Detection and Handling of Frustrating Conversation Situations in a Text-Based Chatbot System

Michael Primetshofer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2019

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, June 25, 2019

Michael Primetshofer

Contents

Abstract

Consider the accelerate digitalization and the growth of new technologies; the way people communicate with each other has significantly changed. That is the point where artificial intelligence (AI) comes into play. Chatbots gain more and more popularity, especially in customer support and sale areas. In this aspect, companies focus on consumers and customer services. Also, customer service scaling is costly and ineffective, as most customer requests are repetitive and do not require intensive manual handling by a service agent. Businesses use chatbots to automate routine work and minimize the associated workload. These systems provide useful information and can increase the working speed considerably.

However, the chatbot technology is far from perfect, and there are also some significant problems. Chatbots can also misunderstand text-based input, do not follow the context of a conversation, or provide the wrong answer in frustrating situations. These issues can negatively impact the user experience and pose a high risk to companies and their brand value. Even AI-based chatbots still need human support today if the knowledge base is exceeded. So, the real added value lies in a hybrid solution that combines the best of a human employee with artificial intelligence to offer an improved service at a lower cost.

This thesis provides an overview of current technologies and methods to handle and detect the problem of a frustrating situation. It also serves the challenges and complications that will be tackled when integrating a text-based hybrid chatbot in a customer support system. The focus of this thesis is to develop a system that recognizes and reacts to frustrating dialogue situations and provide the possibility to transfer the conversation to a human agent.

Kurzfassung

Angesichts der beschleunigten Digitalisierung und des Wachstums neuer Technologien hat sich die Art und Weise, wie Menschen miteinander kommunizieren, erheblich verändert. Dies ist der Punkt, an dem künstliche Intelligenz (KI) ins Spiel kommt. In diesem Aspekt konzentrieren sich Unternehmen immer mehr auf den Kunden Support sowie die Kundendienstleistungen. Darüber hinaus kann die Skalierung eines Kunden Supports sehr schnell kostspielig und ineffektiv werden, da sich die meisten Kundenanforderungen wiederholen und keine intensive manuelle Bearbeitung durch einen Servicemitarbeiter erforderlich ist. Unternehmen verwenden Chatbots, um Routinearbeiten zu automatisieren und den damit verbundenen Arbeitsaufwand zu minimieren. Diese Systeme liefern nützliche Informationen und können die Arbeitsgeschwindigkeit erheblich erhöhen.

Die Chatbot-Technologie ist jedoch alles andere als perfekt, denn es gibt auch einige erhebliche Probleme. Denn Chatbots können auch textbasierte Eingaben missverstehen, den Kontext einer Unterhaltung nicht verfolgen oder in frustrierenden Situationen nicht die richtige Antwort liefern. Diese Probleme können sich negativ auf die Kundenzufriedenheit auswirken und ein hohes Risiko für Unternehmen und deren Markenwert darstellen. Selbst KI-basierte Chatbots benötigen heute noch menschliche Unterstützung, wenn die Wissensbasis überschritten wird. Der eigentliche Mehrwert liegt also in einer Hybridlösung, die das Beste eines menschlichen Mitarbeiters mit der künstlicher Intelligenz kombiniert um daraus einen verbesserten Service zu geringeren Kosten anzubieten.

Diese Arbeit bietet einen Überblick über die heutigen Technologien und Methoden, um das Problem einer frustrierenden Situation zu behandeln und zu erkennen. Zusätzliche werden Herausforderungen und Schwierigkeiten bei der Erstellung eines textbasiertes Kundensupport-System, welches diese Situationen erkennen soll, erläutert. Der Schwerpunkt dieser Arbeit liegt in der Entwicklung eines Systems, welches frustrierende Dialogsituationen erkennt und darauf reagiert und zusätzlich die Möglichkeit bietet, das Gespräch auf einen menschlichen Agenten zu übertragen.

Chapter 1

Introduction

1.1 Motivation

In times of fiercely competitive markets, customer satisfaction and customer service becomes more and more important. As a result, many businesses today face the problem of maintaining high-quality customer service with a growing number of customer inquiries. Besides, customer service scaling is costly and ineffective, as most customer requests are repetitive and do not require intensive manual handling by a service agent. At this point, artificial intelligence (AI) comes into play. Because AI-supported customer service reduces customer latency, and chatbots provide a convenient consumer communication channel instead of emailing or calling the hotline. To get a better solution, combining the intelligence of a human agent and AI technology to a hybrid solution is a promising way to success. The AI can handle the repetitive customer requests, allowing the human support agent to focus on more demanding requests. However, all these systems have their limitations, particularly in terms of emotional recognition and following the context of a conversation.

1.2 Idea

The idea behind this thesis is to use a hybrid model that makes it possible to combine the best of human agent and artificial intelligence to offer a more comprehensive service at a lower cost. In a hybrid solution, if the knowledge base of the chatbot is exceeded and the system does not have an automatic answer to the question a human agent can intervene at any time in the conversation. Another advantage of hybrid solutions is operational readiness. Chatbots can handle customer requests in the absence of human agents. In contrast to employees, a product request that otherwise remains in the mailbox for days can be answered immediately and personalized via a chatbot. The idea behind the thesis project is to create a hybrid chatbot for a web-based ticketing system to offer customer support. The system should provide handover management that prevents frustrating conversations between chatbot and users. The system detects, during the conversation, whether the user wants to communicate with a human agent. When such a situation is detected, the information that had been collected during the conversation is passed to a human support employee, and the human agent can continue the conversation.

1.3 Goals

The goal of this master theses is to give an understanding of how chatbots can be developed to provide a hybrid solution and combine the best of human and artificial intelligence, to offer an improved service at a lower cost. Therefore, the meaning of each part of this system and concept behind it is crucial to the work and is covered by it. During the conversation, the hybrid solution should recognize whether the user likes to communicate with the chatbot or whether he would rather talk to a human employee. The chatbot provides a modular recognition system that provides different modules for detecting a frustrating situation. When the system detects a possible situation, a handover process is triggered, and the user can then continue the conversation with a human agent.

The result is a web application which provides a customer support hybrid chatbot solution. Additionally, the application provides a support management tool for the human agent. Therefore, the web application consists of three different components, the authentication page, the user page, and the administration page. The web application uses the authentication page to verify user identities. The user page allows the user to communicate with the chatbot, and after the handover process, the human agent can use the management tool at any time to participate in the conversation. The administration page illustrates the management tool for the human agent.

1.4 Outline

The following chapter 2 illustrates the fundamental parts of the text-based chatbot concept. In the beginning, the chapter covers a short history of chatbots and their basic functionalities. After that, the thesis describes the overall view of a conversation agent and a short explanation of all components inside a chatbot. Additionally, the chapter illustrates the different kinds of chatbot systems and their possible use-cases. The next chapter 3 describes the state of the art for a chatbot framework, which is used to develop a conversational agent. The chapter compares the frameworks and provides information about which one was used to create this thesis. The second part of the chapter covers current related hybrid chatbot systems. After that, the thesis dives deeper into the thesis project part, with chapter 4. This chapter presents the concept behind the thesis project. It covers the implemented requirements of the system, the use-cases, the system architecture, and the concept of how to detect and react to frustrating conversational situations. After the concept the technical implementation chapter 5 follows. In this chapter, the focus is on the implemented result and provides class diagrams as well as small code snippets. The evaluation of the created hybrid chatbot is given in chapter 6. The last chapter 7 describes the results, the conclusion, and summarizes this master thesis.

Chapter 2

Basic Concepts of Chatbot Systems

This chapter describes the fundamentals of the human-computer interaction model for a text-based chatbot system. The first section is a brief introduction in the history and the terminology of chatbots and gives an overview of already existing chatbots. The second part of the chapter explains the basic concepts of sentiment analysis.

2.1 Conversational Agent

Based on the scientific literature, a chatbot is a conversational agent that can simulate a conversation or a chat with a human user in natural language. A Chatbot is a software that is powered by a rule-driven engine or artificial intelligence (AI) that processes natural language input from the user and produces responses. The user interacts with the system via a text message or voice interface. In every conversation, the user wants to achieve a particular information goal, and this influences the communication flow between the chatbot and the user. Accordingly, a chatbot is a computer program that communicates with a human user humanly to achieve a predefined goal. As reported by [1] a chatbot' s determination is: "A Chatbot is a computer program that have the ability to hold a conversation with human using Natural Language Speech".

2.1.1 History

The history and development of artificial intelligence and chatbots dated back to the 1950s and was a component of early computer science.

1950

Alan Turing developed a simple communication test known as the Turing-Test [16]. The test is used to determine the intelligence of a program. Therefore a person communicates via text inputs with two endpoints. One of the endpoints is a human agent the other one is a machine. Afterwards, a human judge had to determine if the endpoint of text-based communication is a computer program or a human agent Researchers use the test as a method for determining the ability to imitate a person not to measure the intelligence of a machine. Imitation is a testimony of intelligence, but only in a certain area, hence the naming of the Turing test: *The Imitation Game* [16].

1966

The computer scientist Joseph Weizenbaum is one of the pioneers in the research field of artificial intelligence. From 1964 to 1966 he developed a program that enabled a linguistic dialogue via a telegraph console. ELIZA, the computer program that simulated a psychiatrist. The system checked the keywords in the user input and used rules of transformation for the output. The system of ELIZA works without concrete context processing. No personal data is stored or used. The phrases are changed grammatically correct and replaced certain keywords to achieve dynamics. The program handled a relatively simple conversation but gave the illusion of understanding the user's problem and fooled many people [17].

1995–1998

The program *Artificial Linguistic Internet Computer Entity* (ALICE) was originally composed by Richard Wallace and 1998 rewritten in Java. The idea behind ALICE consists of so-called *categories*, which consist of a specified input, output and optional context pattern. The pattern language consists only of letters, numbers, wildcards, scratches and tags. The system stores the pattern in a tree structure. There are three tags available for contextual processing. The keyword *this* refers to the last output of the program. Thus, the program selects the following category with the condition that matches the last output. With the *think* tag the system can store variables, e.g., name, age or gender from the user and use them in the conversation [3].

2001

ActiveBuddy, Inc. (now Colloquis) followed ELIZA's path with some additions, especially in the topics of speech synthesis and emotion detection. The outcome was SmarterChild, a dialog agent that was integrated on AOL Instant Messenger and MSN Messenger. SmarterChild is the forerunner of Siri by Apple[1] and S Voice by Samsung[2] and was later taken over by Microsoft[3] in 2007.

2010-2016

The smartphone era was the beginning of virtual assistants integrated in smartphones such as Apple's Siri or Google's Google Assistant[4]. With the launch of Siri in 2010, a voice-driven bot from Apple, the chatbot technology was embedded in the daily routine of people. With the help of Siri the user can not only talk about different subjects but also control functions via the smartphone.

There are also voice-controlled assistants like Amazon Alexa[5] and Google Home[6], which represent another concept of chatbots. With the development of the software

[1] https://www.apple.com/siri/
[2] https://www.samsung.com
[3] https://www.microsoft.com
[4] https://assistant.google.com
[5] https://alexa.amazon.com/
[6] https://store.google.com/de/product/google_home

Alexa and the combination with the hardware *Amazon Echo*[7], Amazon offers the possibility of controlling smart home elements by voice.

2016 was another significant year in the history of chatbots since Facebook[8] released the Messenger Platform and allowed developers the integration of conversational agents in the Facebook Messenger[9]. However, this does not imply, however, that current solutions are without flaw as will be highlighted in the next sections.

