Fachhochschul-Masterstudiengang
SOFTWARE ENGINEERING
4232 Hagenberg, Austria

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

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science in Engineering

Eingereicht von

Gerald Spenlingwimmer, BSc

Betreuer	Dipl.-Ing. Thomas Reidinger, M.A, 3 Banken IT GmbH, Linz
Begutachter:	FH-Prof. PD DI Dr. Stephan Dreiseitl

September 2020

software engineering
IKM Fakultät Hagenberg

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

The 3 Banken IT wants to use chatbots for two internal use-cases. One use-case is about sickness notifications, and the second one is about vacation requests. The required key feature is the extraction of information from text since both use-cases require person names and dates to be automated.

Before the development phase of a chatbot begins, two main questions arise. What are suitable problems for chatbot solutions and does the given problem satisfy What makes a problem suitable for a chatbot solution and are the given use-cases suitable for a chatbot. A problem is suitable if it is repetitive, can be automated, and only requires simple back-and-forth communication [29]. The given use-cases are repetitive, can be automated, and require only simple back and forth communication. Hence, they are suitable for a chatbot solution.

Prototypes for the use-cases were developed with Dialogflow, LUIS, Rasa, and Watson Assistant to validate the suitability of the use-cases and to do the proof of concept. The technologies were ranked using factors like the entity extraction performance, the intent classification performance in English and German test scenarios, the usability of the frameworks, the customization possibilities, and learnability. Contrary to the sources found on the internet, development relevant aspects of the frameworks are considered to give a development recommendation to the company.

A key factor why companies want to use chatbots is because they are cheap and always available. The prices of the frameworks and the text-to-speech and speech-to-text services were listed and compared. The costs for the cloud chatbots were on an equal level as expected. The same is true for text-to-speech and speech-to-text services. The price comparison is hard because multiple versions are available, and the units differ significantly between the frameworks. Rasa does not list the costs for the enterprise edition on the website, nor does Rasa offer text-to-speech or speech-to-text services. All cloud technologies have a free tier with limited usage, and Rasa is cost-free by default.

The comparison of the frameworks' similarities and differences led to the discovery of common and technology-specific concepts and development-relevant evaluation criteria. All four technologies use intents, entities, and utterances. All three chatbot technologies had a dialog handling mechanism. The cloud chatbots use dialog-nodes while Rasa uses stories to represent the flow of the conversation. The most significant difference is that Dialogflow, LUIS, and Watson Assistant are cloud-based solutions, whereas Rasa is a local solution. The local solution Rasa offers advantages when it comes to offline capabilities, flexible deployment, training, and independence of providers. The cloud technologies were, in general, easier to use and learn; the provider handles the deployment, they scale automatically and perform better on average with small training sets.

The two primary functions of chatbots are entity-extraction and intent classification and were evaluated in detail to determine the best technology. Dialogflow won the entity extraction evaluation by a large margin backed up by the highest f-score. The intent clas-

sification in English worked best with Watson Assistant, closely followed by Dialogflow and LUIS. Rasa was unable to keep up with the cloud technologies in the English test setting. The most comfortable technologies from a usability and learnability point of view are Dialogflow and LUIS, and the recommendation is to use them when possible.

The results were quite different for the German test sentences. The intent classification worked measurably better with all four technologies. The performance of the entity extraction dropped significantly for all three chatbot technologies. LUIS's entity extraction performance increased, contrary to the other three technologies, and is the recommended technology for German. To summarize, the only technology which showed excellent performance on the German tests was LUIS, followed by Rasa, which performed quite well. Rasa is the alternative to LUIS for a German setting. The entity extraction performance of Dialogflow on the German tests was horrible when the input language is German. Annotation of entities in training sentences is only supported in English by Watson Assistant. IBMs Watson Assistant is useless for the German language until the annotation of entities is available for German. LUIS takes the top spot since the performance was good for both languages, followed by Rasa as a good alternative.

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 to develop 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 significantly reduced 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 also able to develop a chatbot.

The major benefits of chatbot systems are that they never complain, are cost-efficient, are available 24 hours a day, seven days a week, and handle enormous service demands without a problem [32]. They are always available if assistance is required at 3 AM the chatbot will be available. In contrast, a customer service hotline is bound to the opening hours and might not be available at 3 AM. Large service demand handling is especially interesting for companies that do not have the resources to hire enough people to match the service demands. They are also used to increase the user acceptance of FAQ pages, for instance. Chatbots can significantly reduce the costs of customer services and increase customer satisfaction [32]. Because of these characteristics, chatbots are becoming popular in the customer support section, where they are used to replace humans [20].

There are various reasons to use chatbots. It is 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 ideal for monotonous, repetitive routine work no employee likes to do. This way, they are used to aid employees instead of replacing them. The employees can then focus on important tasks instead of repetitive work. Another popular reason for the use of chatbots is the price. Employees are expensive, and chatbots are not. Hence, if a chatbot can do a few employees' works, much money can be saved. All these reasons mentioned so far are key factors why chatbot systems are getting popular.

Multiple factors need to be considered to meet the companies needs. The company needs a knowledge base about chatbots in general and for future projects. Furthermore, many chatbot frameworks are available nowadays. The question is, which is the best overall, which is best for the given problem, what are the strengths, weaknesses, similarities, and differences between the frameworks. All of the big tech companies offer chatbot technologies and NLU services. There are the cloud services Dialogflow (Google), Watson Assistant (IBM), Alexa/Lex (Amazon), LUIS + Microsoft Bot Framework. From the found state-of-the-art technologies, four were selected for prototyping.

The chosen technologies were compared with domain-specific data to determine which frameworks can be recommended for further development. The predefined 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 frameworks will be trained and tested with the same data to prevent bias.

No noticeable difference is expected for the final prototypes. 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 gained from the evaluation is used to give a recommendation for future development. Another important aspect is the development of chatbot systems in general. This includes the development approach, the development differences to the regular software development and psychological aspects that need to be considered.

1.2 Motivation

The company currently has no knowledge base about chatbot systems. To justify using a chatbot system, knowledge about those systems needs to be collected and evaluated. The required knowledge includes chatbots in general, 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 is 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 and vacation request use-cases need to be implemented with 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 it to a server for processing.

Chatbots offer various benefits. They 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 use for humans [25]. Anyone who can talk to or write with a person can work with a chatbot interface [29]. It is also a natural way of communication and can increase user acceptance. In summary, a chatbot provides 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

Some prerequisites influence the choosen technologies for this thesis. Based on this thesis, further projects will be developed. The company currently has no experience with chatbot systems. Hence, the collection of information about state-of-the-art technologies is important. The company uses a local container environment for deployment. An 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 uses lots of IBM software. Because of this, the use of Watson Assistant is mandatory. At least three technologies need to be chosen for a comparison. The domain defined by the company are sickness notifications and vacation requests. This means that the chosen chatbots need to support the collection of information in a given domain. The primary purpose is to recommend

a chatbot 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 notifications and vacation requests have been analyzed for the suitability for chatbots. The domain information has been converted to intents, entities, utterances, and dialogs. The dialogs have been implemented with all technologies, if possible. The training and test data sets for the chatbots were 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 frameworks have 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. The synonyms chatterbot and conversational agent appear in the literature. The communication with a chatbot is most often done in written form like in a chat but not restricted to written communication. Speech is another form of natural language used in technologies like Amazon Alexa and Google Home. The basic concepts remain the same for speech and text communication. In general, the communication between man and machine is done in turns until a conversation ends or is aborted [34]. When the chatbot finishes the underlying task, the conversation ends, for instance. As a rule of thumb, a chatbot generates the most likely response base on the user input to drive the conversation forward [17]. In the best case, the generated response is reasonable and intelligent [34]. In the worst case, a generic message like ”I did not understand that” is generated. It is essential to keep in mind that a chatbot can mimic intelligence, human dialog, and personality but is often limited to specific functionality [25, 17].

In general, 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 standard 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 is not connected to a person immediately. The system gives options like ”press one if you would 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 of domain-specific functionality would be a refund desk in a shop. The refund desk only serves customers with refund related issues. The customer is redirected to the information counter when he asks for the location of the TV area inside the shop. A chatbot system in the refund desk domain only offers the functionality necessary to get and collect refund information and issue a refund. The response to issues unrelated to a refund will not be useful in most cases. The

third type is an open-ended system. They handle general conversations. Contrary to the other two approaches, the user can ask open-ended systems anything. 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. It is also impossible to rely on specific information since every conversation varies greatly. Famous examples of open-ended systems are Google Home, Siri (Apple), and Alexa (Amazon) [32]. The most limited approach is the menu-driven approach. The user selects predefined options through numeric input to execute a function. Domain-specific bots communicate with users in natural language and are excellent in achieving tasks and collecting information. Open-ended systems are perfect for general communication and are domain-independent, but it is hard to collect specific information or achieve a domain-specific task. Figure 2.1 shows a simple exemplary chatbot conversation.

In general, an ML-based chatbot system consists of rules and training data. The training data needs to be as realistic as possible to provide a natural conversation and high quality. The training sentences can be handcrafted for prototyping or generated from large sources like e-mails, phone calls, or the internet. ML-based chatbots use sets of rules. These rules define the available functionality, types, and possible input statements. A person's name consists of a first and last name is an example of a simple rule.

A chatbot has two major tasks. It needs to classify the user's intention (intent) and extract the relevant pieces of information from the message. A relevant piece of information is called an entity. An entity has a type and can be a date or person's name, for instance.

2.2 Chatbot/NLU Frameworks

The current chatbot technologies enable non ML experts to develop a chatbot system. There are cloud-based and local chatbot solutions available. The cloud-based solutions are created, edited, and developed through a web-interface. They offer a REST API communication endpoint to build custom applications. In general, the developer is dependent on the cloud provider. All data passes through and in the provider's cloud environment. 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 found in literature is Rasa (Bocklisch et al. [12], ras [8]).

A major concept of the chatbot technologies listed above is slot-filling. A slot is a predefined type/entity which a user needs to provide in a conversation. In Table 2.2 shows the slots of a hotel example. For a hotel reservation, the slots are the hotel name, a start date, an end date, and a name for the reservation. All slots need to be filled before the system can process the hotel reservation automatically.

2.3 Suitable Problems for Chatbots

Not every problem can be solved using chatbots [29]. Chatbots seem like intelligent systems, but they are often designed for specific tasks and are unable to solve complex problems. Chatbots are often used to reduce the workload of humans by aiding them. They can also serve a vast number of requests efficiently. 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. It is possible to solve non-repetitive tasks with chatbots too, but it is not cost-effective. If the problem cannot be automated, an employee is needed for the process, and it is harder to serve a large number of users. Chatbots are only capable of text or voice communication. 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 three statements are valid for a problem, it is suitable for a chatbot. A simple QnA bot for a website's FAQ page is an example of a suitable problem [29]. The task can be automated, is highly repetitive, and the FAQ page predefines the answers.

As mentioned in Section 2.1, there are three major types of chatbots. The menu-driven and domain-specific approaches are suitable for goal-oriented tasks where data needs to be collected to execute an action. They only support a limited set of functionality. 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, the domain-specific approach works best (FAQ chatbot). In the FAQ example, the domain consists of all questions and answers listed on the FAQ page and nothing else. It would be bothersome to navigate through a FAQ forum with the menu-driven approach using numbers. If 1000 topics are available, the user would need to enter a number between one and 1000. Such a system would provide a horrible user experience. In such a case, a user just wants to ask a question to get an appropriate answer. Domain-specific technologies offer the required functionality and are the tool of choice for such problems. The open-ended approach is suitable when no specific data needs to be collected, and the conversation is general without any data collection requirements. Google Assistant can answer general questions like "What is the weather in New York" and gives a general response. It cannot process a request like "Tell my employer I am sick and will not come to work today" since it requires domain-specific data and actions.

2.4 Intent

Intents are the core concept of any chatbot system. An "intent" is the intention of a user and the action he wants to perform [17, 28]. The user wants to book a hotel in sentence two of Table 2.2. 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. The system finds the intent where the training phrases are closest to the user's query or uses the fallback if no intent reached a high enough score. The confidence score is a percentage value between zero and one. Zero is the lower end and means no match, and one is the high end and means a perfect match. The threshold defines the minimum score the intent

classification has to detect to treat the input as a match. The fallback intent handles values below the threshold with a generic response like ”I am sorry, but I did not understand that”. Some intent and entity examples are listed in Table 2.2. The intent of example sentence one listed in Table 2.2 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. As the name suggests, the training phrases are used to train the ML model. Input sentences are classified as the intent if the confidence score is above the threshold and higher than for all other intents. The confidence score reflects how similar the input is to the training sentences. The system then responds with a random pick from the response list. The random pick fakes intelligence and keeps the user interested. It would be annoying to get the same response every time. Such a conversation would be classified as robotic and unnatural, which is the opposite of the desired outcome.

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 chat environments, an utterance is often just a part of a sentence to shorten the message. All sentences listed in Table 2.2 are utterances for instance. The messages of users are also utterances [32, 17]. The chatbot responds to users with utterances. Table 2.1 shows some training phrases and the response list. Table 2.1 shows that a collection of utterances belongs to an ”intent” and builds the base for the classification of user inputs. Training phrases are different versions of the same question or statement. This allows the system to correctly classify utterances that never were part of the training data as long as the utterance is similar to the known sentences. 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. It is legitimate to handcraft utterances

for development and prototyping purposes. The best sources for training data are real-life sources like usage data from live systems, e-mails, phone conversations, or online resources [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 represents data that belongs to an intent an has a type. Entities are volumes, counts, and quantities, for instance [29]. Example sentence seven of Table 2.2 showes an intent wiht multiple entities. The first entity is the shirt color, the second is the shirt as an item, and the third one the shirt size.

Sentences three to six of Table 2.2 show entities in a book hotel example. In this simplified example, to book a hotel, the hotel name (Hotel Item), the check-in and check-out dates (Date), and a name (Person) are required 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 ambiguities like in sentences five and six of Table 2.2. In sentence five, May is the month and needs to be treated as a date, whereas in sentence six, it is the last name and is part of a person entity.

A named entity is a noun phrase relating to individuals. Classic individuals/named entities are persons (Bill Gates), locations (New York), and organizations (Google) [19]. Frameworks provide common predefined named-entities like the ones above. A developer can define named entities for a specific domain like in the hotel example where an entity is needed to represent the valid hotel names. Example sentences four and five of Table 2.2 show a date entity, and sentence six shows a person entity. Table 2.3 lists an example entity for food items like apples and pears. An entity can be a collection of fruit items, a collection of synonyms, or something more complex. The entities are only valid for the current conversation and stored in the session. They form the context of a conversation [32].

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 Section 2.5. The response can be picked at random or in a sequence. Response alternatives keep the conversation interesting for users. If the same response is picked all the time users realize fast that they are talking to a bot, and they get bored. Alternatives also prevent a robotic and unnatural conversation. The alternating responses counteract a robot-like feel of the conversation, especially for returning customers. Table 2.1 shows the default welcome intent of Dialogflow with the response list.

2.8 Story

A story defines the dialog flow. It is 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 a ”sad” path. A ”happy” path is a successful interaction from start to finish without any problems. The ”sad” path handles the behavior when something does not 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 are 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. Figure 2.1 shows an example of a conversation between a person and a chatbot. The story for Figure 2.1 is that the user asks for open hours, the bot responds with the open hours, the user asks for an item, the system validates the item, and generates a response with the item in the response text.

2.9 Webhook

A webhook is a URL endpoint where the frameworks send the metadata and intent data for analysis, completion, and processing. In the case of this thesis, the webhook is a Python 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.

