, t_loss=0.754, i_loss=0.008, i_acc=1.000]
rasa.utils.tensorflow.models  - Finished training.
rasa.nlu.model  - Finished training component.
```

FIGURE 15 Laubot NLU model intent classifier training



```
## intent:ask_diffs
- mikä on koodin [011002](search_terms) toteumavertailu [05/2019-12/2019](timespan)
- mikä on [013504](search_terms) toteumavertailu [5/2018-12/2018](timespan)
- mikä on [062178](search_terms) toteumavertailu
```

FIGURE 16 Example of a training file in Laubot

With additional features of the Rasa command line interface, we can print visualized information of the structure the training creates. As learned in chapter 7.1, stories play a major role in how Rasa decides the following action. Figure 17 presents part of the story connections in Laubot. This is not particularly interesting, since Laubot does not rely heavily on stories, since it aims to achieve quick answers to users right away, instead of keeping conversations. Only some intents are designed to continue a conversation by default.

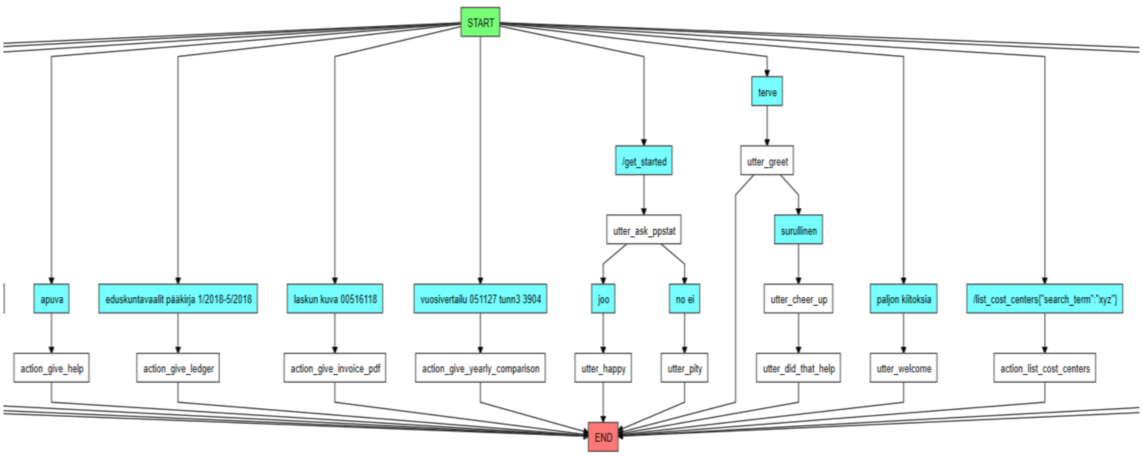


FIGURE 17 Story connections in Laubot

7.2.2 Example, Laubot natural language understanding engine

After having a trained bot, the user can chat with it after starting it. In normal conditions the software would be running already in a server and connected through HTTP. For this purpose using command line interface is enough. And as mentioned earlier, access to database is not possible, so the results will be limited to raw data of the NLU system. This is just to show a conclusive representation to what happens to an input sentence in Laubot.

Assume the question "Mikä on kustannuspaikan työterveyshuolto budjetti välillä 01/2019-12/2019? / *What is the budget of cost center occupational health care during 01/2019-12/2019?)*". What happens to this input text when it is passed to Laubot?

First it enters the NLU pipeline. Referring to Table 6, the first part is the tokenization using white space tokenization. We can take a closer look into how this looks by debugging the Rasa source code locally. As expected, whitespace tokenization contains runs a function that returns a list of the words in an array they were in the input text. Additionally, the list includes the lengths of the words as well as their positioning index in the input string.