2.2 Components of a Conversational Agent

A chatbot is build of multiple components working to achieve a common goal. Figure 2.1 gives an overall summary and visualizes the relationship between each part of the agent. At first, the new message will be processed by the language identification module, which can be a simple tag retrieval or a more complex statistical method. Afterwards, the new message, the language and the potential previous conversation message from the backend are handed over to the intent classifier module. There the intent matching will infer the classification for the user input. This information will be used to determine an appropriate action or sequence of actions. For instance, if the intent is still not clear the chatbot can decide to reply with a question, or it could perform a specific action, for instance, visualize a map or image. Afterwards, the action handler executes the input action and replies to the user with a response message [4].

2.2.1 Language Identification

Language identification is the term to automatically detect the language in a document or text based on the content of the text. The chatbot uses the language identification if the system should provide more than one language. A typical language identification technique assumes that each form writes in a final set of known languages for which training data is available. Afterwards, the most expected language from the set of training languages will be selected. This functionality is a necessary task for creating a conversation agent. Identifying the correct language for the given text is a necessary task. Some algorithms tackle the problem by involving multiple language detection in one single piece of text, but in this work, the focus will be only on a single language [9].

2.2.2 Intent Classification

An intent represents the intention of the user by interacting with the conversational agent. The system has to classify the goal of the user's request. The intent classification has different labels, which represent possible user intentions. The solution for this problem can vary from keyword extraction methods to Bayesian interference and resolve the user's intention based on multiple messages. The dialogue system must fulfil the intent analysis in order to find out the user's assertion during the conversation [10].

[7]https://www.amazon.com/echo
[8]https://www.facebook.com
[9]https://www.messenger.com/

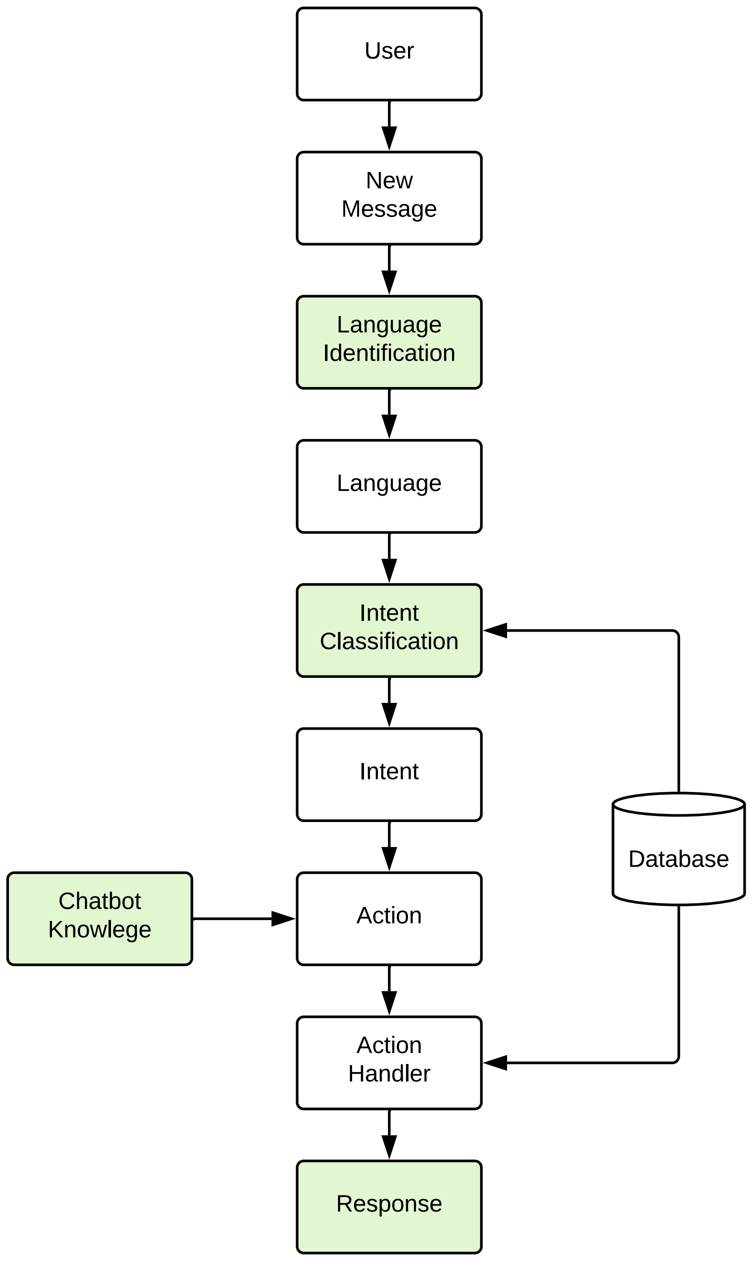


Figure 2.1: Schematic representation of all chatbot components.

2.2.3 Knowledge Management

The knowledge base is fundamental when developing a conversational agent because intent classification and language identification are not sufficient for understanding the user's intention. New techniques usually involve an inference engine in manipulating facts and collecting new knowledge. An intelligent chatbot can only be as good as its knowledge base, so knowledge engineering is highly recommended for answering questions about general facts. Current knowledge management techniques are often used with API requests and advanced database calls. For example, a chatbot could use the knowledge inference method to generate answers from the knowledge base on facts from the web and other sources [6].

2.2.4 Response Generation

The last component of a conversational agent is the ability to produce a reply in order to communicate with the user. The response has to be consistent according to the context. For this problem, there are two different solutions: *retrieval-based* and *generative-based* methods [11].

Retrieval-Based Technique

Figure 2.2 illustrates a retrieval-based technique which uses a repository of predefined responses. Afterwards, a particular heuristic chooses a reasonable response based on the user input and the context of the conversation. The heuristic can be a regular expression that checks for an appropriate sentence structure, or a machine learning model generates the output. This approach has the benefit that the supervisor can control the answers from the conversational agent and avoid unreasonable responses [15].

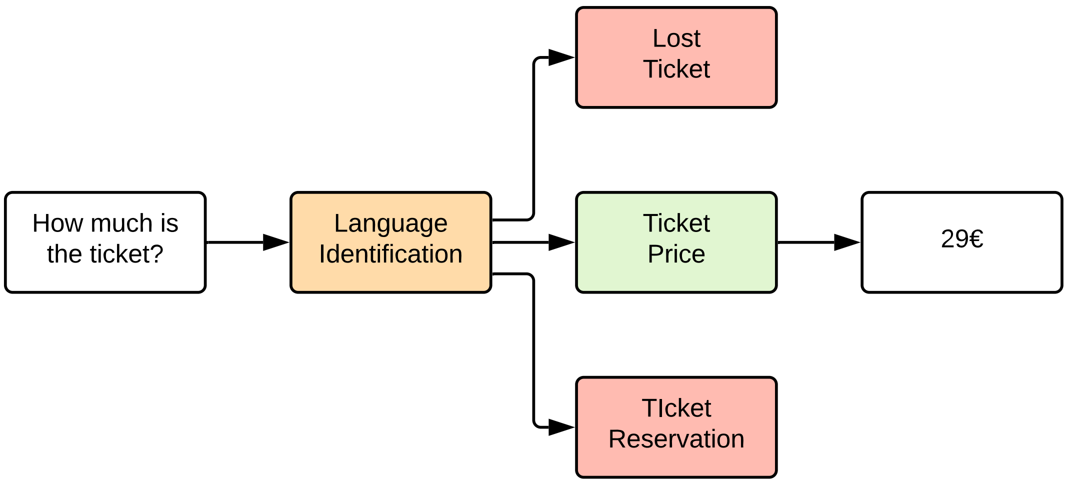


Figure 2.2: Retrieval-based model example.

Generative-Based Technique

The second approach is the generative-based technique that rely on a generative model and generate new responses without the need of predefined responses, seen in Figure 2.3. This technique takes more time than the first one because the model needs to be trained in order to learn how to generate usable responses [15].

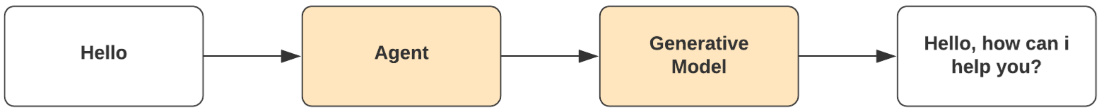


Figure 2.3: Generative-based model example.

2.2.5 Chatbot Categories

The correct identification and planning of the required chatbot type for the system and the selection of the most suitable platform for it are essential for the successful use. Each chatbot has a different task to fulfill, so these systems differ in some functions. The following categories are based on [13].

Personal vs Team Chatbots

A personal chatbot is a system that communicates with the user through direct messages, such as a personal assistant. The conversation is only between user and chatbot, e.g., a shopping assistant or a ticketing system. In contrast, the team chatbot communicates with a single or a group of users. For example, the Lunch-Train[10] chatbot, where teams can choose where they want to have lunch together. A team bot can communicate with multiple users either directly or publicly in a channel setting. The difficulty in developing a team-based chatbot lies in the optimal organization and synchronization of the entered data from all different users. A personal chatbot is much more comfortable to design, but has limitations in some use cases. So, a system can use a personal bot in a team conversation, but the communication focuses only on one user context at a time. The problem is that the chatbot can no longer follow the conversation context when another user joins the conversation.

Super vs Domain-Specific Chatbots

The representation of one specific product, brand or company is a typical use case for a domain-specific chatbot. The bot represents that service and the user interaction will be only about that task, e.g., the chatbot's only service is the representation of a website or a booking application. The advantage of the domain-specific chatbot is that the system specializes a specific type of problem and content. Therefore the user does not need to navigate through different services and tasks. This category is good if the developer only needs the chatbot for one purpose alone.

In contrast to a domain-specific chatbot is the super chatbot, which focuses on multiple services. So, this chatbot can connect different services, for example, a super chatbot can provide weather information in combination with the current traffic status. One example of such a super chatbot would be the Google Assistant, which is a single bot but provides access to different Google services. The significant advantage of a super chatbot is that the system combines different services, and the user only has to communicate with one big bot instead of different smaller ones.

[10]https://lunchtrain.builtbyslack.com/

Super bots and domain-specific have different integration aspects. Super bots give the developer a small overview of the actual user experience, while domain-specific bots can control the user experience much better. Integrating a service into a super-bot is equivalent to providing a service to a third-party library, while a domain-specific bot is similar to building its library.

Business vs Consumer Chatbots

Businesses can use chatbots to automate routine work and minimize the associated workload. Likewise, it can serve as a bridge between the employees. A Business bot aims to require the user with as many as possible support instructions. Consumers Bots, on the other hand, are designed to entertain the user, provide news about or assist in private projects. The conversation is less goal-oriented and focuses on dynamic interactions.

The risk of frustrating user experience, in terms of context, is lower in business chatbots. These programs focus on the demarcation of routine work. If the developer implements these scenarios, there is little danger that the conversation will be frustrating.

Voice vs Speech chatbots

Systems designed for written conversation can be published on platforms such as Facebook Messenger or Slack. Voice-controlled chatbot like *Amazon Alexa*, *Microsoft Cortana* or *Apple Siri* are activated by a button or a specific voice command for the electronic device. The interaction usually consists of a request or question and the confirmation or answer. Voice chatbots usually build for hands-off experience or mobile environment, while text chatbots are good for desktop environment.

2.2.6 Use-Cases

The following use-cases are based on [5] and illustrate an overview of the possible scenarios where a chatbot system can be integrated.

Trade and Advertising

Chatbots are an optimal contribution to the existing communication plan and companies use it for advertising material and e-commerce. For example, the system can place an order during the conversation. Not only a rating can be obtained, but also future offers for the company can be transmitted in order to strengthen customer loyalty.

