Fachhochschul-Masterstudiengang
SOFTWARE ENGINEERING
A-4232 Hagenberg, Austria

Chatbot Prototype Development and Evaluation form a Development Perspective in a Given Domain

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science in Engineering

Eingereicht von

Gerald Spenlingwimmer

Betreuer: FH-Prof. PD DI Dr. Stephan Dreiseitl

Hagenberg, Juli 2020

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, 07.July.2020

Gerald Spenlingwimmer

Contents

Abstract **iv**

Kurzfassung **v**

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Motivation . . . 2
1.3 Company . . . 3
1.4 Research Questions . . . 3
1.5 Prerequisites . . . 3
1.6 Expected Results . . . 3

2 Basics **5**
2.1 Chatbot . . . 5
2.2 Chatbot/NLU Frameworks . . . 6
2.3 Suitable Problems for Chatbots . . . 6
2.4 Intent . . . 7
2.5 Utterance . . . 8
2.6 Entity . . . 9
2.7 Chatbot Response . . . 10
2.8 Story . . . 10
2.9 Webhook . . . 11
2.10 Domain . . . 11
2.11 Evaluation of NLP capabilities . . . 12
2.12 Natural Language Processing/Understanding . . . 12

3 State of the Art **13**

4 Methodology **22**
4.1 Approach . . . 22
4.2 Training and Test Setup . . . 23
4.3 Course of Action . . . 23
4.4 Comparison and Evaluation . . . 24

5 Design **26**
5.1 Architecture . . . 26
5.2 Story/Flowchart . . . 27
5.3 Intents, Entities and Utterances . . . 28
5.4 Problem Analysis . . . 29

5.5 Webhook 30
5.6 Messenger UI 30

6 Prototype Implementation **32**
6.1 Dialogflow 32
6.1.1 Intents and Entities 32
6.1.2 Webhook 33
6.2 Watson Assistant 34
6.2.1 Intents and Entities 34
6.2.2 Dialog and Webhook 35
6.3 Rasa 36
6.4 LUIS 40
6.5 Frontend 41
6.6 Webhook 41

7 Result **42**
7.1 Framework Comparison 42

8 Discussion **59**

9 Conclusion and Outlook **60**
9.1 Conclusion 60
9.2 Outlook 61

10 Appendix **63**

Bibliography **72**

List of Figures **73**

List of Tables **74**

List of Listings **75**

Abstract

Kurzfassung

Chapter 1

Introduction

1.1 Introduction

A chatbot is a conversational agent able to communicate with users in turns using natural language [30, 31, 24, 22]. Chatbots are becoming more and more popular in recent years. The three main reasons for the growing popularity are the rise of chat systems in everyday life, the advancements in machine learning, and the advancements in NLU in recent years. [15] The original chatbot technologies required expertise in machine learning. Hence, a machine learning expert was required for the development of a chatbot and the broad public of developers was unable to implement chatbot systems. Thanks to the recent advances in machine learning no machine learning skills are required for the development of chatbots anymore. This makes chatbot systems attractive to programmers. Another reason for the increasing attention is the development effort which has been reduced greatly in the last few years. Nowadays small to no programming knowledge is needed to develop and implement a chatbot [15]. This means that people without ML and programming skills are still able to develop a chatbot. Because of these characteristics, chatbots are becoming popular in the customer support section where they are used to replace humans [20]. They are also used to increase the user acceptance of FAQ pages. The major benefits of chatbot systems are that they never complain, are cost-efficient, and are available 24 hours a day seven days a week. Chatbots can greatly reduce the costs of customer services and increase customer satisfaction [32]. A chatbot never complains, is available seven days a week 24 hours a day, and can handle large demands [32]. They are always available if e.g. assistance is required at 3 AM the chatbot will be available whereas a customer service is bound to the opening hours and might not be available at 3 AM. Large demand handling is especially interesting for companies which don't have the resources to hire enough people to match the service demands. It's possible to replace humans whit chatbots as mentioned in Rahman et al. [28] or aid them by reducing the workload like in Deshpande et al. [16]. Chatbots are perfectly suited for monotonous routine work which can be used to aid people with daily work where they can focus on more important tasks. All these reasons are key factors why chatbot systems are getting popular. In this master's thesis, multiple important factors need to be considered. The knowledge-base for the company for further development needs to be built. It needs to be researched which technologies are state-of-the-art technologies to build chatbots. Lots of chatbot frameworks have been built where a developer can choose from. The big tech companies offer Dialogflow (Google), Watson Assistant (IBM), Alexa/Lex (Amazon), LUIS + Microsoft

Bot Framework as cloud services. From the found state-of-the-art technologies, some need to be chosen and compared. The chosen technologies need to be compared with domain-specific data to determine which frameworks can be recommended for further development. The chosen domain for this thesis is a combination of sickness notifications and vacation requests. There are some key requirements for the bots. The bot needs to collect information from a natural language conversation of short text messages. The extracted information is used to identify the person and is sent to a server for processing. To ensure a fair comparison in a fair environment the frameworks will be trained and tested with the same data to prevent bias. The end result is expected to look and feel similar with all technologies. The frameworks are expected to share the same or at least similar concepts. They should be easy to develop and the functionality offered is expected to be similar. The NLP performance is expected to be on an equal level like in Braun et al. [15] where the difference between Rasa and LUIS was quite small. Some papers like Braun et al. [15] compare the NLU and NLP capabilities of services. In this thesis, the frameworks are compared from the NLU and development perspective. Hence, things like setup, development difficulty, and the available features of the frameworks are compared. This information is used to give a recommendation for future development. After the evaluation of the framework, a recommendation will be provided. The recommendation provides a ranking of the frameworks based on conditions. Another important aspect is the development of chatbot systems in general. This includes the development approach, the development differences to regular development, and psychological aspects that need to be considered.

1.2 Motivation

The company currently has no knowledge of chatbot systems. To justify the use of a chatbot system knowledge about those systems needs to be collected and evaluated. Knowledge needs to be collected about chatbots, the possible use-cases, the state-of-the-art technologies, the concepts, the evaluation of chatbot systems, and psychological aspects. It needs to be determined if problems like the sickness notification are suitable for a chatbot. It's of general interest to find out which use-cases are suitable for chatbots in general to give recommendations and inspiration for further projects. With the collected knowledge the sickness notification use-case needs to be implemented using multiple frameworks to determine the best framework for the use-case. At the moment, the sickness notification process starts with a call at the company. A person receives the call, enters the information into the computer system and enables an out of office email notification for the sick person. The head of the department also needs to be notified. This repetitive task needs to be solved by a chatbot. The new process also starts with the call but this time the person receiving the call enters the retrieved information into the chatbot system and an out of office email notification is created automatically. This saves time the employees can spend on important, non-repetitive tasks. For this use-case, the chatbot needs to collect the information from natural language and submit the data to a server for processing. Chatbots can serve requests all the time and can serve a larger amount of people immediately [25]. Chatbots are accessible, efficient, 24/7 available, scalable, cheap, and user behavior can be monitored [29]. They are also a great solution when it is impossible to hire a large enough staff to handle lots of user requests [25]. Chatbots can be used to make complicated search operations easier to understand and easier to use for humans [25]. It's also a natural way of communication. Anyone who can talk to or write with a

person can work with a chatbot interface [29]. Hence, it is an easily understandable interface suitable for almost everyone. Other major points are the performance of the different frameworks, the ease-of-use, the development effort, and the supported features. Based on these and similar criteria the frameworks are evaluated to find the best suitable framework and the strengths and weaknesses of these systems. All this information together builds the knowledge-base for further research and development.

1.3 Company

The master's thesis is for the 3 Banken-IT Gmbh. The managing directors are Karl Stöbich and Alexander Wiesinger [2]. The 3 Banken-IT is the general contractor for all IT-services inside the 3 Banken-Gruppe [3]. The core competences of the 3 Banken-IT are the software development and maintenance of banking applications, IT-security, IT-support, operation of the data-centers, and the operation of central and distributed IT-infrastructure [3]. The 3 Banken-Gruppe consists of three independent regional banks namely the Oberbank, Bank für Tirol und Vorarlberg and the BKS Bank [1]. The supervisor of the master thesis from the company side is Thomas Reidinger the chief of enterprise architecture.

1.4 Research Questions

- Where are the differences and similarities between the chatbot frameworks?
- Which framework is the most promising for the given problem?
- Which problems are suitable for the use of chatbots?

1.5 Prerequisites

There are some prerequisites which have an influence on the choosen technologies for this thesis. Based on this thesis further projects will be developed. The company has currently no experience with chatbot systems. Hence, the collection of information about state-of-the-art technologies is a key area for the company and an important prerequisite. The company uses a local container environment for deployment. A offline capable chatbot framework needs to be included in the tests to allow the use in the companies local systems for further projects. The company does use IBM software Hence the use of Watsons Assistant is an option. For a comparison at least three technologies need to be choosen. The domain defined by the company are sickness notifications and vacation requests. This means that the choosen chatbots need to support the collection of information in a given domain. The main purpose is to recommend a technology for further development. For this reason the chatbots need to be compared from a technical view and the development view.

1.6 Expected Results

At least three chatbot frameworks have been selected based on the findings in state-of-the-art. The given problems of sickness notification and vacation request have been analyzed

for the suitability for chatbots. The domain information has been converted to dialogs. The dialogs have been implemented with all technologies if possible. The training and test data for the chatbots have been defined. All technologies are trained and tested with the same data for a fair comparison. Evaluation criteria have been extracted from the state-of-the-art chapter. Additional evaluation criteria have been defined. A detailed comparison has been created for the different technologies showing the strengths and weaknesses from a development point of view. New use cases were found for the use of chatbot systems for further projects. Based on the prototype evaluation the best framework has been recommended for further development.

Chapter 2

Basics

2.1 Chatbot

A chatbot is a conversational agent able to communicate with a user using natural language [30, 31, 24, 22]. The term chatbot is a combination of the words chatting and robot. In literature synonyms like chatterbot and conversational agent are used. The communication with a chatbot is most often done in written form like in a chat but not restricted to it since speech is also classified as natural language. The communication between man and machine is done in turns until a conversation end is reached or the conversation is aborted [34]. A conversation end is when the underlying task has been achieved. The user input is interpreted by the chatbot and the most likely response is generated [17]. In the best case, the generated response is reasonable and intelligent [34]. In the worst case a generic message like ”I didn’t understand that” is generated. It’s important to keep in mind that a chatbot can mimic intelligence, human dialog, and personality but is often limited to specific functionality [25, 17]. Three major chatbot types are present in the literature. There are menu-driven chatbots (Singh et al. [32]), domain-specific/goal-oriented/task-oriented chatbots (Deshpande et al. [16], Williams et al. [35], Braun et al. [15], Williams et al. [36]), and chatbots for general/open-ended/end-to-end conversations (Brandtzaeg and Følstad [14], Singh et al. [32]). Menu-driven chatbots mimic a classical interface and navigate the user through the interface using text or speech. An example of a menu-driven chatbot is a phone call at a support hotline. The caller is guided through the interface by the system and isn’t connected to a person immediately. The system gives options like ”press one if you’d like to request a refund, press two if you need replacement parts”, navigates the caller through the process, collects information like the order number, and connects the user to the correct department. Domain-specific chatbots offer functionality that is limited to a domain. Such systems provide the functionality to achieve specific tasks and are not supposed to handle anything else. One example would be a refund desk in a shop. If the customer asks for refund specific information or wants to issue a refund the customer is served. If the customer asks where the TVs are located in the shop he is redirected to the information counter. A chatbot system in the refund desk domain offers only the functionality necessary to get and collect refund information and to issue a refund. If the bot is asked something else no useful response can be expected. Open-ended systems are made for a general conversation. Contrary to the other two approaches the user can ask open-ended systems anything and the most likely response is generated. There are no fixed rules what users may or may not ask and the conversation can develop

in any direction. These systems have a hard time to collect specific information since there is no way to tell which information is needed. Every conversation can vary greatly hence no information can be presumed. Popular examples for open-ended systems are Google Home, Siri (Apple), and Alexa (Amazon) [32]. The most limited approach is the menu-driven approach since the options are given and a user only inputs numbers to select a functionality. Domain-specific bots communicate with users in natural language and are good in achieving tasks and collecting information. Open-ended systems are perfect for general communication and are not restricted to a domain but it's hard to collect specific information or achieve a domain-specific task with such systems. General tasks are not a problem for open-ended systems. A simple exemplary chatbot conversation is shown in Figure 2.1. In general, an ML-based chatbot system consists of rules and training data. The training data is used to train natural language capabilities and should be as realistic as possible. The training data can be handcrafted for prototyping or generated from large sources like emails, phone calls, or the internet. The rules define the available functionality, types, and possible input statements. One simple rule could be that a person's name is required as input and consists of a first and last name.

2.2 Chatbot/NLU Frameworks

Many chatbot and NLU frameworks were created to enable non ML experts to develop chatbot systems. The frameworks can be divided into cloud-based and local solutions. The cloud-based solutions are created, edited, and developed through a web-interface and are used in applications via a REST API endpoint. The developer is dependent on the provider and the training and analysis are done in the cloud of the provider. This needs to be kept in mind when dealing with sensitive information like bank accounts. The big cloud providers offer chatbot or NLU frameworks in their cloud environment. Microsoft has LUIS (Williams et al. [35], lui [7]), Google develops Dialogflow (dia [4]), IBM offers Watson Assistant (wat [10]), Amazon provides Lex and Alexa. The local solutions are standalone applications running on the development machine or the company server. The data is never passed to another computer or the cloud since it runs locally. A popular local open-source solution is Rasa (Bocklisch et al. [12], ras [8]). All the frameworks listed above use slot-filling. A slot is a predefined type/entity which a user needs to enter in a conversation. In Table 2.2 the slots which need to be filled for a hotel reservation could be the hotel name, start and end date, and a name for the reservation. All of these slots need to be filled before the system can process the hotel reservation automatically.

2.3 Suitable Problems for Chatbots

Chatbots seem like intelligent systems but most often they are designed for specific tasks and are unable to solve complex problems [29]. Therefore, not every problem can be solved using chatbots [29]. A problem can be solved effectively with chatbots when it is highly repetitive, can be automated, and can be done with simple back-and-forth communication. A chatbot system is meant to reduce the workload of employees or serve a large number of users. Therefore, the problem needs to be repetitive to fulfill these requirements. It is possible to solve non-repetitive tasks with chatbots too but it is not cost-effective. The problem needs to be automated to simplify the process. If it can't be automated an employee is needed for the process and it's harder to serve a large number

of users. Chatbots are based on natural language communication with text messages or voice. Hence, the problem must fit in the back-and-forth communication pattern since there are no other means of communication available for chatbots. If all the statements above can be answered with true the problem is suitable for a chatbot. A simple QnA bot for a FAQ page of a website is an example of a suitable problem [29]. The task can be automated, is highly repetitive, and the answers are predefined by the FAQ page. As mentioned in Section 2.1 there are three major types of chatbots. The menu-driven and domain-specific approach are suitable for tasks with a goal where data needs to be collected to execute an action where only a limited set of functions needs to be supported. When the user can select an option from a limited list of actions the menu-driven approach works best (support, helpdesk chatbot). When users should be able to communicate using natural language to ask questions related to a domain then the domain-specific approach works best (FAQ chatbot). In the FAQ example, the domain consists of all questions and answers of the FAQ page. It would be bothersome to navigate through a FAQ forum using numbers because if e.g. 1000 topics are available the user would need to enter a number between one and 1000 and also would need to know the correct number for the question. This would be horrible to use and no one would want to use such a system. A user just wants to ask a question to get an appropriate answer. This works best with natural language hence, the domain-specific approach is suitable. The open-ended approach is suitable when no specific data needs to be collected and a general conversation needs to be held with the users. The Google Assistant can answer general questions like ”What’s the weather in New York” and gives a general response. The Google Assistant can’t process a request like ”Tell my employee I am sick and won’t come to work today” since this would require domain-specific data and actions.

2.4 Intent

Intents are the core concept of any chatbot system. An intent is the action a user wants to perform [17, 28]. In example sentence two in Table 2.2 the user wants to book a hotel. The action the user wants to perform is ”book a hotel”. The intent can be named ”book hotel” for instance. In general, the user queries the system, the system identifies the best matching intent, collects relevant information, and generates a response. The identification of these intents is the core function of chatbots. To identify the intent the system finds the intent where the training phrases are closest to the users query or uses the fallback if no intent reached a high enough score. The score is represented by a percent value between zero and one. One means a perfect match and zero is treated as absolutely no match. A threshold defines the minimum score the system has to detect before an action is executed. All values below the threshold are handled by the fallback action e.g. a generic response like ”I’m sorry but I didn’t understand that”. In Table 2.2 example sentences are listed with their intent, and their entities. In example sentence one of Table 2.2 the intent can be labeled as ”Greet” since the user greets the bot. Table 2.1 shows a small excerpt of the information generated for the default welcome intent created by Dialogflow. The intent consists of training phrases and a response list. The training phrases are used to train the ML model. Everything which is similar to the training phrases and reaches the highest score will be classified as the intent. The system then responds with a random pick from the response list. The random pick fakes intelligence since it would be boring for users to get the same response every time. If the same response is returned all the time users would rate the communication as robotic but the goal is to make

the conversation as natural as possible.

No.	Intent	Type	Sentences
1	Default Welcome	Training Phrases	hey howdy partner heya hello hi hey there hi there greetings howdy
2	Default Welcome	Response List	Hi! How are you doing? Hello! How can I help you? Good Day! What can I do for you today? Greetings! How can I assist?

Table 2.1: Default Welcome Intent of Dialogflow

2.5 Utterance

An utterance is a sentence or a part of a sentence. Especially in the area of chatbots, an utterance is often a part of a sentence since chat messages are often shortened. All sentences listed in Table 2.2 are utterances for instance. The messages of users are utterances [32, 17]. The chatbot also responds to users with utterances. Table 2.1 shows some training phrases and the response list. The sentences listed as training phrases and the response lists are utterances. Table 2.1 shows that a collection of utterances belongs to an intent and builds the base for the classification of user input. The training phrases are different versions of the same question or statement. The different versions allow the system to correctly classify utterances which never were part of the training data as long as the utterance is similar. Example sentence two of Table 2.2 can be used as utterance for the book hotel intent. Utterances need to be chosen with care to prevent overfitting to unimportant features of the sentences. If all utterances contain a specific word at a specific position the model might learn that a specific word must be at that position all the time. Utterances for training purposes can come from multiple sources. They can be handcrafted for development and testing. The data can come from live systems. Real user input from e.g. e-mails, phone conversations, or online resources can be used to get a large amount of training data [32].

No.	Sentence	Intent	Entities
1	Hello	Greet	-
2	I'd like to book a hotel	Book Hotel	-
3	The Twelve Months hotel	-	Hotel Item
4	From the 27th of April	-	Date (From)
5	To the 4th of May	-	Date (To)
6	Ethan May	-	Person (Name)
7	Order one green t-shirt size M please	Place Order	T-Shirt Item, Color, Size

Table 2.2: Examples for Concept Explanation

2.6 Entity

An entity is a piece of information that needs to be extracted from natural language. An entity is metadata which belongs to an intent an has a type. Entities can be represented as volumes, counts, and quantities for instance [29]. An intent can have multiple entities like in example sentence seven of Table 2.2. In sentence seven one entity is the color of the shirt, the second is the shirt as an item, and the third one the size. Sentences three to six of Table 2.2 show entities in a book hotel example. In a simplified example, to book a hotel the name of the hotel (Hotel Item), the check-in and check-out dates (Date), and a name (Person) are needed for the reservation. To extract entities in a context from natural language Named-Entity-Recognition (NER) is used. NER is a sub-area of NLU. Entity recognition deals with word ambiguity. Sentences five and six show a word ambiguity. In sentence five May is the month and needs to be treated as a date whereas in sentence six of Table 2.2 it's the last name and needs to be treated as a person entity. A named entity is a noun phrase relating to individuals. Classic individuals/named entities are person (Bill Gates), location (New York), and organization (Google) [19]. Frameworks sometimes provide predefined named-entities like the ones above which are needed on a regular basis. A developer can define named entities for a specific domain like in the hotel example where an entity is needed which knows all valid hotel names. Example sentences four and five of Table 2.2 show a date entity, and example sentence six shows a person entity. An example entity for food items like apples and peas is shown in Table 2.3. An entity can be a collection of items like for the fruits but it can also be a collection of synonyms. Entities are stored in the session and form the context of a conversation [32]. They are only valid for the current conversation.

No.	Entity	Item	Synonyms
1	Fruit Item	apple banana pea peach	apples bananas peas peaches
2	Sick Item	sick	ill, unwell, poorly, indisposed, sickly, ailing

Table 2.3: Entity examples

2.7 Chatbot Response

The response message of a chatbot is picked from a pool of utterances as described in 2.5. The response can be picked at random or in a sequence. Alternatives are picked to keep the conversation interesting for the users especially if they are using the bot more than once. The alternatives prevent a robot-like feel in the conversation. If the same response is picked all the time users realize fast that they are talking to a bot system and it gets boring fast. An exemplary list of responses for the default welcome intent of Dialogflow is shown in Table 2.1 next to response list.

2.8 Story

A story defines the dialog flow. It's a step-by-step script for a conversation between man and machine. The bot and the user interact in turns using natural language [30, 31, 24, 22]. The user asks a question or makes a statement and the bot processes the request and delivers a response. A story can be a "happy" or "sad" path. A "happy" path is a successful interaction from start to finish without any problems. The "sad" path handles the behavior when something doesn't work out. It could be that the user input is wrong and needs to be re-entered when the validation fails. The user can also quit the conversation at any time. These "sad" paths need to be handled in separate stories. In Figure 2.2 a story for a greeting sequence with a user is shown. The user starts the conversation by entering a greeting phrase. The response of the chatbot is also a greeting phrase and then the conversation ends. An example conversation from a person with a chatbot is shown in Figure 2.1. The story for 2.1 is that the user asks for open hours, the bot respond open hours, the user asks for an item, the system validates the item and a response is generated with the item in the response text.