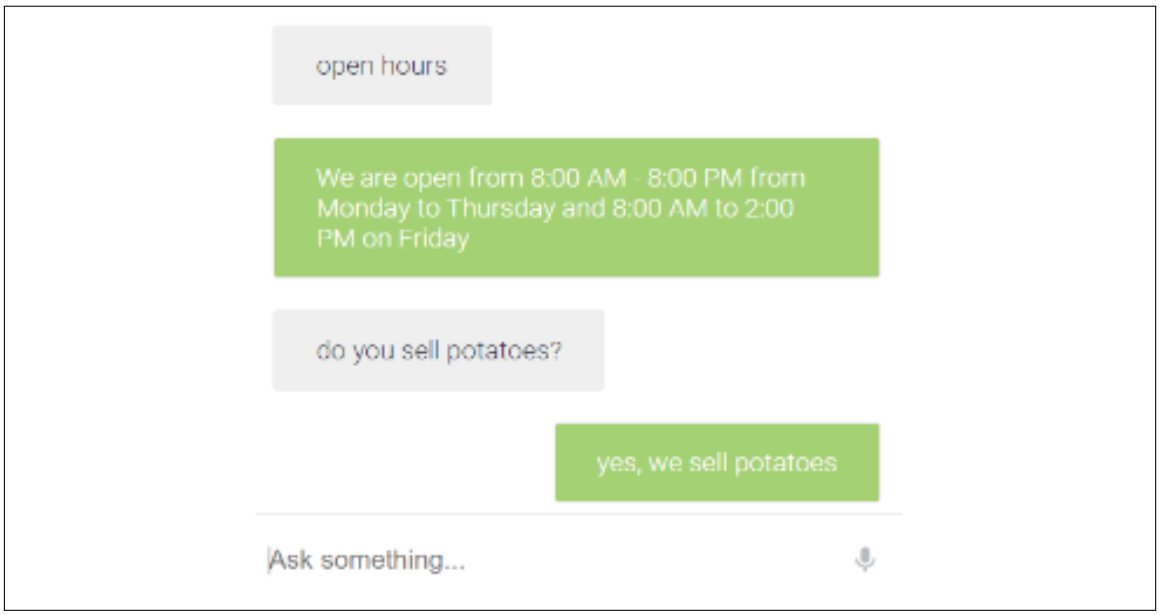

Figure 2.1: Example FAQ Conversation with Dialogflow

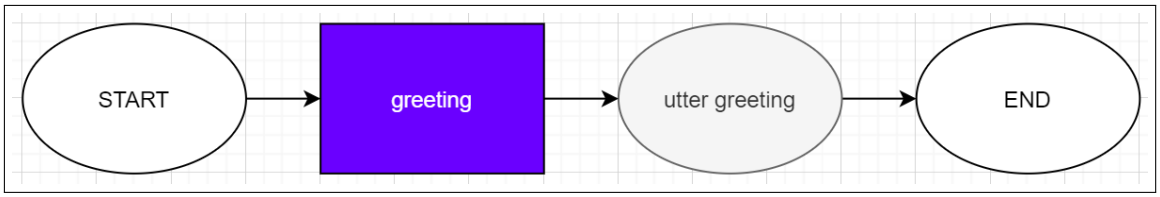

Figure 2.2: Greeting Story


```
[
{
    "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 supported functionality. A domain-specific chatbot has a limited set of operations and works with data relevant to the domain. In the refund-desk scenario, the domain is ”refund” for example. The bot can process and answer refund-related issues, collect refund-specific information, and issue a refund. It does not offer functionality which is unrelated to the refund domain-like ”What is the weather in New York today” and it is not supposed to do so. Open-ended systems escape the domain limitation but have problems executing domain-specific tasks or collecting domain-specific information.

2.11 Evaluation of NLP capabilities

In literature, F-score values are used to evaluate and compare the performance of natural language systems. The scenario for the f-score explanation is a medical test for a disease. Positive means 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 chatbots area, a true positive is when the input is ”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 $2 * \frac{precision*recall}{precision+recall}$. The range of the f-score, precision, and recall values is zero to one where zero is the worst possible result, and one is a perfect result. The precision value shows how many retrieved elements are relevant. The recall value represents how many relevant items were retrieved. To summarize, precision uses the selected elements (tp + fp), whereas recall is based on the relevant elements (tp + fn). Figure 10.3 from the appendix is a visual aid for a better understanding of the difference between precision and recall. The f-score is the average of precision and recall.

In the medical area, recall is more important than precision. The goal is to identify as many sick people as possible correctly. It is not that important if people are false-positive since further checks are always possible to verify the test result. In the area of chatbots, neither false negatives nor false positives are desirable, and the f-score is the most relevant value.

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 (NLP) in the area of chatbots is necessary to analyze the user’s input and figure out the intent and entities necessary to process the given information. NLP is the brain of a chatbot. It receives the user’s information,

processes the data, and retrieves the relevant parts from the input for post-processing. Part-of-Speech (POS) tagging assigns parts of speech like noun, verb, and adjective to each word or token [29]. This information is used to understand the context and is part of NLU.

An essential part of natural language processing is the search for the root of words [29]. Stemming and lemmatization are two conventional approaches. Stemming reduces a word to the base form by cutting off (stem) the endings. This way, ”saying” is reduced to ”say” by removing the ending (ing) of the word. Lemmatization takes a different approach. It tries to find the dictionary form of a word, has better correctness, and is more productive than stemming and hence the preferred method. Lemmatization uses a vocabulary and morphological analysis of words to achieve its task. Both approaches use stop-word removal to get rid of unimportant words. Stop words are words that hold little to no meaning and occur in a high frequency like ”the” and ”a”.

Chapter 3

State of the Art

There are many ways for chatbot type classification, and the category selection depends on the developer or researcher's view and needs. Williams et al. [36] and Bordes et al. [13] state that the two major chatbot types are goal-oriented and end-to-end systems. These two classes define the general type of conversation. Another way of classification is the differentiation between local standalone-applications and web-based solutions like done by Kane [25]. These two categories define the type of development, the source code's location, and the usage type of a chatbot.

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. A combination of these categories is possible as Rasa is an example of a local standalone application built for goal-oriented dialog. The web-based solutions LUIS, Dialogflow, and Watson Assistant, use goal-oriented dialog.

Current chatbot systems use natural language processing (NLP) to respond to user inputs [16]. 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 users, whereas current technologies like Siri use NLP.

The analysis of questions and statements is the main focus of NLP in the area of chatbots [16]. Extraction intents, commands, and actions from user input is the goal. The basic workflow of a chatbot system starts with the user input. Then the NLP engine extracts entities and detects the intent. An entity represents an interesting piece of information. The extracted data is store in entity slots. After the extraction, the user gets a response from the chatbot.

Some of the best chatbot and NLP frameworks are Rasa, LUIS, and Dialogflow [29]. Raj [29] used Dialogflow to build a classical food order chatbot and Dialogflow to build a horoscope bot. To summarize, Raj [29] recommended Rasa, LUIS, and Dialogflow and used two of them for the implementation of chatbots. The collection of state-of-the-art technologies is an important task of this thesis, and these three are candidates.

Raj [29] is an excellent source to learn about the core concepts of chatbots, NLU, and NLP necessary for the development of chatbots. Chatbots do not require complex interfaces since they support natural language. The interaction with a chatbot is natural and easy, and they get more and more popular. They often seem intelligent, but developers and users should be aware that chatbots cannot do everything and often serve one specific task. Such bots fall under the task-oriented category.

A core question before the development of a bot is if the underlying problem is suitable for a chatbot or not [29]. Three factors determine if the problem is suitable for using a chatbot. Analyzing the problem is necessary to answer the question. Chatbots use simple back back-and-forth communication. No other means of communication are available. Hence, a suitable problem requires only simple back-and-forth communication to achieve its task. A suitable problem is also highly repetitive and automatable. It does not make much sense to implement a chatbot for a non-repetitive task, which cannot be automated. The development overhead for non-repetitive tasks is not worth the effort from a financial perspective. Non-automatable tasks require human help to be done, which is what a chatbot should get rid of in most cases. A suitable problem for a chatbot solution is a FAQ help page, for instance. The problem is repetitive since users query a predefined set of questions and answers. The FAQ page is automatable since the user communicates solely with the web-page without human help. The problem scope consists only of questions and answers. This fits the required back-and-forth communication pattern perfectly. The FAQ page is suitable since all three required factors are present.

Named entity recognition (NER) is the core concept used by chatbot tools to extract information from text [19]. The classic examples of named entities are people, locations, and organizations. An important quality factor of NER systems is how good the entity extraction works. The frequency of entities in the training data can influence the NER performance significantly. Since entity extraction is one of the core concepts of chatbots, a similar evaluation is possible for chatbots.

Commonly used cloud chatbot solutions are IBM Watson, Dialogflow, and the Microsoft Bot Framework [28]. Tech giants provide chatbot frameworks for regular developers where programming and ML skills are no longer required. Google provides API.ai/Dialogflow, Microsoft has LUIS, IBM develops Watson, and Amazon provides Lex. Goal-oriented chatbots are most common in the business sector and help users to achieve tasks.

In general, the chatbot architecture consists of intent classification, entity recognition, response generator, and a response selector [28]. The intent classification identifies the best matching intent for the user input. The entity recognition module extracts the information/data from the user's message. The response generator provides response candidates, and the response selector chooses the best matching response. The core concepts used by Dialogflow are intent, entity, and utterance. Intents execute actions based on the user's input.

The state-of-the-art technologies mentioned by Rahman et al. [28] are IBM Watson, Dialogflow, the Microsoft Bot Framework, and Amazon LEX.

To summatize, Deshpande et al. [16], Raj [29], Geyer et al. [19] and Rahman et al. [28] focus on the basics of chatbot frameworks and chatbot related concepts. Deshpande et al. [16] and Rahman et al. [28] describe the general workflow of chatbot systems. Geyer et al. [19] is about NER which is one of the core concepts used by chatbot systems. Additionally, Raj [29] also explains which problems are suitable for a chatbot in general.

End-to-end systems for general conversations are the first major chatbot type. Shawar and Atwell [30] focuses on the application of chat systems in different languages and the evaluation of chatbot systems in general.

The best way to evaluate a chatbot is to check if the specific service or task is achievable [30] The evaluation of chatbot systems depends on the use-case, the application, and the user's needs. A conventional test approach is a system test to acquire user feedback. This helps to evaluate user satisfaction and acceptance. The feedback is used to improve

the system. It is also possible to test the components of a chatbot system individually.

Custom-built solutions can outperform classical approaches in small areas [30]. Shawar and Atwell [30] built a chatbot for frequently asked questions (FAQ). The custom FAQ bot competed against the classical Google search. The participants had to answer a few questions like how many results the system found, which system they preferred, and why. The custom solution found a reasonable answer in more cases than Google search. 47% of the users taking part in the study preferred the custom solution, and 11% preferred Google. The primary reason why the users preferred the FAQ chat was because the chat can often give direct answers and returns fewer links on average, which saves browsing time. The users preferred Google because it is familiar, and it can give different answers when the input query is adjusted slightly.

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. However, The most useful application areas for chatbots are either goal-oriented or transactional settings.

A classical dialog system uses slot-filling to collect information from the conversation [13]. Slot-filling is reliable but has limited scalability. The slots are predefined and hence do not scale well because they can be different for each problem.

The main question is if end-to-end systems perform well on the classical goal-oriented restaurant reservation task [13]. End-to-end systems scale well because they learn directly from conversations and make no assumptions regarding the domain or dialog structure. 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 goal-oriented settings. The results 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, but the end-to-end system never identified all requests of the conversation correctly [13]. The request identification worked well, but 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 a 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. A restaurant reservation is a task-oriented operation that requires domain-specific knowledge. The domain-specific knowledge in a restaurant setting can be the restaurant name and the number of people for the table reservation, for instance.

End-to-end systems lack a mechanism to inject such domain-specific information [36]. The extraction of such information is required to perform well in a goal-oriented setting. There is also no mechanism for constraints in goal-oriented systems. In a banking app, the user needs to log in (constraint) before retrieving account information. A programmer can code such constraints in a few lines, but lots of training data need to be provided to a system to learn such a mechanism on its own.

The hybrid-coding-networks (HCNs) approach extends the open-ended system with domain-specific knowledge [36]. The programmer provides the domain-specific knowl-

edge to the system, which results in more development effort. Since the domain-specific knowledge is provided, the system does not need to learn these parts, and less training data is required compared to existing end-to-end approaches. Real dialogs provide a large source of training data.

The experiments show that the HCN approach performs on an equal level with existing end-to-end approaches in end-to-end settings and need less training data to achieve the same performance [36]. The HCN system also performed better in the task-specific setting than the existing end-to-end approaches. 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. The main benefits are domain-specific information in an end-to-end dialog and the reduced amount of training data.

Other references focus on end-to-end and 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, it is impossible to make a statement 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 an important factor before a chatbot is developed and help to choose the best framework for development. The user's feedback is a valuable source of information for the actual product and is important long after the framework was 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], and 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. Since goal-oriented technologies are the only ones that perform reliably in goal-oriented settings, using them for goal-oriented projects is recommended.

Williams et al. [36] combines 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 HCN comparison does not include goal-oriented approaches, which makes the result irrelevant for this thesis.

The conversation type of a chatbot is wither a general conversation about a broad generic subject or a specific conversation about one product or service Singh et al. [32]. The first step to create a chatbot system is to define a conversational flow. The conversational flow is a decision tree that describes the possible events, decisions, and outcomes of a conversation. An example of a general conversation is when a customer walks into a bank and starts talking to an employee. In this example, we have no idea what the person wants or who the person is. Famous examples of chatbots capable of general conversation are Google Home, Siri, and Amazon Alexa.

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. The domain could be named refund, and the specific rules and information are part of the domain. The customer cannot get any other information than refund related information at the refund desk. The conversation's outcome can be one of the predefined outcomes, the fallback, or a conversation end. A specific conversation ends as soon as the underlying task is achieved or a conversation end is reached. In general, specific conversations are easier to predict and are handled with higher accuracy.

The core component of AI-driven chatbots is the NLP engine, which takes care of data extraction from natural language [32]. The extracted information determines the next steps. There is no difference in functionality between a regular application and a chatbot in terms of functionality. The difference is that a chatbot uses conversation, whereas a regular app is a self-service application. Conversational training data for chatbots can be acquired from lots of sources like emails, phone calls, chats, and social media.

Companies use chatbots because they can save much money and offer a good customer experience, but there are also security issues for cloud solutions [32]. Especially in areas with personal data, cloud chatbot solutions like Dialogflow, Alexa, and Watson are problematic because all the conversation data is sent to and stored at the providers' server. Local chatbot solutions like Rasa are an option for applications whit high-security requirements. Singh et al. [32] built an in-house/local chatbot to provide the required security.

Singh et al. [32] also gives an overview of the basics of Microsoft Bot (LUIS), Rasa, and Google Dialogflow. Furthermore, Singh et al. [32] mentions Alexa, Google Home, and Siri as popular natural language systems. The second big chatbot category is called goal-oriented.

Modern conversational agents require small to no programming knowledge because of NLU services [15]. The recent advancements in machine learning (ML) and NLU and the popularity of messenger platforms has led to huge progress in recent years. Many people use messenger platforms daily, and most are familiar with such platforms. Hence, chatting has become a natural way of communication.

In general, The architecture of a chatbot consists of the request interpretation, the response retrieval, and the response message generation [15]. The purpose of these services is the extraction of information from natural language.

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. All three NLU services share the same basic concepts of intents, entities, and batch import in JSON format. The cloud-based services are secretive when it comes to the ML algorithms and the initial training data. The exception is Rasa, where the developer chooses and modifies the ML backend.

Recent publications discussed the use of NLU services but did not mention why one service was chosen over another [15]. Braun et al. [15] used training data from a production chatbot and two other training sets from StackExchange to evaluate the performance of the NLU services. The comparison included the services LUIS, Rasa, Dialogflow, and Watson Conversation. The services use the same training data to create a fair testing environment. The evaluation of the services NLU capability uses true and false positives and negatives, recall, precision, and F-score. The better the F-score is, the better the NLU service has performed. The evaluation of NLU services is only a snapshot since the services are continuously improving.

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 outperform LUIS through customization. The domain had little to no influence on the ranking of the NLU performance. In other words, a good NLU system performs good independent of the domain. If the training data is sparse, there is no significant difference between the services' performance. Before using an NLU service, different services should be tested with domain-specific data to find the best matching technology for the

given problem.

The comparison and evaluation of Braun et al. [15] included LUIS, Watson Conversation, Dialogflow/API.ai, and RASA. These four technologies are used and mentioned in many articles and online sources.

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 the learning process. The developed chatbot is a web-based education solution using natural language processing (NLP) techniques to answer questions. The bot can participate in small talk to create the illusion of talking to a human. It also helps to motivate learners and might increase interest in the topic.

Popular chatbot platforms are Dialogflow, LUIS, Wit.ai, and Pandorabots [17]. The chatbot systems evaluation is based on their NLP performance and the available features of the platform The training data is the same for all technologies to make a comparison possible. LUIS and Wit.ai showed a slightly better NLP performance than Dialogflow. Dialogflow offers the concept of follow-up intents, which is an important feature for the development of sub-tasks. Thus, Dialogflow was chosen for development over LUIS and Wit.ai. Use-cases for AI-based chatbots are banking systems, customer services, and education, for instance.

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 platforms' available features 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], and Følstad and Brandtzæg [18] go deeper into the psychological aspects of chatbot systems.

The higher the confidence score is, the better the user request matched the training data [22]. Gregori [22] evaluated the NLU capabilities of Wit.ai, Luis, Api.ai/Dialogflow, and Amazon Lex[26] by the confidence score. As other researchers already mentioned, each tool was trained with the same data and was tested with the same questions to ensure fair conditions. LUIS, Wit.ai, and API.ai/Dialogflow performed on an equal level in terms of intent classification. Gregori [22] is another researcher that chose Api.ai/Dialogflow for the development of a prototype. It seems that many researchers use Dialogflow for the development of a chatbot.

Ahmad et al. [11] builds a chatbot 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 give information on the medication intake. 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]. In general, the measurement criteria for chatbots vary depending on the domain of the bot. The general trend is to keep conversations short. Personalized communication is used when recommendations or tips are provided for the user based on the information related to the user. Retail chatbots need a larger amount of conversation steps to provide information and recommendations and hold the user's attention.

Historically, language understanding was implemented via machine learning or handcrafted rules [35]. Williams et al. [35] gives an overview of LUIS, the state-of-the-art language understanding service of Microsoft. The ML model approach is robust but requires

expensive expertise. In general, software developers can build language understanding without the assistance of a framework with handcrafted rules. However, systems with handcrafted rules do not scale well.

Using LUIS, developers do not 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 a cloud-based, domain-specific language understanding models. Developers need to understand the concepts' intent, entity, and utterance to work with LUIS. Developers can create custom entities and use existing entities like location, date, and time. LUIS offers an HTTP endpoint using JSON format for messages. Dialogflow, Rasa, and Watson Assistant also use HTTP endpoints for communication and JSON format for the messages.

Rasa is an open-source tool for natural language understanding and dialog management built for developers [12]. Rasa uses the concepts of intent and entity. The cloud chatbot technologies have these two concepts in common with Rasa. Slots and events represent the state, and the state stores the events which led to the current state. The tracker stores the state information. An action has access to the tracker and determines the next step to take. An action can be something simple like an utterance or something complex, like the execution of a function.

Rasa consists of a natural language and a dialog component accessible via an HTTP API [12]. The format of Rasas training data is either JSON or markdown and is chosen by the developer. Rasa supports live training of the system where the developer can correct the bot while communicating with it. Live training is an effective way to create training data for plausible conversations. With live training, there is a higher possibility of creating a complete and natural conversation because the developer is forced to go through the conversation step by step.

Braun et al. [15] compares some NLU frameworks regarding their natural language capabilities. The result of Braun et al. [15] showed that Rasa performs on an equal level with tools like LUIS. The compared frameworks are suitable for task-specific problems. Shawar and Atwell [30], Bordes et al. [13] and Williams et al. [36] focus on end-to-end systems while Braun et al. [15], Dutta [17], Williams et al. [35], Bocklisch et al. [12], Ahmad et al. [11] and 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] give no arhcitecture overview but evaluate the NLU capabilities.