Entertainment

In addition to using a chatbot with a specific goal, there are also systems which serve as an entertainment platform. There are systems which a user can talk freely about a topic of his choice, or it is also possible to play small games over a chatbot.

FAQ

Many chatbots take over the function as customer service. These systems benefit from their large number of communication partners and permanent accessibility. The frustration free processing is usually guaranteed since the typical question-answer principle takes place, and the user questions can be ideally covered by frequent repetition.

Communication Link

The last use case for a conversational agent could be the communication link between two different humans. Therefore the chatbot can serve as a connection bridge, as the principle of a telephone call with prior forwarding, or the functionally of a handover process from the machine to a human agent.

2.3 Sentiment Analysis

In this thesis the sentiment analysis is used for the detection of frustrating conversation situations. The system should use the sentiment of the user's input message and clarify the current situation.

2.3.1 Definition

Sentiment Analysis describes the extraction of opinions within a text and is a field within Natural Language Processing (NLP). It's also knows as Opinion Mining and beside identifying the opinion, the system also extract aspects of the expression e.g:

- Polarity: Positive or negative opinion.
- Subject: The thing that is being talked about.
- Opinion holder: The person, or entity that expresses the opinion.

Generally there are two major types of text information: *facts* and *opinions*. A Fact describes objective expressions about something and opinions are subjective expressions that characterize people's sentiments or feelings against a subject. The system's structure is as a classification problem where the sentence is classified in two classifications — first the subjectivity classification if the sentence is subjective or objective. The second one is the polarity classification, the positive, negative or neutral opinion [14].

2.3.2 Algorithms

Every method or algorithm to implement an opinion mining system can be classified in one of the following categories:

- Rule-based systems which achieve sentiment analysis based on a set of predefined rules.
- Automatic systems learn from data generated by machine learning techniques.
- Hybrid systems combine rule-based and automatic systems.

The rule-based system is a predefined set of rules which identify subjectivity, polarity, or the subject of opinion. The system takes some types of inputs for classification the opinion, for example, some classic NLP techniques like stemming, tokenization, part of speech tagging and parsing or other references such as a list of words and expressions (lexicons). The disadvantage of this method is the straightforward and naive word detection because it does not include how words are combined in a sequence. Automatic approaches use a machine learning technique for analysing the opinion. Usually, the sentiment analysis task models a classification problem where the system is filled with the text and returns the comparable category. In a polarity analysis system, the return value would be positive, negative or neutral. The first step is the training process, where the model is trained with test patterns and learns to connect the input text to the comparable outputs. The text input will be transferred into a feature vector and paired with the tags, for example, the polarity. The machine learning algorithm and these pairs generate the model. Before the algorithm can classify the text input, it has to transform the text into a simple numerical structure. Commonly a vector representation is used. Every component of the vector describes the frequency of a word or the definition in a predefined lexicon. This procedure is called feature extraction or text vectorization. For evaluating the performance of a classifier *precision*, *recall*, and *accuracy* are standard metrics. The metrics measures how many texts were predicted correctly. Hybrid approaches are very intuitive, which combines the best of both methods, the rule-based and the automatic. The combination of both methods can improve accuracy and precision[2, 8, 23].

2.3.3 Mode of Operation

Nowadays a large amount of research has been done in the field of sentiment analysis and opinion mining. Opinion mining uses natural language analysis to capture the mood or attitudes to a particular sentence. After the opinion classification, the sentiment analyses evaluate the input sentence. The Process, for example, helps to determine if a review or text is more likely to be positive or negative [7, 12, 18].

There are many practical approaches to detect annoying human responses in in text-based chatbot systems. This thesis will focus on the following input scenarios:

- Keywords: The user types “Human Agent” or “Human Assistance”.
- No response: The user gives no response on a request message.
- Emotional sentence: Emotional detection of the user’s input.
- Same phrase or sentence: User repeats sentence or phrase several times during the conversation.

Sentiment analysis is used for example in the “Google Cloud Natural Language API[11]” form Google or the “Watson Tone Analyzer[12]” from IBM to generate reasonable responses. This requires sufficient complexity in interaction, and it is hard to achieve good results without hardcoded rules. Intelligent chatbots are essential in building successful customer service applications.

2.3.4 Challenges

This section gives an overview of some challenges that will generally be encountered during the completion of a text-based communication system with sentiment analysis

[11]https://cloud.google.com/natural-language/

[12]https://www.ibm.com/watson/services/tone-analyzer/

and opinion mining [23]. The following situations should be considered:

- Word definition: The word vocabulary and every word has to be clearly defined because words can have different meanings.
- Reasonable Response: How to generate reasonable response after detecting the user's sentiment without hardcoded rules.
- Sarcasm: This is because sarcasm is extremely contextual and to understand sarcasm, the analyzer needs to understand contextual clues.
- Polarity definition: The polarity of every word has to be categorized.
- Context scope: Words can be considered as being positive in one situation and negative in another situation.

Chapter 3

State of the Art

There is a variety of related work dealing with human-computer interaction technologies and also in combination with sentiment analysis. In this chapter, some examples of these frameworks will be discussed and compared. It also presents the different development platforms for creating a chatbot interface for a web application.

3.1 Chatbot Frameworks

A chatbot framework is a digital workspace for developing conversational agents with predefined functions and classes. These frameworks can help to create a customized natural language chatbot system and integrate it into a wide variety of publishing platforms.

3.1.1 Dialogflow

Dialogflow (previously called API.AI)[1], 2014 released and bought in 2016 by Google, is a platform which offers the developer the possibility to develop of chatbot system for free. The developer can configure the framework via a web-based development environment. The system supports over fifteen different languages, and the developer can easily integrate the framework into different messaging services (e.g., Facebook Messenger or Slack[2]). *Dialogflow* also offers documentation, a forum for users, enables and uses machine learning as well as natural language processing technologies. The framework also provides speech recognition, sentiment analysis and a moderate amount of predefined entities (e.g., weather information, time or date). One of the significant advantages is the unrestricted use of the API calls so the user has no restriction for messaging with the chatbot.

3.1.2 Amazon Lex

Amazon has announced the launch of *Amazon Lex*[3] a service built on the *Amazon Alexa Skill Kit. Amazon Lex* is a service for creating conversational interfaces for speech and

[1]https://dialogflow.com
[2]https://slack.com
[3]https://aws.amazon.com/de/lex/

text in any application. *Amazon Lex* also offers the possibility of speech recognition, sentiment analysis and an advanced amount of predefined entities for text or voice inputs. *Amazon Lex* provides standard integration with *AWS Lambda*, *AWS MobileHub*, and *Amazon CloudWatch*, and integrates easily with other services on the *AWS* platform. The system takes advantage of the power of the *AWS* platform in terms of security, monitoring or storage. The significant disadvantage of the framework is the limited language support, currently only English, the pricing of 0.00075$/text or a one-year trial plan with a limitation of 10k API calls per month.

3.1.3 Wit.ai

Wit.ai[4] is another NLP engine for developers, which provides a framework where developers can build conversational applications. The system acquired by Facebook has a web-based interface, and it is used to understand natural language. *Wit.at* is free to use and offers unlimited API calls. The framework supports over 100 languages for the conversation and additional speech recognition, sentiment analysis, a training module and a basic set of prebuilt entities.

3.1.4 Watson Assistant

The *Watson Assistant*[5], developed by IBM, is a machine learning and natural language understanding system, which offers a cloud-based interface for building conversational interfaces into applications or devices. The Assistant allows the developer to create an application, with the virtual assistant developer toolkit, that understands human language and responds in human-like conversations. The *Watson Assistant* offers a free to use trial plan but with a limitation of 10k API calls per month. The Assistant has ten supported languages and also text and speech recognition. It also provides the possibility for sentiment analysis, training modules and a basic set of prebuilt entities.

3.1.5 LUIS.ai

Luis.ai[6] or Language Understanding (LUIS) is a cloud-based API service that applies custom machine learning intelligence to a user's natural conversation. The system was introduced by Microsoft in 2016 and offered an interface for developing a machine to a human conversational bot. The service has a trial model, where the developer can use up to 10k API Calls per month. The amount of supported languages is thirteen, with also text and speech recognition. Like the other mentioned system, *Luis.ai* also offers a sentient analysis API, a training module and a basic set of predefined entities.

3.1.6 Comparison

Table 3.1 compares for this thesis relevant chatbot frameworks and gives an overview of functionality and pricing. Relating to Table 3.1 *Dialogflow* and Wit.ai have the most relevant functionalities for the favoured chatbot system that must be developed within

[4]https://wit.ai
[5]https://www.ibm.com/cloud/watson-assistant/
[6]https://www.luis.ai

	Dialogflow	**Lex**	**Wit.ai**	**Watson Assistant**	**LUIS.ai**
Organization	Google	Amazon	Facebook	IBM	Microsoft
Pricing	Free	Trial: 1 year Paid: 0.00075$/text	Free	Trial: Free Paid: 0.0025$/text	Trial: Free Paid: 0.75$/1k calls
API calls	Unlimited	Trial: 10k/month Paid: Unlimited	Unlimited	Trial: 10k/month Paid: Unlimited	Trial: 10k/month Paid: Unlimited
Supported languages (German)	15 (✓)	1 (-)	100+ (✓)	10 (✓)	13 (✓)
Speech recognition	✓	✓	✓	✓	✓
Sentiment analysis	✓	✓	✓	✓	✓
Training module	✓	✓	✓	✓	✓
Prebuild entities	intermediate	advanced	basic	basic	basic
Cloud based	✓	✓	✓	✓	✓

Table 3.1: Overview and comparison of chatbot frameworks (2019) [24].

the scope of the thesis. In summary, it can be argued that there are a variety of chatbot frameworks on the market right now. In this thesis, only the most appropriate frameworks for developing the implementation part were selected. Finally, there is no overall best framework, because every chatbot system has a different implementation criteria.

When selecting the application for the development described in this thesis, Table 3.1 shows the criteria that have been taken into account. In term of pricing and the required supported language (German) only two (*Dialogflow*, *Wit.ai*) of the listed frameworks fulfilled the functional scope. After a thorough research and review of the various features, the *Dialogflow* framework was used to develop the chatbot system.

3.2 Related Work

This section illustrates some examples of chatbot systems. The focus will be on web-based conversational agents in combination with hybrid human bot solution. Another criterion is the combination with sentiment analysis for detecting emotion in conversational situations, the limit of chat message and the maximum user interaction. For the comparison, the focus was on B2C (Bussiness to Customer) Support chatbots, FAQ (Frequently Asked Questions) feature and third-party integration (e.g., *Dialogflow*). Table 3.2 visualizes the comparison and overview of the matching systems. The following sections discuss the five evaluated systems in detail.

3.2.1 Kommunicate

The *Kommunicate*[7] software is a real-time chat system in combination with an efficient customer support chatbot. The plugin offers a native mobile SDK for integrating the chatbot in different platforms and provides the integration of a third-party chatbot framework (e.g., *Dialogflow*). *Kommunicate* offers a wide range of functions, e.g., visitor analysis, FAQ support for general customer queries or human fallback. For the human fallback *Kommunicate*, offers different handover triggers. The first one is with an user-driven menu there the system provides a predefined option after every message (e.g., Human Handover). The second trigger uses sentiment analysis for detecting the user's frustration level.