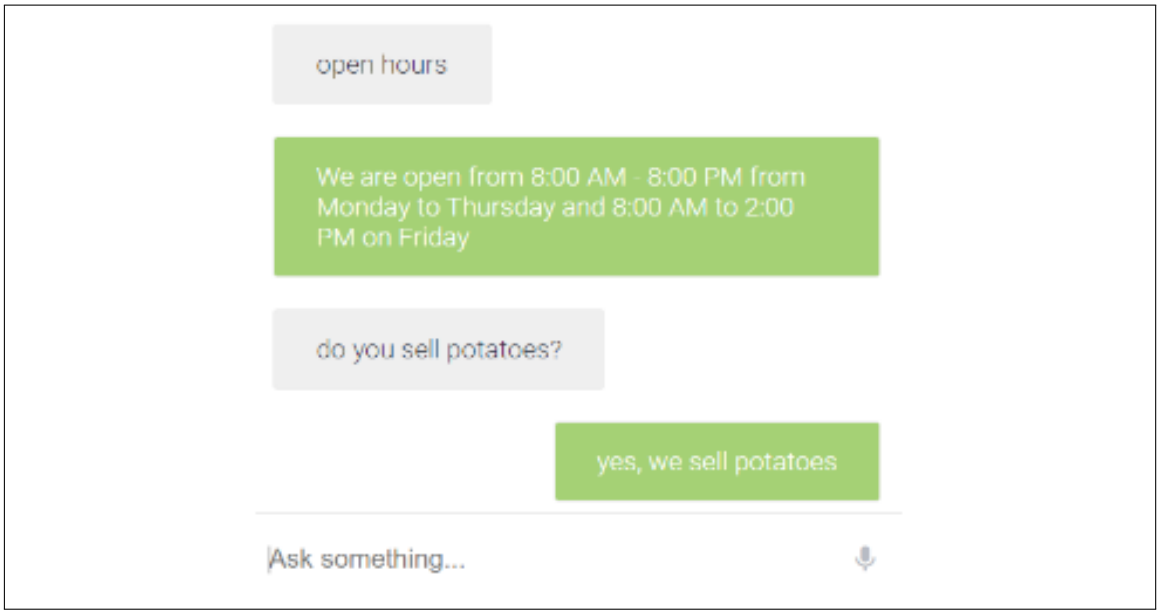

Figure 2.1: Example FAQ Conversation with Dialogflow

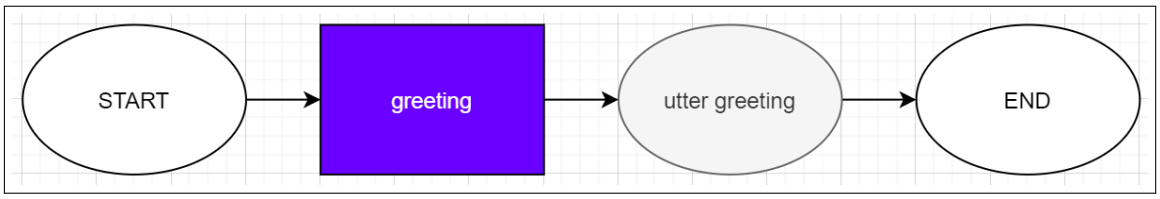

Figure 2.2: Greeting Story

2.9 Webhook

A webhook is a URL where the frameworks send metadata and intent data for analysis, completion, and processing. In the case of this thesis, the webhook is a simple Pyhton REST service implemented with Flask. Each framework gets a route in the service which is ”/watson/webhook” for the Watson Assistant requests for example. In Listing 2.1 a simplified JSON request from Dialogflow is shown for the default welcome intent. In the case of Dialogflow the incoming message can be found under query text, the response message under fulfillment text and the confidence under intent detection confidence.

```
[
{
    "originalDetectIntentRequest": {
        "payload": {}
    },
    "queryResult": {
        "action": "input.welcome",
        "allRequiredParamsPresent": true,
        "fulfillmentMessages": [
        {
            "text": {
                "text": [
                "Good day! What can I do for you today?"
                ]
            }
        }
        ],
        "fulfillmentText": "Good day! What can I do for you today?",
        "intent": {
            "displayName": "Default Welcome Intent"
        },
        "intentDetectionConfidence": 0.72385716,
        "languageCode": "en",
        "parameters": {},
        "queryText": "Hi, how are you doing"
    },
}
]
```

Listing 2.1: Dialogflow Webhook Request Example

2.10 Domain

The domain of a chatbot is equal to the functionality. A domain-specific chatbot has a limited set of operations and works with data relevant to the domain. If the chatbot

is built for a refund-desk scenario then the domain is refund for example. The bot can process and answer refund-related issues, collects refund specific information, and can issue a refund. It doesn't offer functionality which is unrelated to the refund domain-like "What's the weather in New York today" and it is not supposed to do so. Open-ended systems escape the domain limitation but have problems to execute domain-specific tasks or collect domain-specific information.

2.11 Evaluation of NLP capabilities

The scenario for this explanation is a medical test for a disease. Positive is treated as the person has the disease. A true positive (tp) is when the test is positive and the person has the disease. A false positive (fp) is when the test is positive but the person is healthy. A true negative (tn) is when the test is negative and the person is healthy. A false negative (tn) is when the test is negative but the person has the disease. In the area of chatbots, a true positive is when the input is e.g. "Hello" and the identified intent is "Greet Intent". The precision is calculated as $\frac{tp}{tp+fp}$. The recall is calculated as $\frac{tp}{tp+fn}$. The f-score is calculated as $2 * \frac{precision*recall}{precision+recall}$.

2.12 Natural Language Processing/Understanding

The term "natural language" means input is received through voice or writing in the user's language of choice [29]. Natural language processing in the area of chatbots is necessary to analyze the text input provided by the user and figure out the intent and entities necessary to process the given information [29]. Natural language processing is seen as the brain of a chatbot since it receives the information from the user, processes the data, and retrieves the relevant parts from the input for post-processing [29]. Part-of-Speech (POS) tagging is a process where parts of speech like noun, verb, and adjective are assigned to each word/token [29]. This information is used to understand the context and is part of NLU. An important part of natural language processing is the search for the root of words which can be done with stemming and lemmatization [29]. Stemming reduces a word to the base form e.g. "saying" is reduced to "say" by removing the endings of words [29]. Lemmatization tries to return the dictionary form of a word and has better correctness and is more productive than stemming and hence the preferred method [29]. Lemmatization uses a vocabulary and morphological analysis of words to achieve its task [29]. Stop words are words that hold little to no meaning and occur in a high frequency like "the" and "a" which are removed.

Chapter 3

State of the Art

Williams et al. [36] and Bordes et al. [13] state that the two major chatbot types are goal-oriented and end-to-end systems. Kane [25] defines two different categories of chatbots. They can either be classified as a local standalone application or a web-based chatbot [25]. The types/classes of chatbots from Williams et al. [36], Bordes et al. [13] and Kane [25] don't contradict they just look at chatbots from a different perspective. From the design perspective, the conversation type of a chatbot is either goal-oriented or end-to-end. From the development perspective, the used framework is either web-based or local. Goal-oriented or end-to-end is a general statement about the kind of base problem which the chatbot needs to solve. Web-based or local has an influence on where the chatbot is developed. Rasa is one example for a local standalone application chatbot whereas LUIS, Dialogflow and Watson Assistant are cloud-based/web-based chatbot frameworks. Rasa, Dialogflow, LUIS, and Watson Assistant are all capable of goal-oriented dialog.

Deshpande et al. [16] shows the evolution of chatbots from the first chatbots to current state-of-the-art chatbots like Alexa and Siri. The first chatbots created used pattern matching to respond to user inputs [16]. Current technologies like Siri use NLP to respond to user inputs [16]. With NLP the intent is identified and the question is analyzed to detect commands and actions [16]. The basic workflow of a chatbot system starts with the user input [16]. Then the NLP engine extracts entities and detects the intent [16]. Entities are used to retrieve the relevant data [16]. After the relevant data has been retrieved a response is generated [16]. Raj [29] explains the core concepts of chatbots, NLU, and NLP necessary for the development of chatbots. A food order chatbot is built with Dialogflow and a horoscope bot is built with Rasa in Raj [29]. Chatbots are easy to use because it's as natural as talking to a human without the need for complex interfaces [29]. Before starting to implement a bot the problem the bot should solve needs to be analyzed. Chatbots can't do everything, in fact, they are most often designed for one specific task (task-oriented) [29]. Some major questions need to be answered to determine if the problem is suited for the use of a chatbot [29]. If the conversation is simple and consists of simple back-and-forth communication, and the task is highly repetitive and can be automated then the problem is suitable for a chatbot [29]. It doesn't make much sense to implement a chatbot for a non-repetitive task which can't be automated [29]. If the task is non-repetitive then the development of the chatbot would take more time than the communication between the user and an employee. A perfect example use-case for a chatbot is a FAQ web page [29]. The answers are predefined, it's repetitive, it consists solely of questions and answers, and can be automated. Some of the best chatbot and NLP frameworks are Rasa, LUIS, and Dialogflow [29]. Geyer et al. [19] is about named entity recognition (NER) in micro-posts.

NER is the core concept used by chatbot tools to extract information from text. The classic examples for named entities are people, locations, and organizations [19]. Geyer et al. [19] examines how good custom entities are recognized by the system. The frequency of entities in the training data can influence the NER performance [19]. Geyer et al. [19] checked the performance of entity recognition which can be done for chatbot frameworks in a similar fashion. Rahman et al. [28] gives a small overview of the concepts of Dialogflow and the general architecture of chatbot systems. Commonly used cloud chatbot solutions are IBM Watson, Dialogflow, and Microsoft Bot [28]. In general, the chatbot architecture consists of intent classification, entity recognition, a response generator, and a response selector [28]. The intent classification identifies the best matching intent for the users input. The entity recognition module extracts the information/data from the user's message [28]. The response generator provides response candidates and the response selector chooses the best matching response [28]. Goal-oriented chatbots are most common in the business sector and help users to achieve tasks [28]. The tech giants provide chatbot frameworks for regular developers [28]. Google provides API.ai/Dialogflow, Microsoft has LUIS, IBM develops Watson, and Amazon provides Lex [28]. The key concepts of Dialogflow are intent, entity, and utterance. Intents link the user's input to actions which are executed [28]. Deshpande et al. [16], Raj [29], Geyer et al. [19], Rahman et al. [28] focus on the basics of chatbot frameworks and chatbot related concepts. In Deshpande et al. [16], Rahman et al. [28] describe the general workflow of chatbot systems. Geyer et al. [19] is about NER and explains the core concept of entities. Additionally, Raj [29] also explains which problems are suitable for a chatbot in general.

The first major type are the end-to-end systems. Shawar and Atwell [30] focus on the application of chat systems in different languages and the evaluation of the system. The evaluation of chatbot systems needs to be adapted to the application and the user needs [30]. The best way to evaluate a chatbot is to check if the specific service or task can be achieved and the evaluation needs to be adjusted to the use-case [30]. Each component of a chatbot system can be tested individually and the whole system can be tested based on user satisfaction and acceptance [30]. The whole system is evaluated through feedback by users in a test setting [30]. This feedback is then used to improve the system [30]. In Shawar and Atwell [30] a chatbot for frequently asked questions (FAQ) has been built which competed against the classical Google search. The participants had to answer a few questions like how many results the system found and which system they prefer and why [30]. The result of the user study was that the FAQ chatbot found an answer more often than Google [30]. 47% of the users taking part in the study preferred the custom solution and 11% preferred Google [30]. The major reason why the users preferred the FAQ chat was because the chat can often give direct answers and return fewer links on average which saves browsing time [30]. The users preferred Google because it's familiar and it can give different answers when the input query is adjusted slightly [30]. This shows that custom-built solutions can outperform classical approaches in small areas. Bordes et al. [13] analyses the strengths and weaknesses of end-to-end dialog systems in goal-oriented settings. The end-to-end approaches have recently shown promising results with chit-chat/general conversation with a user [13]. However, The most useful application areas for chatbots are either goal-oriented or transactional settings [13]. A classical dialog system uses slot-filling to collect information from the conversation [13]. Slot-filling is reliable but has limited scalability [13]. The slots are predefined and hence don't scale well because they can be very different for each problem. End-to-end systems scale well because they learn directly from conversations and make no assumptions regarding the

domain or dialog structure [13]. The goal-oriented example used by Bordes et al. [13] is the classic restaurant reservation bot. The main question is if the end-to-end system performs well on the classical goal-oriented restaurant reservation task. The restaurant reservation is a domain where goal-oriented systems perform well and is the ideal candidate to determine the strengths and weaknesses of end-to-end systems in comparison to goal-oriented systems [13]. The result of Bordes et al. [13] show that end-to-end systems are not yet ready to replace goal-oriented systems and have to improve before they can perform reliably in goal-oriented settings. Bordes et al. [13] compared the chatbot types based on the number of correctly identified requests (true positives) and the number of dialogs where every request was identified correctly. Multiple conversations were used to test the system and the end-to-end system never identified all requests of the conversation correctly [13]. The request identification as such worked well [13]. No conversation went smoothly since at least one request was identified incorrectly. In other words, each user had at least one problem in the conversation. This would lead to bad user experience. In summary, the most important statement of Bordes et al. [13] is that the current end-to-end systems are not performing reliably enough in goal-oriented settings. Williams et al. [36] introduces an approach that combines end-to-end dialog with domain-specific knowledge. A task-oriented system helps the user to achieve a task using natural language [36]. A restaurant reservation is a task-oriented operation that requires domain-specific knowledge [36]. The domain-specific knowledge in a restaurant setting can be the restaurant name and the number of people for the table reservation for instance. This information needs to be collected from the conversation. End-to-end systems lack a mechanism to inject such domain-specific information [36]. There is also no mechanism for constraints in goal-oriented systems [36]. In a banking app, the user needs to be logged in (constraint) before the account information can be retrieved [36]. A programmer can code such constraints with a few lines of code but lots of training data needs to be provided to a system to learn such a mechanism on its own [36]. The hybrid-coding-networks (HCNs) introduced in Williams et al. [36] extend the open-ended approach with domain-specific knowledge. The domain-specific knowledge is provided by the programmer and requires more development effort [36]. Since the domain-specific knowledge is provided by the programmer the system doesn't need to learn these parts and less training data is required in comparison to existing end-to-end approaches [36]. A large amount of training data can be retrieved from real dialogs [36]. The experiments show that the HCN approach performs on an equal level with existing end-to-end approaches in end-to-end settings and needs less training data for the same performance [36]. The HCN system also performed better in the task-specific setting than the existing end-to-end approaches [36]. HCNs are a mixture of end-to-end and goal-oriented systems. The developer has more control than with the classical end-to-end approach but it also requires more development effort to create a chatbot with HCNs [36]. The main benefits are the usage of domain-specific information in end-to-end dialog and the reduced amount of training data required. Other references focus on end-to-end and/or goal-oriented chatbots whereas Williams et al. [36] combined them to get the best of both worlds. Williams et al. [36] does not compare the HCN approach to a goal-oriented approach like Bordes et al. [13] did for regular end-to-end systems. Hence, no statement can be made if HCNs could replace goal-oriented systems or not. In Shawar and Atwell [30] testing focus lies on the user side whereas the testing focus of Braun et al. [15] lies on the NLU capabilities. The NLU capabilities are important for the developers of the systems before a chatbot is developed and help to choose the best framework for development. The users feedback is a valuable source

of information for the actual product and is important long after the framework has been chosen. These are two very different views on the performance of a chatbot in two different phases of a project. Bordes et al. [13] focuses on end-to-end systems like Shawar and Atwell [30], Williams et al. [36] but uses a goal-oriented setting for the comparison. The important information gained by Bordes et al. [13] is that currently end-to-end systems are not reliable enough for goal-oriented tasks. This leads to the conclusion that a goal-oriented technology should be chosen for goal-oriented projects. Williams et al. [36] is trying to combine the end-to-end and goal-oriented approaches to create a new type of system which can do goal-oriented dialog better than current end-to-end approaches. The result of Williams et al. [36] shows that the HCN system is better in goal-oriented settings than current end-to-end systems but doesn't compare the system to goal-oriented technologies.

Singh et al. [32] states that the conversation type of a chatbot can be categorized as a general conversation which is about a broad generic subject or a specific conversation about one product or service. An example of a general conversation is e.g. when a customer walks into a bank and starts talking to an employee [32]. In this example, we have no idea what the person wants or who the person is [32]. Popular examples for chatbots capable of general conversation are Google Home, Siri, and Amazon Alexa [32]. An example of a specific conversation is the refund desk in a store [32]. Specific rules apply to the refund process and specific information is required from the customer [32]. The domain could be named refund and the specific rules and information are part of the domain. The customer can't get any other information than refund related information at the refund desk [32]. The outcome of the conversation can be one of the predefined outcomes, the fallback, or a conversation end [32]. A specific conversation ends as soon as the underlying task is achieved or a conversation end is reached [32]. In general, specific conversations are easier to predict and are handled with higher accuracy [32]. Chatbot systems for specific conversations communicate with information [32]. Nowadays, chatbots are AI-driven. The core component of AI-driven chatbots is the NLP engine which takes care of data extraction from natural language [32]. The extracted information is used to determine the next steps [32]. There is no difference in functionality between a regular application and a chatbot in terms of functionality [32]. The difference is that a chatbot uses conversation whereas a regular app is a self-service application [32]. Conversational training data for chatbots can be acquired from lots of sources like emails, phone calls, chats, and social media [32]. Companies use chatbots because they can save a lot of money and because they offer a good customer experience [32]. Especially in areas with personal data the usage of cloud chatbot solutions like Dialogflow, Alexa, and Watson is problematic because all the conversation data is sent to and stored at the providers servers [32]. Because of high-security requirements, an in-house/local chatbot has been built in Singh et al. [32]. The first step to create a chatbot system is to define a conversational flow [32]. The conversational flow is a decision tree which describes all events, decision, and outcomes in a conversation [32]. Singh et al. [32] also gives an overview of the basics of Microsoft Bot (LUIS), Rasa, and Google Dialogflow.