Next is the *CRFEntityExtractor* entity extraction. This consumes the data from the tokenization and compares it to the pre-defined entities. It returns a list of detected entities, including the entity, value, length, position, and confidence. Json representation of the Python dictionary returned is the following:

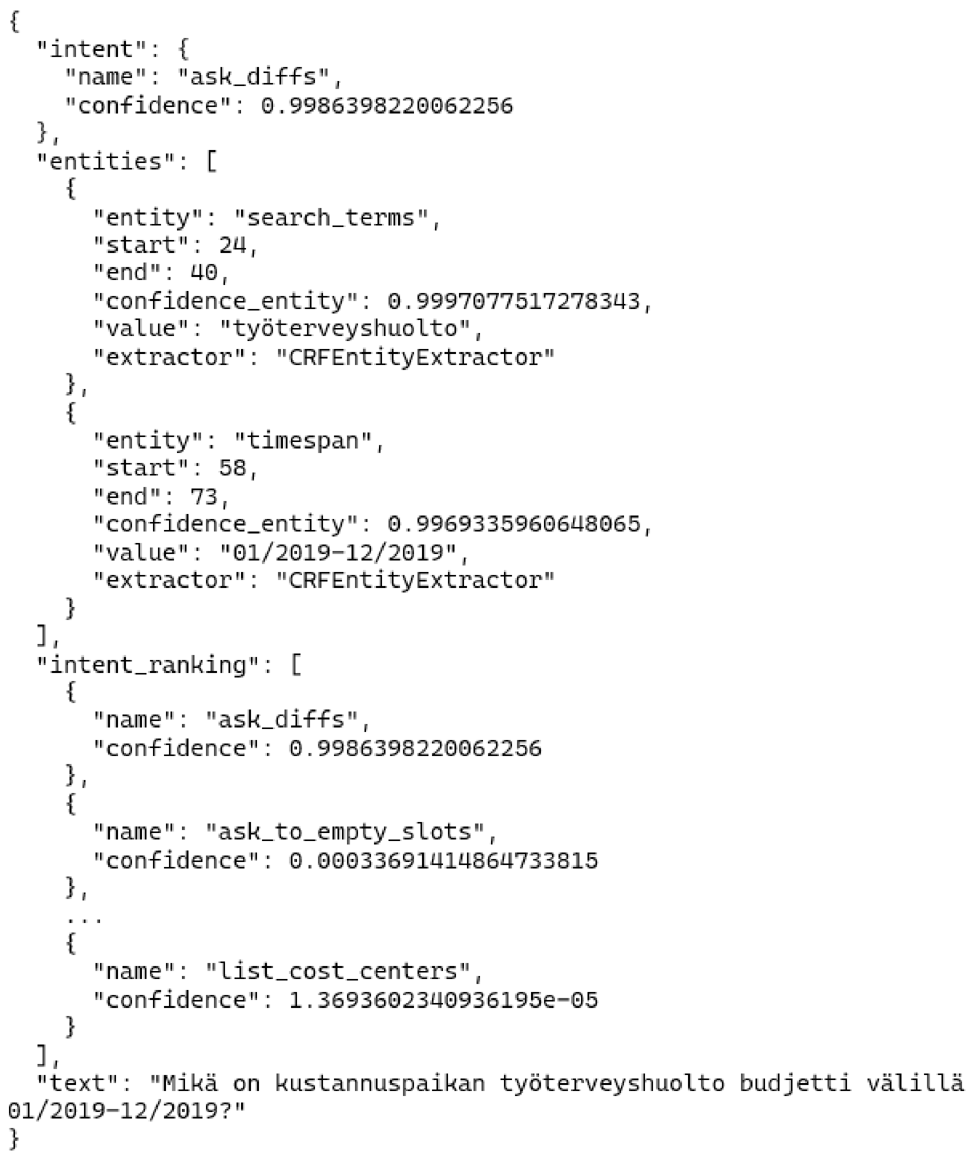
```
{
    [
        {
            "entity": "search_terms",
            "start": 24,
            "end": 40,
            "confidence_entity": 0.9997077283626788
            "value": "työterveyshuolto"
        },
        {
            "entity": "timespan",
            "start": 58,
            "end": 73,
            "confidence_entity": 0.9969335960643355
            "value": "01/2019-12/2019"

        }
    ]
}
```

This is then passed to *EntitySynonymMapper*. For Laubot, there are no entity synonyms defined, so this part does not produce any difference. Entity synonym mapper is a fairly simple algorithm, that checks the values of the give entities from the previous step and compares the "stripped down" values of the "value" attribute to the predefined synonyms for an entity. If they are a match it replaces the value with the value defined as the synonym. Updated representation of the entities is returned, which means no change for Laubot.