3.2.2 Botsify

Botsify[8] is another chatbot builder software where the developer can create a chatbot for messenger platforms or web applications. *Botsify* uses a drag and drop template to create the chatbot. The system also offers integration via plugins (e.g., *Slack* or *Dialogflow*), Smart AI, Machine learning and analytics integration, collect data like email, address or user name and provides a human takeover function for a smooth transition from a bot to a human agent. The handover process will be triggered via a generic keyword (e.g., human help) at any point in the chat-human conversation. After that, an email notifies the agent, and the conversation is handed over to him.

3.2.3 Flow XO

Flow XO[9] is also for building, hosting and managing chatbots on a messaging platform or web applications. With the visual editor and prebuilt templates, the developer can create a chatbot interface without coding experience. It also offers all other functionalities like the two platforms discussed above. Human handover works similar to *Botsify* also only via a generic keyword. Afterwards, the system sends the bot messages to a human agent by email, and the agent can directly message, via the chat surface, with the user.

3.2.4 ActiveChat

ActiveChat[10] is a chatbot builder, which provides an already created template for the development of a conversational agent. The editor and the prebuilt template provides a chatbot interface without coding experience. The chatbot platform uses natural language and can be used for, e.g., customer support. *ActiveChat* can be integrated into messenger platforms and can integrate third-party libraries like *Dialogflow*, or a customer support template for integrating a FAQ. *ActiveChat* offers four different payment models. Besides, the system does not have a handover fallback function with either keyword or sentiment, and there is no way to communicate with a human agent.

[7] https://www.kommunicate.io
[8] https://botsify.com
[9] https://flowxo.com
[10] https://activechat.ai

	Kommunicate	**Botsify**	**Flow XO**	**ActiveChat**	**Intercom**
Number of interactions per month	Unlimited	Unlimited	Free: 500 20$: 5000	Unlimited	Unlimited
Text-based platform	Web Messenger Mobile	Web Messenger	Web Messenger	Messenger	Web Messenger Mobile
FAQ support	✓	✓	✓	✓	✓
German language	✓	✓	✓	-	✓
Third party integration	✓	✓	✓	✓	✓
Mobile SDK	✓	-	-	-	✓
Handover fallback with keyword	✓	✓	✓	-	✓
Handover fallback with sentiment	✓	-	-	-	-
Live-chat after handover	✓	✓	✓	-	✓
Pricing per month	Free	50$ | 300$	Free | 20$	Free | 19$ | 49$ | 249$	87$ | 153$
Number of users per month	Unlimited	30k | Unlimited	Unlimited	500 1k 5k 50k	200

Table 3.2: Overview and comparison of chatbot builder (2019).

3.2.5 Intercom

The *Intercom*[11] can be integrated into websites, messenger platforms or app services. The software offers a real-time messaging approach. The chatbot can understand the German language and offers a third-party library integration. Additionally, the chatbot can transfer the conversation to a human agent by a handover fallback function based on keywords. Therefore the conversation can be transferred to a human agent, and the client can live chat after the handover process.

[11] https://www.intercom.com

Chapter 4

Conceptual Project Design

In this chapter the focus will be on the system design and the concept of the prototype, which was developed during this thesis, and contains information about the requirements and use cases. The chapter also explains the created *Dialogflow* Model to achieve the handover process between the chatbot and the human agent.

4.1 Requirements

The goal for the final prototype is to create and integrate a text-based dialog system in a web-based application for customer support. The chatbot should support employees with customer favours and support requests. Additionally, the chatbot system should provide handover management that prevent frustrating conversations between machine and user. The system should detect, during the conversation, whether the user wants to communicate with a human agent. When a handover process is detected, the chatbot passed, the collected information during the conversation, to a human agent, and afterwards, the human agent can chat with the user. The next section lists the defined requirements for this project. It should give an overview of the various features of the project and clarify the required steps to realize the idea of for the thesis project. The requirements for this project are the following:

1. The creation of a web application and a messaging chat interface.
2. The communication between the web application and *Dialogflow*.
3. The handover process detection (hanodver module).
4. The handover process reaction (handover phases).
5. The communication between human agent and user (live chat).

4.1.1 Web-Application

The project environment should be a web-based application and on top of it, the chatbot system for the user input. Therefore a web application with a dialog messenger interface, for the human-machine communication, should be developed. The application should also provide user authentication, and a managing tool for the handover provided for the human agent. Every human agent should have the possibility to join the chatbot conversation after the user wants to hand it over to a human agent. The managing tool should display all available support tickets and the available transfer options. Additionally, the application should provide the handover process detection, the handover reaction, the data processing and the transfer of the conversation to a human agent.

4.1.2 Communication

The web application should also communicate with the chatbot framework for the natural language processing (NLP), the interaction between computer and the human language. Therefore, the *Dialogflow* API should be integrated, and the two interfaces should communicate via RESTful Web service to access and manipulate the conversational messages between chatbot and user. The communication messsages (user input messages and chatbot output messages) should be transmited over a webhook.

4.2 Use-Case

Diagram 4.1 illustrates the workflow of the application. The use-case shows the process where the chatbot hands over the conversation to a human agent after the handover detection. Therefore the chatbot or the user can create messages, which the application displays on the conversation stack. The system checks every user input for a possible frustrating situation.

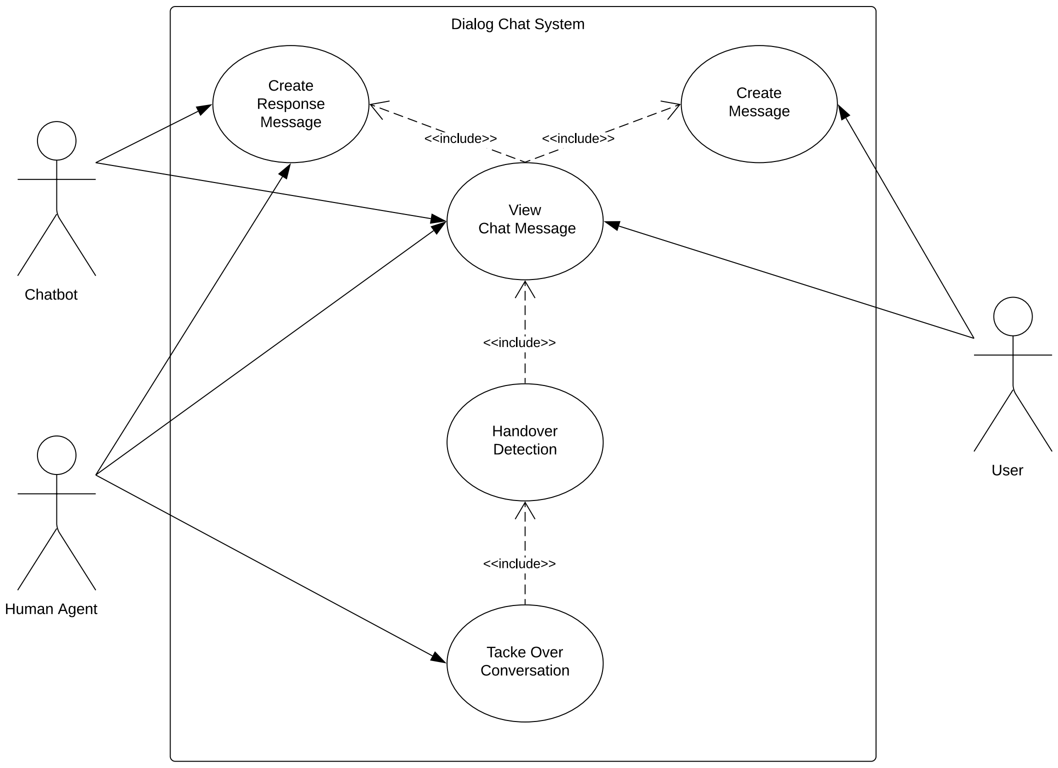


Figure 4.1: Use Case of how the chatbot hands over the conversation to a human agent.

If the system detects such a situation (explained in Section 4.3), it will initialize the handover process. In this situation, the third actor, the human agent, can take over the conversation and can use the managing tool to involve in the conversation and replace the chatbot.

4.3 Handover Detection

There are moments where these systems will stumble and fail to answer the request. It requires a human agent's help and intelligence, in this situation. An alternative to a full machine chatbot is a hybrid solution, which is essential for a good chatbot system.

For this solution, a transition from the machine to a human agent is one of the core features. If the system is not getting the correct balance, it will frustrate the users and might lead to ruin the support chatbot. The application should know when to trigger a handover process and hand it over to an human agent.

This feature triggers the situations for the initialization of the handover process. It should detect frustrating or annoying situations for the user. The following enumeration explain the scenarios that should trigger the handover process in more detail.

User Preference

The simplest and safest approach for initializing a human handover is the hard-coded method. The chatbot will be programmed to provide the user with a predefined option, for communicating with a human agent, after every message. This approach of handing over the conversation does not require any NLP or AI. In this scenario, the bot transfers the conversation to a human agent, whenever one of the following or a similar option is selected:

- Chat with a human agent,
- Human Agent,
- Help,
- Human Assistant,
- Support Employee.

User Sentiment

With the help of natural language processing and sentiment analysis, chatbots have the possibility to conclude the mood of the user. In this scenario, this technique can be beneficial in understanding whether the current conversation is satisfying for the user. So whenever the system detects that the user is frustrated, it can simply provide the option "Talk with Human agent". For this approach, the integration of a sentiment analysis API is necessary.

No Response

Another sequence for the human handover will be if the user gives no response for a request. After a predefined timeout, the system will trigger the handoff process.

Repeated Phrase or Sentence

The last approach is the repeated phrase or sentence case, where the chatbot should detect that the customer request repeats and the system could not respond to the message and types in the same request over and over again. If the system detects one of these four approaches, the handover process will be triggered, and the system will continue with the preparation process.

4.4 Handover Reaction

In this step, the question is how the bot should handle the transition and what information the system prepares for the human agent. For good user experience, the handoff process should be as smooth as possible to minimise any additional frustration. The process will be divided into three phases and discussed in the following section.

4.4.1 Pre-Handover Phase

In the pre-handover phase, one of the described scenarios from Section 4.3 triggers the handover process, and the chatbot needs to pass control to the human agent. This phase illustrates the possibilities of how the user can activate the handover process after the system has understood that it has reached its limitations. For this prototype, the preferred solution is that the chatbot asks the user if he would like to get connected to a human agent and based on the user's response initialize the handover process.