When comparing the frameworks NLU capabilities, the same training data needs to be used for all tested frameworks like done by citetbraunEvaluatingNLU and Gregori [22]. 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 do not build a chatbot with it, nor do they compare frameworks. Dutta [17**?**], and Przegalinska et al. [27] developed actual chatbots for specific use cases. Common chatbot technologies found in the articles and books are Dialogflow, LUIS, Rasa, Watson, and Wit.ai. Many researchers use, mention, and compare those four chatbot technologies, and they are considered state-of-the-art for this thesis.

Chatbots can also be viewed from a psychological perspective that focuses on user

interaction Følstad and Brandtzæg [18]. 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. Many people are already using natural language as the primary input method on the web through mobile messengers and social networks.

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). At platforms like Twitter, machine agents can already be integrated and communicate with human users. The prime example of state-of-the-art conversational interfaces is Google Assistant.

Chatbots are different from classical software interfaces because they use natural language for communication instead of a classical GUI Følstad and Brandtzæg [18]. Currently, developers focus on the design of user interfaces. The design focus for chatbot systems lies in the conversation, which is an entirely different approach than developers are used to at the moment.

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 present around the web. Chatbots all use messenger like interfaces independent from the problem. This means that 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.

Chatbots are natural language interface using text or voice for communication Brandtzaeg and Følstad [14]. They are used to retrieve content or access a service through natural language conversation. Nowadays, people are used to natural language communication because they spend lots of time on messenger platforms. Because of this, chatbot technologies become more and more popular. Brandtzaeg and Følstad [14] focuses on the psychological aspects of chatbot design and gives recommendations independent from the type of chatbot.

It is challenging to design chatbots for open-ended conversations because users can start the conversation in many ways [14]. As an example, person to person communication is like the open-ended chatbot approach. The conversation can develop in any direction, and there is no specific topic given.

It is vital to balance the human and robot aspects of a chatbot to keep the conversation as natural as possible [14]. If the chatbot feels too human, people might ask questions unrelated to the domain. If it is too robot-like, people might complain because the conversation feels unnatural and robotic. 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 in general.

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 about what they cannot do. Three examples for commercial 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. Such conversational interfaces need to be improved based on interactions with humans and their feedback.

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. The impersonal nature of the conversation can be countered with

a high level of message interactivity. Go and Sundar [20] is, in general, about human-like bots and the effects they have on users and compares these bots with current chatbot systems.

There are three approaches to increase the humanness of chatbots [20]. The humanness is increasable through visual cues like human figures, identity cues like a human-associated name, or conversational cues by mimicking the human language. Introducing visual cues and a name to a chatbot is an easy task. Conversational cues are hard to create since they require a deep understanding of the 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 user interaction and expectations change significantly through the user's assumption, as mentioned by [33] and Go and Sundar [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 than when they expect a bot. If the bot is identified as a human, the users automatically expect more from the conversation. The bot needs to be capable of more, or the users will be disappointed. If the bot is identified as a bot, the users expect less from the conversation. The downside is that the conversation is more likely evaluated as robotic or unnatural. This needs to be kept in mind when designing chatbots since false identification as a human can lead to a huge decrease in user satisfaction and acceptance. The user assumption (bot or human) has a high impact on the chatbot evaluation and feedback.

Unlike the other references Følstad and Brandtzæg [18], Brandtzaeg and Følstad [14] and 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 is 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 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 is crucial 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 entirely different from conventional systems from the design perspective. Right now, developers focus on the design of user interfaces (UIs). This approach does not work for chatbots since they all have the same UI requirements independent from the use case. They either use messenger like UIs for text communication or no UI at all through speech interfaces. This means that the design focus of a chatbot is not 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. There are also performance efficiency metrics to check the time behavior of a system with the throughput time. The interoperability of systems can be verified by checking the exchanged information. Usability can be checked through learnability, which determines how easy it is to learn to use the product. Security can be checked by checking for unauthorized access problems. A part of portability is how easy it is to install a system. ISO/IEC 25010 provides evaluation criteria for software systems that can

be used to evaluate chatbot systems.

The researchers in the literature used for this thesis mainly focus on introducing new approaches or comparing different types of chatbots. Some researchers compare chatbots regarding NLP and ML performance. In most cases, the actual frameworks are not compared regarding e.g., offered features, learnability, and the price. Most papers focus on NLP capabilities and ignore other aspects like usability, simplicity, or the comparison of the different frameworks' functionality. Nor is the ISO 25010[6] used, which is a standard for the evaluation of software systems. A major point of this thesis is the recommendation of a framework for future developers and projects in the 3 Banken IT. For this reason, aspects like learnability are considered because they are relevant for developers. Aspects like the costs are essential for companies, but researchers do not address this issue.

Resources like Braun et al. [15] show that the differences between chatbot frameworks are small when it comes to NLU performance. Hence, the impact of the framework on the NLU performance of the system is rather small. The NLU capabilities of the framework are important in general. However, to give a development recommendation, other aspects like the available features and the simplicity of a framework are more important. If it is true that the NLU performance of the chatbots is roughly the same for all frameworks, then a framework cannot be chosen because of the NLU performance. Yet, it is the evaluation found in most papers if there is an evaluation at all.

Common chatbot technologies found in the articles and books are Dialogflow, LUIS, Rasa, Watson, and Wit.ai. Dialogflow, LUIS, Rasa, Watson Assistant are used in this thesis.

Chapter 4

Methodology

4.1 Approach

The first step in the design process of a chatbot is the problem analysis. As mentioned in Følstad and Brandtzæg [18], the design focus of chatbots lies in the conversation. The analysis of the example conversation led to the identification of the intents and entities required for the use-cases. Figure 5.2 shows the example conversation and Table 5.1 the analysis. Figure 5.3 shows the resulting flow-chart for the use-cases. The flow-chart is the blueprint for the chatbot prototype development. This ensured that the dialog structure is the same for all bots making a fair comparison possible.

The problem proofed to be solvable with chatbots based on the problem analysis. As mentioned before, three criteria decide if the problem is suitable or not. The flow-chart clearly shows that both use-cases are solvable through back and forth communication, making the first statement valid. The flow-chart also shows that both use-cases are repetitive data collection tasks. The chatbot needs to identify the type of user message that can either be a sickness notification or vacation request. Then the required data is extracted from the message. For the sickness notification, the required data is a name, and the optional parameter is the return date. The required parameters for the vacation request are the name of the person and a date-span or a start and end date. Hence, the second statement of Singh et al. [32] is also valid since the problem is repetitive. The communication is solely between the user and the bot without the interaction of a third party. This means that the problem is automatable since no interaction with a third party is necessary. To summarize, all three statements are valid for the given problem, and the use-cases are suitable for chatbots.

As mentioned in Chapter 2 Basics, the classification of sentences and extraction of data are the two key features chatbots have to offer.

The given problem is task-oriented and tied to a domain. The three major chatbot approaches are task-oriented, menu-driven, and open-ended. Deshpande et al. [16], Williams et al. [35], Braun et al. [15], Williams et al. [36] use and describe task-oriented approaches. Task-oriented chatbots offer limited, domain-specific functionality. Questions outside of the domain are out of scope and not relevant. The chatbot needs to extract specific data from the sentences, which is an indicator that the problem is task-oriented. The second indicator that the problem is task-oriented is that the chatbot has to collect information from the conversation to achieve a specific task. The chatbot has to be able to answer questions in the domain, which consists of sickness notifications and vacation

requests. It is not required to answer any questions that are unrelated to the given domain. A limited set of operations also fits the requirements. To conclude, the task-oriented approach is a perfect match since the description meets all requirements.

The given problem can be solved theoretically by the menu-driven approach, which is not as suitable as the task-oriented approach in general. The given problem requires a mechanism to input dates and person names in full and partial sentences to provide a natural user experience. The menu-driven approach uses numbers to navigate a user through a menu. The number one could be used for sickness notifications and two for vacation requests. Then the user inputs the name and dates and confirms the information. To summarize, it is possible to use the menu-driven approach, but it would neither be ideal nor pleasant. The flexibility of the menu-driven approach is also limited compared to the task-oriented one, which is a negative trait when further projects are likely.

The third major chatbot type found in literature is the open-ended approach used in Williams et al. [36], Bordes et al. [13], and Rahman et al. [28]. The open-ended approach has no specific goal, is not limited to a domain, and has problems with collecting specific data. The collection of data to achieve a task is crucial to implement the use cases, which is an indicator that this approach is unsuitable. Furthermore, as Bordes et al. [13] mentioned, the open-ended approach is not yet able to perform well enough in task-oriented settings to be considered for this thesis. The analysis showed that both use-cases are task-oriented. This is the second indication that this approach is not the right one.

Williams et al. [36] combined the open-ended and task-oriented approaches by adding state-related information to and open-ended system. The results showed that the hybrid approach works better than the classic open-ended approach, but the comparison with task-oriented technologies is missing. Hence, it is impossible to tell if a hybrid approach makes sense for a task-oriented problem or not. Open-ended approaches usually perform terribly in task-oriented settings, which makes the statement of Williams et al. [36] irrelevant since better than terrible is not precise. To summarize, the task-oriented approach is a perfect fit for the requirements, while the open-ended approach proved unsuitably. The hybrid approach is not used since the comparison with a task-oriented approach was not provided. The menu-driven approach could be used but is not as pleasing as the task-oriented one.

The domain for this thesis is sickness notifications and vacation requests. The prototypes only provide domain-relevant functionality since further functionality is not required. The functionality only covers the two request types sickness and vacation and basic structures like hello, bye, thank you, yes, and no to build the conversations.

Dialogflow, LUIS, Rasa, and Watson Assistant were selected for prototyping. In general, lots of chatbot frameworks are available. As mentioned by Kane [25], the two major categories for chatbot frameworks are cloud-based solutions developed via a website and local standalone applications. The framework selection for this thesis was based on chapter 3 State-of-the-Art and Section 1.5 Prerequisites. Section 1.5 describes the requirements of the company, and Chapter 3 shows the technologies present 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. IBMs Watson Assistant (Rahman et al. [28], Ahmad et al. [11], Godse et al. [21], Gregori [22]) is inevitable because it is on the wish-list of the company. 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 chatbot technologies were Dialogflow, IBM Watson, and Rasa. Additionally, LUIS[35] was added for the NLU comparison. It was the chatbot technology with the best NLU performance in Braun et al. [15]. LUIS has been mentioned or was used by Singh et al. [32], Raj [29], Rahman et al. [28], Dutta [17], and Gregori [22].

All prototypes use the same training data. Figure 5.3 shows the dialog structure, which was implemented with all technologies to ensure a fair comparison. Figure 5.2 shows an example of a conversation of the sickness notification task. The NLU part of all four technologies has also been trained with the same training data to ensure a fair comparison with the same initial state. The data used for training has been listed in Table 4.1, 10.2, 10.1, and 10.3. The same training data also means that all technologies have the same intents, entities, and utterances. The training data for the prototypes has been defined by the author which is a valid approach for prototyping.

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 English

It was impossible to use the same entities since they are different for each technology. It was necessary to create an entity from scratch for Watson Assistant and LUIS (in German) since no predefined person entity was available. In Engish, Dialogflow and LUIS provided all required entities by default. Rasa needed external tools to provide the entities. The missing entities were added to the frameworks to make the NLU performance comparison as fair as possible. The same training data for all technologies approach has been used by Braun et al. [15], Dutta [17] and Gregori [22] and was adopted for this thesis to enable a fair comparison.

The two primary functionalities of chatbots have been tested inside the given domain to answer which chatbot technology is the best to solve the given problem. The primary functions of chatbots are the intent classification and entity extraction. The entity extraction capabilities were tested by Geyer et al. [19]. The intent classification and entity extraction was evaluated in Braun et al. [15] using precision, recall, and f-score values. The precision, recall, and f-score values are the base values for the evaluation of the entity extraction and intent classification capabilities. This approach provides an objective numeric value for a chatbot system's performance and prevents subjective bypass for tests. The intent classification data used for all technologies has been listed in Table 7.10, 7.11, 10.4 and 10.5. The entity extraction test data has been listed in Table 7.5, 7.6, 7.7, 7.8, and Table 7.9. The test evaluation focuses solely on domain-specific information since the primary purpose of this thesis is a proof of concept through prototyping.

The result of Braun et al. [15] showed that the domain had small to no influence on the performance of the tested technologies. If the results can be trusted, good technologies will perform excellently no matter if the environment is domain-specific or not. A lousy technology will perform poorly in any setting. As Braun et al. [15] said, different technologies should be tested with domain-specific data to determine which is the best fit for the given problem. The domain-specific performance is the relevant performance since it has a big influence on the product. A chatbot technology that performs great in general but poorly in the given domain is of no value. All in all, The best technology is used to implement the actual product and not the technology that performs best in general domain-independent cases. The prototypes of this thesis were implemented and tested solely with domain-specific training and test data.

As mentioned by Shawar and Atwell [30], user studies are another common approach to test chatbot systems. They are recommended for customer-facing bots to increase the quality of the product and gather data about the human bot interaction. User-studies are not part of this thesis. A proof of concept through prototyping user studies are not required. Furthermore, a user study does not provide any information about the development perspective, which is a vital part of the thesis.

The psychological aspects mentioned by Følstad and Brandtzæg [18], Brandtzaeg and Følstad [14] and Go and Sundar [20] have been partially applied. The general rule that a chatbot should tell the user that he is talking to a bot was applied. For further projects, the humanness of bots is an important factor and should be evaluated and implemented. The humanness is an exciting topic for user-studies and can significantly increase user experience and acceptance. Standard approaches are giving the bot a human name, gender, and face. Amazons Alexa is one example of a bot with a human name, gender, and a voice. However, it is not relevant for the proof of concept because there are no user studies in this thesis.

4.2 Comparison and Evaluation

A major part of this thesis has focused on the evaluation and comparison of the developed prototypes. The training and test data used for the comparison are domain-specific. The evaluation of the entity extraction and intent classification capabilities of the framework relies on the provided data sets.

It was impossible to test the entity extraction capabilities of the frameworks unbiassed. The person entity is unavailable for Watson Assistant, and LUIS had none for the German language. The bias came from the creation of the entities since the training data quality is of different quality compared to the predefined ones. An entity created by the developer might overfit or suffer from a small amount of training data. This is a bias in the test setting since the performance is dependent on the training and test data provided by the developer and not on the technology alone. With Watson Assistant, it was not possible to mark entities in the training sentences for the German language. This rendered Watson Assistant useless for the German input language. Watson Assistant provides no date span entity. Two separate date entities were used instead to get the same result.

The Rasa framework does not offer any predefined entities itself and relies on other optional technologies like Spacy and Duckling. Spacy offers date and date-span entities, but the result date is not in a standardized format. The resulting dates are not machine-processable. The automation criterion of chatbots requires dates in a standardized form. To summarize, the date and date span entities of spacy were unsuitable. This lead to the

use of Duckling since it provided the necessary standardization for date and date-span entities out of the box. Dialogflow offered two entities that can handle date-spans. Both entities were tested to get the best result possible.

The entity extraction evaluation uses the recall, precision, and f-score values for the comparison and the ranking, as done by Braun et al. [15]. The result of each test case is either TP, FP, FN, or TN. The entities defined by the problem analysis of the user-cases are the person, date, and date-span entity. The entity extraction test only included these three entities to get a result relevant to the use-cases. Furthermore, the calculation in groups showed the performance for specific entities and intents. This allowed a detailed performance comparison for the three relevant entities. The combined score across all entities determined the best technology. As mentioned before, the framework comparison has been difficult since some entities are not available using certain technologies. In the case of Dialogflow, two date-span entities are available, which increased the testing overhead. Table 7.3 shows the supported entites next to the technology to get an overview.

The second primary function of chatbots is the intent classification. The test setup used the same training and test data for each technology to provide a fair environment. The intent classification evaluation uses the same approach as the entity extraction evaluation. The performance measurement relies on precision, recall, and f-score values. As mentioned in Chapter 2 Basics, the NLU services of all four frameworks provide a confidence value. The average confidence score is another performance quality metric. The confidence value is a response that represents how well the input fits an intent. Hence, it is logical to include the confidence score in the evaluation. It indicates how good the training worked for the frameworks and also if additional training data would have been necessary. Furthermore, the standard deviation shows how consistent the chatbots' confidence scores were. As a footnote, a chatbot always returns a confidence score for each intent. For each test, the correct intent's confidence score was listed no matter if it was identified correctly or not. This allowed the calculation of the average standard deviation across an intent and as a whole. A high average confidence score combined with a high f-score indicates that the chatbot classifies intents reliably and vice versa. For the use-cases, English and German are potentially interesting as input language. Hence, the intent classification tests include both. The expectation for the German tests was that there is no significant difference to the English tests. The performance expectation was that the German bot is on the same level as the English bot regarding entity extraction and intent classification. Contrary to the expectation, each technology showed significant differences. The intent performance increased noticeably while the entity extraction performance dropped for Rasa, Dialogflow, and Watson Assistant. LUIS had no person entity for the German input language. Dialogflow disappointed in the entity extraction department. This came as a surprise as the entities are not that different for the two languages.

As Singh et al. [32] mentioned, the best evaluation criterion is if the underlying task can be achieved or not. This evaluation criterion has been adopted for this thesis since it is the most relevant for a technology recommendation. As a result, value for this evaluation, a binary boolean criterion was chosen. Either it was possible to build a prototype that implemented the design, or it was impossible. The evaluation did not include an opinion or how hard the implementation was. LUIS is an NLU technology and was excluded from this test because it offers no dialog handling mechanism. NLU technologies do not provide dialog handling in general. The Microsoft Bot Framework uses LUIS and offers dialog handling but was not used in this thesis and is therefore not part of the evaluation. As expected, the implementation was possible with all three chatbot technologies.