Next step is the *CountVectorsFeaturizer.* This produces a sparse feature matrix for the classifier. Sparse matrix is a multi-dimensional numerical matrix representing the different tokens of the input. Rasa uses *Scikit learn* open software to produce these feature vectors. (Rasa Technologies, 2020) Sklearn's (other name for scikit learn) CountVectorizer is used to produce feature matrices of an input corpus, or the input tokens and entities most importantly in this case. Feature vector is a numerical representation of an object to be used (most commonly) in a neural network or another statistical algorithm. In the case of Laubot, it functions as a numerical representation of tokens and their cumulative occurrences in the input. The featurizer does the same for intents and responses to be used in the following step, as well as response selection in the later phases of the conversation (Rasa Technologies, 2020). Counts of tokens become the "defining features" for the next step, which is intent classification.

Technically speaking, the featurizer does not produce an output, but works in close conjunction with the following step, *EmbeddingIntentClassification.* At this point, the input text "Mikä on kustannuspaikan työterveyshuolto budjetti välillä 01/2019-12/2019?" has been tokenized, it has it's entities recognized and it is transformed into a numerical representation of the features (tokens in this case[3]). The neural network then assigns the final "score" of intent classification confidence. This data and additional information, represented in figure 18, is passed to the Rasa core for dialogue management. It recognized the intent correctly to the desired *ask_diffs* intent. It recognized two entities *search_terms* and *timespan.* All these are custom-made features of Laubot to reflect the needs of the user.

[3] If additional steps for lemmatization (chapter 2.2.3) or other featurization (2.2.7) were provided in previous stages of the pipeline, features could be different.



```
{
  "intent": {
    "name": "ask_diffs",
    "confidence": 0.9986398220062256
  },
  "entities": [
    {
      "entity": "search_terms",
      "start": 24,
      "end": 40,
      "confidence_entity": 0.9997077517278343,
      "value": "työterveyshuolto",
      "extractor": "CRFEntityExtractor"
    },
    {
      "entity": "timespan",
      "start": 58,
      "end": 73,
      "confidence_entity": 0.9969335960648065,
      "value": "01/2019-12/2019",
      "extractor": "CRFEntityExtractor"
    }
  ],
  "intent_ranking": [
    {
      "name": "ask_diffs",
      "confidence": 0.9986398220062256
    },
    {
      "name": "ask_to_empty_slots",
      "confidence": 0.00033691414864733815
    },
    ...
    {
      "name": "list_cost_centers",
      "confidence": 1.3693602340936195e-05
    }
  ],
  "text": "Mikä on kustannuspaikan työterveyshuolto budjetti välillä
01/2019-12/2019?"
}
```

FIGURE 18 Final result of the example in the Laubot's NLU model

8 Results, analysis, discussion

This chapter discusses the different aspects found in the case study, and the connections to literature and previous work. Furthermore, some discussion is in place for the usability and accuracy of this case, and the general usage of bots.

The question "How does a chatbot process language?" turned out to be a multi-faceted question with no clear answers. The general answer is a vast landscape of methods and ideas, all very estranged from the development work that goes into conversational agents. Case study provided with a reasonable answer to analyze upon.