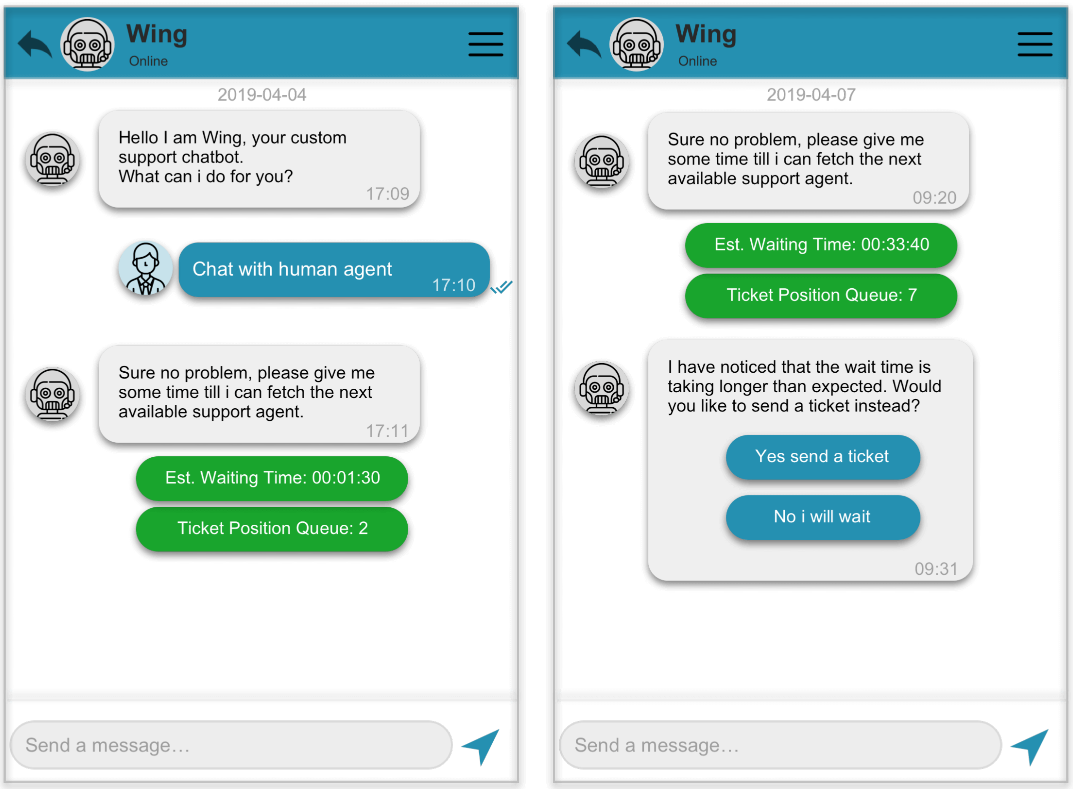
Another essential task is the transparency of the current transition while it is in progress. In this case, the system should inform the user about the process status and that the support ticket is currently unassigned. Additionally, the message from the user, during that time, will only be answered when an human agent is assigned. Figure 4.2 (a) illustrates the idea behind the pre-handover phase and the desired solution for the conceptional prototype design.

4.4.2 Wait-Handover Phase

Immediately after the user initiates the handover process, the system queues the support ticket and notifies the support human agent. During this phase, the application informs the user about the estimated waiting time and that the wait time might be long. For a good user experience to the system should also display the user's position in the waiting queue and provide the option to send the issue via an email when the user has no desire in waiting any longer. The allocation of the tickets should depend on the availability of the human agent's. Figure 4.2 (b) shows the idea behind the wait-handover phase and the prototype design.

4.4.3 Post-Handover Phase

The final step of the human handover process is when the human agent finally joins the conversation between chatbot and user. In this phase, it is important to understand that the human agent took over the conversation from the chatbot. Additionally, the



2019-04-04

Hello I am Wing, your custom support chatbot.
What can i do for you? 17:09

Chat with human agent 17:10

Sure no problem, please give me some time till i can fetch the next available support agent. 17:11

Est. Waiting Time: 00:01:30

Ticket Position Queue: 2

(a)

2019-04-07

Sure no problem, please give me some time till i can fetch the next available support agent. 09:20

Est. Waiting Time: 00:33:40

Ticket Position Queue: 7

I have noticed that the wait time is taking longer than expected. Would you like to send a ticket instead?

Yes send a ticket

No i will wait

09:31

(b)

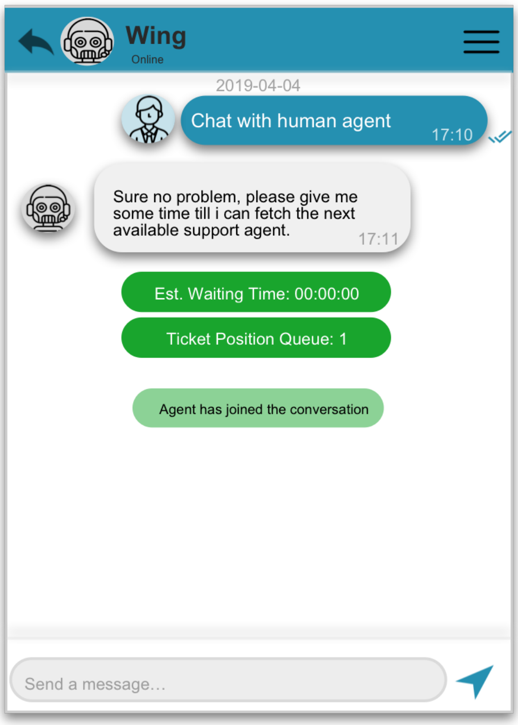
Figure 4.2: Concept and design of the handover process reaction. The result (a) shows the pre-handover phase and the result (b) illustrats the wait-handover phase.

system should display that the handover process was successful, and the human agent joined the chat.

The second option is, as already mentioned that the system sends the issue via an email to the employee if that is so the chatbot should additionally collect some user-specific information and send this together with the issue to the human agent.

For a smooth handover flow, the system should also ensure that the chatbot provides the entire previous conversation to the human agent. Therefore the text messages should already be displayed in the human agent's chat window.

After the post-handover phase, the last method to complete the system is to find out the reason of the handover process. That is to trace back to the original question. In this system, it is the message which triggers the handoff process and where the user requested help from a human agent. So it can be beneficial to trace each human help call back to the origin handover source message, and with the help of this method, the system improves the conversation. Figure 4.3 illustrates the conceptional prototype design for the post-handover phase.



2019-04-04

Chat with human agent 17:10

Sure no problem, please give me some time till i can fetch the next available support agent. 17:11

Est. Waiting Time: 00:00:00

Ticket Position Queue: 1

Agent has joined the conversation

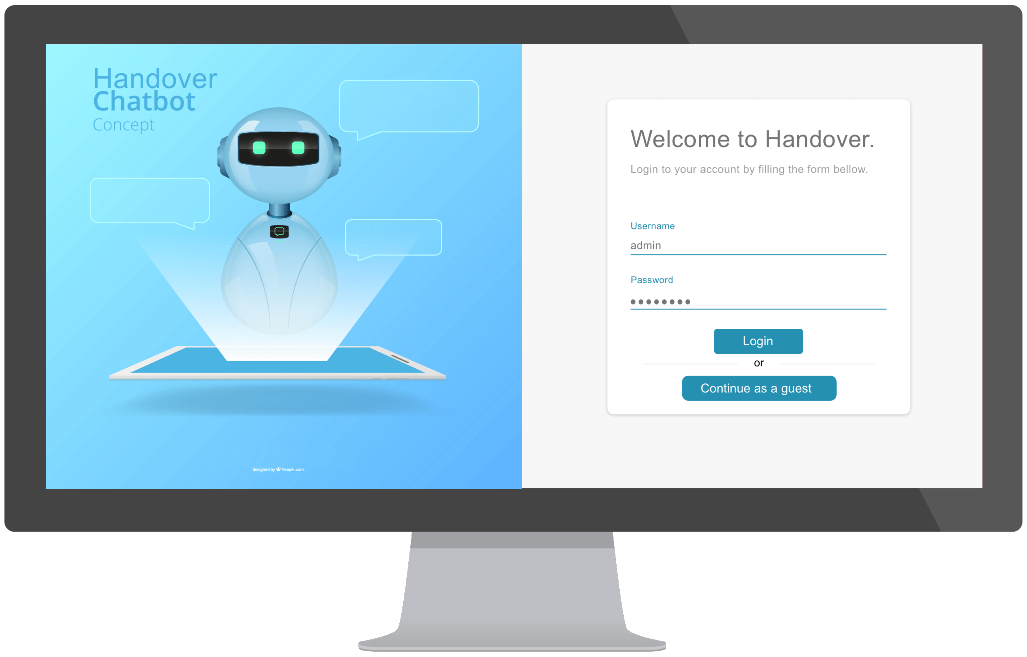
Figure 4.3: Concept and design of the handover process reaction of the post-handover phase.

4.5 Technical Design

This section focuses on the design and the required functionalities for the chatbot and the web application. Therefore the workflow describes the functions and the concept created concerning the requirements.

4.5.1 Web-Application

The web application should fulfil two major functionalities. At first, the integration and visualization of the dialog messenger (chatbot) for the communication between the user and the chatbot. Second, the administration of the handover tickets with the help of the human agent. Therefore the web application has to provide user management and a ticket management functionality. For administration tasks, the human agent can log in (Figure 4.4) and can edit support request tickets or can enter a conversation that is currently taking place. The conceptional design of the administration view is visualized in Figure 4.6. The left table shows the support tickets where the human agent can send the response for the problem by email. To the right, is the list which visualizes the live chat conversations where the human agent can take over the conversation and join the chat. Figure 4.5 illustrates the user page and the message box with the chatbot dialog.



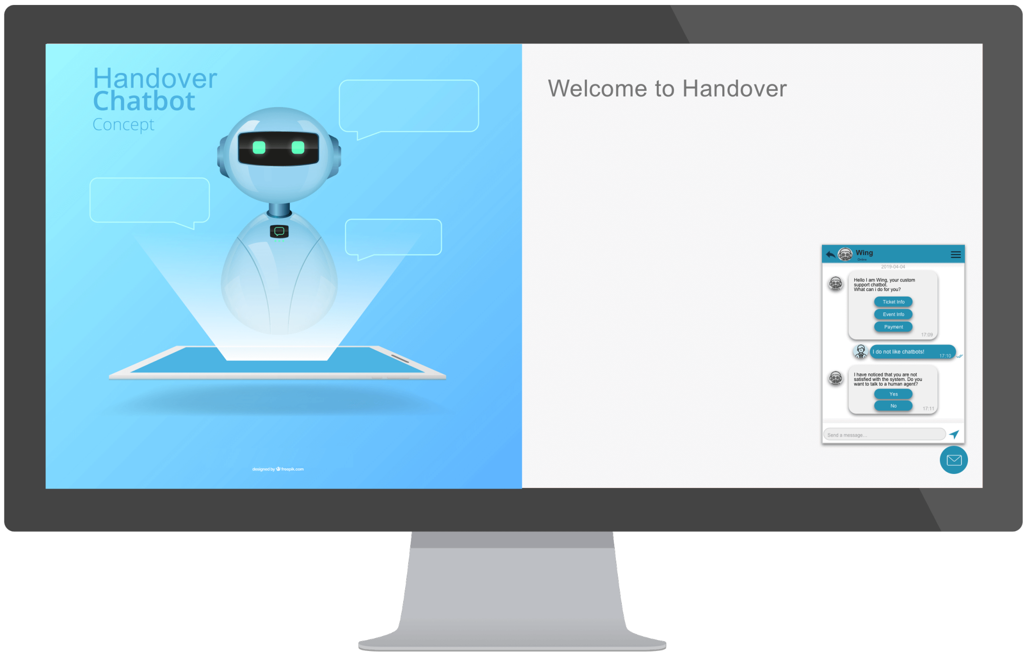
Welcome to Handover.

Login to your account by filling the form bellow.

Username: admin

Password: ••••••••

Figure 4.4: Conceptional Design: Login page of the web application.