The second major type are goal-oriented systems. Braun et al. [15] states that modern conversational agents can be built without programming knowledge because of natural language understanding services (NLU) [15]. The recent advancements in machine learning (ML) and NLU and the popularity of messenger platforms has led to huge progress in recent years [15]. People are familiar with messengers because many use them on a daily basis. Hence, chatting has become a natural way of communication. A big question is how

chatbots can be evaluated. Recent publications have discussed the use of NLU service but not why one service has been chosen over another [15]. In general, The architecture of a chatbot consists of three elements [15]. The request needs to be interpreted, the response needs to be retrieved and the response message needs to be generated [15]. The purpose of these services is the extraction of information from natural language [15]. Popular NLU services are the cloud services LUIS, Watson Conversation, Dialogflow/API.ai, and RASA as an open-source alternative [15]. The cloud-based solutions have advantages when it comes to hosting and scalability and Rasa has advantages when adaptability and data control are needed [15]. All three NLU services share the same basic concepts of intents, entities, and batch import in JSON format [15]. The cloud-based services are secretive when it comes to the ML algorithms and the initial training data [15]. The exception is Rasa where the ML backend can be chosen by the developer [15]. For the evaluation of the NLU services training data from a production chatbot and two other training sets from SatckExchange were used [15]. The NLU capabilities of the frameworks were compared. The comparison included the services LUIS, Rasa, Dialogflow, and Watson Conversation [15]. The same training data was used for all services [15]. This is important to get a comparable result. The evaluation of the services NLU capability is based on the true and false positives and negatives, the recall, precision, and F-score [15]. The better the F-score is the better the NLU service has performed [15]. The evaluation of NLU services is only a snapshot since the services are continuously improving [15]. LUIS performed best on all datasets but in the area of chatbots, Rasa and LUIS performed on an equal level [15]. Rasa can be customized further than LUIS and could perform better than LUIS through customization [15]. The domain had little to no influence on the ranking of the NLU performance [15]. In other words, a good NLU system performs good independent of the domain. If the training data is sparse then there is no significant difference between the services [15]. Before using an NLU service different services should be tested with domain-specific data [15]. In Dutta [17] a chatbot is developed that assists high school students with learning general knowledge subjects and analyses the impact the chatbot had on learning. AI-based chatbots are used for banking systems, customer services, and education [17]. Popular chatbot platforms are Dialogflow, LUIS, Wit.ai and Pandorabots [17]. The developed chatbot is a web-based education solution using natural language processing (NLP) techniques to answer questions [17]. It is also able to participate in small talk [17]. The chatbot systems are evaluated based on their NLP performance and the available features of the platform [17]. All platforms are trained with the same knowledge base for comparison [17]. LUIS and Wit.ai performed slightly better in the NLP part than Dialogflow [17]. Dialogflow offers the concept of follow-up intents which is an important feature for the development of sub-tasks and was chosen for development over LUIS and Wit.hai [17]. To create the illusion of talking to a human being some small talk is implemented [17]. This can motivate learners and might increase the interest in the topic [17]. Dutta [17] uses goal-oriented technologies like Braun et al. [15] and both compare the technologies based on their NLU capabilities. Additionally, Dutta [17] also focuses on the available features of the platforms and selects Dialogflow over the other technologies because of these features. Dutta [17] also touches the psychological aspects of chatbot systems like the motivation of users through small talk. Go and Sundar [20], Brandtzaeg and Følstad [14], Følstad and Brandtzæg [18] go deeper into the psychological aspects of chatbot systems. Gregori [22] evaluated the NLU capabilities of Wit.ai, Luis, Api.ai/Dialogflow, and Amazon Lex[26] by the confidence score. The higher the confidence the better the user request matched the training data [22]. Each tool

was trained with the same data and was tested with the same questions to ensure fair conditions [22]. LUIS, Wit.ai, and API.ai/Dialogflow performed on an equal level in terms of intent classification in Gregori [22]. Gregori [22] chose Api.ai/Dialogflow for the development of a prototype. In Ahmad et al. [11] a chatbot is built to aid customers with questions regarding medication for a pharmacy company using IBM Watson. The bot can provide medications for an illness, give information about a specific medicine, and can give information on the medication intake [11]. The bot is a domain-specific bot since it answers questions to medication only. Important performance metrics for chatbots are the conversation length and structure, the ability to provide personalized communication, and the number of conversation steps [27]. The general trend is to keep the conversations short [27]. Personalized communication is used when recommendations or tips are provided for the user based on the information related to the user [27]. Retail chatbots need a larger amount of conversation steps to provide information and recommendations and hold the user's attention [27]. The measurement criteria for chatbots vary depending on the domain of the bot. Williams et al. [35] gives an overview of LUIS the state-of-the-art language understanding service of Microsoft. Historically, language understanding could be implemented via machine learning or handcrafted rules [35]. The ML model approach is robust but requires expensive expertise [35]. In general, software developers can build language understanding without the assistance of a framework with handcrafted rules [35]. But systems with handcrafted rules don't scale well [35]. With LUIS developers don't need machine learning knowledge to build language understanding models nor do they need to write handcrafted rules [35]. LUIS allows regular developers without ML expertise to develop cloud-based, domain-specific language understanding models [35]. Developers need to understand the concepts intent, entity, and utterance to work with LUIS [35]. Developers can create custom entities and use existing entities like location, date, and time. The communication with LUIS is done through an HTTP endpoint in JSON format [35]. Dialogflow, Rasa, and Watson Assistant also use an HTTP endpoint for communication and JSON format for the messages. Bocklisch et al. [12] describes the architecture and function of Rasa. Rasa is an open-source tool for natural language understanding and dialog management for developers [12]. Rasa uses the concepts of intent and entity [12]. The cloud chatbot technologies have these two concepts in common with Rasa. The state is represented by slots and the events which led to the current state [12]. This information is stored in the tracker [12]. An action has access to the tracker and determines the next step to take [12]. An action can be something simple like an utterance or something complex like a function which is executed [12]. Rasa consists of a natural language and a dialog component that can be accessed via an HTTP API [12]. The training data for Rasa can be specified in JSON or markdown format [12]. Rasa supports live training of the system where the developer can correct the bot while communicating with it [12]. Live training is an effective way to create training data for plausible conversations [12]. With live training, there is a higher possibility to create a complete and natural conversation because the developer is forced to go through the conversation step by step. Braun et al. [15] showed that Rasa performs on an equal level with tools like LUIS. Braun et al. [15] compares some NLU frameworks regarding their natural language capabilities. The compared frameworks are suitable for task-specific problems. Shawar and Atwell [30], Bordes et al. [13], Williams et al. [36] focus on end-to-end systems, Braun et al. [15], Dutta [17], Williams et al. [35], Bocklisch et al. [12], Ahmad et al. [11], Gregori [22] focus on task-oriented systems, and Singh et al. [32] is about both. Singh et al. [32] gives a general overview of the chatbot types, gives development advice, and introduces

popular chatbot technologies and their basics. Braun et al. [15] gives an overview of the architecture of chatbot systems and evaluates the NLU capabilities of chatbot frameworks. Gregori [22] and Dutta [17] also evaluate the NLU capabilities. When frameworks are compared for their NLU capabilities the same training data needs to be used for all tested frameworks like Braun et al. [15], Gregori [22] did. Dutta [17] also introduces a framework comparison based on the available features and some psychological aspects of chatbot communication. Unlike other references which either use or evaluate frameworks Williams et al. [35] and Bocklisch et al. [12] introduce the frameworks LUIS[35] and Rasa[12]. They focus on the technology itself and don't build a chatbot with it nor do they compare frameworks. In Dutta [17], Ahmad et al. [11], Przegalinska et al. [27] actual chatbots for specific use cases have been implemented.

Chatbots can also be viewed from a psychological perspective that focuses on user interaction. Følstad and Brandtzæg [18] investigates how users are communicating on the web and shows how the recent technological advances in the area of chatbots could influence the human-computer interaction in the future. Lots of people are already using natural language as the main input method on the web through mobile messengers and social networks [18]. At the moment the natural language conversation online is from human to human through a machine interface [18]. The people are using messenger platforms (machine-interface) to communicate with other people (human to human). Some platforms already offer chatbot integration. At platforms like Twitter machine agents can already be integrated and communicate with human users [18]. Chatbots are different from classical software interfaces because they use natural language for communication instead of a classical GUI. Currently, developers are focused on designing user interfaces but for chatbots the design focus lies on the conversation itself [18]. Regular UI systems are designed by experts and improved by qualitative data which is rather sparse [18]. A big advantage of conversational interfaces is the massive amount of training data that is present all around the web [18]. Chatbots all use messenger like interfaces independent from the problem. Developers don't have to think about designing user interfaces for chatbots. The developers need to switch to a goal-oriented view where the main focus lies on understanding what the user wants and how they can be served [18]. The state-of-the-art technology of conversational interfaces is Google Assistant [18]. Brandtzaeg and Følstad [14] describes chatbots as a natural language interface using text or voice. They are used to retrieve content or access a service through natural language conversation [14]. People are used to natural language communication because they spend lots of time on messenger platforms [14]. Because of this chatbot technologies become more and more popular [14]. Person to person communication is like the open-ended chatbot approach. It is difficult to design chatbots for open-ended conversations because users can start the conversation in many ways [14]. The conversation can also develop in any direction. The communication with a chatbot should feel natural. Hence, it is important to balance the human and robot aspects of a chatbot [14]. If the chatbot is too human people might ask questions unrelated to the domain [14]. If it's too robot-like people might complain because the conversation feels unnatural [14]. Developers also need to think about how friendly a chatbot should be, how fast the bot should answer if the bot should have a gender, and how human-like the bot should be [14]. Successful chatbots inform the users what they have to expect and clarify that they are talking to a bot right from the start [14] Chatbots should inform users about what they can do and it can help to inform them what they can't do [14]. Conversational interfaces need to be improved based on the interactions with humans [14]. Three examples for chatbots are Microsofts Heston Bot

for food, cooking opportunities and fashion, H&Ms bot for shopping suggestions based on photos, and Ikeas shopping assistant bot [14]. Brandtzaeg and Følstad [14] focus on the psychological aspects of chatbot design and give recommendations independent from the type of chatbot. Go and Sundar [20] is about human-like bots and the effects they have on users and compares these bots with current chatbot systems. The main functions of current online chatbots are interaction with users, address of concerns, and question answering [20]. These chatbots often lack humanness when communicating with users [20]. The impersonal nature of the conversation can be countered with a high level of message interactivity [20]. To increase the humanness of a chatbot three approaches can be used [20]. A chatbot can be given visual cues like human figures, identity cues like a human-associated name, or conversational cues by mimicking the human language [20]. Introducing visual cues and a name to a chatbot is an easy task. Conversational cues are hard to create since they require deep understanding of human conversation and the context. These three approaches can be used to make people believe that they are talking to a human or a bot. The way a user interacts with a bot and the things a user expects change through this assumption [33, 20]. If a user expects a human agent then the chatbot is more likely evaluated as human-like and the conversation feels more natural for users compared to when they expect a bot [33]. If the bot is identified as a human than the users automatically expect more from the conversation and the bot needs to be capable of more otherwise the users will be disappointed [20]. If the bot is identified as bot the users expect less from the conversation but the conversation is more likely evaluated as robotic or unnatural [20]. If on the other hand, the bot is falsely identified as a human being by a user the expectation of the user regarding interactivity rises [20]. This needs to be kept in mind when designing chatbots since the false identification as a human can lead to a huge decrease in user satisfaction and acceptance. Go and Sundar [20] shows that the user assumption (bot or human) has a great impact on the chatbot evaluation and the feedback. Unlike the other references Følstad and Brandtzæg [18], Brandtzaeg and Følstad [14], Go and Sundar [20] focus on the psychological aspects of chatbots and the communication with machines in general. Go and Sundar [20] is about the humanness of bots. It's important to keep in mind how human a bot should be and how the user assumption influences the feedback. If the bot appears to be human the expectations rise. If the bot is presented as bot right from the start the expectations are in general lower. Brandtzaeg and Følstad [14] also focuses on the psychological aspects and the humanness of bots. The disadvantages of too human bots are listed with real-life examples. Real chatbot conversations have shown that users ask unrelated and inappropriate questions if a chatbot is too human. Hence, it's important to balance the robot and human aspects when designing a bot [14]. Dutta [17] is about chatbot design aspects. The important message of Dutta [17] is that chatbot systems are completely different from regular systems from the design perspective. Right now, developers focus on the design of user interfaces (UIs). But chatbots all have the same UI requirements independent from the use case. They either use messengers like UIs for text communication or no UI at all through speech. This means that the design focus of a chatbot isn't the UI. It is the conversation. Hence, the main design focus needs to shift from UI design to conversation design.

ISO/IEC 25010 defines quality characteristics for the evaluation of software systems [6]. Functional suitability is split into functional completeness, correctness and appropriateness [6]. There are also performance efficiency metrics to check the time behavior of a system with the throughput time [6]. The interoperability of systems can be verified by checking the exchanged information [6]. Usability can be checked through learnability

which determines how easy it is to learn to use the product [6]. Security can be checked by checking for unauthorized access problems [6]. A part of portability is how easy it is to install a system [6]. ISO/IEC 25010 provides evaluation criteria for software systems which can be used to evaluate chatbot systems.

The ML and NLP capabilities of the frameworks are tested in some papers but the actual frameworks are not compared in most cases. Most papers focus on the NLP capabilities and not on other aspects like usability, simplicity or the comparison of the the functionality of the different frameworks. Papers like Braun et al. [15] show that the differences of chatbot frameworks is small when it comes to NLU performance. Hence, it doesn't make a huge difference which framework is chosen when the focus lies solely on NLU. The NLU capabilities of the framework are important in general but to give a development recommendation other aspects like the available features and the simplicity of a framework are more important.

Chapter 4

Methodology

4.1 Approach

The problem (sickness notification and vacation notification) needs to be evaluated to determine if the problem is suitable for chatbots. Sickness notifications are a repetitive task, the task can be automated and requires simple Back-and-Forth communication hence the problem fits the description of Raj [29]. The same is true for holiday notifications since the idea of both remains the same. The problem is suitable for chatbots since all three questions can be answered with yes. The next step is to determined the converstaion type. The two major chatbot conversation categories mentioned in Chapter 3: State-of-the-Art are goal-oriented and end-to-end conversations [36, 13, 28]. A goal-oriented system helps a user to achieve a task [28]. In the case of a sicknes notification the system needs to aid and guide the user through the process to reduce the workload. This description fits perfectly for goal oriented dialog. The domain is clear and of limited size since the user can only ask about the sickness/vacation notification process and nothing else. The bot also needs to retrieve specific data from the conversation to identify the person. The retrieval process is a specific task with specific entities and fits a goal-oriented system. In open-ended systems the conversation is not limited to a domain and the conversation can evolve in any direction. This is definitely not the case for the sickness and holiday notifications because the problems are domain specific. Therefore, goal-oriented technologies fit the the problem description. End-to-end systems won't be considered in this thesis because their performance in goal-oriented settings is not good enough yet as Bordes et al. [13] mentioned. The given problem of sickness notifications is a domain-specific/goal-oriented (Deshpande et al. [16], Williams et al. [35], Braun et al. [15], Williams et al. [36]) task hence domain-specific frameworks are suitable. The framework selection for this thesis is based on chapter 3 State-of-the-Art and Section 1.5 Prerequisites. Section 1.5 describe the requirements of the company and chapter 3 shows which technologies are used in papers, articles, and books. To match the prerequisites at least one cloud (Braun et al. [15], Rahman et al. [28]) and one local (Braun et al. [15]) chatbot are needed and IBM Watson Assistant needs to be taken because an IBM technology is on the wish-list. A common cloud technology is Dialogflow (Braun et al. [15], Dutta [17], Singh et al. [32], Raj [29], Rahman et al. [28], Godse et al. [21]) hence it is selected. A common local standalone technology is Rasa (Braun et al. [15], Singh et al. [32], Bocklisch et al. [12], Raj [29], Gregori [22]) and is the local tool of choice. The chosen technologies are Dialogflow, IBM Watson (Rahman et al. [28], Ahmad et al. [11], Godse et al. [21], Gregori

[22]), and Rasa to fulfill all requirements. All three prototypes will be used to implement the same task. An example convsersation of the sickness notification task is shown in Figure 5.2 They will be trained with the same data listed in Table 4.1, 10.2, 10.1, and 10.3. They will have the same intents, entities, and utterances. These steps are necessary to ensure a fair comparison. The analysis of the sickness notification dialog is shown in Table 5.2.

No	Utterance	Person Entity	Date Entity
1	I am sick	✗	✗
2	sickness notification	✗	✗
3	I am ill	✗	✗
4	[Alfred Mayer] is sick	✓	✗
5	my colleague [Maria Müller] is sick until [wednesday]	✓	✓
6	colleague [Simone Bauer] is ill	✓	✗
7	report [Franz Dorfer] sick	✓	✗
8	[Stefan Weber] is ill till [Friday]	✓	✓
9	sick [Emma Wagner] [6th of June]	✓	✓
10	[Sophia Richter] feels sick today	✓	✗

Table 4.1: Sickness Training Utterances

4.2 Training and Test Setup

To test chatbot systems they need to be implemented. A chatbot is based on natural langage processing and needs training data.

4.3 Course of Action

First of all, the problem needs to be analyzed and classified. The problem can either be suitable or unsuitable for the use of chatbots. It's suitable if the problem can be solved by simple back and forth communication, is repretitive, and can be automated Singh et al. [32]. Then the conversation needs to be analyzed. In general, the conversation needs to be solvable through general conversation or task-oriented communication. The conversation can be represented as flowchart like Singh et al. [32] did. The required intents, entities, and actions need to be defined based on the conversation. Training data needs to be created for the intents and entities where necessary. The same trainig data is used for every technology to ensure a fair comparison of the results. The prototypes are then developed based on the training data. The resulting prototypes need to implement the same dialog structure. It's either possible or impossible to implement the dialog structure with the chatbot technologies. If it's impossible to implement the dialog structure the technology won't be recommended. Afterwards, the test data needs to be created for the comparison and evaluation of the chatbots. The test data is the same for all bots.

4.4 Comparison and Evaluation

A main part of this thesis is the comparison and evaluation of the frameworks and the prototypes. The frameworks and prototypes will be compared in the sickness/vacation domain. The two major tasks of chatbot systems are the entity extraction and the intent classification. Geyer et al. [19] compared frameworks based on their entity extraction capabilities. Based on the work of Geyer et al. [19] the frameworks will be tested for their entity extraction capabilities. This is measured by counting the true/false positives and true/false negatives. These four values are used to calculate precision and recall which are used to calculate the f-score. The f-score is the measurment criteria for the performance. An f-score of 1.0 is the maximum which can be achieved. The frameworks will be ranked based on their f-score. The f-score will be calculate for each use-case relevant entity individually (date, date-span, person) and for all relevant entities together. This will show how good the individual entities of the frameworks are and how good the entity extraction works in general. Rasa doesn't provide entities but the pipeline can be adjusted to include technologies which provide entities. A popular technologies for date, time, and all sorts of numeric values is Duckling (Wit.ai). Spacy is a technologie which provides the required person entity and it's available for Rasa. The cloud frameworks all have built in system entities. The same process will be used for the intent classification task. The intent classification provides a confidence score value. The confidence score will be used to rank the intent classification capabilites in a second ranking. The range is from zero to one where values close to zero are bad and values close to one are good. The confidence score alone is not enough to do the second ranking since the technologies always pick the best fitting intent. The best fitting intent doesn't have to be the correct/expected intent. The technologie could say it's 60% sure the intent none is correct and the expected sick intent only reaches 40%. In this case the none intent would be picked. The confidence scores alone are biased since no matter which intent is recognized there will always be a value for each intent no matter the input. The average confidence score will be calculated for the two use-cases individually and combined for both. This is an indicator if a technology has problems with or excells on a specific use-case. The average confidence score should be as high as possible. An average confidence below 50% is low and indicates that the technology is not sure that the intent was classified correctly. A low confidence score can be improved through more training data in most cases. The intent classification capabilities need to be checked for the English and German language. The potential users for further projects use German as input language. The expectation is that there is small to no difference in the development process, that the same functionality is available for Germand and English, that the performance is on an equal level, that the entity extraction works for both languages, and that the same predefined entities are available for both languages.

Singh et al. [32] says the best evaluation criteria is if the task can be achieved. This will be used as evaluation criteria and will be based on the design and prototype chapters. The design chapter provides the blueprint which needs to be implemented with the different technologies. The prototype section shows if the blueprint can be implemented successfully with a specific technology. The technologies will be classified binary as true if it's possible and flase if it's impossible. The expectation is that the implmentation is possible with all chatbot frameworks namely Dialogflow, Watson Assistant, and Rasa. A successful implementation can't be expectd with LUIS. LUIS is a NLU service and not a chatbot technology. Hence, it doesn't offer a mechanism to handle dialogs which is neces-

sary to implement a chatbot. The Microsoft Bot Framework can be used to build chatbots and uses LUIS. One of the main reasons why companies use or want to use chatbots is because they are cost efficient. For this reason the prices of the chatbot frameworks need to be compared. In general, cheap is treated better in this ranking than expensive. This criteria does not make a statement about the quality of the framework. For this reason the ranking should be reconsidered after the test when the quality ranking is available. If a technolgy is good it's a candidate for the recommendation. If a technology is bad the price doesn't matter because it will not be recommended. Hence, a good and cheap technology is wanted. For the usage in a business application the prices of all version of the frameworks are relevant especially if an enterprise verison is offered. The expectation is that the frameworks cost roughly the same and that the prices are easy to compare. The ISO 25010[6] standard defines portability as an evaluation criterion for software systems. Portability is a big issue with chatbot technologies. One sub criteria for the evaluation is the number of providers where the chatbot can be deployed. It's expected that the cloud technologies can run only in the cloud environment. This criteria focues on Rasa to see where and how it can be deployed. Another aspect fo portability is the number of devices a developer can work with with a focus on the hardware and software which needs to be present to run the framework. The expectation is that a cloud chatbot can be developed from everywhere as long as a web-browser and an internet connection are available. It's expected that Rasa is at a disadvantage in this category. The migration from one technology to another is an aspect of portability. It's expected that all technolgies safe the data differently and that it's not easy to migrate between them.

Chapter 5

Design

5.1 Architecture

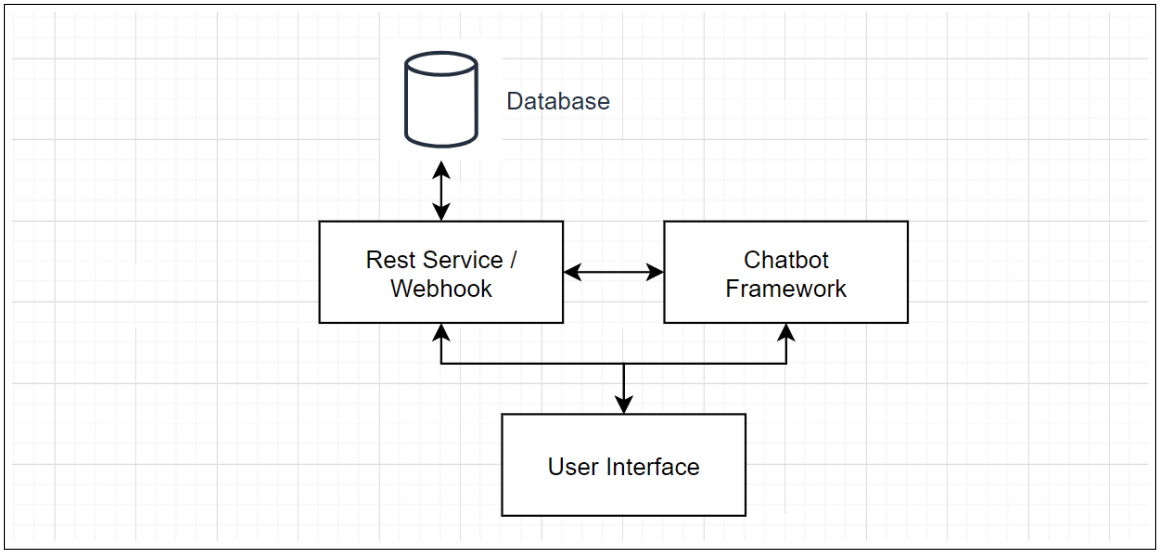

Figure 5.1: Minimalistic Chatbot Architecture

One possible architecture for a minimalistic chatbot system with database communication is shown in Figure 5.1. The chatbot framework takes care of all conversation related tasks. Conversation related tasks are the intent recognition, entity extraction, and the general flow of the dialogue for instance. The user interface of Figure 5.1 fetches the input from the user, submits it to the chatbot framework and displays the response messages. The UI can additionally communicate with the database via the REST service or webhook if necessary. The DB handles data-related tasks like the possible items and can be used for the validation of the extracted entities by the chatbot framework in combination with the webhook. The webhook separates the data logic from the application logic and is used for validation, data submission and retrieval. It's also used for other chatbot related issues like the validation of intent related data and error responses to the chatbot framework.