For future projects, speech recognition capabilities are on the wish-list of the 3 Banken IT. Thus, the costs of the speech-to-text and text-to-speech services were compared based on the price. These services enable voice interaction between man and machine. All chatbot providers should provide such services to extend the functionality of the bot. Rasa is the only tested framework that does not offer such services. It is possible to use the speech services of other providers. The cloud providers offer speech services at comparable prices.

A key factor for the use of chatbots is the price. That the price plays an important role was mentioned in Raj [29], for instance. The literature found and used for this thesis did not compare the prices of the available technologies. The goal of the price comparison was to find out which technologies are cheap and which are expensive. The goal was to find a cheap technology that performs well. The costs of a chatbot have been calculated for different categories since the prices listed online were intransparent and hard to compare. The first category was the price for the chatbot technologies, the second one is the price for hosting, and the third category was the costs for speech-to-text and text-to-speech services. The websites of Rasa and IBMs Watson Assistant do not list the prices for the enterprise edition. It is calculated individually for each customer. To build a chatbot with Microsoft technologies LUIS and the Microsoft Bot Framework are combined. The added price for both builds the actual price for the comparison. IBM calculates the price for Watson Assistant per user per month. Dialogflow, LUIS, and the Microsoft Bot Framework calculate the price based on the number of requests while Rasa is free by default. The prices for the text-to-speech and speech-to-text services were comparable for Watson Assistant, Microsoft Speech Services, and Dialogflow. Rasa did not offer text-to-speech and speech-to-text services at all. The general expectation was that the frameworks cost roughly the same, but the prices for the chatbots were hard to compare or not comparable at all. The speech services cost roughly the same. All in all, there is no exceptionally cheap or expensive technology.

Another evaluation criterion used for this thesis is the portability of software systems, as defined in the ISO 25010[6] standard. The portability is interesting for a comparison of cloud frameworks with a local chatbot technology like Rasa. For the development of a chatbot, portability can play a crucial role.

The only tested technology which is portable was Rasa. The portability of the development environment of the local technology Rasa in contrast to the cloud chatbot technologies was the main focus of the first portability evaluation. The number of providers that allow a deployment was the second factor for the portability evaluation. The main focus lay on cloud technologies since the expectation was that they can only run in the environment of the provider and are not portable at all. The third portability category chosen was how flexible the development of chatbots is regarding the devices. The cloud providers use simple web interfaces. No special setup and no additional tools are required to start the development of a chatbot. The local technology required a development environment and is inflexible in comparison.

Another aspect of portability is the migration from one technology to another. The migration from one technology to another is hard or impossible, which meets the expectation. Every technology uses the JSON format to store data, which seems perfect at first. The information of every bot is stored differently, making it impossible to copy the information from one bot to another. To summarize, this makes it impossible to migrate from one framework to another in a fast and straightforward way since the format of the data is incompatible.

The project setup complexity plays a crucial role in the development of a chatbot. The main focus of this test was Rasa since the setup complexity was interesting for the local technology. That Rasa is at a disadvantage in this category was clear from the start. Cloud chatbots require no setup at all since the provider takes care of everything. The real question is how much harder is the local setup. It was harder to set everything up for a Rasa project. With Docker, the setup of Rasa was easy and fast.

The deployment of chatbot systems is an important aspect of portability for local and cloud technologies. Cloud chatbot technologies should not produce any deployment overhead at all. They are hosted in the cloud by default, which makes a setup unnecessary. For Rasa, the opposite was true in this category. No cloud deployment is available by default, and the developer has to take care of it if required.

This may seem like a grand disadvantage at first glance, but thanks to the Docker compatibility Rasa come with some huge advantages that cloud chatbots cannot offer. Rasa can run in a Docker container. This means it can run almost everywhere, while the cloud technologies are limited to the providers' cloud. The deployment in cloud environments, on local machines, and internal servers is possible.

The ISO 25010[6] defines learnability as a sub-criterion of usability as a measurement metric for software systems. The learnability is important from a development perspective since a technology that is easy to learn and use is the optimal case. The performance is still important and needs to be high nonetheless. Evaluating the project setup and deployment complexity was also part of the learnability evaluation since it plays a role in the development process.

The predefines entities can save lots of development time and are often of high quality. As mentioned before, the chatbot technologies use entity extraction to extract data from text. The creation of entities is a part of the development process of a chatbot, and it takes time to define good entities. All of the chatbot technologies offer predefined entities that are ready to use. As a criterion, the number of predefined entities was defined as a measure since lots of predefined entities can save lots of development time for future projects. If a required entity is available for a given problem, the developer does not need to create one. The second measurement criterion of the entity category is if the chatbot technology predefined all required entities for the use-cases of this thesis. The technologies where all required entities were predefined are better suitable to solve the given problem. Another aspect was how hard the creation of new intents, entities, training phrases, and dialogs was. The expectation was that it is equally complicated for each technology. This proofed to be the case for cloud technologies.

As mentioned by Følstad and Brandtzæg [18], the design of chatbots is all about the conversation. Hence, the most important aspect was the creation of dialogs. The frameworks Dialogflow and Watson Assistant offered dialog nodes for the creation and structuring of conversations. A dialog node is one step of the conversation and optionally followed by other dialog nodes. A dialog node was the classification of the sickness notification intent followed by the collection of the name and return date and the resulting response of the system, for instance. Rasa offered stories to represent a dialog. A story represents a whole conversation path from start to finish. As an example, a story starts with a greeting phrase and ends with a goodbye message.

Rasa's stories were adjustable fine granular compared to the dialog node approach of Dialogflow and Watson Assistant. A story can execute multiple actions and display multiple responses in a row. Watson Assistant cannot execute multiple actions but can display multiple messages. Dialogflow cannot execute multiple actions nor display multiple re-

sponses.

The extraction of form-data is a required feature for this thesis. It was necessary to extract such data from the conversation. The main focus of the evaluation was the complexity of the form-data extraction. The cloud bots provided the required extraction mechanisms out of the box. Rasa, on the other hand, was the only technology tested where programming skills were necessary to extract form data from the conversation. To conclude, the chatbots were easy to use in this category, while Rasa was not.

Rasa's action server is more complicated to implement but also has advantages over the form-data approach of the cloud bots. Programming skills were required to implement form-data extraction with the action server, while the cloud bots do not require any programming skills at all. This means that the advantages of programming languages are available at the action server. Data validation, the introduction of logic, and calling external services is easy in programming languages. The two approaches are quite different since the focus appears to be different. The form-data extraction of the cloud bots is easy to use and very simple, while Rasa is more complicated but offers the advantages of a programming language.

Customization and fine-tuning of chatbot systems might be important for developers when the uses-cases have very specific requirements or need fine-tuning. Rasa is the only technology that allows adjustments in the frameworks pipeline. This allows a developer to replace components or add custom components to the pipeline and makes fine-tuning possible. The pipeline of the other tested technologies is not even visible for developers. Adjustments of Rasa's pipeline were necessary to get the required entities of external tools into the application. Duckling and Spacy were added to the pipeline to get the three required entities as predefined entities. Something like that is impossible with the other frameworks since the pipeline cannot be adjusted at all. The pipeline also allows detailed adjustments of ML parameters, which is not the case for the cloud bots. Developers can fine-tune the application by adjusting ML parameters, adding additional components, or replacing components.

As discussed in Chapter 2 Basics, the concepts used by the frameworks are interesting. The base concepts intent, entity, and utterance are present in literature, and the expectation was that these concepts are also present in the frameworks. This proofed to be the case for all four technologies.

Another important aspect is the communication with the chatbot services. In general, JSON is the request and response format for all tested technologies. All the technologies offered a REST endpoint for communication. The bots send different message content to external services. LUIS and Dialogflow sent metadata while Watson Assistant and Rasa did not. The evaluation focused on these two approaches since they are quite different and have positive and negative aspects. The meta-information is valuable for the creation of statistics and performance tracking, for instance. The other approach is simpler and has no predefined structure. Hence, the developers can choose the format freely without any restrictions. It is also possible to add the required meta-data to the chatbot message.

External services can send additional information to chatbots through the response messages. The question was, how easy is the processing of the additional information using the chatbot framework. The expectation was that it is easies with Rasa since the action server handles the response. As a reminder, the action server uses Python, and it is easy to process data using a programming language.

Another big difference between Rasa and the other technologies were the training capabilities. Live training is offered by Rasa while all the other technologies were solely

trained with the training data defined by the developer. With live training, the created stories should be closer to a real-life scenario because the developer has to go through the conversation step by step to create the story. This can prevent unnatural feeling conversations, and it is also impossible to forget a step of the conversation.

In the ISO 25012[6] defines the user interface aesthetics under the section usability. Hence the user interfaces of the frameworks were also compared regarding simplicity. The evaluation of the UI is subjective and contains the author's opinion and objective, measurable values where possible. The interface of a framework was evaluated as easy to use when the overall structuring is easy to understand, and the prototype was easy to implement.

4.3 Course of Action

The first step was the problem analyzation and the creation of the flow-charts for the dialogs. Based on the flow-chart, the required intents and entities were defined. The required intents are sickness notification and vacation request. For each use-case, one intent was necessary. The conversation built in the flow-chart made the required entities visible. The sickness notification requires the name of a person and an optional return date. The vacation request needed the name of the person and a date-span. To summarize, the resulting entities were person, date, and date-span. For each intent, ten training sentences in English and ten training sentences in German were created. Additionally, test sentences for the validation of the result were created. The entity extraction was tested with sentence fragments.

The prototypes used the training sentences for the creation of the intents. The result of each test is either TP, FP, TN, or FN. Based on these values, the precision, recall, and f-score values were calculated to provide an objective evaluation. The same approach was used to evaluate the intent classification. The confidence and standard deviation values were added for the intent classification to provide more evaluation factors. For each test category, a summary compared the performance of the technology in the given category. The frameworks evaluation is based on the criteria listed in Section 4.2. The ranking of the frameworks uses the results in the categories to show strengths and weaknesses. This made a final recommendation for further projects possible. Furthermore, the entity extraction and intent classification result in English and German showed which technology performed best in the given domain and language.

Chapter 5

Design

5.1 Architecture

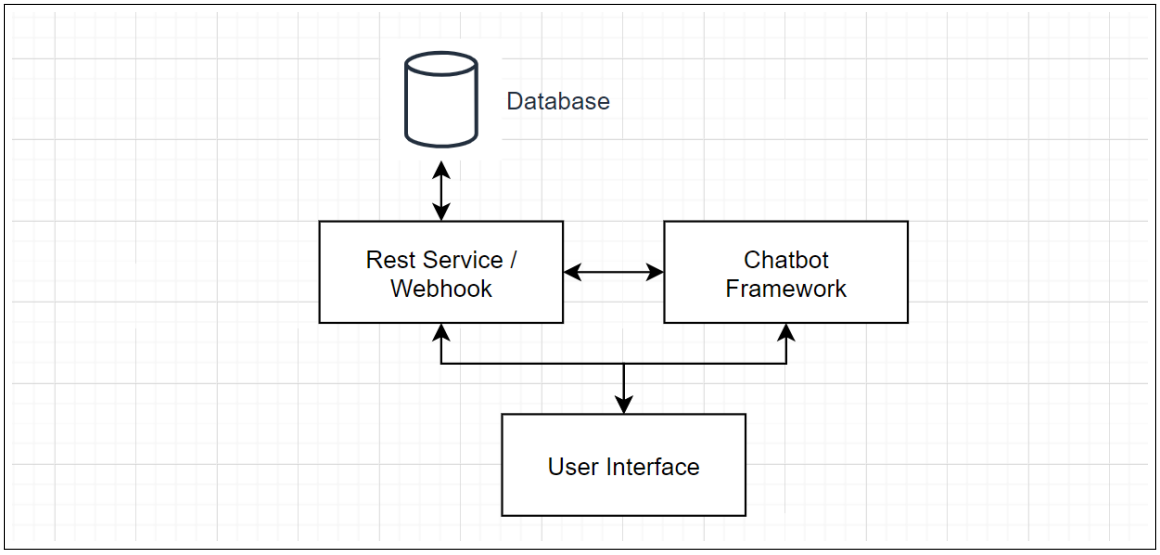

Figure 5.1: Minimalistic Chatbot Architecture

Figure 5.1 shows one possible architecture for a minimalistic chatbot system with database communication. 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. 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 is used to store and retrieve the relevant data. The extracted entities can be validated using the webhook. Furthermore, the webhook is called by the chatbot frameworks when the entity extraction is finished and handles the insertion and retrieval of data from the DB. The dialog flow can is adjustable at the webhook in error cases, for instance.

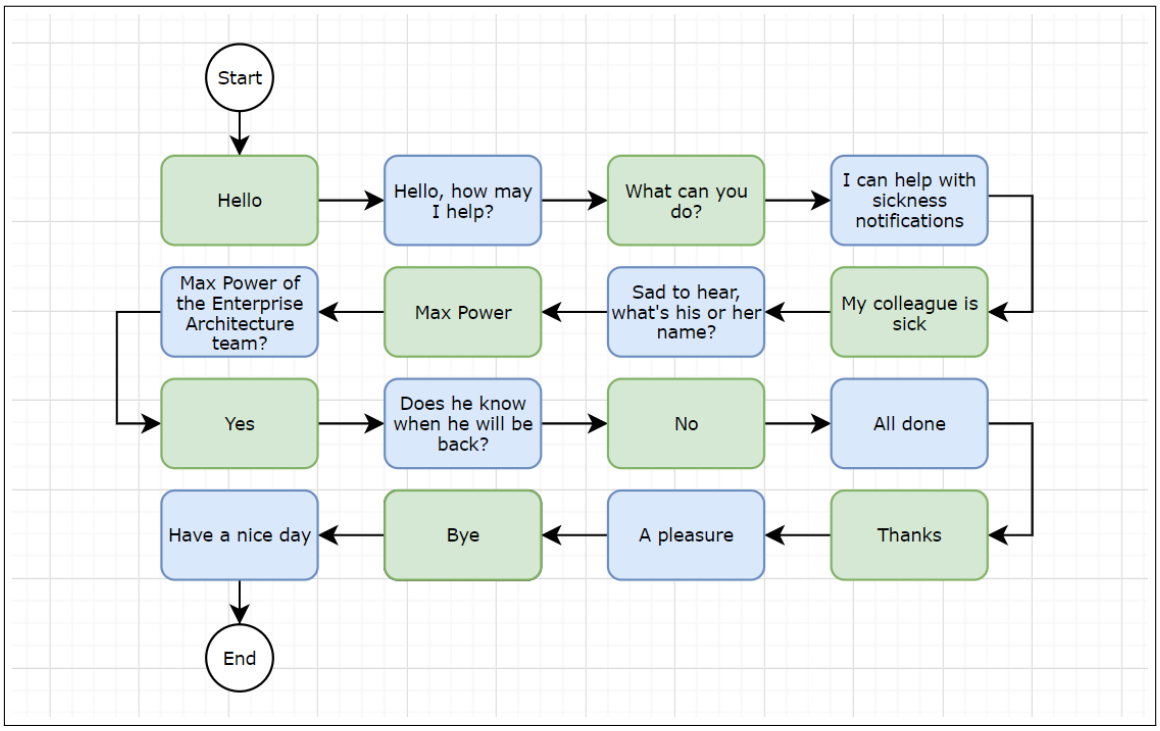

Figure 5.2: Sickness Notification Example Conversation

5.2 Story/Flowchart

As mentioned in Følstad and Brandtzæg [18], the design focus for chatbot systems lies in the conversation instead of the GUI. A story is an example of a dialog/conversation between user and chatbot. Figure 5.2 shows an example dialog to report a colleague sick. The user input is green, and the chatbot response is highlighted blue. The knowledge base for the domain is created by finding the intent of each user input. The intent for the user input ”Hello” can be marked as greeting. The second input is some general information about 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. In this case, the interesting entities 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 the return date is often unknown.

Table 5.2 lists the resulting intents, entities, and utterances for the example sentences of the conversation shown in Figure 5.2. Figure 5.3 shows the detailed flowcharts for the sickness notification and vacation request. The flowcharts are the base for the implementation section since it is the blueprint for all prototypes. User inputs are highlighted red, bot responses are green, and general actions are marked purple. Both use cases start with the identification of the person. If the entered name cannot be validated, the user can correct it 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 like before. Afterward, a date needs to be entered in both cases. The sickness intent asks for an optional return. The date is optional since the yes, and no paths 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 are present, 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. 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. The blueprint is valid for tools using intents, entities, and utterances.

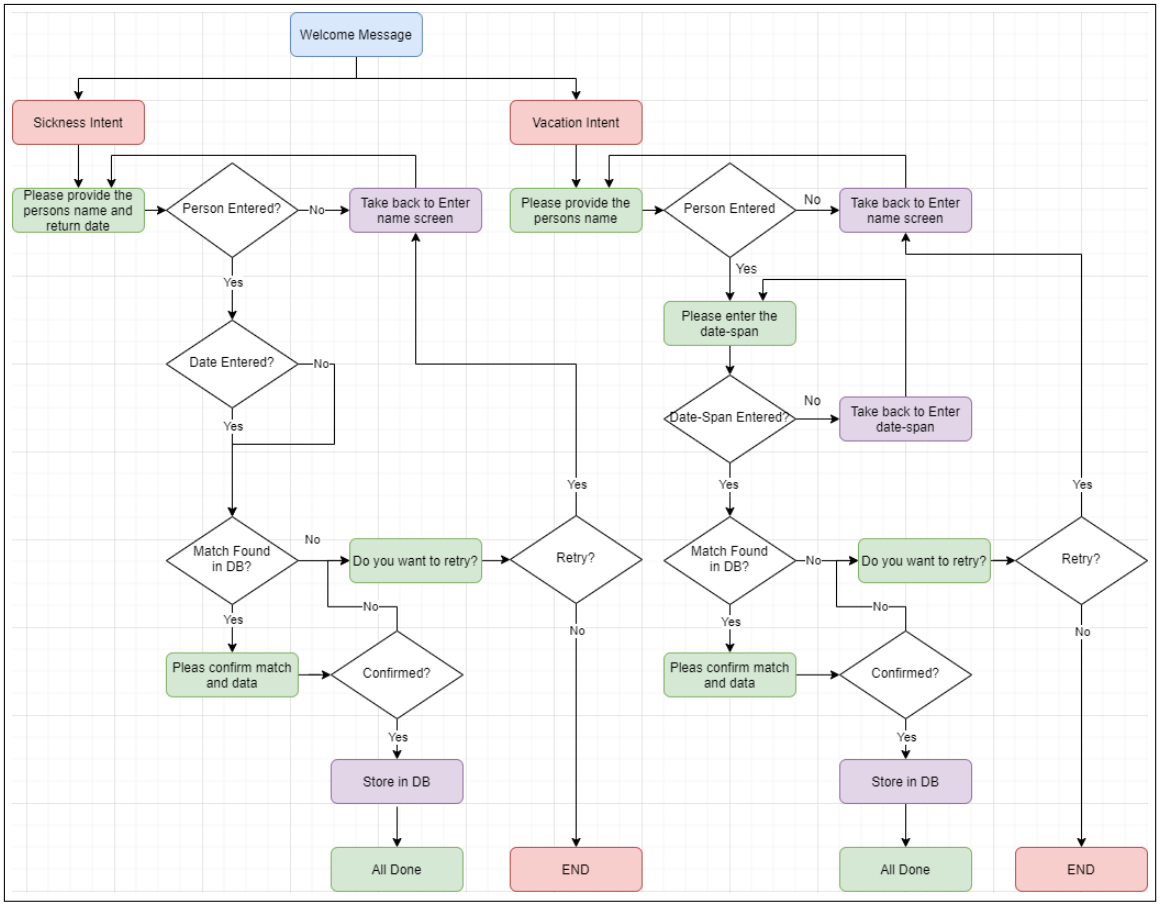

Figure 5.3: Chatbot Conversation Flowchart

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. In this thesis, the chatbot is the crucial part, and the backend is mocked for simplicity. The identified information from the conversation is displayed in Table 5.2. In general, the greeting and goodbye intents are default in almost any chatbot conversation. A ”thank you” intent is also something common. The user should have the option to ask for the available options. The core intents of the whole process are the sickness notification and vacation request intents. For the use-cases, the needed entities are person, date, and date-span. 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. Table 5.2 lists the basic concepts extracted from Figure 5.2. There are the intents discussed above, the entities and their types, and the bot actions.

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 utter-

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

ance. 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. The only change in the process is that there needs to be a mandatory start and end date.

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

5.4 Psychological Aspects

Psychological aspects paly a vital role in the development process of chatbots Brandtzaeg and Følstad [14]. The user's impression and feedback depend significantly on the mindset. If the user thinks he is talking to a bot, the user expects less, but the conversation is more likely evaluated as unnatural. If the user assumes he is talking to a human and the bot is unable to achieve the task, the user is very disappointed Hence, the user should be informed that he is talking to a bot to prevent wrong assumptions.

There are three ways to adjust the humanness of a chatbot. Two options are easy to implement, and one is hard. A human name increased the humanness of a bot signifi-

cantly. Examples for bots with human names are Cortana and Alexa. The second way is to introduce a gender which is also part of Alexa and Cortana. The hard way is to model human conversations in detail. An important factor for the humanness of a bot is the message response time. When chatting with a friend, the response happens not immediately. This concept can also be used for chatbots since a small response delay already has a significant impact on the fell of the conversation. In general, if the response is too fast, it feels unnatural.

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 an 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. Additionally, the frameworks can optionally call the endpoint after each recognized intent 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. Figure 10.4 shoes the messenger UI developed for this thesis. A user can enter, send, and view messages. Message cards show the sender, time, and message and are colored differently for the user and the bot. A user can enter a new message at the bottom input 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 10.4.

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. The input language and a name for the agent are required to create a new agent. The selected language is the interaction language of the users, not the development language. Dialogflow offers the option to create a mega agent by combining multiple agents. Figure 6.1 shows the web interface with the menu entries titled intents and entities. Dialogflow uses the basic concepts of intents, entities, and utterances. The implementation uses the intents, entities, and utterances listed in Table 5.1.

6.1.1 Intents and Entities

An intent in Dialogflow has training phrases, actions, parameters, a response list, a context, and the fulfillment. As described in the basics, the training phrases are a list of utterances a user would enter and belongs to an intent. The parameters are the slots to fill in a conversation and have an entity as type. The parameters for the sickness intent are the rows of Table 2.2, for instance. The response list is a list of utterances used to respond to the user's question at the end of an intent. The context passes parameters from one intent to the next.

By default, an intent is handled by Dialogflow without communication to another service. The use-cases require communication with an external service for data processing. The fulfillment section offers the concept of webhooks for this reason. When enabled, the webhook can handle all kinds of requests like entity extraction or processing of the extracted information after the form-data collection. For the sickness notification use-case, the webhook processes the name of a person, and an optional return date after the chatbot collected the data. The webhook functionality covers all data submission requirements for the use-cases.

Dialogflow provides many predefined entities and intents. Dialogflow predefines the date and person entities, and it is 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. With the concept of follow-up intents, the structuring of intents is possible. One or multiple intents can follow after an intent, hence the name. The final structure is a sickness intent which is followed by either the yes or the no intent. 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. Both conversation paths end the conversation, which

is related to data collection and processing. They are optionally followed by thank you, goodbye, or chit-chat, but this is independent of data collection and storage.

The process looks the same for the vacation use case. The vacation intent needs to be created and has the follow-up intents yes, and no for the input confirmation. The entities required are person and date-span. Dialogflow offers two predefined entities for date-span handling. The first entity is called date-period, and the second one is called date-time. The advanced features of these entities show why predefined entities are better than custom entities for standard problems. Both date entities parse dates like 06.27.2020 correctly. They are also capable of processing weekdays (Monday) and periods like Monday to Friday and tomorrow correctly. It is possible to implement such behavior for a custom entity but is additional work They also convert dates to the correct values. If Monday is in the past for the span Monday to Friday, the resulting date period will be from the future Monday to Friday after next Monday. The provided standard functionality for dates and date-spans is very convenient and user friendly.

When the slots are part of the training sentences, experienced users can shorten the workflow. Usually, the user enters the required information slot by slot. The user is not queried for a slot when the information is already present. With these concepts, sending all the information with the first request is possible. This speeds up the process for experienced users. In a sentence like ”Max Power is sick and will return Monday” all the required information is present, and the request can be processed immediately.

In general, custom entities are a list of entries or a list of entries with synonyms. An advanced feature of Dialogflow is small-talk. It can be enabled in the settings with a single click. When small-talk is enabled, the bot responds to requests like ”What are you”, ”Bye”, and ”sorry”. This is another useful feature of Dialogflow, which is predefined and requires no development effort.

6.1.2 Webhook

The Dialogflow webhook requires an HTTP POST URL as an entry point. The communication relies on JSON as a format. Listing 6.1 shows the entry point of the webhook. A request (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. Listing 6.3 shoes a skeleton for intent handling for the success and error scenario of 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
elif intent == 'Vacation':
    ...
```