8.1 Results and analysis

To answer the very difficult question this research builds upon, a literature review was conducted to learn about chatbots and text processing. While conversational agents are the main focus, understanding of NLP methods and technologies is absolutely crucial to understanding the inner workings of modern chatbots. A lot of the modern chatbot building tools rely heavily on different methods that were originally developed for document processing purposes. A conscious effort was made to not include human interaction research in the reviews, since the aim is to gain a technical view, regardless of the use cases or perceptions of quality.

The literature review resulted in a general understanding of different technical structures and architectures of conversational agents. Nevertheless, the answer to the question was not quite what was expected, as the sheer number of different methods to achieve conversational artificial intelligence was overwhelming, and provided very little clear distinctions towards a general answer on how do chatbots process language. Some distinctions were found, especially depending on the use case. To achieve general AI-imitating chatterbots, most beneficial methods seemed to be pattern matching algorithms utilizing vast amounts of data. Simultaneously more scalable architectures utilizing a set of different text processing and machine learning algorithms seem to better support domain-specific work, especially in commercial usages.

A case study was conducted on one commercial usage, domain specific chatbot. The setting of the chatbot development landscape made this the best option to gain deeper understanding of bot technologies. Case study was conducted as a combination of document analysis and a supporting interview. Access to the source code of both the technology stack and the underlying systems was crucial to the results of the case study.

What was found in the case study was a moderate connection to underlying language processing techniques and a heavy connection to the domain specific language utilized in the chatbot. Laubot processes text by relying on manually defined conversation paths (stories), intents, entities, and responses. Training data is also hand-crafted (however, this process could be automated to a degree). This works as a foundation for getting more knowledgeable on the relevant topics, as well as achieving scalability. Specific hand-crafted training examples also benefits the bot at answering exact and specific questions correctly. Bugs and missing features can be easily fixed, as the underlying data the bot uses to teach itself is disconnected from the more complicated language processing task. This creates opportunities to maintain more deterministic results in specific cases, as the more generative end-to-end models and data-reliant pattern matching chatbots are difficult to fine-tune to achieve a specific outcome.

Laubot utilizes a set of language processing tasks to convert natural language input text into a structured (and numerical) format. This includes tokenization, named entity recognition, and feature extraction. This data is used in a generative feed-forward neural network stack to match inputs to predefined intents and entities. Dialogue management after the intent is found works in a similar fashion. It uses a neural network to map the input to an output, coupled with deterministic triggers to "circumvent" nondeterministic behavior of neural networks.

This corresponds with the outcome of the literature review. Modern development tools designed for commercial use benefit from a more scalable architecture, and it is clearly visible in the system architecture of Rasa and Laubot. This achieves better accuracy and maintainability for knowledge-based domain specific conversational agents, due to the sheer control of the pipelines and the customizable components.

While only one bot was studied, this is somewhat extendable to understanding bot structures in general. Especially the commercial use bot development tools were found to be very similar in structure. Commercial development frameworks do not necessarily want to provide their technology stack to the public, since it might be of competitive advantage compared to other providers. This was found to affect the documentation as well, where most subjects were not explained deeply enough to gain understanding of the technical functionalities and limitations.

8.2 Discussion

Only one chatbot was studied for the case-research. This is not optimal, but nearly impossible circumvent. To achieve a level of precision in the technical analysis requires access to the source code of the system, as well as the framework. This kind of access is not usually available, and due to the landscape of development tools used in commercial systems, finding more cases to achieve the same level of technical access turned out to be impossible. Another method would be to build a separate system, which would provide even more detailed access to the technical components, as well as the ability to produce additional test cases for accuracy and maintainability. This is better left to further research.