5.2 Story/Flowchart

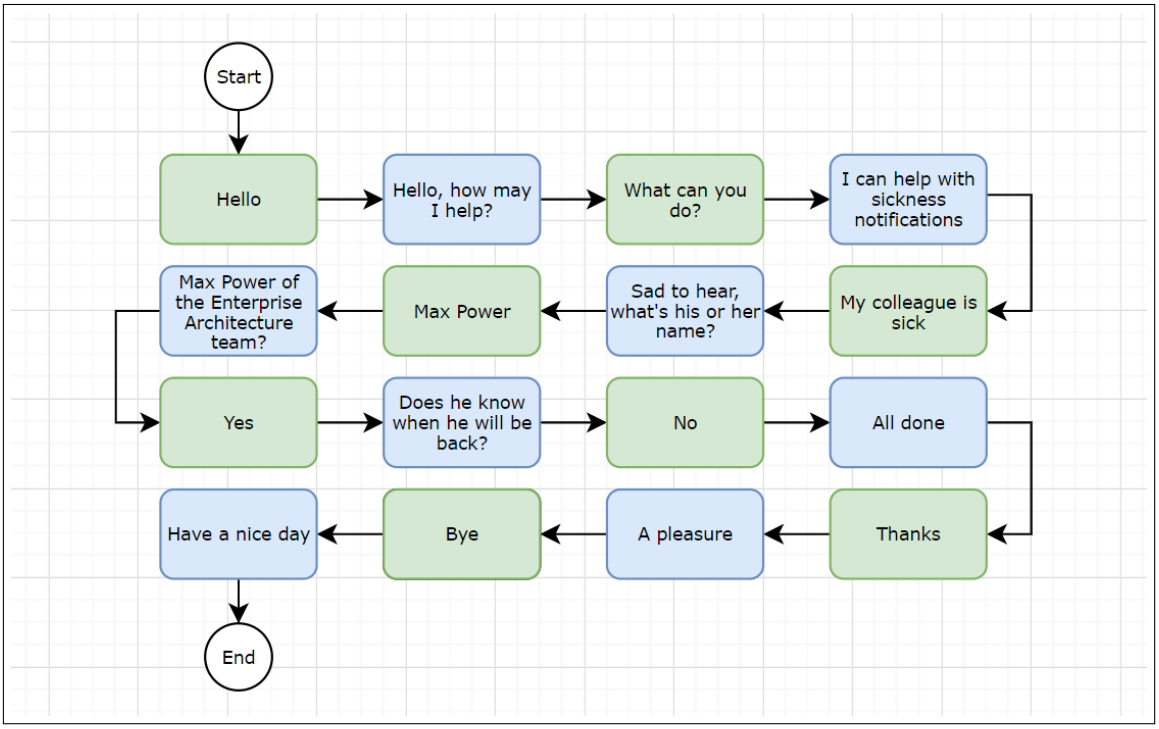

Figure 5.2: Sickness Notification Example Conversation

As mentioned in Følstad and Brandtzæg [18] the design focus for chatbot systems lies on the conversation itself. A story is an example of a dialog/conversation between user and chatbot. An example dialogue to report a colleague sick is shown in Figure 5.2 where the user input is highlighted green and the chatbot response is highlighted blue. The intent of each user input needs to be found to create the knowledge base for the domain. The intent for the user input ”Hello” can be marked as greeting, the second input is the general information what the chatbot can do, the third input is the sickness notification intent. Then the necessary information is collected followed by some chit-chat and the end of the conversation. The interesting entities, in this case, are the name of the employee which is required all the time, the confirmation that the found employee is the correct one and the return date which is optional since it’s not possible to tell in every case. The resulting intents, entities, and utterances for the example sentences of the conversation shown in Figure 5.2 are listed in Table 5.2. The detailed conversation structure for the use cases of this thesis is shown in Figure 5.3. The figure shows the flowcharts for the sickness notification and vacation request. User inputs are highlighted red, bot responses are green and general actions are marked purple. Both use cases start the same way because the person needs to be identified. If the entered name can’t be validated the user can correct the name or enter a different name. Then the user needs to confirm that the correct person was found. If the user responds with no the name of the person can be corrected. Afterward, a date needs to be entered in both cases. The sickness intent asks for an optional return date since the yes and no path lead to the same action and the conversation ends afterward. The vacation intent requires a start and return date. The user is asked for the date until a valid date has been entered. After both dates have been entered the vacation request can be processed and the conversation end is reached.

Figure 5.3 describes every step of the conversation from start to end. Hence, it defines the functionality of the chatbot and the flow of the conversation. Table 5.1 defines the intents, name of the intents, entities, the data types of the entities, and the names of the utterances based on the information present in Figure 5.3. The combination of Figure 5.3 and Table 5.1 is the blueprint for the implementation of the chatbot. It is valid for tools using the concepts intent, entity, and utterance.

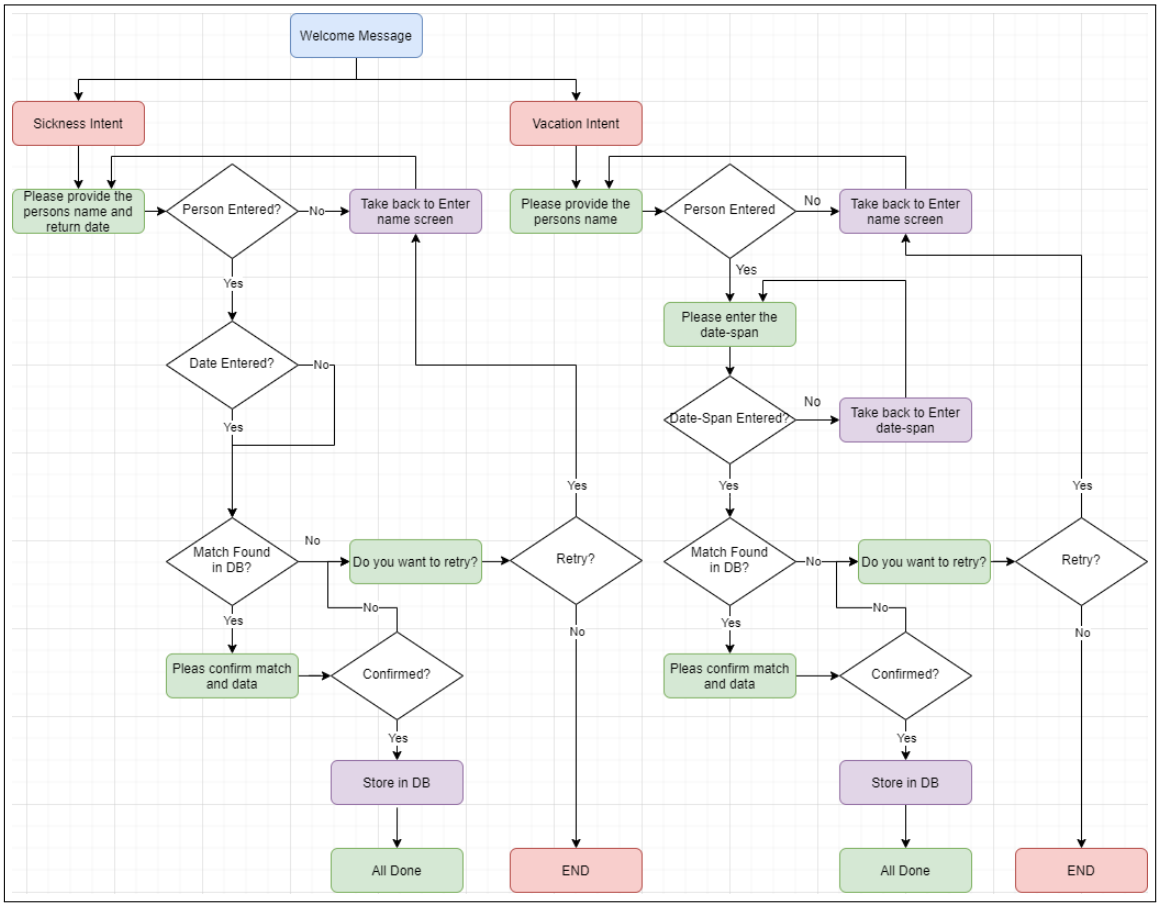

Figure 5.3: Chatbot Conversation Flowchart

5.3 Intents, Entities and Utterances

The base for this section is Figure 5.2 which shows an example conversation for a sickness notification from start to end. The identified information from the conversation is displayed in Table 5.2. In general, the greeting and goodbye intents are expected to be in almost any chatbot conversation. A thank you intent is also very likely. The user should have the option to ask for the available options. The core intent of the whole process is the sickness intent. To do the sickness notification the user needs to enter the name of a person. If the person can't be found the user needs to be notified and the operation is aborted. If the person was found the user needs to confirm that the right person was found. In production, there could be multiple employees with the same name.

Because of this ambiguity, the confirmation is part of the dialog. Then the optional return-date needs to be extracted. It's optional because especially in the case of a sickness the return date is often unknown. The other case is e.g. a broken arm where the employee

Intent	Reference from Figure 5.3	Type	Utterances and Entities
-	Welcome Message	Utterance	Utter_Welcome
Sickness Intent	provide sick persons name	Utterance	Utter_Enter_Sick_Name
	person name	Entity	Person
	Confirm person name	Utterance	Utter_Confirm
	enter return date	Utterance	Utter_Enter_Return
	return date	Entity	Date
	all done	Utterance	Utter_All_Done
Vacation Intent	proivde persons name	Utterance	Utter_Enter_Name
	person name	Entity	Person
	Confirm person name	Utterance	Utter_Confirm
	enter start date	Utterance	Utter_Enter_Start_Date
	start date	Entity	Date
	enter return date	Utterance	Utter_Enter_Return_Date
	return date	Entity	Date
	all done	Utterance	Utter_All_Done

Table 5.1: Analysis of Figure 5.3

might return tomorrow. Then the sickness notification is done. In the case of this thesis, the chatbot is the important part hence the backend is mocked for simplicity. The user might thank the bot or end the conversation with a goodbye message, or the user might not respond after the all done message. These are the three possible ends of a conversation. The basic concepts extracted from Figure 5.2 are shown in Table 5.2. There are the intents discussed above, the entities and their types, and the bot actions. For this use case, the needed entities are a person or the name of a person, a confirmation of boolean type, and a date. Alternatively for the confirmation, the name of the department the person works in can be used. The department is modeled as a custom entity as described in Chapter 2: Basics in Section 2.6. The bot responds to the user with utterances. The conceptual names of the utterances are shown in Table 5.2 under Bot Action. In the simplest case, a bot action is an utterance. For the utterances, different versions need to be written to respond to the user with alternating sentences. For the vacation notification, the process remains the same. The greeting, thanks, goodbye, and list options intents remain the same. The new intent is the vacation intent. To validate a vacation request the name of the person and the department are needed again. The only change in the process is that there needs to be a mandatory start and end date. There are no new datatypes/entities needed for the second use-case.

5.4 Problem Analysis

The problem is task-oriented.

Some psychological aspects of Brandtzaeg and Følstad [14] have to be included. Brandtzaeg and Følstad [14] stated that the user impression depends greatly on the users mindset. If the user thinks he's talking to a bot the user expects less but the conversation is more likely evaluated as unnatural [14]. If the user assumes he's talking to a human and the bot is unable to achieve the task the user is very disappointed [14]. Since the problem is a task-oriented one the general conversation capabilities are not the primary focus. This

Sentence	Intent	Entity	Bot Action
Hello	Greet	-	Utter Greeting
What can you do?	List Options	-	Utter Options
my colleague is sick	Sick	-	Utter Ask Name
Max Power	-	Person Name	Utter Confirm
Yes	-	Boolean	Utter Return Date or Failed
No	-	Optional Date	Utter Done or Failed
Thanks	Thanks	-	Utter You'r Welcome
Bye	Goodbye	-	Utter Goodbye

Table 5.2: Sickness Notification Intents, Enitities, and Actions of Figure 5.2

means the bot is unable to fake a human conversation which goes beyond the expected functionality. Hence, the user needs to be informed that he's talking to a bot. For further development steps, a gender can be introduced to make the bot feel more human-like if necessary. Another way to give the bot a more natural feel is to look at the message response times. They should not be to fast. If the response comes too fast it feels unnatural because a human being can't answer as fast as a machine.

5.5 Webhook

As explained in Chapter 2: Basic in Section 2.9 the webhook is an endpoint where services can send requests to. In this thesis three frameworks are present and each framework gets and endpoint. The service is also used to access general information about the state of the data. The Dialogflow endpoint is reachable under "/dialogflow/webhook", the Watson endpoint is located at "/watson/webhook", and the Rasa service submits post requests to "/sick" and "/vacation" to submit data to the REST service. For development purposes, the last request of Watson can be viewed at "/watson/request", and the last request of Dialogflow at "/dialogflow/request". To see all successful sickness notification entries "/sick/all" can be used, and under "/vacation/all" all successful vacation entries can be viewed.

The service fakes a database and stores the request. After the information has been collected successfully by a framework a request is submitted to the service. The service can do everything with the data and is the point where the company can execute actions. All frameworks can call the webhook at the end when the data has been collected but also after each user request if necessary. This way the information can be validated and modified by the service.

5.6 Messenger UI

The design of a chatbot focuses on the conversation and not on the UI. The UI requirements are always the same for chatbots. The basic features required are writing a new message, sending a message, and displaying a message. The messenger UI shown in Figure 5.4 can enter and send a message, and displays message cards.

A user can enter a new message at the bottom and click the send button to send a message.

Brandtzaeg and Følstad [14] states that successful chatbot should inform users that they are talking to a bot. Therefore, the heading has been adjusted to remove ambiguities and includes the term chatbot. The messenger UI is shown in Figure 5.4.

Figure 5.4: Chatbot UI

Chapter 6

Prototype Implementation

6.1 Dialogflow

Dialogflow[4] formerly known as API.ai is the chatbot framework of Google and is hosted in the Google cloud. To create a new agent the main language needs to be selected and the name of the agent needs to be defined. The selected language is the interaction language of the users. Dialogflow offers the option to create a mega agent by combining other agents. The web interface of Dialogflow is shown in Figure 6.1 where the menu shows the intents and entities tabs. Dialogflow supports the basic concepts of intents, entities, and utterances. Hence, Table 5.1 is used for the implementation of the intents, entities, and utterances.

6.1.1 Intents and Entities

When a new intent is created some options are present. A Dialogflow intent has training phrases, actions, parameters, a response list, a context, and the fulfillment. The training phrases are a list of utterances a user would enter. Parameters are the slots that need to be filled in the conversation. The parameters for the sickness intent are the rows of Table 2.2 where the entity cell is filled with text. The response list is a list of utterances which is used to respond to the users question. The context is used to pass parameter values to other intents. In the fulfillment section, the webhook can be enabled. By default, an intent is handled by Dialogflow without communication to another service. In the fulfillment section, a webhook call can be enabled. For the sickness intent, two parameters are required. The name of the person is a non-optional parameter, the return date is an optional parameter. Dialogflow provides many predefined entities and intents. The date and person entities are predefined by Dialogflow and it's not necessary to create them. Additionally, yes and no intents are also available and can be used and adjusted for the current sickness notification use case. The final structure is a sickness intent which is followed by either the yes or the no intent. For this, the follow-up intent concept is used where follow-ups can be entered. At the end of the no intent the chatbot responds with a retry response. At the end of the yes intent, the bot responds with an all done response. This ends the conversation of the sickness use case and the yes and no intent are marked as the conversation ends. The process looks the same for the vacation use case. The vacation intent needs to be created and has a follow-up intent yes and no to confirm the user input. The entities required are a person and a date-span. Dialogflow offers a predefined entity called date-period which offers this functionality and it's not required to create one. The

date and date-period entities offer advanced features. A valid date period is for instance Monday to Friday. This is translated to the correct values if e.g. Monday is in the past the resulting date period will be from the next Monday to Friday which follows next Monday. For the example sentences, the entities can be used to create alternative ways to fill the slots. The user can be prompted slot after slot or enter them in the first request to speed up the process for more experienced users. In a sentence like ”Max Power is sick and will return on Monday” all the required information is already in the sentence. The entities need to be defined in the training sentences. Then the system learns that there are multiple ways to acquire the information. With suitable training sentences, the chatbot can extract the information directly and jumps directly to the confirmation. In general, custom entities are defined as a list of entries or a list of entries with synonyms. An advanced feature of Dialogflow is small talk which can be enabled in the settings. When small talk is enabled the bot is able to respond to requests like ”What are you”, ”Bye”, and ”sorry”. This is another useful feature which is predefined and requires no development effort and can be enabled with one click.

6.1.2 Webhook

The Dialogflow webhook requires a HTTP POST URL as an entry point. The communication relies on JSON as a format. The entry point of the webhook is shown in Listing 6.1. A request like in Listing 2.1 can be accessed like a dictionary or an array shown in Listing 6.2 where the name of the intent gets extracted. After the intent has been extracted the parameters are extracted to process the request. In the error case, a new fulfillment message is returned. A fulfillment message as return value prevents follow up intents. In the success case, an empty response is returned and the bot continues normally. In Listing 6.3 a skeleton for intent handling is shown for a success and error scenario for an intent called ”Sickness Confirmed”.

```
@restService.route('/dialogflow/webhook', methods=['POST'])
def dialogflowRequestEntryPoint():
    return asJsonResponse(dialogflowRequestHandler()), 200
```

Listing 6.1: Dialogflow Webhook Entry Point

```
req = request.get_json(force=True)
intent = req.get('queryResult').get('intent').get('displayName')
```

Listing 6.2: Dialogflow Request Parameters

```
if intent == 'Sickness Confirmed':
 params = req.get('queryResult').get('outputContexts')[0].get('
    parameters')
 employee_name = params.get('employee').get('name').lower()
 return_date = params.get('return_date')
 if !employeeExists(employee_name):
  return {
    "fulfillmentMessages": [
      {"text": {"text": ["Employee not found. Enter sick to retry."]}}
    ]
  } # RETURN ERROR MESSAGE
  else:
   db.store_sickness(employee_name, return_date) # PROCESS DATA
   return {} # CONTINUE NORMALLY
```

```
14 elif intent == 'Vacation':
15     ...
```

Listing 6.3: Dialogflow Intent Handling

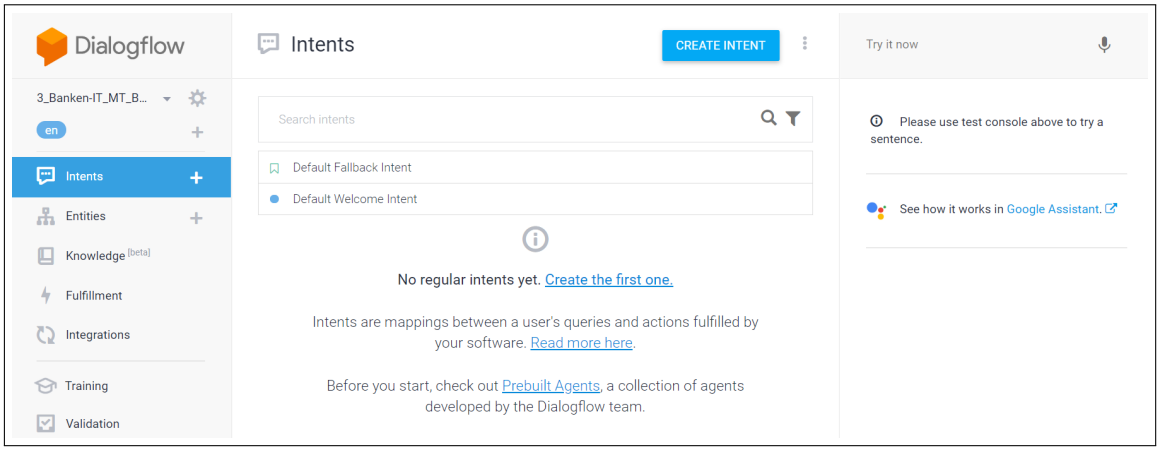

Figure 6.1: Dialogflow Web Interface

6.2 Watson Assistant

Watson Assistantwat [10] is the cloud chatbot technology of IBM hosted in the IBM cloud. An IBM cloud account is required to use the services. First, a new assistant needs to be created which requires a name. Then dialog skills can be created. For the sickness notification process, a sick dialog is created. The web interface of Watson Assistant is shown in Figure 6.2. The interface offers the concepts intent, entity, and dialog.

6.2.1 Intents and Entities

The required entities are person and date. Watson Assistant offers system entities for persons and dates. In the settings under options, a webhook URL can be entered. The person entity is marked deprecated and is available in English but not in German. This means a person entity needs to be defined as a replacement for the person entity using entity annotations like recommended by IBM. To do this every occurrence of person names in the test sentences need to be highlighted and marked as person. For the sickness use case, a new intent needs to be created. Then the training phrases need to be added. It is possible to create new entities by using examples or regular expressions. After the intent had been defined a new dialog needs to be created. An intent needs to be selected for a dialog. The dialog is triggered when the intent condition is met. In this case, the intent is the sickness intent. In the settings menu, the dialog can be customized. A customization is necessary to enable slot filling and the webhook call which are both necessary for the sickness use case. The required slots are person of type person and the return date of type date. The person is mandatory and the date is optional. After the information has been collected the webhook is called.