Listing 6.3: Dialogflow Intent Handling

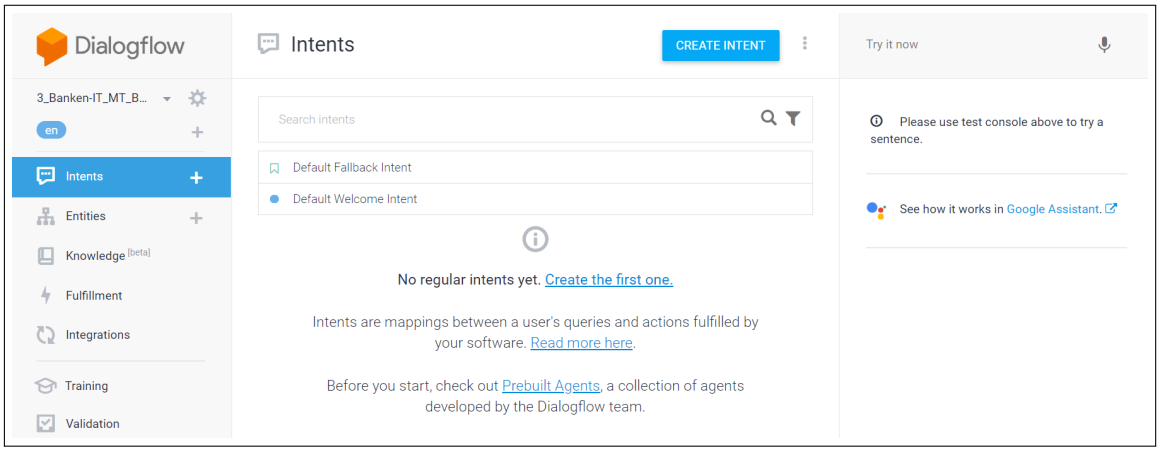

Figure 6.1: Dialogflow Web Interface

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. Figure 6.2 shows the web interface of Watson Assistant. The interface offers the concepts of intent, entity, and dialog. A dialog node represents the actions when an intent is recognized. The structuring of dialogs works the same way as with Dialogflow.

6.2.1 Intents and Entities

The required entities are person and date. Watson Assistant offers system entities for persons and dates. Watson uses the same concept as Dialogflow for communication with external services. After entering a webhook URL in settings, the endpoint can be used in the application.

The person entity is marked deprecated and is available in English but not in German. It is possible to create new entities by using examples or regular expressions. IBM recommends the creation of a custom person entity with entity annotations in the test sentences to eliminate the deprecated person entity. Every occurrence of person names in the test

sentences needs to be highlighted and marked (annotated) as a person entity. The sickness use-case requires a new intent. Then the training phrases are added to the intent.

A condition triggers a dialog node. For the sickness use-case, the dialog node triggers every time the bot recognizes a sickness intent. The settings menu of a dialog node offers customization options. Customization options are slot filling and calling a webhook, for instance. The sickness-notification use-case requires both options. The required slots for the form filling are a person of type person and the return date of type date. The person is mandatory, and the date is optional. The webhook receives a request at the defined URL after the form-data collection.

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 the return date are the information stored in the backend for a successful request. Listing 6.4 shows that the requests of Watson Assistant contain no meta-information by default. The developer defines the information and format submitted to the webhook. The webhook returns a JSON object. A context variable stores the JSON object. It is accessible in the chatbot code like a regular struct in C++.

Based on the webhook response, the dialog can go in different ways. It is possible to define a response for each possible response. If the request only contained the name of a person, only the name of the person is present in the response message. If the request contained a name and a date, both are present in the chatbots response message. Whenever the webhook finds a person in the backend, the user needs to confirm that the entered information is correct. If the person was not found the retry message is displayed Watson Assistant offers conditional statements to handle such cases.

Table 6.1 shows the conditions, responses, and actions of 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 is the next thing to implement. No system entity or dialog for the confirmation logic is present. Two intents or an entity are options to represent the confirmation. An entity with two possible values (true, false) is a logical and straightforward approach. 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 is selected based on the value. The webhook response shown in Table 6.1 used similar logic. The confirm or abort logic requires an additional dialog node. The jump option in the settings of the dialog node allows a dialog node to be triggered based on the value of the variable. For the confirm or abort dialog requires form-data and a webhook call after the confirmation. Both options were enabled in the setting menu, like for the other nodes. When the user aborts the operation, the webhook is not called. The confirmation slot is of type confirmation entity and is required. The webhook receives a request with the collected information. If everything went fine, an all done message is displayed. If something went wrong, an error message is displayed instead. This ends the sickness dialog.

The vacation use-case is quite similar in structure. The vacation use-case requires a date-span entity. Watson Assistant offers no date-span entities, but it is possible to use two separate dates instead. It is impossible to create compound entities with Watson.

A compound entity cloud store two dates. The options are to create an intent with two dates in it or a dialog node which fetches the required dates as form-data. This thesis uses the dialog node approach 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 is created with the example sentences. The dialog needs a mandatory person slot and two mandatory date slots. The date slots are 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. The confirmation dialog works the same way for both use-cases. 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 implemented.

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

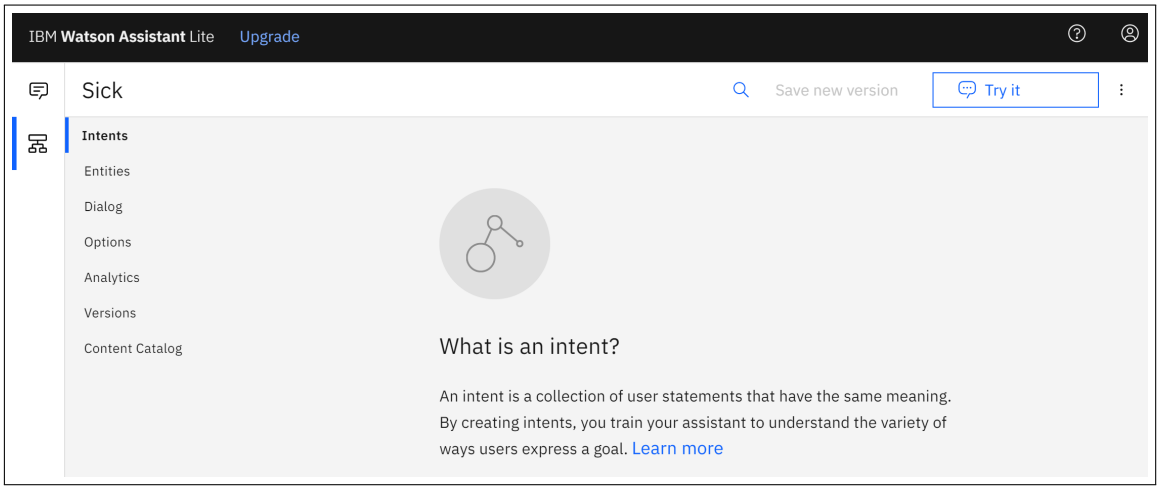

Figure 6.2: Watson Assistant Web Interface

6.3 Rasa

Rasa[8] is an open-source framework for NLU and chatbot implementation. Rasa runs inside a Docker container or in a local Python environment. Rasa uses Python as a de-

velopment language and can be installed using pip. Listing 6.5 shows a Docker compose file for Rasa, and the action server. Rasa offers a standalone NLU application, the action server, and chatbot development.