This thesis does not discuss the technical limitations to an extent that might interfere with development. While it would be helpful to understand the current technical limitations of conversational systems, a simple guideline does not exist. Where limitations apply is solely dependent on the general architecture of a system. The technical theme-interview raised issues on Laubots current state, where performance was enough to meet the expectations when entity-count was low, the performance took a significant hit when more and more entities were introduced in a single input. Whether the system can be optimized further to meet the increasing expectations is a question unanswered. If hard limits exist, a different system altogether would be more suitable for this use case. Additionally, a systematic review on different commercial systems could also answer the research question form a different perspective. This would obviously mean a general view instead of a detailed technical view but would certainly be of great support for future research on the subject.

The case study also provided its own limitations. While access to the source code was granted, access to the actual data or database were not. These challenged were circumvented by the debugging tools and running the code locally, instead of the very confined environment. But how Rasa is built, this did not hinder the research as much as it could have. Accessing any databases or endpoints is the final step in the pipelines, and everything related to text processing is done beforehand.

The project studied is an ongoing project as the time of writing, and a version used in this thesis might not reflect the future changes or a "final product". This deemed not to be an issue, since the project and the system is already well established, and the study does not rely on a finalized product. Moreover, agile projects such as this, especially of this scale, tend to continue further than a single "delivery date".

Since the project was already established in an earlier date, certain parts of the technologies provided by the Rasa framework have been updated since. Certain parts of the pipeline are already deprecated. This does not provide issues to already built applications and they can be used as is, but documentation of these components is not as readily available as the new ones. Some information was fetched from the legacy documentation, as well as a version of the open source code that might not be in the current version of the system.

8.3 Further research

This thesis opened more questions than it answered. The landscape of conversational agents is a vast and disconnected collection of literature and commercial use cases. Both literature review and the case study showed a major disconnect between underlying technologies and development of chatbots.

Due to the plethora of different approaches to chatbots, optimization issues are not easily researchable. While research exists, it can be nearly impossible to find the appropriate sources to apply to a certain project. This creates the problem that also works as the motivation for this study. Continuing this type of research of connecting the underlying technologies and techniques to the actual development work, the obvious next step would be to optimize the systems being built. This type of research should focus on the singular components of the framework being used and finding how much they contribute to the final outcome. Understanding this can help finding optimization strategies for chatbots.

The concept of 'hard limit' mentioned in chapter 8.2 is another issue that is related to optimization. Understanding the technologies helps understanding the capabilities. Based on this type of research, more detailed research can be done on the question "is a chatbot right for this problem?" or "Can a chatbot solve this problem?". While new knowledge and techniques are found on a frequent basis, they may not solve the underlying fundaments in a problem being solved. This creates a balancing act between usability and the capabilities of a chatbot. If the chatbot is not 'smart' enough, the usability suffers. It needs to ask for clarifications or produce more incorrect outcomes. If the usability suffers enough, a more conventional user interface would most likely solve the issue better.

Lastly, a proper study on the effects of increased focus on the target language could benefit the optimization issue. Even for semi-marginal languages like Finnish, a good body of NLP-literature exists. However, this knowledge is not appropriately applied to chatbots. More meta-level information could be extracted from the input by using advanced NLP-components designed for a specific input-language. This could benefit the outcome, since more 'popular' languages already have improved pipelines in place.

9 Conclusion

This thesis looked into the world of conversational agents, also known as chatbots, from a detailed technical perspective. While the vast landscape of literature is readily available for the topics revolving around chatbots and "artificial intelligence", it often falls into a systematic division where technically oriented literature does not reach the final use cases. By researching the source code and documentation of Laubot in a systematic manner, a more detailed view connecting the theory and practice was achieved.

Conversational agents process text in multitude of different manners. Domain specific chatbots rely on scalability over perceived artificial intelligence. Case study of this thesis falls into this category. Laubot processes text through a processing pipeline, that functions as a formatter for machine learning algorithms. Dialogue management, or response generation, is done through a system of intents and entities reflecting the original inputs and the additional data relating to said intent. This data is again utilized in selecting the appropriate response or an action for the input. Laubot is connected to a relational database that provides data if the user is requesting any. This is done to customized actions, that take place if the core finds that action to be to most appropriate one. User can then continue conversating with the bot, which might start this cycle again, or fit into a pre-defined conversation paths, or stories.