6.2.2 Dialog and Webhook

The format of the webhook call is JSON. The parameters need to be specified by hand. The required parameters are the name of the intent, the person name, and the return date. The name of the intent is necessary to execute the correct action at the webhook. The name and return date are the information that needs to be stored. It can be seen that no meta-information is sent in the request of Listing 6.4. The information and format submitted to the webhook is defined by the developer. The returned information from the webhook is stored as a context variable in Watson Assistant and can be accessed like a struct in C++. The format for the webhooks response is also up to the developer. Based on the webhook response the dialog can go different ways. If the person was found and a date was entered the date needs to be displayed in the chatbot response. If the person was found but no date was entered the chatbot response needs to ignore the date. Whenever a person was found by the webhook the user needs to confirm that the entered information is correct. Whenever the person wasn't found the user needs to be prompted to retry. These cases need to be implemented with the conditional statements of Watson Assistant. Table 6.1 shows the conditions, responses, and actions that need to be implemented for the use case. Chatbot responses can be picked sequentially, at random, or as a multiline response. The multiline response displays all response lines at once. The sequential version picks them in order and the random variant picks the response at random from the available responses. The confirmation logic needs to be implemented next. No system entity or dialog for the confirmation logic is present. This can be implemented with two intents or an entity. An entity with two possible values is a logical approach. One value is true the other is false. The entity is called confirmation. For these two cases, synonyms are defined like confirm, yes, and OK for true. When the entity is detected the correct path can be selected with a condition. Similar conditional logic was used for the webhook response shown in Table 6.1. A new dialog node needs to be created for the confirmation or abort logic. There is a jump option in the settings of the dialog node. In the success cases of the sickness dialog a jump to the newly created confirm or abort dialog needs to be entered. A slot needs to be filled and a webhook call is necessary to store the collected information. These two settings also need to be enabled for the confirm or abort dialog. The slot is of type confirmation entity and is required. After the information has been gathered a request is sent to the webhook. The two possible scenarios are that the information is stored or something goes wrong. If the information has been stored successfully an all done message is displayed. Else a generic message is displayed indicating that something went wrong. This ends the sickness dialog. For the vacation use-case, a date span is needed. No predefined entity exists for date-spans but a date entity is present. It's not possible to create entities containing entities with Watson. The options are to create an intent with two dates in it or a dialog node which fetches the required data. The dialog node approach is chosen because it comes close to the behavior implemented with Dialogflow. A new dialog node named vacation is created. For the dialog node, an intent called vacation needs to be created with the example sentences. The dialog needs a mandatory person slot and two mandatory date slots. The start and return dates of the vacation. After the information has been collected the information can optionally be validated through the webhook. Then the user needs to confirm the information like in the sickness use-case. Then the final request is sent to the webhook which stores the collected information. The final all done message is displayed and the vacation use-case is finished.

Condition	Chatbot Response	Jump To Action
$response.OK && $date!=null	Please confirm $person is sick until $date.	Confirm or abort
$response.OK	Please confirm $person is sick.	Confirm or abort
$response.ERROR	$person not found. Enter sick to retry.	Wait for input

Table 6.1: Conditional Responses of Watson Assistant

```
[
    {
        "intent": "Sick",
        "person_name": "Anna Nass",
        "return_date": "2020-05-27"
    }
]
```

Listing 6.4: Watson Assistant Request Format

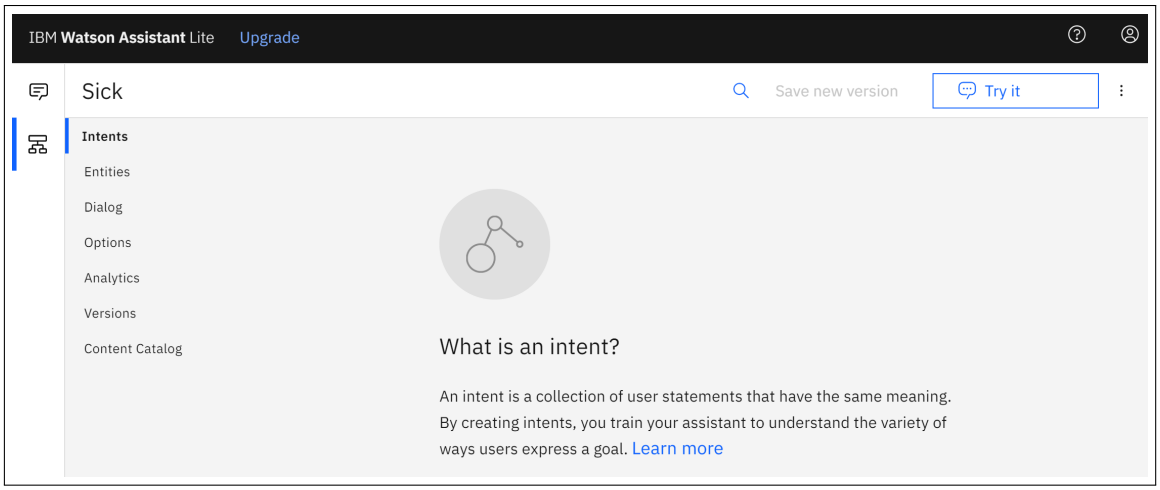

Figure 6.2: Watson Assistant Web Interface

6.3 Rasa

Rasa is an open-source framework for NLU and chatbots. Rasa[8] can either be used as a Docker container or it can be installed locally. Rasa uses Python as a development language and can be installed using pip. A Docker compose file for Rasa and the action server is shown in Listing 6.5. Rasa is split into a natural language and logic part. The natural language part takes care of everything which is related to the conversation like intents, entities, the story, and the conversation flow. The second part handles the slot filling and the actions. For each part, a Docker image can be created. The Docker compose file of 6.5 can be used in similar fashion for the Azure deployment in combination with the images. Azure also offers a Docker registry to upload and pull custom images. It's important to choose a compatible version of rasa and rasa-sdk. The Rasa image is responsible for the NLU part and the Rasa sdk is the actions server.

```
version: '3.0'
services:
    rasa:
    image: rasa/rasa:1.4.6
    ports:
        - 80:5005
    networks:
        - app_net
    volumes:
        - ./:/app
    command:
        run
        --enable-api
        --cors *

    action_server:
    image: rasa/rasa-sdk:1.4.0
    volumes:
        - ./actions:/app/actions
    expose:
        - 5055
    networks:
        - app_net
networks:
    app_net:
```

Listing 6.5: Rasa Docker Compose File

The information for a Rasa chatbot is split in multiple files. The nlu.md file stores the intents with the utterances used for training. The intent format is shown in Listing 6.6.

```
## intent:sick
    - My colleague [Max Power](PERSON) is sick until [Friday](date)
    - somebody is sick
    - a colleague is ill today
```

Listing 6.6: Rasa Intent Format

The stories.md file stores the stories. A story defines the flow of the dialog. The sickness story is shown in Listing 6.7.

```
## Sickness Affirm
* sickness
    - sickness_form
    - form{"name":"sickness_form"}
* submit{"PERSON":"Anna Maria Mayer"}
    - sickness_form
    - slot{"PERSON":"Anna Maria Mayer"}
* submit{"time":"2020-05-30T00:00:00.000-07:00","DATE":"today"}
    - sickness_form
    - form{"name":null}
    - slot{"time":"2020-05-30T00:00:00.000-07:00"}
* affirm
    - action_submit_sickness_data
    - action_restart
```

Listing 6.7: Rasa Story Format

All the actions are defined in actions.py. The sickness form slot filling is an action for instance. The intents, entities, and actions need to be listed in the domain.yml file. Otherwise, they can't be used. The defined actions can be simple utterances or something complex like the slot filling of a form. In utterance actions, the response is chosen at random. In the endpoints.yml file, the action server needs to be entered to use the actions server for slot filling. The actions server runs separately from the NLU service at the standard port 5055. The NLU service is accessible at default port 5005. Rasa offers great freedom when it comes to configuration. This is important because Rasa does not offer predefined entities as such. But spacy does offer the required date and person entities for the use cases. In the config file shown in Listing 6.8 the pipeline includes the spacy entity extractor to enable the use of predefined entities. spa [9] lists the supported entities under named entity recognition. The entity recognition of spacy does work but the dates are stored as they are. The best case is when the date is converted in a standard format like Dialogflow and Watson Assistant do. To use a more powerful date parsing the Duckling[5] Rasa Docker image is used. Duckling converts entered dates and times to a standard format as the cloud services do. To add duckling to the project it needs to be added under pipeline in the config file. It recognizes strings like "next Monday" as dates and converts them correctly. The person entity of Spacy works as expected. The used entities need to be added under entities in the domain.yml file. People with machine learning experience can also define their own pipelines or adjust parameters. The detailed pipeline is listed in the appendix in Listing 10.1. The config file defines policies. The form policy enables the use of forms and the fallback policy enables a fallback action which is triggered when the confidence score is below the defined threshold. In the sickness use case, a generic utterance is the fallback response.

```
language: en
pipeline:
    - name: "SpacyNLP"
    - name: "SpacyTokenizer"
    - name: "SpacyFeaturizer"
    - name: "RegexFeaturizer"
    - name: "DucklingHTTPExtractor"
        url: "http://localhost:8000"
    - name: "SpacyEntityExtractor"
    - name: "EntitySynonymMapper"
    - name: "SklearnIntentClassifier"
policies:
    - name: MemoizationPolicy
    - name: TEDPolicy
    - name: MappingPolicy
    - name: FormPolicy
    - name: FallbackPolicy
    nlu_threshold: 0.4
    core_threshold: 0.3
    fallback_action_name: action_default_ask_rephrase
```

Listing 6.8: Rasa Configuration

The general command required to work with Rasa are train, shell, run actions, and x. To train Rasa the command "rasa train" is used. The shell command starts an interactive shell where the bot can be tested. The action server needs to be run separately. If Rasa-X is installed the command "rasa x" starts a web UI shown in Figure 6.3. The web interface offers the discussed files in the training section with additional features like the

conversation graph (right) called flow for a story (middle) shown in Figure 6.3. Rasa X offers interactive training where the chatbot is built through live interaction with the bot. Rasa stores a model every time it's trained. These models can be exchanged in the models section. The whole history is stored which can be used for features. If the feature does not work out the model can be set back to the original version. For the

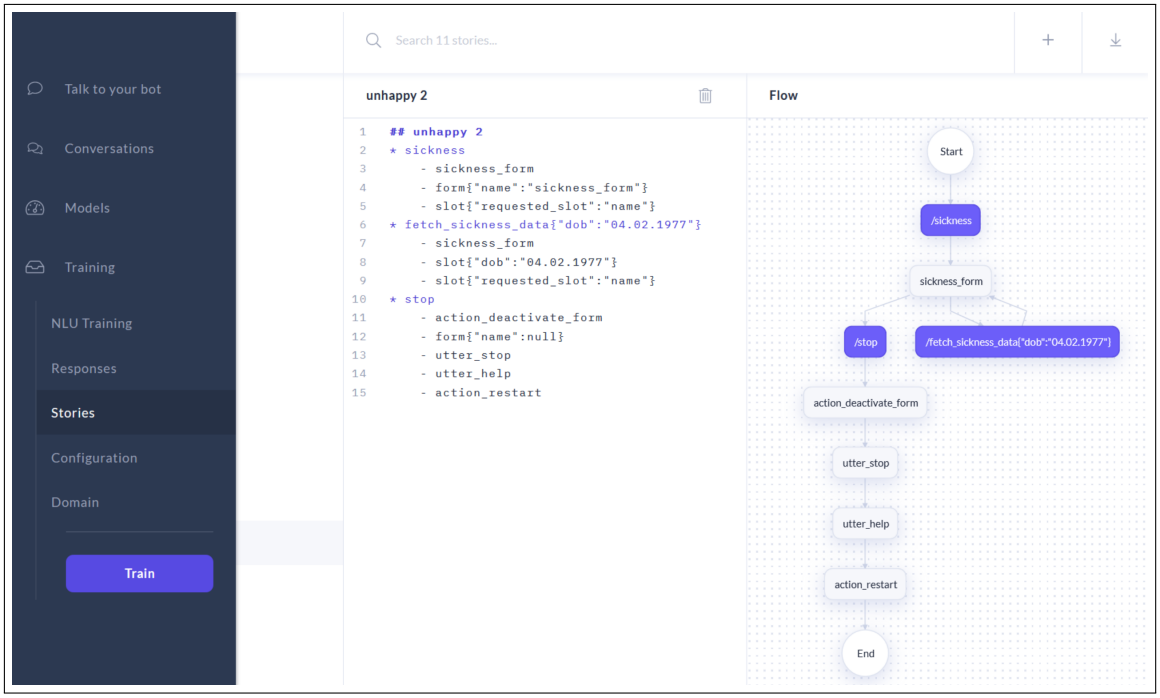

Figure 6.3: Rasa X Interface

sickness use-case, a new intent called sickness needs to be defined in the domain file. The sickness intent needs slot filling and the required entities are the person and the return date. The slots need to be defined in the slots section. Every slot needs an utterance which is named utter_ask_¡SLOTNAME¿. The utterance is displayed when the bot fills a slot. For the slot filling an action needs to be defined. An example slot fill action is shown in Listing 10.2. The action needs the name function which returns a unique name. The required slots need to be defined. If a slot is required it needs to be filled. The slot mappings function defines the type of the slots. The submit function is called as soon as the slots have been filled. In the submit function the collected information can be sent to the external endpoint for instance. Custom validation rules can be defined for each slot. This can be used to validate that the entered person name exists. Now the intent needs to be defined in the NLU file. The sickness intent and the training phrases need to be defined in the NLU file in the format of 6.6. The entities are marked with square brackets and the types with parentheses. Now the sickness story needs to be defined. When a phrase is recognized which is classified as sickness intent then the slot filling needs to be triggered. The next step is the confirmation dialog where the user confirms the entered information. A submit sickness data action is required. To classify the response the intent affirm and deny are created with some example phrases. After affirm is recognized the confirmation action needs to be triggered. In the run method of the submit action, the data can be validated and processed. If deny is recognized the slots need to be cleared and a retry message needs to be displayed. The sickness story for the successful cases is shown in Listing6.7. The vacation use case is not different from the sickness use-case in Rasa. The

Duckling time entity can process times, time-spans, dates, and date-spans. For Duckling there is no difference between a single date and a date-span. Both are stored in the time slot and are of type time. Hence, the same person and time slot of the sickness use-case can be used for the vacation use-case. The response messages need to be adjusted. For the vacation use-case, a new intent needs to be created in the domain file. The vacation training phrases need to be listed in the NLU file. A form and a submit action need to be created for the vacation intent. The structure can be copied from the sickness use-case. Then the response messages and conditions are adjusted as needed. The actions need to be listed in the domain file to be usable in the stories. Then the new stories need to be created. With Rasa-X the stories are created with live interaction with the bot. The stories are structured like in the sickness use-case.

6.4 LUIS

With LUIS[7], the language understanding service of Microsoft, NLU services can be created. NLU is only a part of a chatbot and it's not possible to create a chatbot with LUIS alone. The chatbot features are available through the Microsoft bot services in combination with LUIS. A LUIS account is necessary to create NLU services. When a new project is created the name and a language need to be defined for the NLU application. The chosen language is the language the users will use for the communication with the LUIS application. The modification is done in the build tab. The interface shown in Figure 6.4 offers intents and entities like described in Chapter 2. To create the NLU part with LUIS Table 5.1 will be used. 5.1 defines the intents entities and response utterances that need to be created in LUIS. Figure 5.3 has no influence on LUIS because the conversation handling is not part of the NLU service. To create an intent only a name and a list of training phrases (utterances) are necessary. Multiple options are available to create different kinds of entities. There are simple entities like e.g. a city. They describe exactly one concept. There are composite entities that are built from multiple entities (parts). E.g. a ticket order entity could consist of an order number, the number of tickets, and the event. An entity list is used when the valid items are limited like in the book hotel example. There is only a limited number of valid hotel names and the list holds all valid hotels. The invalid hotels are uninteresting in such a case. Regular expressions[23] can also be used to define an entity. It is also possible to add prebuilt entities and prebuilt domain entities. The entities required for the sickness intent are person name and date. LUIS offers the prebuilt entities date-time and person-name. This means no new entities need to be created for the sickness use-case since the are prebuilt. To test the NLU functionality two additional intents need to be created to simulate the conversation. A submit name and a submit return date intent are required. The submit name intent extracts the person name entity. The submit return date entity extracts a date-time entity. The advantage of predefined entities is the saved time and the advanced features. The date-time entity is able to match inputs like Monday or tomorrow to a valid and correct date by default. The year is also automatically added when the date e.g. first January lies in the past. This is important for the vacation use case when the vacation is requested for the next year.

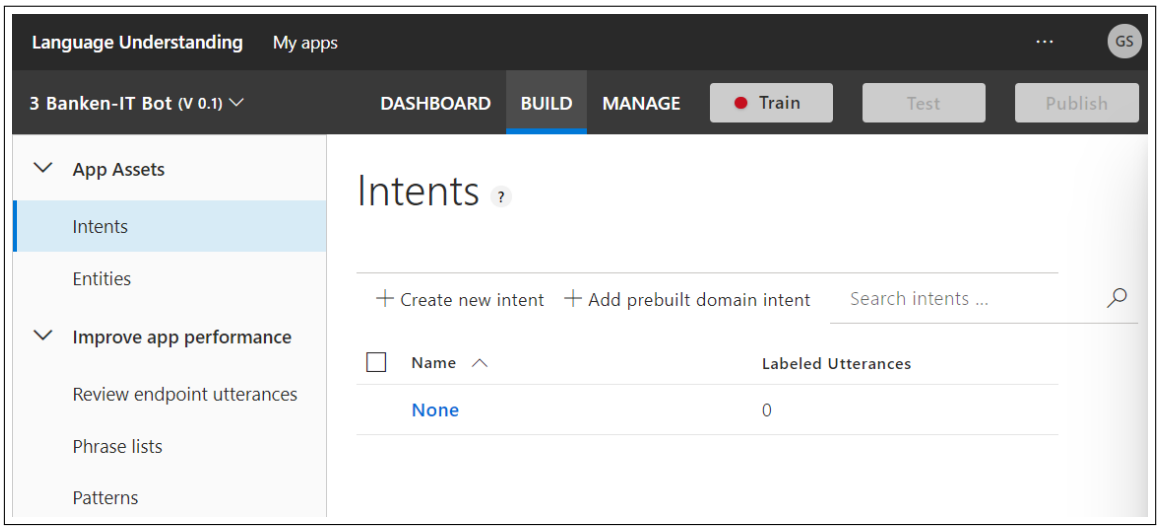

Figure 6.4: LUIS Web Interface

6.5 Frontend

The main purpose of the frontend is message handling. It tracks the user input, submits the input to the chatbot service when the submit button is pressed, and displays the response. A classic message in messengers has the properties sender, timestamp, and message. The message format of the frontend is shown in Listing 6.9

```
messages.push({
    sender: "<Sender>",
    message: "<Message>",
    timestamp: this.getTimestamp()
})
```

Listing 6.9: Message Format

A toolbar with title, message area, and submit area are necessary for a messenger UI. The structure is shown in Listing 6.10

```
<Box width={"25em"}>
    <MessengerToolbar title="3 Banken IT Dev Chatbot" />
    <MessageArea entries={this.state.messages} />
    <Divider light />
    <MessengerSubmitArea
        message={this.state.message}
        textHandler={this.handleMessage.bind(this)}
        submitMessage={this.handleSubmit.bind(this)}
    />
</Box>
```

Listing 6.10: Messenger Structure

The UI of the React messenger frontend is shown in action in Figure 5.4.

6.6 Webhook

Chapter 7

Result

7.1 Framework Comparison

Portability

An important factor for software systems is the portability. Portability is defined in ISO 25010 [6]. As a measurement criterion, the number of providers that support the technology is defined. A subfactor is how tied to a provider a technology is. When it comes to the chatbot cloud services (LUIS, Dialogflow, IBM Watson Assistant) the provider can't be changed. All three services run in the cloud of the provider and nowhere else. The data processing is done in the cloud and can't be done locally or on a company server. This means that the three cloud services are not portable at all and the provider can't be changed. With Rasa things are different. It is a local solution which means it can run on a local machine and a company server. The data processing is also done locally and never goes into the cloud of a provider. Docker images for Rasa are provided too. It's possible to run Docker containers in the clouds of all big providers. This means Rasa can run everywhere where Docker or a Python environment are possible and the provider can be changed easily. Another aspect of portability is the number of devices where the chatbot can be developed from. The cloud frameworks are accessible via the internet and can be used and developed from every device which has an internet connection and a browser. Rasa can only be developed on machines which have access to the source files. It is per default not accessible via the internet. However, it can be deployed in a cloud environment. Then it can be used like the cloud services but it still can't be developed easily on any device. The third aspect of portability is the migration from one technology to another. Rasa offers an import functionality (beta version) which allows The import of Dialogflow, LUIS, Wit.ai, and IBM Watson chatbots. In general, all chatbot frameworks use JSON format to store the training data. But evert technology saves information differently and it's not easy to migrate an existing chatbot to another technology.

Implementation of the Design

like Singh et al. [32] said, the best evaluation criteria is if a task can be achieved with a technology or not. The goal is to implement the dialogs shown in Figure 5.3. The result of this evaluation is a binary criterion which is either true when the implementation was possible or false if it wasn't possible. The prototype implementations in Chapter 6 shown that the dialogs can be implemented with Dialogflow, Watson Assistant, and Rasa. It is

not possible to implement the dialogs with LUIS. This was to be expected since LUIS is a NLU service and not a chatbot technology and offers no dialog control. The Microsoft Bot framework is the chatbot technology which uses LUIS but is not considered in this thesis. The implementation of the dialogs of Figure 5.3 is possible with all three chatbot technologies.