Rasa consists of two parts for chatbot development. The natural language part takes care of everything related to the conversation like intents, entities, the story, and the conversation flow. The second part handles the slot filling and other actions. For each part, a Docker image can be created.

The Docker compose file of 6.5 can be used similarly for the Azure deployment in combination with the images. Azure also offers a Docker registry to upload and pull custom images. It is important to choose a compatible version of rasa and rasa-SDK. The Rasa image is responsible for the NLU part and the Rasa sdk runs 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 scattered across multiple files. The nlu.md file stores the intents with the utterances used for training. Listing 6.6 shows the format for intents.

```
## 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 from start to end. Listing 6.7 shows the sickness story.

```
## 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

The actions.py file defines all the actions which are callable in the chatbot files. The sickness form slot filling is one action, for instance. The domain.yml file lists all intents, entities, and actions. If they are not listed in the domain.yml file, they cannot be used in the application. 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 from the list of responses. 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. However, spacy does offer the required date and person entities for the use cases. Duckling is another option for data and date-span extraction. In the config file shown in Listing 6.8, the pipeline includes the spacy entity extractor and Duckling to enable the use of predefined entities. spa [9] lists the entities supported by Spacy 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 gain access to 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. It recognizes strings like "Monday" as dates and converts them correctly. Duckling works similar to the date and date-span entities of Watson Assistant and Dialogflow. The person entity of Spacy works as expected. The used entities need to be added under entities in the domain.yml file. If they are not listed in the domain.yml file, they cannot be used in the application. People with machine learning experience can define custom pipelines or adjust parameters of components. Listing 10.1 shows the detailed pipeline used for this thesis.

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. The command "rasa train" trains Rasa with the current data. The shell command starts an interactive shell to test the bot. The action server runs separately from the NLU service. After the installation of Rasa-X, the command "rasa x" starts a web UI for the development of the chatbot. Figure 6.3 shows the UI of Rasa-X. The web interface offers the files discussed in the training section (left) with additional features like the conversation graph (right) called flow for a story (middle) shown in Figure 6.3.

Rasa X offers interactive training, which allows developers to build chatbots through live interaction with the bot. Rasa stores a model every time it is trained. This means the training history is available and old bots can be loaded. If a feature does not work out, just load the old model.

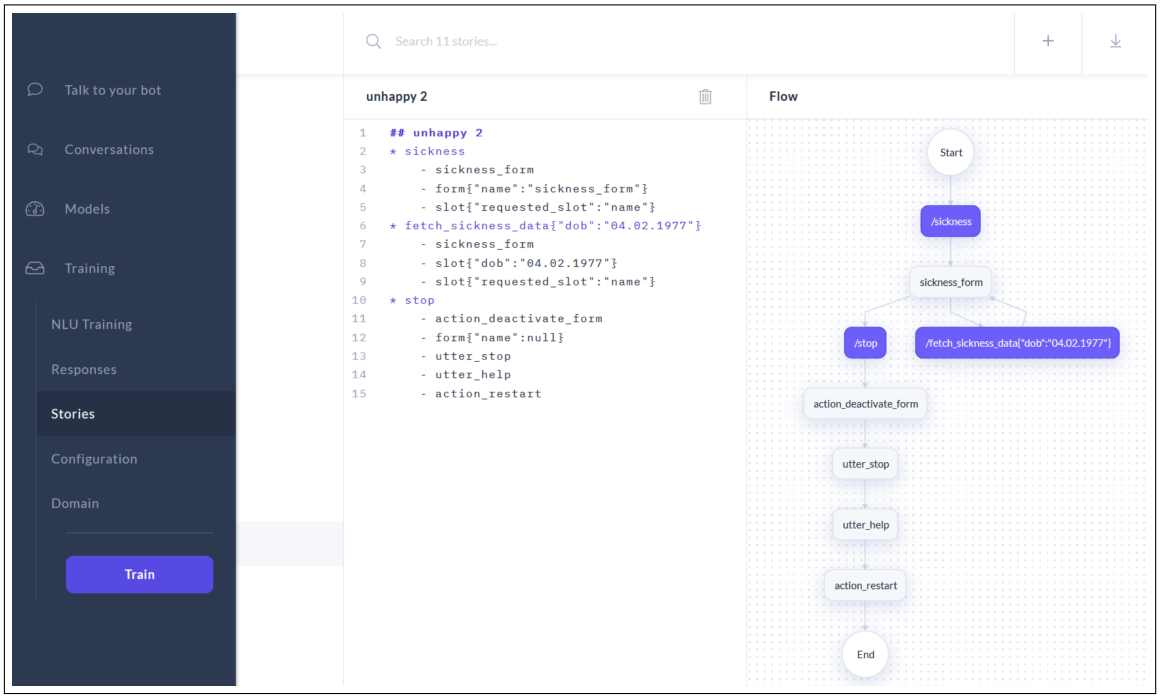

Figure 6.3: Rasa X Interface

For the 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 asks for a slot. Slot filling requires an action. An example slot filling 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. The bot does not ask for unrequired slots. 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. Each slot allows the definition of custom validation rules. This way, the validation of person names is possible at the action server. 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 shown in Listing 6.6. Square brackets mark entities and parentheses mark the types. Now the sickness story needs to be defined. Phrases trigger the slot filling mechanism when the correct intent is recognized. The next step is the confirmation dialog, where the user confirms the entered information.

A submit sickness data action is required. The submit action requires the ”affirm” and ”deny” intents. When the affirm intent is recognized, the confirmation action is triggered. In the run method of the submit action, the data can be validated and processed. The deny conversation path clears the slots and displays the retry message. Listing6.7 shows the affirm conversation path of the sickness use-case. 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. Both use-cases use the same slots and types. There is absolutely no difference between the use-cases regarding the entities. The response messages are different, however.

For the vacation use-case, a new intent needs to be added to the domain file. The vacation training phrases need to be listed in the NLU file. The vacation intent requires a form and a submit action. The structure of the form and submit actions is the same as for the sickness use-case. The actions need to be listed in the domain file to be usable in the stories. The response messages and conditions are adjusted as needed. Then the new stories need to be created. With Rasa-X, the stories are created through live interaction with the bot. The structure of the vacation story is the same as for the sickness story.

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 is impossible 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. A new NLU project requires a name for the service and an input language. The chosen language is the language the users will use for communication with the LUIS application. The build tab provides the interface to modify and build an NLU service with LUIS. The interface shown in Figure 6.4 offers intents and entities like described in Chapter 2. All four tested technologies have intents and entities and use utterances for training. Marking of entities in training sentences is also possible with LUIS.

Table 5.1 and 5.1 list the required intents and entities. Figure 5.3 does not influence LUIS because the conversation handling is not part of the NLU service. Only a name and a list of training phrases (utterances) are necessary to create an intent with LUIS.

Multiple options are available to create different kinds of entities. There are ”simple” entities that describe a single concept like a city, for instance. There are composite entities that combine multiple entities (parts). A ticket with an id, event name, the price, and the number of tickets is an example of a composite entity. For a list of valid entries, entity

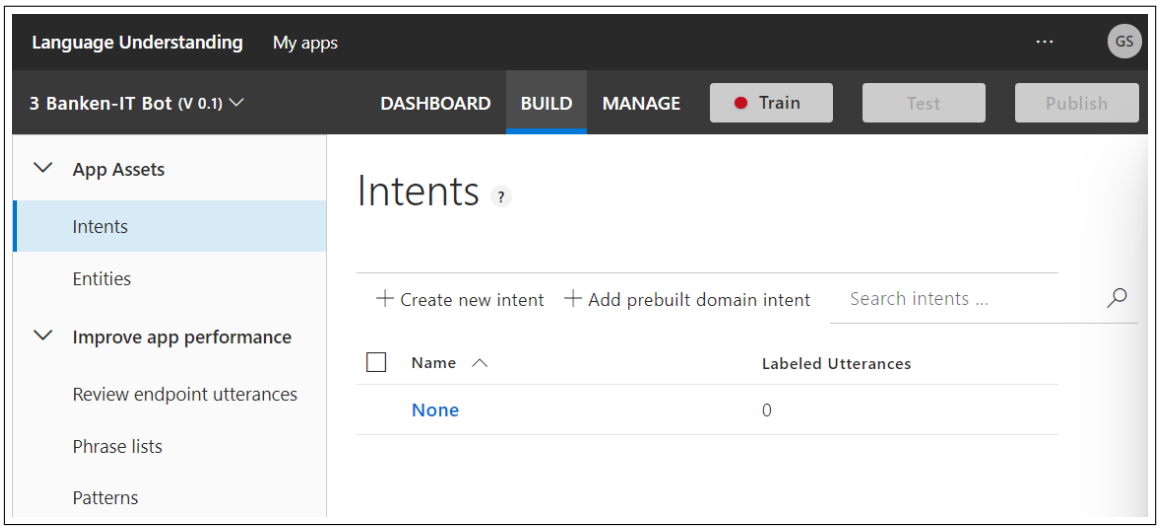

Figure 6.4: LUIS Web Interface

lists are the right choice. In the book hotel example, a list of valid hotel names existed. This list is a suitable example of entity lists since the number of valid entries is limited. Invalid hotel names are uninteresting in such a case. Entity definition by a pattern is another possibility. LUIS offers regular expressions[23] (regex) for the entity definition. All four frameworks offer regex entities, and regex entities are, therefore, considered to be state-of-the-art.

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. LUIS defines all required entities as prebuilt entities, and there is no need to add custom entities for the use-cases. The submit name and a submit return date intent allowed the testing of the entity quality. 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 matches inputs like Monday or tomorrow and converts them to a valid and correct date by default. The year is also automatically added when the date lies in the past. Hence, the data-span entity parses vacation requests for the upcoming year correctly by default. The required intents and entities were defined with LUIS for the performance comparison with the chatbot frameworks.

Chapter 7

Result

Portability

The portability is an important factor for software systems and especially interesting for chatbot technologies. The ISO 25010 [6] defines portability as a measurement criterion for software systems in general. One measurement criterion was the number of providers who support the deployment of the chatbot. It is interesting how tied to a provider technology is. When it comes to the chatbot cloud services (LUIS, Dialogflow, IBM Watson Assistant), the provider cannot be changed. All three services run in the cloud of the provider and nowhere else. The data processing is done in the cloud and cannot be done locally or on a company server. All in all, the tested cloud technologies are tied to the provider and cannot run anywhere else.

With Rasa, things are different. It is a local solution and runs on a local machine or a company server without any problems. Local data processing is no problem, and communication with a cloud service is not required at all. Docker deployment is available for Rasa, which is a flexible way for deployment. It is possible to run Docker containers in the clouds of all big providers. All in all, Rasa can run almost everywhere, while the cloud services can run only in the cloud of the provider. Furthermore, when Rasa runs in a Docker container switching the cloud provider is an easy task.

The second evaluated aspect of portability is the development from different devices and how easy the setup for such a scenario is. The cloud services only require an internet connection and a browser for development. Developing a chatbot is possible from any device with a browser. No setup is required to start developing. For the development of a bot with Rasa, access to the source file is required. 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 other cloud services, but it still cannot 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 every technology saves information differently, and it is 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 the chosen technology or not. The goal is to implement the dialogs shown in Figure 5.3. The evaluation result is a binary criterion which is either true when the implementation was possible or false if it was impossible. Chapter 6 shows that the implementation of the dialogs was possible with Dialogflow, Watson Assistant, and Rasa. It is impossible to implement the dialogs with LUIS, which met the expectation since it is an NLU technology and not a chatbot service. As mentioned before, NLU technologies have no dialog handling mechanisms. The Microsoft Bot framework is the chatbot technology which uses LUIS. Using both Microsoft frameworks, the implementation of a chatbot is possible. The implementation of the dialogs of Figure 5.3 is possible with all three chatbot technologies.

Project Setup Complexity

The number of required tools is a measure for the setup complexity. The setup of Watson Assistant, Dialogflow, and LUIS, requires a web-browser. This means only one tool, a browser, is required. The setup of Rasa is possible with Docker or in a Python environment. Rasa offers Rasa-X, which includes a web-browser based development environment similar to the UI of the cloud services. The local setup requires a Python environment and a package manager like pip. A text editor or IDE is required to modify the files efficiently. This results in a count of at least three (Python + package manager+ text editor). The second way is a Docker container setup, which requires Docker. Like before, a text editor is needed. The final count for the Docker deployment is two. With Rasa-x, a web-interface replaces the text-editor, but the scores remain the same since the web-interface requires a browser. To summarize, the could technologies have a score of one, and Rasa has a final score of two or three depending on the used approach.

The second measurement criteria for setup complexity is the setup time. The cloud technologies require an account to start the development. With an account, a new project can be created instantly. As mentioned before, after creating the project, everything is ready for development. No additional software needs to be installed. For Rasa, the tools and the environment needs to be provided. For the development, setting up a Python environment or pulling the Docker image is required. The setup for Rasa takes more time since setting up the development environment is required. All in all, the first place is shared by the cloud services, and the last/second place goes to Rasa in the setup time comparison.

Development Complexity

The easiest framework is LUIS since it is an NLU framework it doesn't offer dialog handling mechanisms. The development complexity for the NLU part is comparable 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 handled inside

of Watson which is a useful feature at times but makes the application more complex in general. Rasa is the most complex chatbot and NLU framework. The information about intents, entities, and utterances is scattered across multiple files. By default, form-filling actions are not supported and require Python programming. Luis, Dialogflow, and Watson Assistant offer form-filling actions by default without any programming.

Rasa does not offer any predefined entities by default. The pipeline needs to be adjusted to add predefined entities, which is more work than the solutions of the cloud providers. The creation of stories is also hardest with Rasa since the form filling action, for instance, requires additional information inside of the story, shown in Listing 10.3 where the form, person and time slots need to be set correctly. Trying to set the forms and slots by hand is not recommended. The usage of live training is recommended for the creation of stories. Live training is a useful feature to create natural conversations. However, it is also more complex than the creation of stories with the dialog node systems of Watson Assistant and Dialogflow. Figure 10.2 shows the dialog node system of Dialogflow.

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 languages, as does Rasa. Watson Assistant does not offer a person entity in German, and the one for the English language is marked deprecated. It also has no date-span entity, but two separate dates can replace it. Hence, Watson Assistant offers only two of the three entities for both languages. LUIS offers all three entities for the English language but has no person entity for the German language.

Pricing

The costs for a framework are important for companies. Companies can use Dialogflow for free. The limits for the free tier are 180 text requests per minute. The Essential and Plus versions allow 600 requests per minute. Rasa is free to use and open-source. There is an enterprise edition with some benefits but the price is calculated individually and not listed on the website. 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. Five requests per second are 300 requests per minute, which is more than the free tier of Dialogflow offers. Watson Assistant also offers 10.000 messages per month. According to IBM, 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. 7.1 lists the costs for the frameworks and editions of the frameworks.

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. 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 and essential edition. The standard edition also offers 50 transactions per second, which are 3,000 requests per minute. This is far more than the 600 requests per minute offered by 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 plus 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, 7]

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 is easier to develop a chatbot with Dialogflow than with Watson Assistant or Rasa. The Watson Assistants UI is intransparent 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, 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. Watson Assistant and Dialogflow require no programming skills, while Rasa requires programming skills, which makes it harder to learn. It is also harder to define a dialog structure with Rasa since a story has the format shown in Listing 6.7, whereas with Dialogflow and Watson dialog nodes are created. The dialog nodes are structured in a GUI with drag-and-drop.

It is harder to create form filling actions in Rasa. Python code defines the form-actions for Rasa. Listing 10.2 shows an example for such a form-action. Dialogflow and Watson support form-actions through a simple GUI shown in Figure 10.1. Using the GUI of Rasa-X defining new entities and intents works the same way for all frameworks. The pipeline of Rasa needs to be modified to enable the use of predefined entities. This 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 given problem, a developer does not 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

A major development factor is a deployment, where it is possible, and how easy the deployment process is. For the tested cloud technologies, the provider handles the deployment, and there is no development overhead for the developer. The deployment of Rasa needs to be done by the developer. Rasa can either run in a Docker container or Python environment. The Docker container approach was chosen for the deployment of Rasa since all cloud providers support Docker. The cloud technologies share the first place in the deployment ranking, as there is no overhead for the developer. The only technology tested which produces deployment overhead is Rasa.