The literature review partly gives basis to the case study, and partly support the findings of the analysis. It was found how commercial usage bots differ from certain generative or pattern matching bots, which sole purpose is to function as an intelligent assistant or a chatter bot. The literature review provided an important foundation on understanding the techniques found in the case study. For many chatbot applications and the development tools, the natural language processing is not the interesting or driving factor for success, but rather a building block to build more sophisticated systems on. However, it can be assumed that understanding the fundamental structures of text processing while developing a chatbot can be an excellent resource for optimizing the performance and recognizing faults in the system in the first place.

LITERATURE

Abdul-Kader, S. A., & Woods, J. C. (2015). Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, *6*(7).

Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural language interfaces to databases-an introduction. *arXiv preprint cmp-lg/9503016*.

Arsovski, S., Cheok, A. D., & MuniruIdris, M. R. A. R. (2017). ANALYSIS OF THE CHATBOT OPEN SOURCE LANGUAGES AIML AND CHATSCRIPT: A Review. In 9th DQM International Conference on life cycle engineering and management.

Bocklisch, T., Faulkner, J., Pawlowski, N., & Nichol, A. (2017). Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*.

Brandtzaeg, P. B., & Følstad, A. (2017, November). Why people use chatbots. In *International Conference on Internet Science* (pp. 377-392). Springer, Cham.

Braun, D., Mendez, A. H., Matthes, F., & Langen, M. (2017, August). Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue* (pp. 174-185).

Cahn, J. (2017). CHATBOT: Architecture, design, & development. *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science*.

Canonico, M., & De Russis, L. (2018). A comparison and critique of natural language understanding tools. *Cloud Computing*, *2018*, 120.

Chowdhary, K. R. (2020). Natural language processing. In *Fundamentals of Artificial Intelligence* (pp. 603-649). Springer, New Delhi.

Goddeau, D., Meng, H., Polifroni, J., Seneff, S., & Busayapongchai, S. (1996, October). A form-based dialogue manager for spoken language applications. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96* (Vol. 2, pp. 701-704). IEEE.

Gómez-Rodríguez, C., Alonso-Alonso, I., & Vilares, D. (2019). How important is syntactic parsing accuracy? An empirical evaluation on rule-based sentiment analysis. *Artificial Intelligence Review*, *52*(3), 2081-2097.

Huang, J., Zhou, M., & Yang, D. (2007, January). Extracting Chatbot Knowledge from Online Discussion Forums. In *IJCAI* (Vol. 7, pp. 423-428).

Korenius, T., Laurikkala, J., Järvelin, K., & Juhola, M. (2004, November). Stemming and lemmatization in the clustering of finnish text documents. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management* (pp. 625-633).

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Lee, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M., & Jurafsky, D. (2013). Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational linguistics*, *39*(4), 885-916.

Lee, K., He, L., Lewis, M., & Zettlemoyer, L. (2017). End-to-end neural coreference resolution. *arXiv preprint arXiv:1707.07045*.

Liddy, E. D. (2001). Natural language processing.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014, June). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations* (pp. 55-60).

Masche, J., & Le, N. T. (2017, June). A review of technologies for conversational systems. In *International Conference on Computer Science, Applied Mathematics and Applications* (pp. 212-225). Springer, Cham.

Mauldin, M. L. (1994, August). Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI* (Vol. 94, pp. 16-21).

McKeown, K. (1982). The TEXT system for natural language generation: An overview.

Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, *30*(1), 3-26.

Nivre, J., De Marneffe, M. C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., ... & Tsarfaty, R. (2016, May). Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)* (pp. 1659-1666).

Plank, B., Søgaard, A., & Goldberg, Y. (2016). Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *arXiv preprint arXiv:1604.05529*.