Project Setup Complexity

As a measurement criterion, the required tools will be used. To set up Watson Assistant, Dialogflow, and LUIS a web-browser is needed. Those three technologies require nothing else since the development environment, files, and data are built into the websites of the technologies. This means only one tool is required. For Rasa three approaches are evaluated. Rasa can be installed locally or with Docker and Rasa-X can be used for development. Rasa-X provides a web-interface similar to the cloud technologies. A local setup requires a Python environment and a package manager like pip. A text editor or IDE is required to edit the files. This results in a count of at least three. The second way is a Docker container setup. For this approach, Docker and a text-editor are required and results in a count of two. With Rasa-x a web-interface replaces the text-editor but the score remains the same. This means the could technologies have a score of one and Rasa has a score of two or three depending on the approach. As second measurement criteria for setup complexity a setup time ranking will be used. To create a new project from scratch with the cloud technologies only an account is required. Then a new project can be created instantly. No tools or additional software needs to be installed to have everything development ready. For Rasa, the tools and the environment needs to be created. For the development, either Rasa needs to be installed or the Docker image needs to be pulled. The setup for Rasa takes more time because the development environment needs to be provided. This results in the first place for cloud services and the last/second place for Rasa in the setup time comparison.

Development Complexity

The easiest framework is LUIS since it is a NLU framework it doesn't offer dialog handling mechanisms. The development complexity for the NLU part is compareable to Dialogflow and Watson Assistant. They use the same concepts (intent, entity, utterance) in the UI and the mechanisms to create intents and entities are the same. The easiest chatbot framework is Dialogflow since it offers a simpler UI than Watson Assistant, more predefined entities, and the option to include features like small talk. Watson Assitant UI is close to Dialogflow but is not as transparent and additional logic can be handeled inside of Watson which is a useful feature at times but makes the application more complex. Rasa is the most complex chatbot and NLU framework. The creation of intents, entities and utterances is scattered across multiple files. By default form filling actions are not supported and need to be created using Python. Luis, Dialogflow and Watson Assistant offer form filling by default and it can be enabled with a few clicks in the UI without a programming language. Rasa doesn't offer any predefined entities by default. To add predefined entities the pipeline needs to be adjusted which requires additional effort from the user. The creation of stories is also hardest with Rasa since the form filling action for instance requires additional information iside of the story, shown in Listing 10.3 where the the form, person and time slots need to be set correctly. Trying to set the forms and

slots by hand is not recommended. The stories should be created using live training. It is a useful feature to create natural conversations but it's also more complex than the creation of stories with the dialog node systems of Watson Assistant and Dialogflow. The dialog node system of Dialogflow is shown in Figure 10.2. When the framework development complexity is evaluated based on the use-cases of this thesis the predefined entities are very important. The creation of new entities is equally hard with each framework. Hence, the more of the required entities are predefined the easier the development process gets. Dialogflow offers all three required entities for the German and English language as does Rasa. Watson Assistant doesn't offer a person enity in German and the one for the English language is marked depricated. It also has no date-span entity but it can be replaced by two seperate dates. This means that Watson Assistant offers two of the three entities for both required languages. LUIS offers all three entities for the English language but has no person entity for the German language.

Pricing

Dialogflow can be used by companies for free. The limits for the free tier are 180 text requests per minute. With the Essential and Plus version, 600 requests per minute are possible. Rasa is a free to use open-source tool and costs nothing. There is an enterprise edition with some benefits but the price is calculated individually. Watson Assistant offers a free tier with up to 1.000 users per month, 10,000 messages per month. The free tier of the Microsoft Bot Framework offers five requests per second and 10,000 messages per month which is the same amount of messages IBM offers. Five requests per second are 300 requests per minute which is more than the free tier of Dialogflow offers. A user is a person who interacts with the bot at least once. The Plus edition costs $120 for 1.000 users per month with no message or user limit. The pricings and editions of the framework are shown in Table 7.1. The price of Dialogflow is based on the number of requests, IBM offers a fixed number of requests and users, and with Rasa the developer has to take care of the number of users and requests in soft- and hardware. The Microsoft Bot Framework charges per 1,000 users like IBM does. The price of the Microsoft Bot Framework is calculated by combining the prices for the framework and LUIS. The resulting cost for the Microsoft Bot Framework and LUIS is $0.002 per request which is the same Dialogflow charges for the enterprise essential edition. The standard editon also offers 50 transactions per second which are 3,000 requests per minute which is far more than the 600 requests per second of Dialogflow. In the free tier, Dialogflow provides the best value because there is no message or user limit. In the priced tier Watson costs $0.12 per user per month ($120 for 1.000 users per month). Dialogflow and the Microsoft Bot Framework using LUIS charge $0.002 per request each. Watsons cost of $0.12 per user per month equals 60 requests per user per month with Dialogflow or the Microsoft Bot Framework plus LUIS.

Framework	Edition	Price
Dialogflow	Standard	Free
	EE Essentials	$0.002 per request
	EE Plus	$0.004 per request
Watson Assistant	Lite	Free
	Plus	$120 for 1,000 users/month
	Premium	Individual
Rasa	Rasa Open Source	Free
	Rasa X	Free
	Rasa Enterprise	Individual
Microsoft Bot Framework	Free	Free
	S1	$0.50 per 1,000 requests
		$0.0005 per requests
LUIS	Free	Free (10.000 requests per month)
	Standard	$1.50 per 1,000 transactions
		$0.0015 per request

Table 7.1: Pricing of Frameworks [8, 4, 10]

Customization Possibilities

Learnability

The easiest of the chatbot frameworks is Dialogflow. Compared to Rasa the setup is easy. The UI is minimalistic and easy to understand. It's easier to develop a chatbot with Dialogflow than with Watson Assistant or Rasa. The Watson Assistants UI is intransparent when compared to Dialogflow and a developer may need more time to orient. Rasa requires the most time to develop a chatbot and to get everything ready. As described in Section 7.1 the setup of cloud services is easy since the cloud providers take care of the setup. Rasa has to be installed or used as a Docker container which is harder than the cloud setup and requires more time. To use Watson and Dialogflow no programming skills are required while they are needed for Rasa which makes it harder to learn. It's also harder to define a dialog structure with Rasa since a story needs to be written in the format shown in Listing 6.7 whereas with Dialogflow and Watson dialog nodes are created which can be strucutred in a GUI with drag-and-drop. Furthermore, it's harder to create form filling actions in Rasa. In Rasa the form filling needs to be defined through code like in Listing 10.2 while Dialogflow and Watson support it through a simple GUI shown in Figure 10.1. When the GUI of Rasa-X is used defining new entities and intents works the same way for the three frameworks. To enable the use of predefined entities the pipeline of Rasa needs to be modified which is more complicated than with Dialogflow where they are enabled by default and Watson where the entities need to be enabled in the settings.

Another measurement criteria for learnability is the number of predefined entities. If a predefined entity is available for a problem a developer doesn't need to invest time in creating an entity or collecting training data. Table 7.2 shows the amount of predefined entities present for the tested technologies. Dialogflow provides the most predefined entities, followed by LUIS. Compared to the two above the other technologies provide a small amount of predefined entities.

Rank	Framework	Entities
1	Dialogflow	over 400
2	LUIS	circa 150
3	Spacy	18
4	Duckling	11
5	Watson	7
6	Rasa	0

Table 7.2: Predefined Entities

Deployment Complexity

Communication

All of the chatbot and NLU technologies offer Rest API endpoints for communication. There is no difference in the type of communication. Dialogflow and Watson Assistant communicate with a webhook when the extraction of information was successful. Rasa communicates with the action server which serves a similar purpose as the webhooks. The communication between the frameworks and the external service is done with JSON messages. While Dialogflow and LUIS offer metadata, shown in Listing 6.2, 10.4 and 10.5, Rasa and Watson Assistant offer nothing by default. The required information needs to be defined by the developer. This means more logic can be handeled with Rasa and Watson Assistant on the chatbot side since the format and processing of the result is up to the developer. Especially with Rasa the logic can be freely defined since the action server is written in Python and all features of programming languages are available for the result processing. The behavior of Watson Assistant and Rasa can be mimicked by shifting the logic from the chatbot to the webhook.

Entity Recognition and Extraction

Entity recognition and extraction is a key feature of chatbots. For the use-cases of this thesis a date, date-span, and person entity are required. The entity recognition and extraction tests focus solely on the required entities. In Table 7.3 the supported entities are listed alongside the technologies. Rasa does not offer any entities but the pipeline can be adjusted. One part of the pipeline is Spacy which offers person and date entities. Spacy does not convert the date into a standard format. For this reason, Duckling was added to the pipeline. Duckling focuses on numeric values, dates, and times and converts dates into a standardized format. Dialogflow offers the system entities date, date-period, and person. LUIS offers a person and date-time entity. The date-time entity can also be a date-span. Hence, no entities need to be created in LUIS or Dialogflow for the use-cases of this thesis.

Framework	Subtype	Person	Date	Date Span	Standardized Output Date
Rasa	-	✗	✗	✗	✗
	Spacy	✓	✓	✓	✗
	Duckling	✗	✓	✓	✓
Dialogflow	-	✓	✓	✓	✓
Watson	-	✗	✓	✗	✓
LUIS	English	✓	✓	✓	✓
	German	✗	✓	✓	✓

Table 7.3: Framework Entity Recognition and Extraction

Watson Assistant offers the system entity sys-date but no date-span entity. When Watson recognizes two sys-dates the slots are filled correctly as start and return date and the result is comparable to a date-span entity. Dialogflow extracts the date, time, and timezone in the format ”2020-06-02T12:00:00+02:00”. Watson and LUIS extract only the date in the format ”2020-06-02”. Duckling extracts dates (time entity) in the format ”2020-06-07T00:00:00.000-07:00”. The second date of date spans is always set to one day after the recognized day in Duckling. This means that the second dates of Duckling differ from the dates of the other technologies shown in Table 7.8 in the ”Extracted Value” column. The column ”Extracted Value” of Tables 7.6, 7.7, and 7.8, 7.9 show that Spacy doesn't convert the dates into standard format. The date and date-span tests focus on different date inputs. It is to be expected that a recognized date format works for all instances of the format. If Monday is recognized correctly the other weekdays will work too.

The person entity has been added to Watson Assistant with custom training data. The expectation is that the person entity of Watson Assistant performs worst because it's no system entity and is built with a low amount of training data. The results of the date recognition and extraction are listed in Table 7.6 and 7.7. The expectation of the results is listed as true positive(TP) when the date should be recognized and extracted or as true negative(TN) when the entity shouldn't be recognized and extracted. The test result shows that the date entity of Dialogflow worked best with just one error followed closely by Duckling with two errors. Spacy performed worst with five errors in ten tests followed by Watson Assistant with four errors out of ten. No technology worked correctly on test case eight of Table 7.6. This is true for the english language selection. When German is selected as a language the performance on dates separated by ”.” increases. It also shows that weekdays and tomorrow work with all technologies. The extracted value column shows that Spacy recognizes many date inputs correctly but doesn't convert them.
The detailed result of the entity recognition comparison is shown in Table 7.4. The column name p stands for precision and r for recall. Date sum shows the framework result based on the date and date-span recognition without the person entity for the comparison with Duckling which has no person entity. The last row of Table 7.4 shows the result for the combination of Duckling for dates and date-spans and Spacy for persons. This makes the comparison of Rasa with the other technologies possible. The true negatives (TN) are not part of the f-score calculation. Table 7.4 shows that the technology with the best f-score across all tests (0.914) is Dialogflow followed by a combination of Duckling for dates and date-spans and Spacy for persons is used. Rasa can use Duckling and Spacy and reaches the second highest f-score. The best f-score (0.929) for the combination of dates and date-spans was achieved by Dialogflow. Dialogflow has the best date entity (f-score of 0.94) when the date-time entity is used instead of the date-period entity. The

Framework	Type	TP	TN	FP	FN	p	r	F-Score	Tests
Dialogflow date-period	Date	8	1	0	1	1.0	0.888	0.94	10
	Date-Span	2	0	0	4	1.0	0.333	0.5	6
	Person	3	2	0	1	1.0	0.75	0.857	6
	Summary	13	3	0	6	1.0	0.684	0.813	22
Dialogflow date-time	Date	8	1	0	1	1.0	0.888	0.94	10
	Date-Span	5	0	0	1	1.0	0.833	0.909	6
	Person	3	2	0	1	1.0	0.75	0.857	6
	Summary	16	3	0	3	1.0	0.842	0.914	22
Watson	Date	6	0	1	3	0.857	0.666	0.75	10
	Date-Span	5	0	0	1	1.0	0.833	0.909	6
	Person	2	1	1	2	0.666	0.5	0.571	6
	Summary	13	1	2	6	0.866	0.684	0.765	22
LUIS	Date	6	0	1	3	0.857	0.666	0.75	10
	Date-Span	3	0	0	3	1.0	0.5	0.666	6
	Person	4	0	2	0	0.666	1.0	0.8	6
	Summary	13	0	3	6	0.8125	0.684	0.743	22
Spacy	Date	4	1	0	5	1.0	0.444	0.615	10
	Date-Span	2	0	0	4	1.0	0.333	0.5	6
	Person	4	0	2	0	0.666	1.0	0.8	6
	Summary	10	1	2	9	0.833	0.526	0.645	22
Duckling	Date	7	1	0	2	1.0	0.777	0.875	10
	Date-Span	5	0	0	1	1.0	0.833	0.909	6
	Summary	12	1	0	3	1.0	0.8	0.888	16
Dialogflow	Date Sum	13	1	0	2	1.0	0.866	0.929	16
Watson	Date Sum	11	0	1	4	0.916	0.733	0.815	16
LUIS	Date Sum	9	0	1	6	0.9	0.6	0.72	16
Spacy	Date Sum	6	1	0	9	1.0	0.4	0.571	16
Spacy + Duckling	Summary	16	1	2	3	0.888	0.842	0.865	22

Table 7.4: Entity Recognition Result Evaluation

date-period entity of Dialogflow reached a low f-score of 0.5 and is not recommended for further use. Duckling, Watson, and Dialogflow were on an equal level when only date-spans are considered. All three reached an f-score of 0.909. Dialogflow also has the best f-score for the person entity. Watsons person entity performed worst as expected. With more training data the performance of Watsons custom person entity can be increased. In test case two of Table 7.8 Spacy recognized two separate dates instead of a date-span. This is counted as a failure because in the application there is only one date slot present and the information can't be processed correctly. In test case one Dialogflows date-period entity and Luis convert Monday to a date in the past. This is now what is needed for the use-cases since it should be the next matching weekday and not the last. Test case two gives a hint that the next occurring weekday is the correct one. The result is not influenced and Dialogflows date-period entity and LUIS still take the wrong day. The functionality was retested on another day. LUIS still failed to extract the correct date but Dialogflow managed to do so. In the test result it will be counted as a failure of Dialogflows date-period entity since it doesn't deliver reliable results.

No	Input	Expected	Technology	Recognized Correctly	Extracted Value
1	Franz Bauer	TP	Watson	✓	Franz Bauer
			Dialogflow	✓	Franz Bauer
			Spacy	✓	Franz Bauer
			LUIS	✓	Franz Bauer
2	franz bauer	TP	Watson	✓	franz bauer
			Dialogflow	✓	franz bauer
			Spacy	✓	franz bauer
			LUIS	✓	franz bauer
3	Anna Maria Mayer	TP	Watson	✗	Maria Mayer
			Dialogflow	✗	Maria Mayer
			Spacy	✓	Anna Maria Mayer
			LUIS	✓	Anna Maria Mayer
4	Anna Mayer-Bauer	TP	Watson	✗	Anna Mayer
			Dialogflow	✓	Anna Mayer-Bauer
			Spacy	✓	Anna Mayer-Bauer
			LUIS	✓	Anna Mayer-Bauer
5	Asdfgh	TN	Watson	✓	-
			Dialogflow	✓	-
			Spacy	✗	Asdfgh
			LUIS	✗	Asdfgh
6	Asdfgh Qwert	TN	Watson	✗	Asdfgh Qwert
			Dialogflow	✓	-
			Spacy	✗	Asdfgh Qwert
			LUIS	✗	Asdfgh Qwert

Table 7.5: Person Entity Recogniton and Extraction

Intent Recognition

The main task of a chatbot is to find the intent best matching the user input. The training utterances for the sickness intent are listed Tabel 4.1 in the Appendix. The phrases for the vacation intent can be found in Table 10.2. As an evaluation criterion, multiple parameters are used. The confidence score represents how well the input can be classified. Building upon the entity extraction the correct extraction of the entities in combined inputs is tested. A combined example is training phrase five of Table 4.1 where the entities person and date are built into the training sentence. Furthermore, the correctly identified intents are used to calculate the f-score like it was done for the entities. For the intent recognition task, only date and date-spans are used which were recognized by all relevant frameworks to ensure unbiased tests. Spacy has no relevance for the application in terms of date and date-span extraction and is not considered for dates in the intent recognition tests. The detailed test are listed in Table 7.10 and Table 7.11. The test evaluation is shown in Table 7.12. The tests show that Dialogflow, LUIS, and Watson have almost the same average confidence score. Dialogflow has a smaller deviation than the other three. Dialogflow always scores above 50% confidence and often reaches 100%. The scores of Watson Assistant are similar to Dialogflow with the difference that the confidence score was below 50% at times. The confidence scores of Rasa are lower than the ones of Dialogflow, LUIS,

No	Input	Expected	Technology	Recognized Correctly	Extracted Value
1	Tomorrow	TP	Watson	✓	2020-06-01
			Dialogflow	✓	2020-06-01
			Spacy	✓	Tomorrow
			Duckling	✓	2020-06-01
			LUIS	✓	2020-06-01
2	Friday	TP	Watson	✓	2020-06-05
			Dialogflow	✓	2020-06-05
			Spacy	✓	Friday
			Duckling	✓	2020-06-05
			LUIS	✓	2020-06-05
3	6.June	TP	Watson	✗	2020-06-01
			Dialogflow	✓	2020-06-06
			Spacy	✗	-
			Duckling	✗	-
			LUIS	✗	2020-06-01

Table 7.6: Date Entity Recogniton and Extraction

and Watson on average, shown in Table 7.12. Dialogflow and Watson managed to classify all intents correctly. LUIS classified all vacation intents correctly. The entity extraction of Dialogflow failed in test case 4 of Table 7.10 since ”myself” has been extracted as a person. This means that the person slot was set incorrectly. LUIS extracted sick as person in test sentence one and didn't extract the person in test sentence six, shown in Table 7.10. The test sentences get more complex as the number increases. Dialogflow and Watson were able to correctly classify all test sentences. LUIS failed to classify test sentence seven of Table 7.10. Rasa failed to correctly classify the more complex sentences. This leads to the conclusion that Dialogflow, LUIS, and Watson work better with complex sentences when a small amount of training data is used. The result in Table 7.12 shows that Dialogflow and Watson reach a perfect f-score of 1.0. Rasa was able to reach a much higher f-score on the vacation use-case than on the sickness use-case. This implies that the vacation test sentences were better classified by the system than the sickness related sentences. The average confidence was highest with Dialogflow followed closely by LUIS and the Watson Assistant. Compared to Dialogflow, LUIS, and Watson Assistant the average confidence score of Rasa is a lot lower.

The same experiment is also interesting with German as language since the potential users are from a German speaking country. The training data is listed in the Appendix in Table 10.1 and Table 10.3. The translates test sentences are listed in Table 10.4 and Table 10.5 with a boolean value which indicates if the correct intent was identified, a boolean that indicates if the entities were extracted correctly, and the confidence score for each test case. The finalized result is listed in 7.13 and shows that the average confidence score for the German test sentences is a higher than for the English test sentences for each technology individually. The f-score also increased significantly which indicates that the German test sentences are recognized better than the English test sentences. The ranking of the Technologies is also different from before since three technologies achieved perfect f-scores the ranking is based on the average confidence. The highest confidence was achieved by Watson. The second place goes to LUIS. LUIS was able to increas the

average confidence from about 75% on the English tests to about 85% on the German tests. Dialogflow only reaches the third place. The intent recognition are top notch for all technologies and all can be recommended from that point of view. The results in Table 10.4 and 10.5 show that the entity extraction results drop in the German setup with all technologies except LUIS. Dialogflow has problems to extract the person entity correctly in German sentences although German has been selected in the settings. The system entity date-time of Dialogflow is unable to handle German date formats although the dialog language is set to German. Watson doesn't offer entity annotation in training sentences when German is selected as language. This means that a main feture is only available for English language bots. Until the annotation of entities in training phrases is enabled for other languages Watson should only be used in English. LUIS offers a person entity when English is selected but not for the German language. The date-time entity of LUIS is able to handle German date formats. This means that LUIS is the only technology which achieved a satisfying result in the entity extraction category. All in all, the intent recognition result is very good for the German language tests but the entity extraction often fails.