The second deployment is the local variant. It is possible to deploy Rasa locally using Docker or a Python environment. As expected, it is impossible to deploy a cloud chatbot locally or in a non-native cloud environment. This ties a user to the provider and is not flexible. The only technology used in this thesis that supports local deployment or deployment in more than one cloud environment is Rasa.

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 send a request to the webhook when the extraction of information is successful. Rasa communicates with the action server, which serves a similar purpose as the webhooks. The frameworks use JSON messages for the communication between the chatbot and external services. While Dialogflow and LUIS offer metadata in the messages, shown in Listing 6.2, 10.4 and 10.5, Rasa and Watson Assistant offer no metadata by default. If metadata is necessary, the developer has to define it. This means more logic can be handled 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, logic handling is simple since the action server uses Python. Hence, all features of programming languages are available for result processing. By shifting the logic to the webhook, the cloud bots gain access to the advantages of programming languages.

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. The pipeline also includes Duckling, which converts dates into a standardized form as required. Duckling focuses on numeric values, dates, and times. 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”. Duckling always sets the second date of a date-span to the recognized date plus one day. This means that the second dates of Duckling differ from the dates of the other technologies. Table 7.8 shows the dates 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 formats and phrases as inputs. The expectation is that a recognized date format works for all instances of the format. If Monday is recognized correctly, the other weekdays should 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 is no system entity and is built with a low amount of training data. Table 7.6 and 7.7 list the results of the date recognition and extraction. The expected results are listed as true positive(TP) when the date should be recognized and extracted, or as true negative(TN) when the entity should not 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. The result also shows that weekdays and tomorrow work with all technologies.

The extracted value column shows that Spacy recognizes many date inputs correctly but does not convert them.

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

Table 7.4 shows the detailed result of the entity recognition comparison. 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 Rasa when with the combination of Duckling for dates and date-spans and Spacy for persons. Dialogflow achieved the best f-score (0.929) for the combination of dates and date-spans. 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 date-period entity

of Dialogflow reached a low f-score of 0.5. The use of the date-period entity is not recommended since the performance was horrible. Duckling, Watson, and Dialogflow were on an equal level (f-score of 0.909) on the date-span tests. Dialogflow also has the best f-score for the person entity. The person entity of Watson Assistant performed worst as expected. With more training data, the performance of the custom person entity of Watson Assistant can be increased. In test case two of Table 7.8 Spacy recognized two separate dates instead of a date-span. This behavior is counted as a failure because, in the application, there is only one date slot present, and the information cannot be processed correctly. In test case one, the date-period entity of Dialogflow and Luis convert Monday to a date in the past. That is not what is needed for the use-cases as 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 hint does not influence the result as the date extraction delivered the same result as before. The functionality was retested on another day. LUIS still failed to extract the correct date but Dialogflow managed to do so. The test result counts as a failure of Dialogflow's date-period entity since it does not 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

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

Intent Classification

The main task of a chatbot is to find the intent best matching the user input. Table 4.1 in the Appendix lists the training utterances for the sickness intent. The phrases for the vacation intent can be found in Table 10.2.

The intent classification evaluation uses multiple parameters. 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 part of the training sentence. Furthermore, the correctly identified intents are used to calculate the f-score like it was done for the entities. The test sentences only use date and date-span, which all frameworks recognized correctly to ensure unbiased tests. Spacy has no relevance for the application in terms of date and date-span extraction.

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 those of Dialogflow, LUIS, 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. LUIS extracted sick as a person in test sentence one and did not extract the person in test sentence six, shown in Table 7.10. The test sentences with higher numbers are more complex than the ones with smaller numbers. Dialogflow and Watson were able to classify all test sentences correctly. LUIS failed to classify test sentence seven of Table 7.10 correctly. Rasa failed to classify the more complex sentences correctly. 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 sentence classification worked better than the sickness test sentence classification. The average confidence was highest with Dialogflow followed closely by LUIS and the Watson Assistant. Compared to Dialogflow, LUIS, and Watson Assistant, Rasa's average confidence score is a lot lower.

The same experiment is also interesting with German as input language since the potential users are from a German-speaking country. Table 10.1 and Table 10.3 list the German training sentences. The test results 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 of Table 7.13 shows that the average confidence score for the German test sentences is higher than for the English test sentences for each technology. 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 average confidence is an additional value for the comparison since the f-score puts three technologies on par.

Watson achieved the highest confidence. The second place goes to LUIS. LUIS was able to increase 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 classification is 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 for all technologies except LUIS. Dialogflow has problems extracting the person entity correctly in German sentences. The system entity date-time of Dialogflow is unable to handle German date formats, although German is the selected input language.

Watson does not offer entity annotation in training sentences for German. This essential feature is only available for English language bots with Watson. Without entity annotation in training sentences, the creation of bots with Watson cannot be recommended.

LUIS offers a person entity when English is selected but not for the German language. The date-time entity of LUIS can handle German date formats. LUIS is the only technology that achieved an excellent entity extraction result on the German test sentences. To summarize, the intent classification result is excellent for the German language tests, but the entity extraction often fails.

Training Capabilities

One major factor for chatbot technologies is the training of intents, entities, and stories/dialogs. The development of new intents and entities works the same for all four technologies. An intent is trained with utterances. An entity is created either with examples or annotations. The training sentences for the intents can contain entities. They can be marked as entities by annotating them. Then the values are used to train the entity. There is no difference in the creation process of intents and entities. The creation of stories/dialogs only works for chatbot technologies. LUIS will not be part of the dialog evaluation.

Dialog structures are created with dialog nodes in Dialogflow and Watson, while Rasa uses stories. A dialog node represents a recognized intent and defines the slots, actions, and responses regarding the intent. In the UI, follow up dialog nodes can be created

to build a structure where one intent follows after another. Rasa uses stories that also defined the recognized intent, slots, actions, and responses. The real difference is that Rasa's stories can be structured in any imaginable way, while the dialog nodes follow a fixed structure.

The second big difference is that live training should be used with Rasa since the story files' structuring can get rather complicated. Neither Dialogflow nor Watson Assistant offers live training. With live training, a developer starts a conversation with the bot and goes through the whole conversation step-by-step. Every time the bot makes a mistake, the developer can correct the bot. The second option is to define the conversation step-by-step from scratch, where the developer tells the bot what to do. When the end of a story is reached, it is stored and used for training. The training capabilities of Rasa exceed the ones of Dialogflow and Watson Assistant and results in the first rank. There is no difference between Dialogflow and Watson Assistant when the dialog structure is considered, and they share the second place.

Speech Recognition Capabilities

Speech recognition is an option for further projects. The result of this evaluation is a boolean value. Frameworks that support this feature can be used with speech input instead of text. Either the technology supports speech recognition, or it does not.

Rasa does not support speech functionality itself, but it can use speech-to-text and text-to-speech frameworks like Alexa. Dialogflow offers audio in- and output by default. The speech-to-text functionality of Dialogflow 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, resulting 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 Microsoft Speech service costs $1 per hour of speech, which results in $0.00416 per 15 seconds. It is less than the costs of Dialoflows speech services. Text-to-speech costs are the same for the Microsoft 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 Microsoft Speech Services or Dialogflow. To summarize, the only technology which does not support speech recognition is Rasa. The other three technologies support the speech-to-text and text-to-speech features 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 four frameworks offer intents, entities, and utterances. The UIs 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 that with Watson Assistant, two entities have to be modeled for the use-cases. LUIS offers a date-time entity that supports dates and date spans, but it does not work as reliable as the one from Dialogflow. LUIS did not identify ”22-06 to 30-06” correctly, for instance, which was no problem for Dialogflow.

Usability of the Framework

The evaluation of the frameworks' usability is based on subjective impressions from the prototype development section. A major part of the chatbot development process is the creation of intents entities and utterances. The concepts are implemented in the same way for all technologies except Rasa, where they need to be defined in multiple files to achieve the same result. This means that it is equally easy to implement intents, entities, and utterances with Dialogflow, LUIS, and Watson Assistant and harder with Rasa in comparison. The structuring of the user interfaces of the cloud bots is similar. The interface of LUIS was the easiest, which is not surprising since it has no dialog handling mechanism. The UI of Dialogflow was the simplest of three chatbot frameworks because it was more transparent than the interface of Watson Assistant. The UI of Rasa-X was between Dialogflow and Watson Assistant. Rasa alone had no graphical user interface and is on the last place in this category.

Customization Possibilities

A major part of chatbot frameworks is the pipeline since it defines how the training of the intents and entities is done. The cloud chatbot technologies do not show the pipeline they are using, nor can it be adjusted. Rasa is different in that regard since the pipeline can be adjusted freely. Furthermore, technologies like Spacy show information about the components online. Spacy offers different models based on small, medium, and large training sets for the predefined entities when English is the target language. Figure 10.1 shows the adjusted pipeline used for this thesis. It is even possible to include custom components into the pipeline of Rasa. Rasa is the only technology of the four where the pipeline is visible, and it offers customization on top of that. The other three technologies do not offer mechanisms to adjust the pipeline.

It is possible to use external services instead of adjusting the pipeline by moving the entity extraction to the webhook where necessary. Since especially Dialogflow and LUIS offer a vast amount of predefined entities, the use of other entity extraction tools might not be necessary at all. It is different for Watson Assistant since it has a small number of predefined entities and would surely benefit from external services.

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

Table 7.8: Date Span Entity Recogniton and Extraction 1

No	Input	Expected	Technology	Recognized Correctly	Extracted Value
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
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	15	1	1.0	0.94	0.97	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 Table 7.1 shows that all tested frameworks offer free tiers. The non-free tiers of Dialogflow and LUIS plus the Microsoft Bot framework are charged based on the number of requests while the price for Watson Assistant is calculated based on the number of users. The price for the enterprise edition of Rasa and the premium edition of Watson Assistant are calculated individually and not listed on the website. The prices are hard to compare except for the bots of Google and Microsoft, which cost roughly the same per request. Since Watson Assistant calculated the price based on the number of users, it offers a better value when the users communicate with the bot frequently or have long conversations. Rasa's disadvantage in this category is the deployment since the infrastructure costs are unclear for a similar workload.

The prices for the speech functionality are comparable for Dialogflow and the Microsoft speech-services. The IBM text-to-speech service is more expensive than the one of Dialogflow and Microsoft. Rasa does not offer any speech services at all. To summarize, the Microsoft speech-service and Dialogflow are cheaper than IBMs speech services when it comes down to the price.

A significant disadvantage of cloud-based solutions is the dependency on the provider. It is not easy to change the provider and migrate the current chatbot 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. Rasa is the best technology when deployment flexibility is essential.

A major factor for chatbots is security. All cloud providers process the data in their cloud, which is not surprising, but it might be a security issue. The cloud provider has access to all the information entered by the user. This can be problematic for bank-account or health-related information, for instance. Rasa has no such problem since the data is processed inside 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 does not 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. Dialogflow achieved the best overall performance, followed by Rasa. No technology performed exceptionally bad in this category. The date-span entities of Duckling, Dialogflow, and Watson Assistant all reached the same f-score and performed on an equal level.

The intent classification result of Table 7.12 shows that the cloud services performed on an equal level when it comes to precision, recall, the f-score, and the average con-

fidence. Rasa was unable to keep up with the cloud technologies, and especially the average confidence score is low in comparison. IBMs Watson Assistant and Dialogflow were able to reach a perfect f-score of 1.0. The standard deviation is lowest for Watson, which puts Watson Assistant in the top spot closely followed by Dialogflow and LUIS. Dialogflow extracted an entity incorrectly in test case four of Table 7.10. Rasa showed 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.

The german language result of Table 7.13 shows some interesting differences to the English tests of Table 7.12. The average confidence increased, and the standard deviation decreased significantly for all four technologies. For the German language, the f-score values also increased. All in all, the German language result looks outstanding on the surface when only the intent classification is considered. However, the entity extraction performance dropped significantly on the test sentences. The German tests are shown in detail in the appendix in Table 10.4 and 10.5. There were a total of 21 failed extractions for the German test sentences and three failed extractions for the English test sentences on a total of 64 test sentences each. This means the entity extraction failed on every third test sentence for the German language tests. The only technology which delivered a satisfying entity extraction result on the German test sentences with zero mistakes total is LUIS. Rasa's performance was worse than the performance of LUIS, but the performance was still good enough for a recommendation. The date entity of Dialogflow and Watson Assistant is unable to cope with the German date format. The person entity of Dialogflow did not work reliably in German, although the names are not different from those used in the English test sentences.

Braun et al. [15] stated that if the training data is sparse, there is almost no performance difference between the chatbot frameworks regarding NLU. 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.

The communication of the frameworks works the same way for all the technologies. There is no significant difference, no matter the technology used. LUIS and Dialogflow offer metadata by default, while Rasa and Watson Assistant do not. All four offer REST endpoints and use JSON format for messages.

The easiest technology is LUIS because it is just an NLU framework and does not handle the dialog structure. Dialogflow is the easiest chatbot framework because it has the most straightforward UI, the most predefined entities, predefined intents for everyday tasks like a yes/no path and small talk can be enabled in the menu. All of these factors reduce the development time and can save money. Watson is comparable to Dialogflow when it comes to creating entities and intents. However, the UI is less intuitive, there is no small talk feature, there are almost no predefined entities, and it offers no predefined intents for everyday tasks. The most complex is Rasa since the base version has no UI, and programming skills are mandatory to develop a chatbot. The pipeline needs to be adjusted to add predefined entities, it has no predefined intents, and a small talk feature is not available. The creation of dialogs with Rasa is far more complicated, as is the collection of form data. Compared to the cloud frameworks, Rasa is incredibly complicated and does not offer many features by default.

The setup of Rasa is complicated compared to the cloud chatbots. It is not surprising but needs to be mentioned nonetheless. Only a browser is needed to develop a cloud chatbot, for Rasa, either Docker or a Python environment is required. It is harder to set up Rasa because the cloud technologies run already in the cloud and require no additional deployment. The comparably complex setup has advantages when it comes to deployment.

Rasa can be deployed locally and in every cloud environment while the cloud services only run in the provider's cloud. The deployment evaluation met the expectations. Rasa is flexible, and the cloud chatbots are simple when it comes to deployment.

Rasa is deployable almost anywhere, does not rely on the cloud, and can run without an internet connection. It is the only real choice of the four for security-critical applications.

The implementation of the design is possible with all three chatbot technologies. This means that all three technologies can be recommended when only the implementation of the design is considered.

The basic concepts for the implementation are the same for all technologies. There are intents, entities, and utterances. Dialogflow and Watson Assistant use dialog nodes for the conversation structure while Rasa uses stories. All offer REST endpoints, they communicate using JSON, they have a webhook or REST endpoint where information is sent to after the entity extraction. In the end, there is no significant difference when it comes to the concepts.

The only technology which allows customization is Rasa. If something very specific is needed or tuning is required, then Rasa should be used.

Chapter 9

Conclusion

The domain consisting of the sickness notification, and the vacation request is suitable for the use of chatbots. All in all, Dialogflow wins the implementation based ranking in front of Watson Assistant, followed by Rasa. It was possible to implement the design with all three chatbot technologies. Based on Chapter 6, the implementation is easiest and fastest with Dialogflow, then comes Watson Assistant followed by Rasa, which is the most complicated by far. It is 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, and yes/no logic is missing and needs to be created by the developer. Rasa is the most complex one since it does not provide any entities itself. The pipeline needs to be adjusted to enable the entities of other technologies like Spacy and Duckling. Adjusting the pipeline means additional development effort in comparison to the cloud bots. 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 cloud technologies do not require any programming skills to model the dialog. The implementation with Rasa took by far the longest, Watson Assistant is in the middle, and Dialogflow requires the least time.

Table 7.2 shows the number of predefined entities. Dialogflow and Rasa provide the entities required by the domain, whereas Watson Assistant lacks a person entity. Dialogflow also wins the number of predefined entities ranking by a large margin, and LUIS is a solid second. Rasa and Watson Assistant do not offer many predefined entities in comparison. The number of predefined entities is important for future use-cases since it reduces the development time significantly when the correct entities are already predefined. In general, predefined entities perform better because of the large and carefully selected 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. Dialogflow achieves the best overall entity extraction performance with the best f-score in all categories. Of the entity extraction capabilities alone, the use of either Dialogflow or Rasa is recommended. To summarize, the ranking shows that Dialogflow and Rasa performed better than LUIS and Watson Assistant, but no technology performed poorly.

Watson Assistant and Dialogflow dominate the intent classification since both were able to classify all test cases correctly. LUIS is a close behind, but Rasa could not keep up.

All three cloud chatbots performed on an equal level. The detailed result is shown in Table 7.12. Rasa was unable to keep up with Dialogflow, LUIS, and Watson on the average confidence value. This is an indication that Rasa needs more training data to reach the same performance level as Dialogflow, LUIS, 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 spot has to be Dialogflow since the performance in both categories was excellent. Additionally, Dialogflow has the best average confidence value. It is also possible to use LUIS and Watson Assistant since the performance was close to Dialogflow. It is possible to improve the performance and confidence scores of Rasa with more training data.