Welcome to Handover

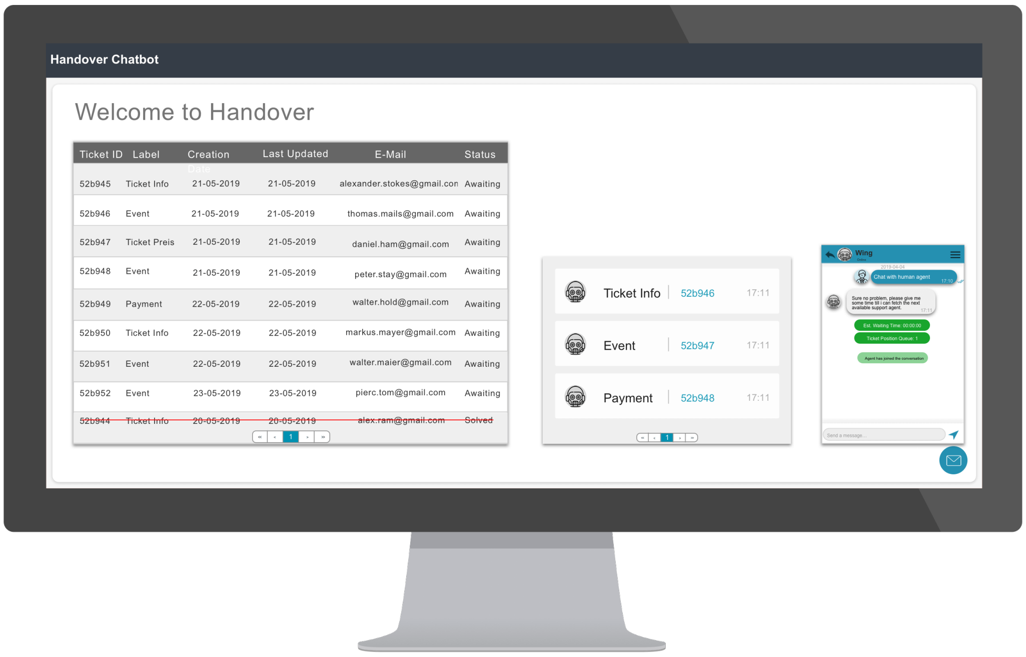
2019-04-04

Hello I am Wing, your custom support chatbot.
What can i do for you? 17:09

I do not like chatbots! 17:10

I have noticed that you are not satisfied with the system. Do you want to talk to a human agent? 17:11

Figure 4.5: Conceptional Design: User page of the web application.



Ticket ID	Label	Creation Date	Last Updated	E-Mail	Status
52b945	Ticket Info	21-05-2019	21-05-2019	alexander.stokes@gmail.con	Awaiting
52b946	Event	21-05-2019	21-05-2019	thomas.mails@gmail.com	Awaiting
52b947	Ticket Preis	21-05-2019	21-05-2019	daniel.ham@gmail.com	Awaiting
52b948	Event	21-05-2019	21-05-2019	peter.stay@gmail.com	Awaiting
52b949	Payment	22-05-2019	22-05-2019	walter.hold@gmail.com	Awaiting
52b950	Ticket Info	22-05-2019	22-05-2019	markus.mayer@gmail.com	Awaiting
52b951	Event	22-05-2019	22-05-2019	walter.maier@gmail.com	Awaiting
52b952	Event	23-05-2019	23-05-2019	pierc.tom@gmail.com	Awaiting
~~52b944~~	~~Ticket Info~~	~~20-05-2019~~	~~20-05-2019~~	~~alex.ram@gmail.com~~	~~Solved~~

Ticket Info	52b946	17:11
Event	52b947	17:11
Payment	52b948	17:11

Chat with human agent 17:10

Sure no problem, please give me some time till i can fetch the next available support agent. 17:11

Est. Waiting Time: 00:00:00

Ticket Position Queue: 1

Agent has joined the conversation

Figure 4.6: Conceptional Design: Human agent page of the web application.

4.5.2 Chatbot Agent

The chatbot should simulate an online ticketing system, where the bot can answer support questions. For the initial conversation, there will be a rough overview of some example requests, and the user can choose from the given examples or enter a sentence or phrase. The user's input message triggers the application *Dialogflow* backend.

The environment is built of a few different components. The user message is a dynamic input which the chatbot agent can receive at any given time. The representation of the message consists of the original text message sent by the user and the time the message was sent. The system transfers the message to the NLP, where the corresponding response for the message will be generated. The chatbot reply, which also consists of the answer text message from the system and the time the reply was sent, can add some possible answering choices (e.g., *Do you want to talk with a human agent?* or *Continue conversation*) for the user, but the chatbot can only read the user messages. A visual representation of the chatbot agent's environment is demonstrated in Figure 4.7.

4.5.3 Dialogflow API

Dialogflow uses a web-based developer environment to configure the chatbot environment. The framework is used for the human–computer interaction based on *Natural Language Processing*, in particular the text analyzer for the user input. The system is build with one *agent* and multiple *intents* as seen in Figure 4.8.

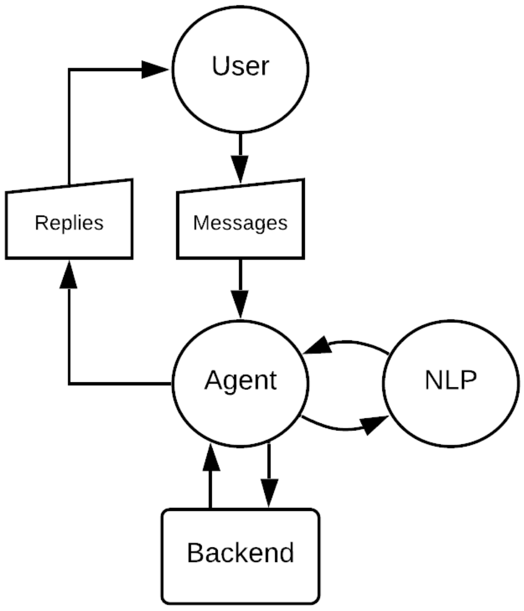


Figure 4.7: The chatbot agent's environment.

Agent

The system uses the dialogflow agent for the text understanding and the categorising of the user's input message. It describes the *Natural Language Understanding* (NLU) module, which translates the text from the request into actionable data. Therefore *intents* represent possible user messages and the translation starts after the message matches a specific *intent* from the dialogflow agent module.

Intent

Intents form the base of a chatbot system, which reflect each possible user input. In Dialogflow these consist of the following parts:

- the *intent name* for identifying and matching the *intent*,
- the *input* and *output context*, which can be used to remember parameter values, for passing the values between *intents*,
- *events*, which allow to call an *intent* without matching any user input,
- *phrases*, which will trigger the *intent*,
- *action* and *parameters*, define how the system extracts the user utterances from the relevant information (parameters) (e.g., ticket category or date),
- *response* from the chatbot and
- the *fulfillment*, the integration of an external *action* for this *intent.*

If an *intent* matches the user request *Dialogflow* delivers a response back to the user. This response can be a simple text or a *fulfillment* response that includes information from the handover system. The answers that the chatbot should give are managed in the

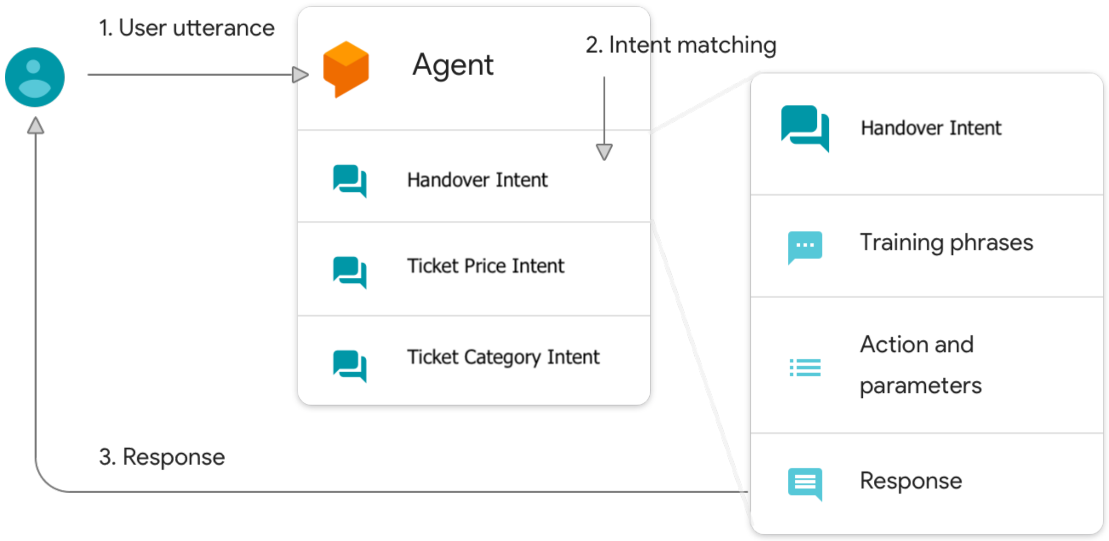


Figure 4.8: Schematic example of how *Dialogflow* matches user input to an intent and responds [19].

response section. Multiple responses can be created and sent as individual messages in the chat. In each text response, several executions can be specified, which the developer can select. The intent can be individually configured depending on the desired platform (e.g., the chatbot should response with additionally suggestions (Yes or No) when it is used for the *Google Assistant*). The default output is always used if no specific platform is defined in the intent's response section [22].

Entity

Entities simplify and improve the recognition of a suitable intent. An *entity* consists of several entries that are named in *Dialogflow* synonyms. These entries determine which input will trigger the *entity*. For example, using time is a common use of entities. With the use of intents, for determining the motivation behind a particular user message, entities should find out a piece of specific information in the user's request. In this manner, each corresponding entity reflects all the essential data that the system receives from the user's message [21].

In the case of this project, the *ticket category* is used for the support question conversation. Additionally the *entity* named *handover* is the corresponding trigger for the mentioned methods in Section 4.3 from every handover detection.

Webhook

A webhook is a process for communicating between server endpoints. An invocation consists of an HTTP-Post request that sends data to the webhook. After this has processed the data, it will be sent back via a response. In Dialogflow, the integration of a webhook is possible to further process data entered by the user or to perform functions that can not be mapped by Dialogflow. In Dialogflow, fulfilment is a code that represents a webhook that allows the system, with the extracted information, to generate responses or trigger actions on the handover backend. The handover service returns a response in JSON format [20].

4.5.4 Aylien API

For the user sentiment as described in Section 4.3, the system should detect the frustration level of the user messages. Therefore the Sentiment Analysis *AYLIEN*[1] API should be integrated as an additional module into the system. The API provides a specific Text Analysis API for detecting the sentiment from the textual content. The analyzer categorizes the tone into positive, neutral or negative different polarities. Additionally, the API can divide the text into subjective (the user's opinion) or objective (meaning a fact).

4.5.5 Dandelion API

Additionally, to the user sentiment (Section 4.3), the system should also provide the handover repeated phrase functionality. Therefore, the chatbot system should add the Text Similarity API from *Dandelion*[2] as a module. With this API, it is possible to compare two sentences and get a score of their semantic similarity. It works even if the two sentences do not contain similar words.

4.6 System Design

This Section describes the overall abstract system architecture of the project with all components, each solving one particular problem. The system design also includes an interface description and the communication between each component. As can be seen in Figure 4.9 the system consists of three main components: *Client + Chatbot Interface*, *Server*, *Database*. The component labelled as *Client* is the generic client program which makes use of the chatbot. It also provides an interface where the human agent can join and take over the conversation. So, therefore the handover process has to be initialized with the subcomponents of the *Server* backend.In particular the *Server* component has three submodules to achieve the handover process. The first one is the NLP *Dialogflow* module, which exchanges the data messages between the user and the chatbot with a *webhook*. The second module is the *AYLIEN* interface, which uses the input message to extract the user emotion. Moreover, the last submodule is the *Dandelion* interface, which compares the similarity of the latest user messages. The last component, the *Database*, is used for the storage of the handover data and authentication information of the human agent.