Training Capabilities

Integration

Transfer of Knowledge

Usability of the Framework

Frontend

Speech Recognition Capabilities

For the further development which is based on the results of this thesis speech recognition is an option. The result of this evaluation is true or false. Frameworks which support this feature can be used with speech input instead of text. Either the tchnology supports speech recognition or it doesn't. Rasa doesn't support speech functionality itself but it can be used in combination with speech-to-text and text-to-speech frameworks like Alexa. Dialogflow offers audio in- and oputput. The speech-to-text functonality of Dialoflow costs $0.0065 per 15 seconds of audio. The IBM Watson Assistant also offers speech functionalities through Watson Speech to Text and Watson Text to Speech. The Watson text-to-speech service costs between $0.01 and $0.02 per minute which results in $0.0025 to $0.005 per 15 seconds. The speech-to-text service costs $0.02 per 1,000 chars which results in $20 per million chars. Microsoft offers the Speech Services for the speech-to-text and text-to-speech functionality. The Speech service costs $1 per hour of speech which results in $0.00416 per 15 seconds which is less than the costs for Dialoflow. The costs for text-to-speech are the same for the Speech Services and Dialogflow with a rate of $4 per one million characters. The speech-to-text services are on a similar pricing level but the text-to-speech service of Watson costs five times as much as the Speech Services or Dialogflow. The only technology which doesn't support speech recognition is Rasa. The other three technologies support the speech-to-text and text-to-speech features which are required for speech processing.

Framework Concepts

The user interfaces of the cloud frameworks are similar. The interfaces are shown in Figure 6.1, 6.2, and 6.4. All three offer intents and entities. They are all structured similarly with the menu to the left. The information for the bots is always entered as text. Intents are always built with training phrases. All offer prebuilt system entities. Dialogflow has the most prebuilt entities, followed by LUIS, then comes Rasa, and Watson Assistant offers the fewest entities. For the sickness and vacation use-case the entities of Dialogflow suit best because no entity has to be defined by the developer. Watson offers a person system entity in English but the entity is marked as deprecated. It offers a date entity but no date span entity. This means with Watson Assistant two entities have to be modeled for the use-cases. LUIS offers a date-time entity which can be used for dates and date spans but it's not working as reliable as the one from Dialogflow. It did not identify "22-06 to 30-06" correctly which is no problem for Dialogflow.

No	Input	Expected	Technology	Recognized Correctly	Extracted Value
4	June 6	TP	Watson	✓	2020-06-06
			Dialogflow	✓	2020-06-06
			Spacy	✓	June 6
			Duckling	✓	2020-06-06
			LUIS	✓	2020-06-06
5	6th of June	TP	Watson	✓	2020-06-06
			Dialogflow	✓	2020-06-06
			Spacy	✓	6th of June
			Duckling	✓	2020-06-06
			LUIS	✓	2020-06-06
6	06/06	TP	Watson	✓	2020-06-06
			Dialogflow	✓	2020-06-06
			Spacy	✗	-
			Duckling	✓	2020-06-06
			LUIS	✓	2020-06-06
7	06-06	TP	Watson	✗	-
			Dialogflow	✓	2020-06-06
			Spacy	✗	-
			Duckling	✓	2020-06-06
			LUIS	✗	-
8	06.06	TP	Watson	✗	-
			Dialogflow	✗	-
			Spacy	✗	-
			Duckling	✗	2020-05-31
			LUIS	✗	-
9	06.06.2020	TP	Watson	✓	2020-06-06
			Dialogflow	✓	2020-06-06
			Spacy	✗	-
			Duckling	✓	2020-06-06
			LUIS	✓	2020-06-06
10	06/34/2020	TN	Watson	✗	2020-06-01
			Dialogflow	✓	-
			Spacy	✓	-
			Duckling	✓	-
			LUIS	✗	2020-01-01

Table 7.7: Date Entity Recogniton and Extraction 2

No	Input	Expected	Technology	Recognized Correctly	Extracted Value
1	Monday to Friday	TP	Watson	✓	2020-06-01 2020-06-05
			Dialogflow date-period	✗	2020-05-25 2020-06-01
			Dialogflow date-time	✓	2020-05-25 2020-06-05
			Spacy	✓	Monday to Friday
			Duckling	✓	2020-06-01 2020-06-06
			LUIS	✗	2020-05-25 2020-05-29
2	from next tuesday to wednesday	TP	Watson	✓	2020-06-02 2020-06-03
			Dialogflow date-period	✗	2020-05-26 2020-06-03
			Dialogflow date-time	✓	2020-06-02 2020-06-03
			Spacy	✗	next tuesday wednesday
			Duckling	✓	2020-06-02 2020-06-04
			LUIS	✗	2020-06-09 2020-06-10
3	June 6 to June 7	TP	Watson	✓	2020-06-06 2020-06-07
			Dialogflow date-period	✓	2020-06-06 2020-06-07
			Dialogflow date-time	✓	2020-06-06 2020-06-07
			Spacy	✓	June 6 to June 7
			Duckling	✓	2020-06-06 2020-06-08
			LUIS	✓	2020-06-06 2020-06-07
4	06/06 to 06/07	TP	Watson	✓	2020-06-06 2020-06-07
			Dialogflow date-period	✓	2020-06-06 2020-06-07
			Dialogflow date-time	✓	2020-06-06 2020-06-07
			Spacy	✗	- -
			Duckling	✓	2020-06-06 2020-06-08
			LUIS	✓	2020-06-06 2020-06-07

Table 7.8: Date Span Entity Recogniton and Extraction 1

No	Input	Expected	Technology	Recognized Correctly	Extracted Value
5	06/07 to 06/06	TP	Watson	✗	2020-06-07 2020-06-06
			Dialogflow date-period	✗	2020-06-07 2020-06-06
			Dialogflow date-time	✓	2020-06-07 2021-06-06
			Spacy	✗	- -
			Duckling	✗	2020-06-07 2020-06-07
			LUIS	✗	2020-06-07 2020-06-06
6	from tomorrow to 06.06.2020	TP	Watson	✓	2020-06-02 2020-06-06
			Dialogflow date-period	✗	- -
			Dialogflow date-time	✗	2020-06-02 -
			Spacy	✗	tomorrow -
			Duckling	✓	2020-06-02 2020-06-07
			LUIS	✓	2020-06-02 2020-06-06

Table 7.9: Date Span Entity Recogniton and Extraction 2

No	Input	Technology	Intent Match	Entity Match	Confid. [%]
1	sick	Watson	✓	-	41.0
		Dialogflow			84.2
		Rasa			78.5
		LUIS		✗	99.2
2	Emilia Schwarz is ill	Watson	✓	✓	94.0
		Dialogflow			78.2
		Rasa			63.1
		LUIS			83.4
3	Lea Schmitt is sick until wednesday	Watson	✓	✓	92.0
		Dialogflow			73.9
		Rasa			66.2
		LUIS			83.3
4	I need to report myself as sick	Watson	✓	-	38.0
		Dialogflow		✗	63.3
		Rasa	✗	-	9.7
		LUIS	✓		87.2
5	Paul Armstrong is sick and will be back on Monday	Watson	✓	✓	89.0
		Dialogflow			69.6
		Rasa	✗		23.8
		LUIS	✓		73.1
6	sick Anna Maier 06/10	Watson	✓	✓	90.0
		Dialogflow			100.0
		Rasa			80.5
		LUIS		✗	67.6
7	I feel sickly today and won't come to work	Watson	✓	✓	43.0
		Dialogflow			51.6
		Rasa	✗		4.3
		LUIS			17.9
8	Anna Lehner is feeling sick. She will be back on Thursday	Watson	✓	✓	91.0
		Dialogflow			72.5
		Rasa	✗		26.2
		LUIS	✓		68.1

Table 7.10: Sickness Intent Classification

No	Input	Technology	Intent Match	Entity Match	Confid. [%]
1	vacation	Watson	✓	-	100.0
		Dialogflow			100.0
		Rasa			91.7
		LUIS			97.1
2	my colleague Franz Bauer wants to go on vacation	Watson	✓	✓	91.0
		Dialogflow			64.9
		Rasa			33.7
		LUIS			64.9
3	vacation Martin Huber 07/12 to July 15th	Watson	✓	✓	44.0
		Dialogflow			100.0
		Rasa			57.4
		LUIS			84.6
4	holiday from 07/12 to July 15th	Watson	✓	✓	94.0
		Dialogflow			64.7
		Rasa			67.4
		LUIS			87.9
5	vacation from 28th September to October 4	Watson	✓	✓	95.0
		Dialogflow			100.0
		Rasa			55.5
		LUIS			88.1
6	I really need a long vacation on a distant island	Watson	✓	-	44.0
		Dialogflow			79.0
		Rasa			35.0
		LUIS			92.3
7	colleague Helmut Kerschbaum requested holidays	Watson	✓	✓	93.0
		Dialogflow			61.7
		Rasa			46.6
		LUIS			41.9
8	Colleague Helmut Kerschbaum requested holidays. He will be unavailable from 09/10 to 09/24	Watson	✓	✓	45.0
		Dialogflow			70.2
		Rasa	✗		21.6
		LUIS	✓		70.8

Table 7.11: Vacation Intent Classification

Framework	Tested	TP	FN	p	r	f-score	Confid. avg. [%]	Confid. Median	std. dev.
Watson	sickness	8	0	1.0	1.0	1.0	72.3	89.5	26.2
	vacation	8	0	1.0	1.0	1.0	75.8	92.0	26.1
	sum	16	0	1.0	1.0	1.0	74.0	90.5	25.4
Dialogflow	sickness	8	0	1.0	1.0	1.0	74.2	73.2	14.3
	vacation	8	0	1.0	1.0	1.0	80.1	74.6	17.3
	sum	16	0	1.0	1.0	1.0	77.1	73.2	15.6
Rasa	sickness	4	4	1.0	0.5	0.67	44.0	44.7	31.3
	vacation	7	1	1.0	0.88	0.93	51.1	51.1	22.1
	sum	11	5	1.0	0.69	0.82	47.6	51.1	26.4
LUIS	sickness	7	1	1.0	0.88	0.93	72.5	78.2	24.5
	vacation	8	0	1.0	1.0	1.0	78.5	86.3	18.3
	sum	5	11	1.0	0.31	0.48	75.5	83.4	21.1

Table 7.12: Intent Classification Result

Framework	Tested	TP	FN	p	r	f-score	Confid. avg. [%]	Confid. Median	std. dev.
Watson	sickness	8	0	1.0	1.0	1.0	94.1	94.0	1.88
	vacation	8	0	1.0	1.0	1.0	96.4	96.5	1.99
	sum	16	0	1.0	1.0	1.0	95.3	95.5	2.21
Dialogflow	sickness	8	0	1.0	1.0	1.0	80.8	78.6	19.1
	vacation	8	0	1.0	1.0	1.0	89.5	100	16.5
	sum	16	0	1.0	1.0	1.0	85.1	90.3	17.8
Rasa	sickness	7	1	1.0	0.88	0.93	54.0	44.2	20.2
	vacation	8	0	1.0	1.0	1.0	58.3	57.0	10.9
	sum	15	1	1.0	0.94	0.97	56.2	53.6	15.9
LUIS	sickness	8	0	1.0	1.0	1.0	79.6	77.5	15.1
	vacation	8	0	1.0	1.0	1.0	91.0	90.9	5.01
	sum	16	0	1.0	1.0	1.0	85.3	89.7	12.3

Table 7.13: Intent Classification Result German

Chapter 8

Discussion

The price comparison of Section 7.1 shows that all tested frameworks can be used for free. Dialogflow is better when there is a lot of users with little chatbot interaction. Watson offers a better value when the users interact a lot with the bot. With Rasa the costs depend on the infrastructure provided by the company. A big disadvantage of cloud-based solutions is the dependency on the company. It's not easy to change the provider and take the current chatbot with you if it is a cloud-based chatbot. Rasa differs at that point since it can run in any cloud environment inside a docker container. Every cloud provider offers the option to host docker containers in their cloud. A major factor is security since the users enter sensitive information in the chatbot communication and the data is processed in the cloud of the provider. The cloud provider hence has access to the sensitive information which can be problematic with bank account information for instance. Rasa has no such problem since the data is processed inside of the docker container or in a local environment. The entity extraction section shows that Dialogflow and LUIS define the required entities while Watson lacks a person entity. Rasa doesn't offer any entities but can use the entities of Spacy and Duckling. If both are used then no entities need to be defined in Rasa. The result in Table 7.4 shows that Dialogflow has the best person, date, and date-span entity and dominated the entity extraction tests. The best overall performance was achieved by Dialogflow followed by Rasa. No technology performed exceptionally bad in this category. The intent classification result of Table 7.12 shows that Dialogflow and Watson outperformed Rasa and LUIS. Both reached a perfect precision, recall, and f-score of 1. Watson takes the top spot because Dialogflow extracted an entity incorrectly in test case four of Table 7.10.

Rasa showed a good performance when the f-score is used as a measure but has a low average confidence score of 46%. The confidence scores of Dialogflow, LUIS, and Watson are far better with 77%, 75%, and 74%. In this category, Dialogflow, LUIS, and Watson performed good enough to be recommended. The confidence scores of Rasa can be improved through more training data. Rasa achieved by far the lowest f-score and average confidence values.

Braun et al. [15] stated that if the training data is sparse there is almost no performance difference. This can be confirmed for the cloud frameworks since there is no big difference reagarding the f-score, average confidence, and entity extraction, shown in in Tables 7.4, 7.12, 7.13. The local solution Rasa was measurably worse.

Chapter 9

Conclusion and Outlook

9.1 Conclusion

The domain consisting of the sickness notification and the vacation request can be solved with chatbots. It was possible to implement the design with all three chatbot technologies namely Dialogflow, Rasa, and Watson Assistant. Based on Chapter 6 the implementation is easiest with Dialogflow then comes Watson Assistant followed by Rasa which is the most complicated by far. It's easiest with Dialogflow because it already has prebuilt entities, it offers the utility intents yes and no, and small talk can be included with a single mouse click. Watson Assistant is second because it takes more time to get accustomed to the UI, the person entity needs to be defined by the developer, and the yes/no logic needs to be implemented. Rasa is the most complex one since it doesn't provide any entities itself. To enable the entities of other technologies like Spacy and Duckling the pipeline needs to be adjusted. To connect to Duckling and Additional Docker container needs to be started. The chatbot developer also needs to implement the form filling with Python when using Rasa. The other technologies don't require any programming skills to model the dialog. The implementation with Rasa took by far the longest. When only the implementation section is used for the ranking Dialogflow wins in front of Watson followed by Rasa. Table 7.2 shows the number of predefined entities. When only the domain of this thesis is considered then the only technology where an entity is missing would be Watson Assistant since it provides no person entity. When the number of entities is considered for a general case then the top technology by a large margin is Dialogflow. The number of predefined entities is an important number for future use-cases since lots of development time can be saved when the entities are already predefined. They are also trained well because they use large training sets. Table 7.4 ranks the frameworks based on their entity extraction capabilities. The best person, date and date-span entities are present at Dialogflow because they achieved the highest f-score values. In the date-span entity category Duckling and Watson achieved the same f-score as Dialogflow. The best overall entity extraction performance is achieved by Dialogflow. From the entity extraction capabilities alone either Rasa or Dialogflow needs to be recommended. The other frameworks performed worse in the entity extraction department. The ranking shows that some technologies performed better than others. It also shows that no framework performed badly. The intent classification is dominated by Watson and Dialogflow since both were able to classify all test cases correctly. LUIS failed most of the intent detection tests and can't be recommended. Dialogflow wins the average confidence score comparison in front of

Watson. The detailed result is shown in Table 7.12. Rasa and Luis were unable to keep up with Dialogflow and Watson when the average confidence is compared. This is an indication that Rasa and LUIS need more training data to reach the same performance level as Dialogflow and Watson. An average confidence value below 50% is not satisfactory. Based on the entity extraction and intent recognition results the recommended technology in the top stop has to be Dialogflow since the performance in both categories was great. Additionally, Dialogflow has the best average confidence value. It's also possible to use LUIS and Watson Assistant since the performance was close to Dialogflow. Rasa needs to improved to be recommended. It's possible to improve the performance and confidence scores of Rasa with more training data. The intent classification was also done for the German language and the results differ greatly from the English results. The average confidence increased for each technology. The f-score of LUIS and Rasa increased significantly. This puts LUIS on par with Dialogflow and Watson with a perfect f-score value of 1.0. This leads to the conclusion that the intent classification works better for the German language in general since all technologies performed better. Countrary to the intent classification, the entity-extraction performance decreased significantly. In the English test sentences of Tables 7.10 and 7.11 the entity extraction didn't work correctly in two cases and worked in 46 cases. For the German language, test shown in Tables 10.4 and 10.5, a total of 21 wrong and 26 correct extractions are counted. The only technology that achieved a satisfying entity extraction result on the German test sentences is LUIS. Rasa is the only technology that is portable. It can run in a Docker container and Docker containers are supported by every cloud provider. It can also run locally on a development machine and on an internal server. The cloud technologies are tied to the cloud provider who offers the bot. Rasa wins the portability comparison. The migration from one chatbot technology to another is not easily possible with any of the technologies tested. Hence, there is no evaluation possible and no ranking can be done. The pricings of the frameworks shown in Table 7.1 are hard to compare and intransparent for the premium version of Watson Assistant and the enterprise edition for Rasa. All three offer a free tier which can be used by companies. Because of the lack of transparency, the comparison can't be used for a recommendation or a ranking. The real difference is that Watson Assistant is measured by the number of users per month and Dialogflow is based on the number of requests. This makes Dialogflow cheaper for small amounts of requests and lots of users with short conversations. Watson is cheaper for returning users with lots of conversation or very long conversations. When security is an important factor then a local technology needs to be used. The cloud technologies all process the requests in the cloud and this has to be kept in mind. If it's impossible to process information in the cloud because there are security restrictions then the recommendation is to use Rasa.

When all the factors discussed above are combined the clear winner is Dialogflow. The design could be implemented fasted with Dialogflow, it's the easiest of the chatbot technologies, it offers the most predefined entities, provides utility intents, had the best person and date entity in the entity extraction tests, has the highest average confidence score, and performed great in the intent classification tests.

9.2 Outlook

In Singh et al. [32] a chatbot is used for a banking system which is a possible use case for the 3-Banken-IT.

A common use case for chatbots are FAQs(Shawar and Atwell [30], Raj [29], Huang

et al. [24], Go and Sundar [20]) and is another possible use case for most companies.

Like Go and Sundar [20] there are multiple ways to make users believe that they are talking to a bot. For an actual chatbot, there needs to be decided if the users should know that they are talking to a bot or not. If they believe to talk to a bot they expect less but the conversation is generally treated more machine-like [20]. When users assume to talk to a human the conversation is judged as more natural but if the bot performs badly the experience is also evaluated as more negative [20]. Like Shawar and Atwell [31] said, a chatbot should not imitate humans they should be used to reduce their workload and help them.

The best source for conversational data is real-life conversations of real customers with the expert to get the most realistic training data possible [32].