The intent classification results looked quite different for the German language tests and varied considerably compared to the English ones. The average confidence increased for each technology. The f-score of LUIS and Rasa increased significantly. In the German setting, LUIS is 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 showed a performance increase. Contrary to the intent classification, the entity-extraction performance decreased significantly. In the English test sentences of Table 7.10 and 7.11 the entity extraction didn't work correctly in two cases and worked correctly in 46 cases. For the German language, test shown in Table 10.4 and 10.5, a total of 21 wrong and 26 correct extractions are counted. The only technology that achieved an excellent entity extraction result on the German test sentences is LUIS.

Rasa is the only portable technology. It can run in a Docker container, and every cloud provider supports docker containers. It can also run locally on a development machine or a company's internal server. The cloud technologies are tied to the cloud provider who offers the bot and cannot be deployed anywhere else. All in all, Rasa wins the portability comparison.

The migration from one chatbot technology to another is not easily possible with any of the technologies tested. Hence, an evaluation of the migration possibilities is impossible.

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 of Rasa, the prices are calculated individually. All three offer a free tier, which can be used by companies. A technology recommendation because of the price is impossible since the prices are intransparent and hard to compare. The only thing which can be said is that Dialogflow and LUIS cost roughly the same. The units used for the price calculation vary between the cloud frameworks. Dialogflow calculates the price per request, while Watson Assistant calculates the price per 1000 users per month. LUIS mixes both approaches and calculates the price per 1000 requests per month This makes Dialogflow and LUIS 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.

The prices for the speech capabilities of Dialogflow and the Microsoft Speech Services are comparable. IBM's speech services are more expensive as a whole. Worst in this category is Rasa, since it offers no speech services. It is possible to use the speech services of cloud providers in combination with Rasa. The winners of the speech capability evaluation are Dialogflow and the Microsoft Speech Services.

When security is an essential factor, then local technologies like Rasa need to be used. Cloud chatbots process and analyze all data in the cloud, and this can be a problem for

safety-critical applications. If it is impossible to process information in the cloud because of security restrictions, then the recommendation is to use Rasa.

When all the factors discussed above are combined, the clear winner for English bots is Dialogflow, and the winner for German bots is LUIS. The design could be implemented fastest with Dialogflow, and it is 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. The only downside of Dialogflow is the entity extraction performance on the German test sentences. The best overall technology is LUIS and is the recommended framework for future development. Rasa's only negative aspect is the low average confidence score, but this is improvable through more training data. Rasa is a good local alternative to LUIS when more training data is provided.

When looking at the differences and similarities of the chatbot frameworks, the conclusion is that they are very similar and differ only in small aspects. Especially the cloud frameworks are almost identical in functionality, the provided services, and the costs. All tested frameworks use intents, entities, and utterances. All of the chatbot frameworks use a mechanism to structure the dialog. Dialogflow and Watson Assistant use dialog nodes, and Rasa uses stories for dialog structuring. The development effort and complexity between the technologies vary. Cloud chatbots require less knowledge and are easier to use while the local technology (Rasa) offers more customization possibilities. The pipeline of Rasa can is adjustable to the developer or company's specific needs. No similar function is available for the tested cloud chatbot technologies.

The third research question is about the suitability of problems for the use of chatbots. The three major criteria mentioned by Singh et al. [32] can be confirmed. A problem needs to be solvable via simple back and forth communication because it is the only thing a chatbot can do. The problem is not required to be repetitive, but it is only cost-efficient if it is. It should also be automatable to be efficient and to make the chatbot work independent from employees. In general, all problems which satisfy those three criteria are suitable for the use of chatbots.

Possible future use cases for the 3 Banken IT is the use of a chatbot for a banking application like mentioned in Singh et al. [32]. The banking app of Singh et al. [32] is used for transactions and querying account information. A popular use case for chatbots are FAQs (Shawar and Atwell [30], Raj [29], Huang et al. [24], Go and Sundar [20]).

The psychological aspects of chatbots play a vital role in the development phase [20]. When literature like Følstad and Brandtzæg [18], Brandtzaeg and Følstad [14] and Go and Sundar [20] are taken into account, the psychological aspects of chatbot development play an essential role for future work. There are multiple ways to make users believe that they are talking to a bot. Although uses should not be tricked, the concepts can increase the humanness of bots, which is valuable for customer-facing bots. User expectations will decrease when a bot is expected, but the conversation is evaluated more often as robotic. When users assume to talk to a human, the conversation is judged as more natural, but if the bot performs poorly, the experience is also evaluated as very negative. Imitation of humans is not a goal of chatbot development for business. The real goals are to reduce the workload of humans, aid humans, and replace humans to save costs.

For prototyping, the generation of training data by the developer is appropriate. It is not appropriate for the training of the actual product since the performance requirements are much higher. The training data for future work on the actual product can be collected through various real-life conversation sources like emails and conversations between cus-

tomers and experts. With real-life data, the training data is as realistic as possible, and the quality of the chatbot should increase significantly. User-studies are recommended to further increase the quality of a chatbot since actual users' feedback is the most valuable to improve the actual systems.

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 English

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

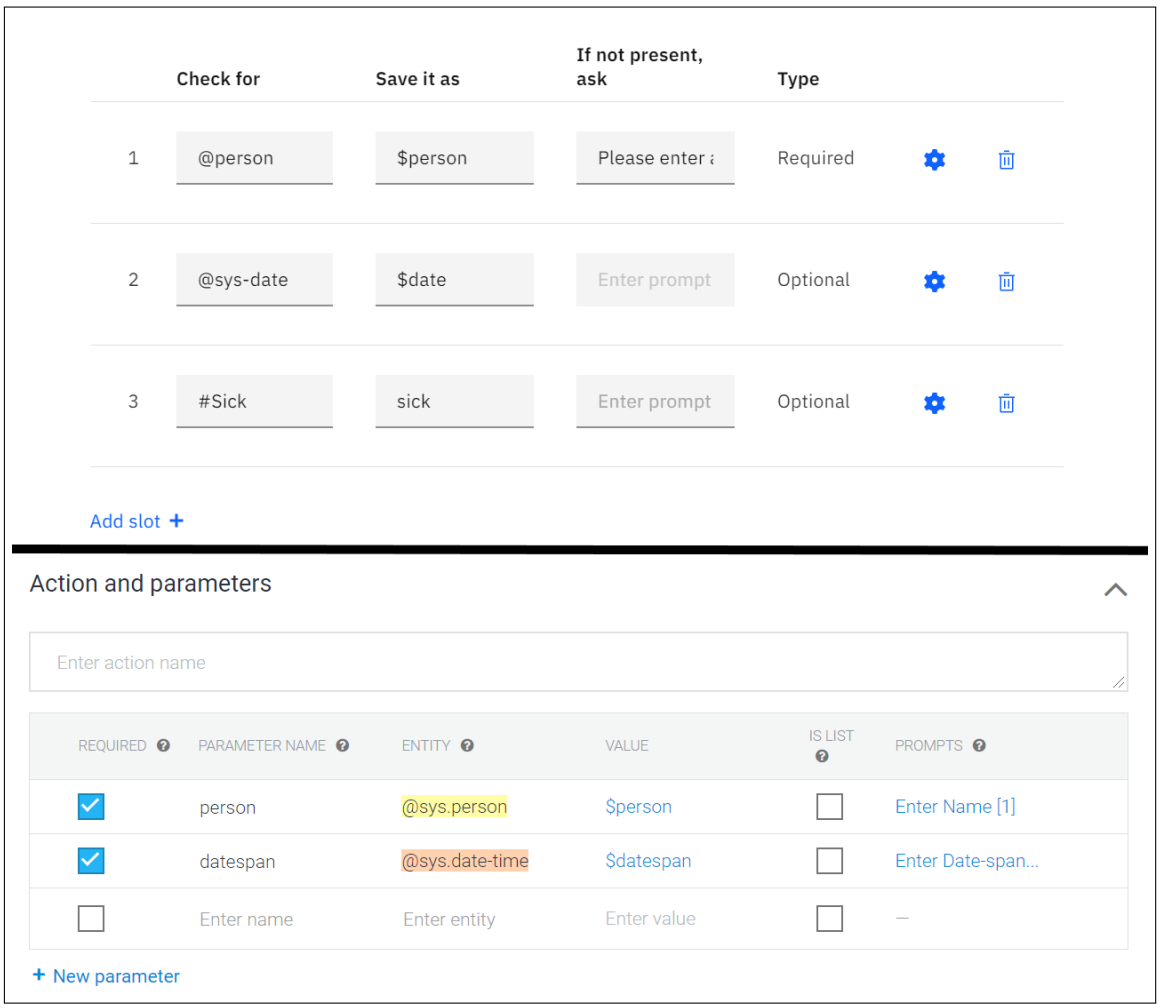

Figure 10.1: From Filling in Watson Assistant (top) and Dialogflow (bottom)

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

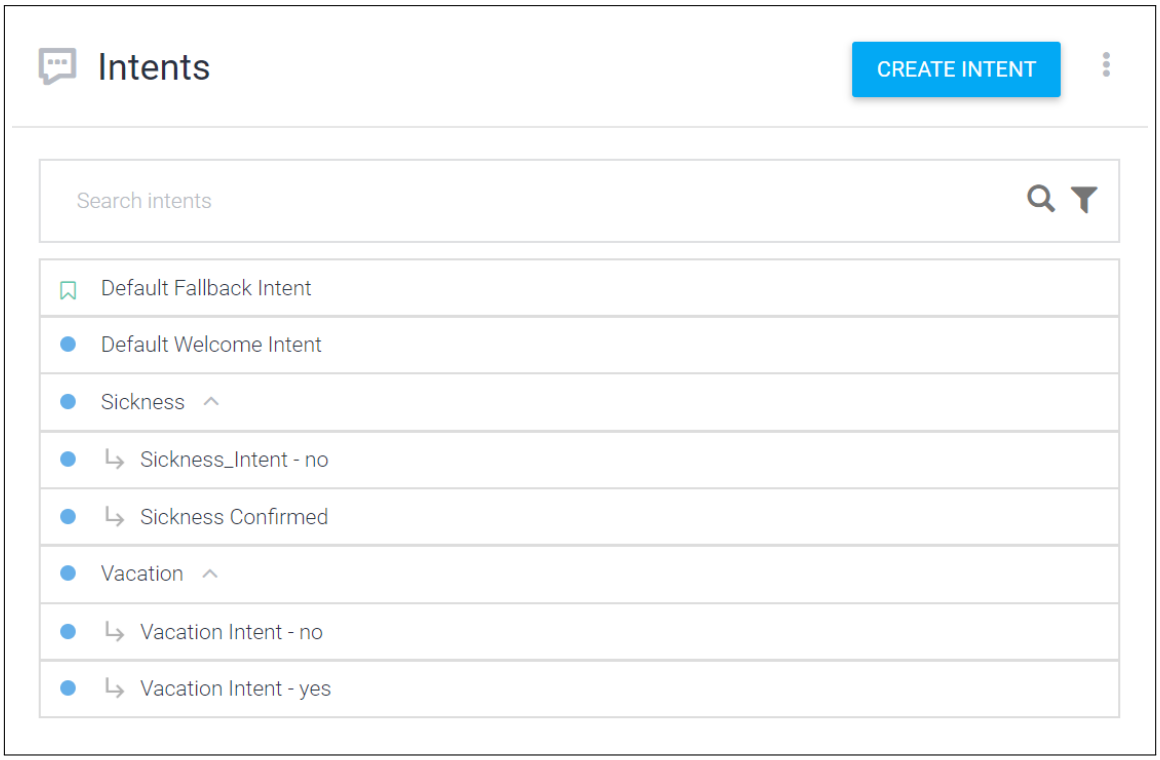

Figure 10.2: Dialog Nodes in Dialogflow

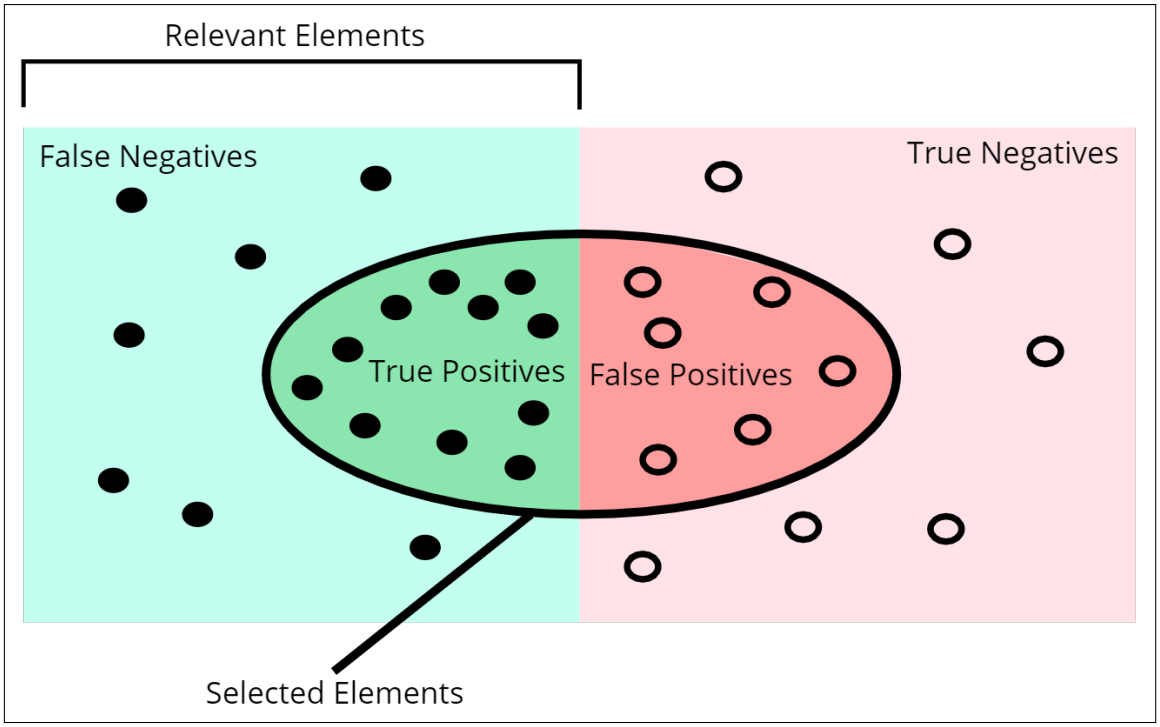

Figure 10.3: True and False Positives and Negatives

Figure 10.4: Chatbot UI

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, 39, 52, and 86.)

[5] Duckling. https://duckling.wit.ai/. Accessed: 2019-05-29. (Cited on page 45.)

[6] Iso/iec 25010. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. Accessed: 2020-05-09. (Cited on pages 22, 23, 29, 30, 32, and 49.)

[7] Language understanding (luis) documentation. https://docs.microsoft.com/en-us/azure/cognitive-services/luis/. Accessed: 2019-12-26. (Cited on pages 6, 47, 52, and 86.)

[8] Build contextual assistants that really help customers. https://rasa.com/. Accessed: 2020-05-09. (Cited on pages 6, 43, 52, and 86.)

[9] Spacy api. https://spacy.io/api/annotation. Accessed: 2019-05-28. (Cited on page 45.)

[10] Watson assistant. https://www.ibm.com/cloud/watson-assistant/. Accessed: 2020-05-09. (Cited on pages 6, 41, 52, and 86.)

[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 19, 20, and 25.)

[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, 20, and 25.)

[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 14, 16, 17, 20, and 25.)

[14] Petter Bae Brandtzaeg and Asbjørn Følstad. Chatbots: changing user needs and motivations. *interactions*, 25(5):38–43, 2018. (Cited on pages 5, 19, 21, 22, 27, 36, 37, and 72.)

[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, 17, 18, 19, 20, 23, 24, 25, 26, 27, 28, and 68.)

[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, 14, 15, and 24.)

[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, 19, 20, 22, 25, and 26.)

[18] Asbjørn Følstad and Petter Bae Brandtzæg. Chatbots and the new world of hci. *interactions*, 24(4):38–42, 2017. (Cited on pages 19, 21, 22, 24, 27, 30, 34, and 72.)

[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, 15, and 26.)

[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, 19, 21, 22, 27, and 72.)

[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 25.)

[22] Eric Gregori. Evaluation of modern tools for an omscs advisor chatbot. 2017. (Cited on pages 1, 5, 10, 19, 20, 25, and 26.)

[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 48.)

[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 72.)

[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 3, 5, 14, and 25.)

[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 19.)

[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 19 and 20.)

[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, 15, 25, and 26.)

[29] Sumit Raj. *Building chatbots with Python: using natural language processing and machine learning*. Springer, 2019. (Cited on pages iv, 3, 7, 9, 12, 13, 14, 15, 25, 26, 29, and 72.)

[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, 15, 16, 17, 20, 27, and 72.)

[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, and 10.)

[32] Abhishek Singh, Karthik Ramasubramanian, and Shrey Shivam. Building an enterprise chatbot. (Cited on pages 1, 5, 6, 8, 9, 17, 18, 20, 24, 25, 26, 28, 50, and 72.)

[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 22.)

[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, 19, 20, 24, and 26.)

[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, 14, 16, 17, 20, 24, and 25.)

List of Figures

List of Tables

List of Listings