Diagram 4.10 illustrates the abstract system architecture of the mentioned components from Figure 4.9 in more detail. It shows the complete application and gives an overview of all used components and how they are related to each other. The client component consists of the chatbot, which is integrated into the web application. For communication between the user and the chatbot, the system must be extended by a

[1] https://aylien.com/text-api/

[2] https://dandelion.eu/docs/api/datatxt/sim/v1/

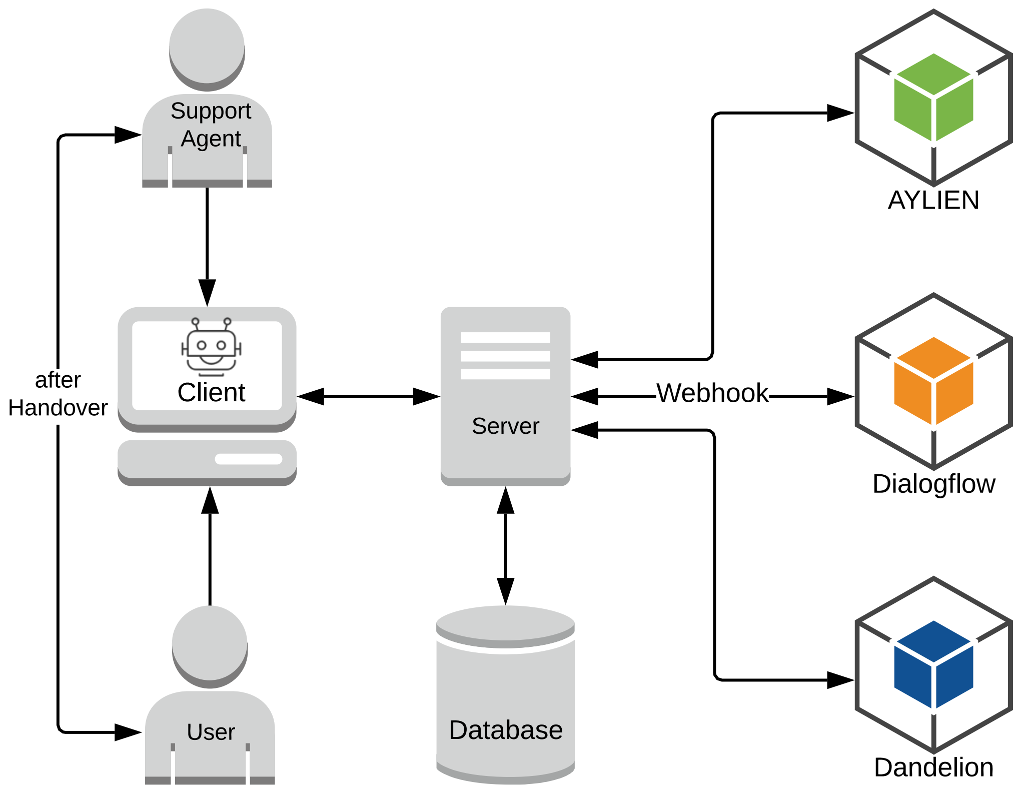


Figure 4.9: Conceptional Project Components.

component, the *Dialog Component*, that exchanges messages with the *Dialogflow API*. The component illustrates the dialog messenger, where the communication partners, the user, the human agent or the chatbot, exchange messages. The *User Input Service* creates the messages and will be structured in the *Dialog Component* as a chat dialog.

The *User Input Service* represents the input messages from the user and the *Dialogflow Component* represents the chatbot response messages created from the *Dialogflow API*. After the user types in a message the *Dialogflow Component* expects a request message from the NLP. The system uses the *FAQ Component* to provide the ticket information system (e.g., ticket price or ticket availability information).

Afterwards *Dialogflow* transfers the given user message via the *webhook* to the *Dialogflow Service*. There, the *Handover Module* tries to find out the frustration level of the user's message with the four approaches as described in Section 4.3. With the help of the *AYLIEN* API, the system can detect the sentiment of the user's input message. Additionally, the system uses the *Dandelion* API for the similarity check of the chat history. The communication between the system and both API's is via REST. Each of the four modules can trigger the handover process.

In the last step, the conversation is taken over by a human agent. For that the

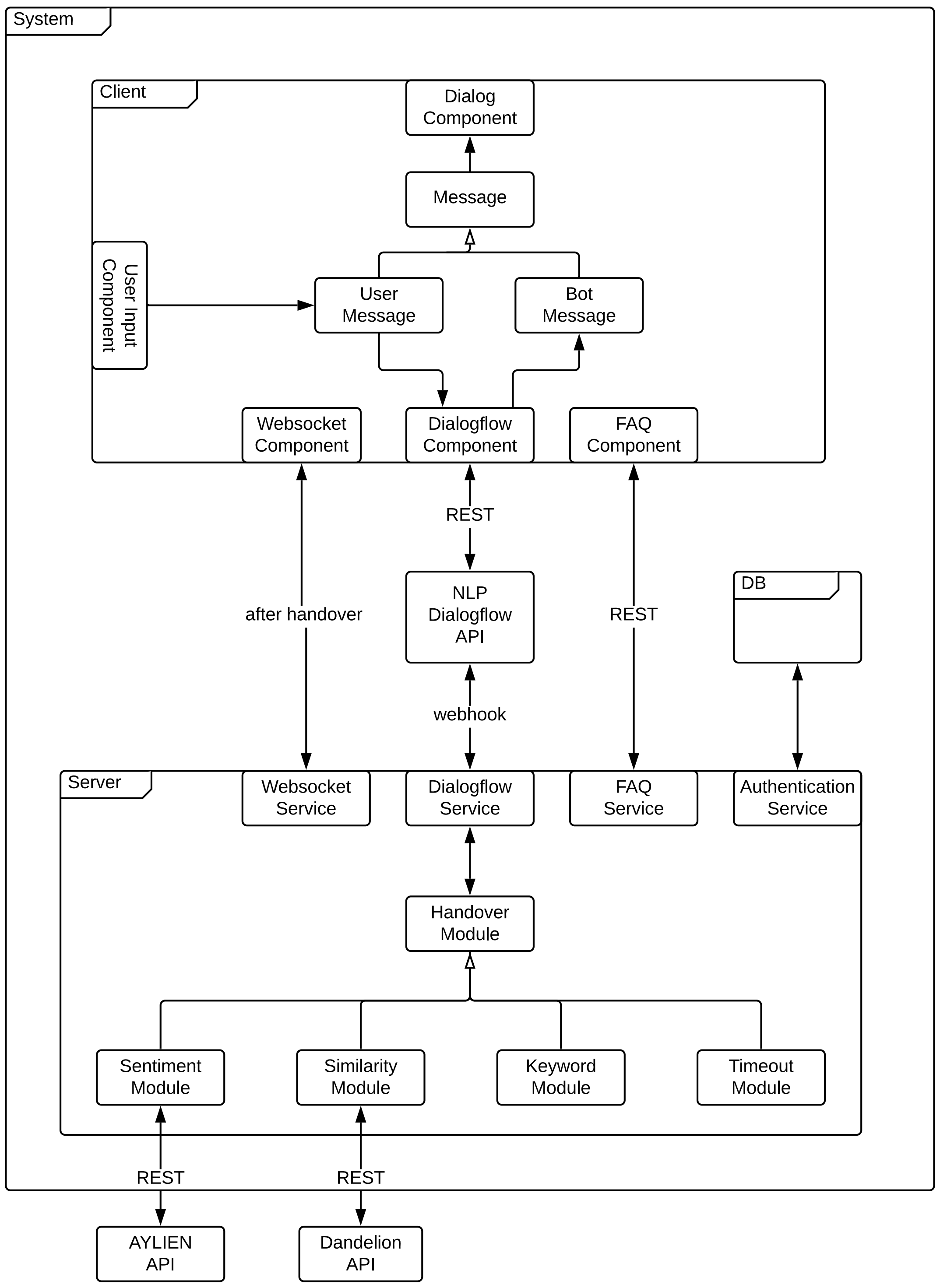


Figure 4.10: Conceptional Project Relations.

Authentication Service allows the human agent to log in as already mentioned in Section 4.5.1. After the human agent has chosen to join a conversation, the *Websocket Service* connects the human agent with the frustrated user and the chatbot is excluded from the conversation. The human agent can now solve the problem that the machine could not solve.

Chapter 5

Technical Implementation

In this chapter, the used methods for implementing the given requirements will be discussed. It focuses on how the system was technically implemented and describes which methods were used for development.

5.1 Technology Stack

Attached are the technology used to implement the system.

- JHipster[1]: The *JHipster* framework is a free and open-source application generator which was used for the project architecture.
- Angular: The TypeScript[2]-based open-source web application framework is used for the frontend of the chatbot application.
- Spring: The project's backend is built with the *Spring* framework, which is an application framework and inversion of control container for the *Java* platform and has extensions for building a web application's backend.
- MySQL[3] database is an open-source relational database management system It is used for the authentication data from the support agent and the support tickets.
- Dialogflow API: The NLP based human-computer interaction technology is used for the system's conversational interface.
- AYLIEN Text Analysis API: The Natural Language Processing API is used for the emotional detection of the handover module.
- Dandelion API: The last used technology is the Dandelion API, which is a Semantic Text Analytics service. The analyzer is used for the similarity detection of the handover module.

The server-side implementation of the project is written in *Java* because of the integration of the *Spring* Framework and the easy integration of the desired *REST* functionality. Because of the use of the *Angular* framework, the client side component is mainly written in *TypeScript*. Another reason why the client is written in *TypeScript* is the large community, the open source libraries and the easy setup of the web service. *MySQL* is

[1]https://www.jhipster.tech
[2]https://www.typescriptlang.org
[3]https://www.mysql.com/: The *MySQL*

the preferred database management system because it can be easily integrated with the *JHipster* framework. The *AYLIEN* and the *Dandelion* API's provide a *Java SDK* and allow the easy integration in the existing system.

5.2 System Architecture

Chapter 4 already mentioned that the handover module is the core and offers the user the frustration-free transition to a human agent. The following chapter describes the implementation of the user client with the chatbot, the human agent handover interface, its communication with the handover detection modules, the server-side implementation and the natural language processing integration with the *Dialogflow* API.

5.2.1 Client

Figure 5.1 shows the structure of the web application's client classes of the project. The *ClientMainComponent* class is the entry point of the user interface and consists of the dialog messenger.

The messenger contains of two components the *MessengeItemComponent* and the *MessengeFormComponent*. The *MessengeItemComponent* visualizes the messages between the chatbot and the users, whereas the *MessengeFormComponent* is used for visualisation of the user input form. The *DialogflowComponent* is the interface between the web application and *Dialogflow*:

```
@Injectable()
export class DialogflowComponent {
    private baseURL = DIALOGFLOW_URL;
    private token = DIALOGFLOW_TOKEN;
    private resourceUrl = SERVER_API_URL + 'api';

    constructor(private http: HttpClient) {
    }

    public getResponse(query: string): Observable<any> {
        const data = { query, lang: 'de', sessionId: '12345' };
        return this.http.post(`${this.baseURL}`, data, this.getHeaders());
    }

    private getHeaders(): { headers: HttpHeaders } {
        return { headers: new HttpHeaders({
            Authorization: `Bearer ${this.token}`})};
    }

    public getSupportTickets(): Observable<EntityArrayResponseType> {
        return this.http.get<SupportTicket[]>(
        `${this.resourceUrl}/supportTickets`, { observe: 'response' });
    }
}
```

Therefore the *DialogflowComponent* posts the user message to *Dialogflow* and then *Dialogflow* responses with the chatbot message. When *Dialogflow* detects a frustrating situation, and the user wants to talk to a human agent, the *WebSocketService* connects the user with the human agent. Therefore the *WebSocketService* provides the configuration possibilities, like setting ports for the websocket server or exchange messages with the human agent.