Pundge, A. M., Khillare, S. A., & Mahender, C. N. (2016). Question answering system, approaches and techniques: a review. *International Journal of Computer Applications*, *141*(3), 0975-8887.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering, 14(2), 131.

Schmid, H. (1994). Part-of-speech tagging with neural networks. *arXiv preprint cmp-lg/9410018*.

Schwaber, K., & Beedle, M. (2002). Agile software development with Scrum (Vol. 1). Upper Saddle River: Prentice Hall.

Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P., & Zue, V. (1998). Galaxy-II: A reference architecture for conversational system development. In *Fifth International Conference on Spoken Language Processing*.

Shum, H. Y., He, X. D., & Li, D. (2018). From Eliza to XiaoIce: challenges and opportunities with social chatbots. *Frontiers of Information Technology & Electronic Engineering*, *19*(1), 10-26.

Soares, M. A. C., & Parreiras, F. S. (2018). A literature review on question answering techniques, paradigms and systems. *Journal of King Saud University-Computer and Information Sciences*.

Song, Y., Yan, R., Li, C. T., Nie, J. Y., Zhang, M., & Zhao, D. (2018). An Ensemble of Retrieval-Based and Generation-Based Human-Computer Conversation Systems.

Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., ... & Meteer, M. (2000). Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational linguistics*, *26*(3), 339-373.

Straka, M., Hajic, J., & Straková, J. (2016, May). UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)* (pp. 4290-4297).

Sun, S., Luo, C., & Chen, J. (2017). A review of natural language processing techniques for opinion mining systems. *Information fusion*, *36*, 10-25.

Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003, May). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 conference of the North American chapter of the association for computational linguistics on human language technology-volume 1* (pp. 173-180). Association for Computational Linguistics.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36-45.

Williams, J., Raux, A., & Henderson, M. (2016). The dialog state tracking challenge series: A review. Dialogue & Discourse, 7(3), 4-33.

Xing, C., Wu, W., Wu, Y., Liu, J., Huang, Y., Zhou, M., & Ma, W. Y. (2017, February). Topic aware neural response generation. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Yang, L., Hu, J., Qiu, M., Qu, C., Gao, J., Croft, W. B., ... & Liu, J. (2019, November). A hybrid retrieval-generation neural conversation model. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (pp. 1341-1350).

Zamora, J. (2017, October). I'm sorry, dave, i'm afraid i can't do that: Chatbot perception and expectations. In *Proceedings of the 5th International Conference on Human Agent Interaction* (pp. 253-260).

https://www.apple.com/siri/

https://www.cleverscript.com/

https://cloud.google.com/dialogflow

https://www.laukaa.fi/tietoa-laukaasta/hankinnat

https://rasa.com/docs/

ATTACHMENT 1 : FULL OUTPUT OF LAUBOT RASA SHELL NLU WITH A GREETING

```
{
  "intent": {
    "name": "greet",
    "confidence": 0.9620822072029114
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "greet",
      "confidence": 0.9620822072029114
    },
    {
      "name": "apua",
      "confidence": 0.015260818414390087
    },
    {
      "name": "ask_invoice_pdf",
      "confidence": 0.004917561542242765
    },
    {
      "name": "bot_challenge",
      "confidence": 0.0049173650331795216
    },
    {
      "name": "thank",
      "confidence": 0.004144945181906223
    },
    {
      "name": "deny",
      "confidence": 0.003176956670358777
    },
    {
      "name": "goodbye",
      "confidence": 0.002531921025365591
    },
    {
      "name": "ask_to_empty_slots",
      "confidence": 0.0012326458236202598
    },
    {
      "name": "mood_unhappy",
      "confidence": 0.0009485589107498527
    },
    {
      "name": "ask_yearly_comparison",
      "confidence": 0.0007870752015151083
    }
  ],
  "text": "Moikka!"
}
```