Chapter 10

Appendix

```yaml
language: "en"
pipeline:
    - name: "SpacyNLP"
    - name: "SpacyTokenizer"
    - name: "SpacyFeaturizer"
    - name: "RegexFeaturizer"
    - name: "DucklingHTTPExtractor"
        url: "http://localhost:8000"
    - name: "SpacyEntityExtractor"
    - name: "EntitySynonymMapper"
    - name: "SklearnIntentClassifier"
```

Listing 10.1: Detailed Pipeline Configuration for Rasa

```python
class SicknessForm(FormAction):

def name(self) -> Text:
    return "sickness_form"

@staticmethod
def required_slots(tracker: Tracker) -> List[Text]:
    return ["PERSON", "return_date"]

def slot_mappings(self) -> Dict[Text, Union[Dict, List[Dict]]]:
    return {
            "PERSON": [self.from_entity(entity="PERSON"),
                self.from_text()],
            "return_date": [self.from_entity(entity="DATE"),
                self.from_text()]
    }

def submit(
    self,
    dispatcher: CollectingDispatcher,
    tracker: Tracker,
    domain: Dict[Text, Any],
) -> List[Dict]:

    return []
```

Listing 10.2: Rasa Slot Filling Action

```
## Sickness Affirm
* sickness
    - sickness_form
    - form{"name":"sickness_form"}
* submit{"PERSON":"Anna Maria Mayer"}
    - sickness_form
    - slot{"PERSON":"Anna Maria Mayer"}
* submit{"time":"2020-05-30T00:00:00.000-07:00","DATE":"today"}
    - sickness_form
    - form{"name":null}
    - slot{"time":"2020-05-30T00:00:00.000-07:00"}
* affirm
    - action_submit_sickness_data
    - action_restart
```

Listing 10.3: Rasa Sickness Story

No	Utterance	Person Entity	Date Entity
1	Ich bin krank	✗	✗
2	Krankmeldung	✗	✗
3	krank melden	✗	✗
4	[Alfred Mayer] ist krank	✓	✗
5	meine Kollegin [Maria Müller] ist bis [Mittwoch] krank	✓	✓
6	Kollegin [Simone Bauer] ist krank	✓	✗
7	melde [Franz Dorfer] krank	✓	✗
8	[Stefan Weber] ist krank bis [Freitag]	✓	✓
9	krank [Emma Wagner] [6.Juni]	✓	✓
10	[Sophia Richter] ist heute krank	✓	✗

Table 10.1: Sickness Training Utterances German

No	Utterance	Person Entity	Date-Span Entity
1	vacation	✗	✗
2	holiday	✗	✗
3	I need a vacation	✗	✗
4	colleague [Johan Schmidt] requested a vacation	✓	✗
5	I'm on holiday [from Monday to Friday]	✗	✓
6	vacation [Mia Koch] [06/20 to 06/30]	✓	✓
7	[Adam Schneider] is on holiday [from June 6 to June 14]	✓	✓
8	holiday request	✗	✗
9	vacation [from 12th of August to 30th of August]	✗	✓
10	[Lea Klein] requested a vacation	✓	✗

Table 10.2: Vacation Training Utterances

No	Utterance	Person Entity	Date-Span Entity
1	Urlaub	✗	✗
2	Urlaubsantrag	✗	✗
3	Ich brauche Urlaub	✗	✗
4	Kollege [Johan Schmidt] hat Urlaub beantragt	✓	✗
5	Ich bin von [Montag bis Freitag] im Urlaub	✗	✓
6	urlaub beantragen [Mia Koch] [20/06 bis 30/06]	✓	✓
7	[Adam Schneider] ist [vom 6.Juni bis 14.Juni] im Urlaub	✓	✓
8	Ich beantrage Urlaub	✗	✗
9	Urlaub [vom 12.August bis zum 30.August]	✗	✓
10	[Lea Klein] hat einen Urlaubsantrag eingereicht	✓	✗

Table 10.3: Vacation Training Utterances German

```
"query": "sick Anna Maier 06/10",
"prediction": {
    "topIntent": "sick_",
    "intents": {
    "sick_": {
        "score": 0.6760214
    },
    "vacation_": {
        "score": 0.224106133
    },
    "None": {
        "score": 0.006858599
    }},
}}
```

Listing 10.4: LUIS Intent Response

```
"entities": {
    "datetimeV2": [{
        "type": "date",
        "values": [{
            "timex": "XXXX-06-10",
            "resolution": [
            { "value": "2020-06-10" },
            { "value": "2021-06-10" }]}]
    }],
    "$instance": {
    "datetimeV2": [{
        "type": "builtin.datetimeV2.date",
        "text": "06/10",
        "startIndex": 16,
        "length": 5,
        "modelTypeId": 2,
        "modelType": "Prebuilt Entity Extractor",
        "recognitionSources": [ "model" ]}]
    }
}
```

Listing 10.5: LUIS Entity Response

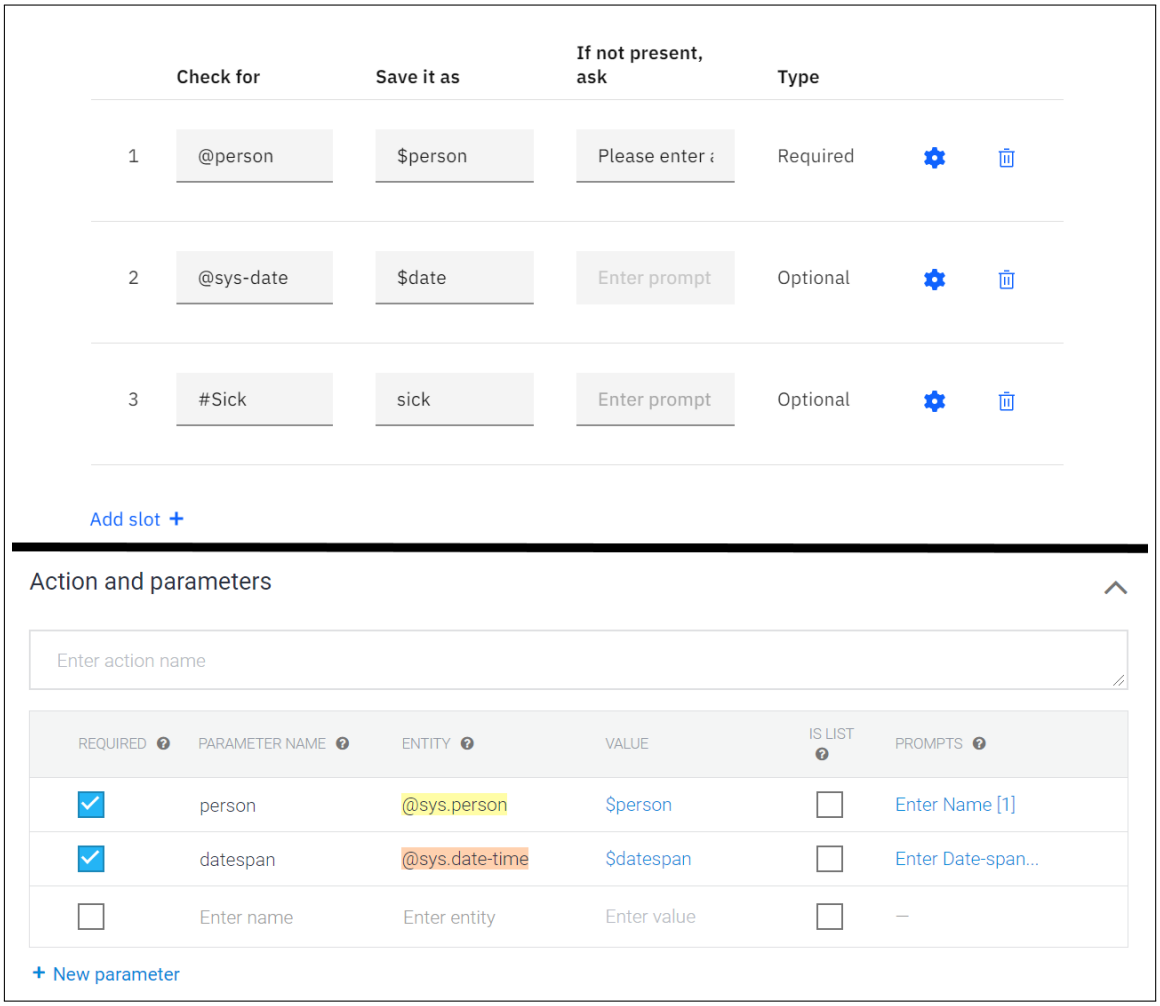

Figure 10.1: From Filling in Watson Assistant (top) and Dialogflow (bottom)

No	Input	Technology	Intent Match	Entity Match	Confid. [%]
1	krank	Watson	✓	-	91.0
		Dialogflow			77.1
		Rasa			93.6
		LUIS			99.8
2	Emilia Schwarz ist krank	Watson	✓	✗	94.0
		Dialogflow		✓	100
		Rasa			72.7
		LUIS			92.7
3	Lea Schmitt bis Mittwoch krank gemeldet	Watson	✓	✗	95.0
		Dialogflow		✓	80.0
		Rasa		✗	43.1
		LUIS		✓	79.1
4	Ich muss mich krank melden	Watson	✓	-	97.0
		Dialogflow			70.5
		Rasa			45.3
		LUIS			95.0
5	Paul Armstrong ist krank und kommt am Montag wieder	Watson	✓	✗	93.0
		Dialogflow		✓	73.5
		Rasa			41.6
		LUIS			64.8
6	krank Anna Maier 06/10	Watson	✓	✗	93.0
		Dialogflow		✓	100.0
		Rasa		✗	60.2
		LUIS		✓	72.1
7	Ich komme heute nicht zur Arbeit weil ich krank bin	Watson	✓	-	96.0
		Dialogflow			45.2
		Rasa	✗	✗	33.4
		LUIS	✓		75.9
8	Anna Lehner fühlt sich krank. Sie wird Dienstag wieder da sein.	Watson	✓	✗	94.0
		Dialogflow		✓	100
		Rasa			42.4
		LUIS			57.6

Table 10.4: Sickness Intent Classification German

No	Input	Technology	Intent Match	Entity Match	Confid. [%]
1	Urlaub	Watson	✓	-	100.0
		Dialogflow			100.0
		Rasa			59.8
		LUIS			98
2	mein Kollege Franz Bauer möchte Urlaub beantragen	Watson	✓	✓	93.0
		Dialogflow			100.0
		Rasa			45
		LUIS			88.8
3	Urlaub Martin Huber 12.Juli bis 13.07.2020	Watson	✓	✗	95.0
		Dialogflow			100.0
		Rasa			44.5
		LUIS		✓	91.4
4	Urlaubsantrag 12/07 bis 13.Juli	Watson	✓	✓	100.0
		Dialogflow		✗	100.0
		Rasa			91.7
		LUIS		✓	98.3
5	Urlaub beantragen für den Zeitraum 28.September - 2.Oktober	Watson	✓	✗	97.0
		Dialogflow			54.9
		Rasa		✓	69.1
		LUIS			96.4
6	Ich brauche umbedingt Urlaub auf einer einsamen Insel	Watson	✓	-	96.0
		Dialogflow		✗	80.5
		Rasa		-	54.2
		LUIS			83.2
7	Kollege Helmut Kerschbaum hat Urlaub beantragt	Watson	✓	✗	97.0
		Dialogflow		✓	100.0
		Rasa			66.7
		LUIS			90.5
8	Kollege Helmut Kerschbaum will Urlaub. Er wird von 10/09 bis 24/09 nicht erreichbar sein.	Watson	✓	✗	96.0
		Dialogflow			100.0
		Rasa			53.0
		LUIS		✓	93.8

Table 10.5: Vacation Intent Classification German

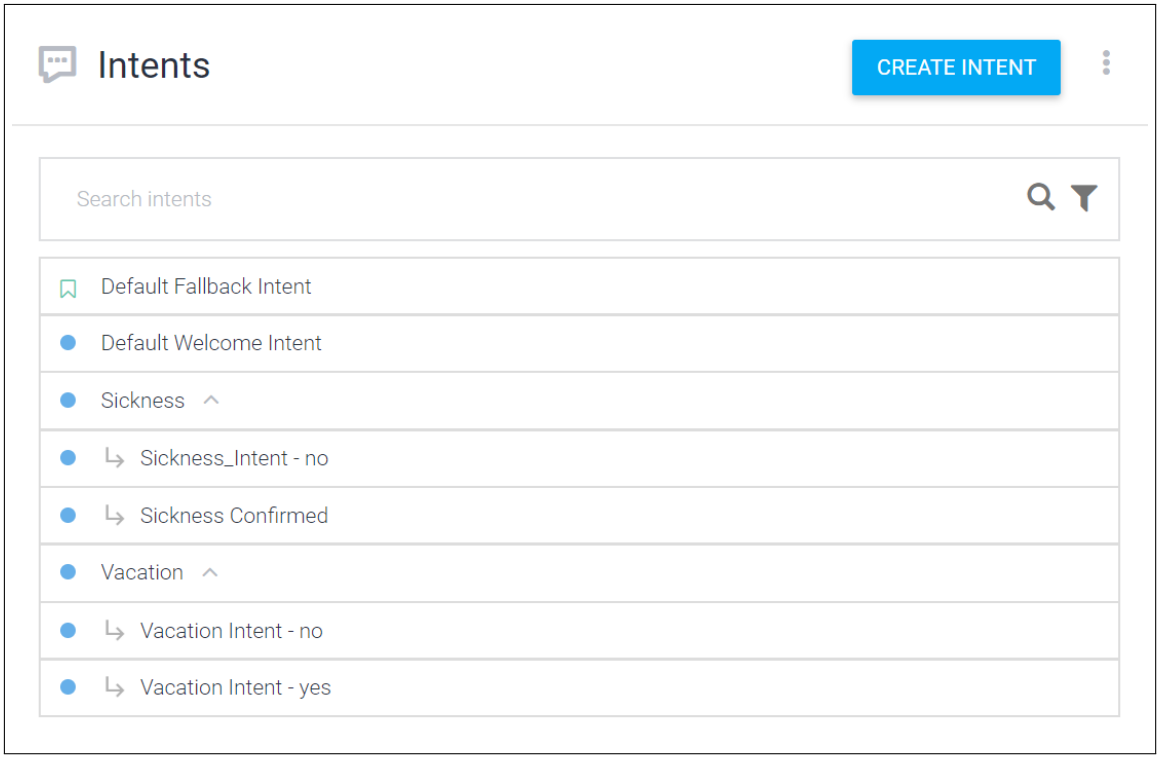

Figure 10.2: Dialog Nodes in Dialogflow

Bibliography

[1] 3 banken it company. https://www.3bankenit.at/company/company.xhtml, . Accessed: 2020-03-07. (Cited on page 3.)

[2] 3 banken it organigram. https://www.3bankenit.at/company/organigram.xhtml, . Accessed: 2020-03-07. (Cited on page 3.)

[3] 3 banken it services. https://www.3bankenit.at/competence/services.xhtml, . Accessed: 2020-03-07. (Cited on page 3.)

[4] Dialogflow. https://cloud.google.com/dialogflow. Accessed: 2020-05-09. (Cited on pages 6, 32, 45, and 74.)

[5] Duckling. https://duckling.wit.ai/. Accessed: 2019-05-29. (Cited on page 38.)

[6] Iso/iec 25010. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. Accessed: 2020-05-09. (Cited on pages 20, 21, 25, and 42.)

[7] Language understanding (luis) documentation. https://docs.microsoft.com/en-us/azure/cognitive-services/luis/. Accessed: 2019-12-26. (Cited on pages 6 and 40.)

[8] Build contextual assistants that really help customers. https://rasa.com/. Accessed: 2020-05-09. (Cited on pages 6, 36, 45, and 74.)

[9] Spacy api. https://spacy.io/api/annotation. Accessed: 2019-05-28. (Cited on page 38.)

[10] Watson assistant. https://www.ibm.com/cloud/watson-assistant/. Accessed: 2020-05-09. (Cited on pages 6, 34, 45, and 74.)

[11] N. S. Ahmad, M. H. Sanusi, M. H. Abd Wahab, A. Mustapha, Z. A. Sayadi, and M. Z. Saringat. Conversational bot for pharmacy: A natural language approach. In *2018 IEEE Conference on Open Systems (ICOS)*, pages 76–79, Nov 2018. doi: 10.1109/ICOS.2018.8632700. (Cited on pages 18, 19, and 22.)

[12] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*, 2017. (Cited on pages 6, 18, 19, and 22.)

[13] Antoine Bordes, Y-Lan Boureau, and Jason Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016. (Cited on pages 13, 14, 15, 16, 18, and 22.)

[14] Petter Bae Brandtzaeg and Asbjørn Følstad. Chatbots: changing user needs and motivations. *interactions*, 25(5):38–43, 2018. (Cited on pages 5, 17, 19, 20, 29, and 31.)

[15] Daniel Braun, Adrian Hernandez Mendez, Florian Matthes, and Manfred Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, Saarbrücken, Germany, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-5522. (Cited on pages 1, 2, 5, 15, 16, 17, 18, 19, 21, 22, and 59.)

[16] Aditya Deshpande, Alisha Shahane, Darshana Gadre, Mrunmayi Deshpande, and Prachi M Joshi. A survey of various chatbot implementation techniques. *International Journal of Computer Engineering and Applications*, 11, 2017. (Cited on pages 1, 5, 13, 14, and 22.)

[17] Debasatwa Dutta. Developing an intelligent chat-bot tool to assist high school students for learning general knowledge subjects. Technical report, Georgia Institute of Technology, 2017. (Cited on pages 5, 7, 8, 17, 18, 19, 20, and 22.)

[18] Asbjørn Følstad and Petter Bae Brandtzæg. Chatbots and the new world of hci. *interactions*, 24(4):38–42, 2017. (Cited on pages 17, 19, 20, and 27.)

[19] Kelly Geyer, Kara Greenfield, Alyssa Mensch, and Olga Simek. Named entity recognition in 140 characters or less. In *# Microposts*, pages 78–79, 2016. (Cited on pages 9, 13, 14, and 24.)

[20] Eun Go and S. Shyam Sundar. Humanizing chatbots: The effects of visual, identity and conversational cues on humanness perceptions. *Computers in Human Behavior*, 97:304 – 316, 2019. ISSN 0747-5632. doi: https://doi.org/10.1016/j.chb.2019.01.020. URL `http://www.sciencedirect.com/science/article/pii/S0747563219300329`. (Cited on pages 1, 17, 20, and 62.)

[21] N. A. Godse, S. Deodhar, S. Raut, and P. Jagdale. Implementation of chatbot for itsm application using ibm watson. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–5, Aug 2018. doi: 10.1109/ICCUBEA.2018.8697411. (Cited on page 22.)

[22] Eric Gregori. Evaluation of modern tools for an omscs advisor chatbot. 2017. (Cited on pages 1, 5, 10, 17, 18, 19, 22, and 23.)

[23] Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching for xml. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 67–80, 2001. (Cited on page 40.)

[24] Jizhou Huang, Ming Zhou, and Dan Yang. Extracting chatbot knowledge from online discussion forums. In *IJCAI*, volume 7, pages 423–428, 2007. (Cited on pages 1, 5, 10, and 62.)

[25] Danielle A Kane. The role of chatbots in teaching and learning. *E-Learning and the Academic Library: Essays on Innovative Initiatives*, 131, 2016. (Cited on pages 2, 5, and 13.)

[26] Amazon Lex. Conversational interfaces for your applications. *Powered by the same deep learning technologies as Alexa. URL https://aws. amazon. com/lex*. (Cited on page 17.)

[27] Aleksandra Przegalinska, Leon Ciechanowski, Anna Stroz, Peter Gloor, and Grzegorz Mazurek. In bot we trust: A new methodology of chatbot performance measures. *Business Horizons*, 62(6):785 – 797, 2019. ISSN 0007-6813. doi: https://doi.org/10.1016/j.bushor.2019.08.005. URL `http://www.sciencedirect.com/science/article/pii/S000768131930117X`. (Cited on pages 18 and 19.)

[28] AM Rahman, Abdullah Al Mamun, and Alma Islam. Programming challenges of chatbot: Current and future prospective. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 75–78. IEEE, 2017. (Cited on pages 1, 7, 14, and 22.)

[29] Sumit Raj. *Building chatbots with Python: using natural language processing and machine learning*. Springer, 2019. (Cited on pages 2, 3, 6, 7, 9, 12, 13, 14, 22, and 61.)

[30] Bayan Abu Shawar and Eric Atwell. Different measurements metrics to evaluate a chatbot system. In *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies*, pages 89–96. Association for Computational Linguistics, 2007. (Cited on pages 1, 5, 10, 14, 15, 16, 18, and 61.)

[31] Bayan Abu Shawar and Eric Atwell. Chatbots: are they really useful? In *Ldv forum*, volume 22, pages 29–49, 2007. (Cited on pages 1, 5, 10, and 62.)

[32] Abhishek Singh, Karthik Ramasubramanian, and Shrey Shivam. Building an enterprise chatbot. (Cited on pages 1, 5, 6, 8, 9, 16, 18, 22, 23, 24, 42, 61, and 62.)

[33] S Shyam Sundar, Saraswathi Bellur, Jeeyun Oh, Haiyan Jia, and Hyang-Sook Kim. Theoretical importance of contingency in human-computer interaction: effects of message interactivity on user engagement. *Communication Research*, 43(5):595–625, 2016. (Cited on page 20.)

[34] Dana Vrajitoru, Jacob Ratkiewicz, et al. Evolutionary sentence combination for chatterbots. In *International Conference on Artificial Intelligence and Applications (AIA 2004)*, pages 287–292, 2004. (Cited on page 5.)

[35] Jason D Williams, Eslam Kamal, Mokhtar Ashour, Hani Amr, Jessica Miller, and Geoff Zweig. Fast and easy language understanding for dialog systems with microsoft language understanding intelligent service (luis). In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 159–161, 2015. (Cited on pages 5, 6, 18, 19, and 22.)

[36] Jason D Williams, Kavosh Asadi, and Geoffrey Zweig. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *arXiv preprint arXiv:1702.03274*, 2017. (Cited on pages 5, 13, 15, 16, 18, and 22.)

List of Figures

2.1 Example FAQ Conversation with Dialogflow 10
2.2 Greeting Story 11

5.1 Minimalistic Chatbot Architecture 26
5.2 Sickness Notification Example Conversation 27
5.3 Chatbot Conversation Flowchart 28
5.4 Chatbot UI 31

6.1 Dialogflow Web Interface 34
6.2 Watson Assistant Web Interface 36
6.3 Rasa X Interface 39
6.4 LUIS Web Interface 41

10.1 From Filling in Watson Assistant (top) and Dialogflow (bottom) 66
10.2 Dialog Nodes in Dialogflow 69

List of Tables

2.1 Default Welcome Intent of Dialogflow 8
2.2 Examples for Concept Explanation 9
2.3 Entity examples 9

4.1 Sickness Training Utterances 23

5.1 Analysis of Figure 5.3 29
5.2 Sickness Notification Intents, Enitities, and Actions of Figure 5.2 30

6.1 Conditional Responses of Watson Assistant 36

7.1 Pricing of Frameworks [8, 4, 10] 45
7.2 Predefined Entities 46
7.3 Framework Entity Recognition and Extraction 47
7.4 Entity Recognition Result Evaluation 48
7.5 Person Entity Recogniton and Extraction 49
7.6 Date Entity Recogniton and Extraction 50
7.7 Date Entity Recogniton and Extraction 2 53
7.8 Date Span Entity Recogniton and Extraction 1 54
7.9 Date Span Entity Recogniton and Extraction 2 55
7.10 Sickness Intent Classification 56
7.11 Vacation Intent Classification 57
7.12 Intent Classification Result 58
7.13 Intent Classification Result German 58

10.1 Sickness Training Utterances German 64
10.2 Vacation Training Utterances 64
10.3 Vacation Training Utterances German 65
10.4 Sickness Intent Classification German 67
10.5 Vacation Intent Classification German 68

List of Listings

2.1 Dialogflow Webhook Request Example 11
6.1 Dialogflow Webhook Entry Point 33
6.2 Dialogflow Request Parameters 33
6.3 Dialogflow Intent Handling 33
6.4 Watson Assistant Request Format 36
6.5 Rasa Docker Compose File 37
6.6 Rasa Intent Format 37
6.7 Rasa Story Format 37
6.8 Rasa Configuration 38
6.9 Message Format 41
6.10 Messenger Structure 41
10.1 Detailed Pipeline Configuration for Rasa 63
10.2 Rasa Slot Filling Action 63
10.3 Rasa Sickness Story 64
10.4 LUIS Intent Response 65
10.5 LUIS Entity Response 65