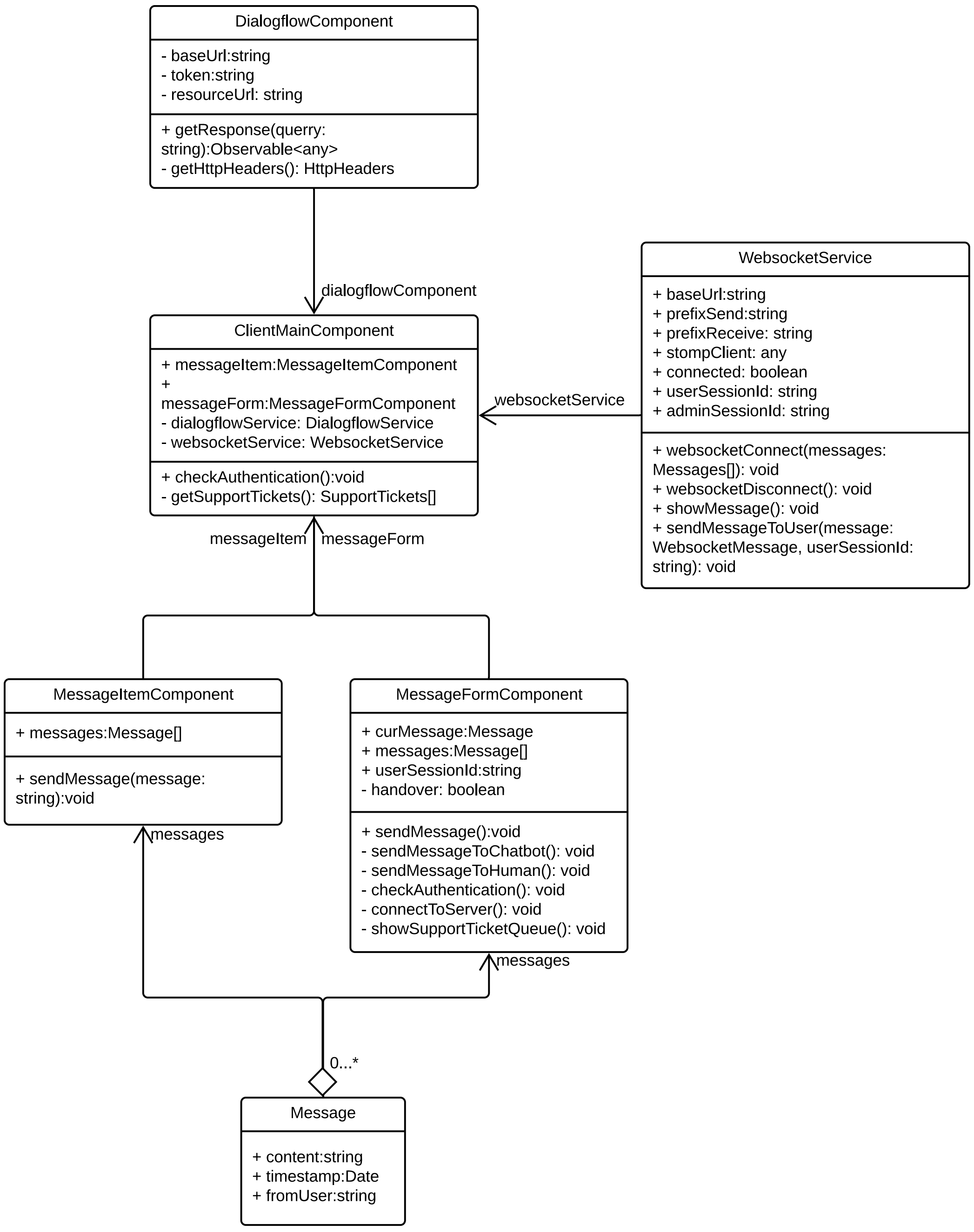

DialogflowComponent
- baseUrl:string
- token:string
- resourceUrl: string
+ getResponse(querry: string):Observable<any>
- getHttpHeaders(): HttpHeaders
dialogflowComponent
ClientMainComponent
+ messageItem:MessageItemComponent
+ messageForm:MessageFormComponent
- dialogflowService: DialogflowService
- websocketService: WebsocketService
+ checkAuthentication():void
- getSupportTickets(): SupportTickets[]
websocketService
WebsocketService
+ baseUrl:string
+ prefixSend:string
+ prefixReceive: string
+ stompClient: any
+ connected: boolean
+ userSessionId: string
+ adminSessionId: string
+ websocketConnect(messages: Messages[]): void
+ websocketDisconnect(): void
+ showMessage(): void
+ sendMessageToUser(message: WebsocketMessage, userSessionId: string): void
messageItem
messageForm
MessageItemComponent
+ messages:Message[]
+ sendMessage(message: string):void
MessageFormComponent
+ curMessage:Message
+ messages:Message[]
+ userSessionId:string
- handover: boolean
+ sendMessage():void
- sendMessageToChatbot(): void
- sendMessageToHuman(): void
- checkAuthentication(): void
- connectToServer(): void
- showSupportTicketQueue(): void
messages
messages
0...*
Message
+ content:string
+ timestamp:Date
+ fromUser:string

Figure 5.1: UML diagram of the system's client.

The *sendMessage* function generates a new message, with the user input string and adds it to the message array:

```
public sendMessage(): void {
    this.clearSuggestions();

    if (this.message.content !== '') {
        this.message.timestamp = new Date();
        this.messages.push(this.message);

        if (this.handover) {
            this.sendMessageToHuman();
        } else {
            this.sendMessageToChatbot();
        }
        this.message = new Message('', 'user');
    }
}
```

Afterwards the *MessengeItemComponent* lists the array messages on the chat messenger. If no handover process is detected the *sendMessageToChatbot* function sends the user input to the NLP:

```
private sendMessageToChatbot() {
    this.dialogFlowService.getResponse(this.message.content).subscribe(res => {
        this.messages.push(new Message(res.result.fulfillment.speech,
                                        'bot', res.timestamp));

        if (res.result.action === 'HandoverIntent.HandoverIntent-yes') {
            this.handover = true;
            this.connectToServer();
        }
    });
}
```

If the result action of the NLP request is *HandoverIntent.HandoverIntent-yes* the system starts the handover process. Therefore the *connectToServer* function connects the user with the websocket server:

```
private connectToServer() {
    if (!this.webSocketService.connected) {
        this.webSocketService.webSocketConnect(this.messages);
        this.webSocketService.connected = true;
    }
}
```

When the handover process has started, the user can communicate with the human agent. Therefore, the *sendMessageToHuman* function sends the messages to the human agent and not to the NLP any more:

```
private sendMessageToHuman() {
    const websocketMessage = new WebSocketMessage(this.message.content, this.
    websocketService.userSessionId);
    this.websocketService.sendMessageToUser(websocketMessage, this.userSessionId);
}
```

This is where the chatbot system is no longer required and the successful handover process is complete.

5.2.2 Server

For the system's server-side implementation Figure 5.2 illustrates the structure of the classes.

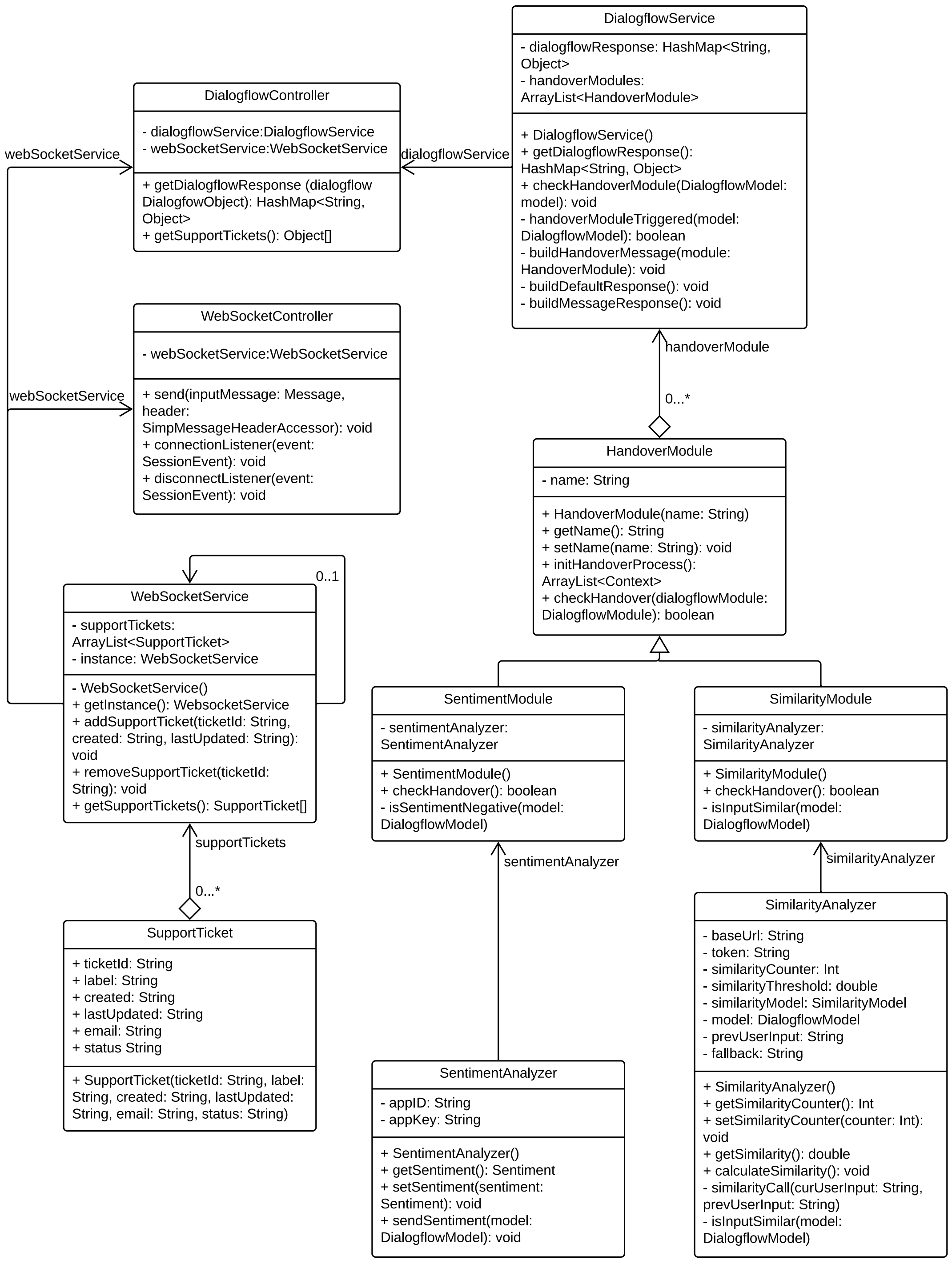

DialogflowService
- dialogflowResponse: HashMap<String, Object>
- handoverModules: ArrayList<HandoverModule>
+ DialogflowService()
+ getDialogflowResponse(): HashMap<String, Object>
+ checkHandoverModule(DialogflowModel: model): void
- handoverModuleTriggered(model: DialogflowModel): boolean
- buildHandoverMessage(module: HandoverModule): void
- buildDefaultResponse(): void
- buildMessageResponse(): void
DialogflowController
- dialogflowService:DialogflowService
- webSocketService:WebSocketService
+ getDialogflowResponse (dialogflow DialogfowObject): HashMap<String, Object>
+ getSupportTickets(): Object[]
webSocketService
dialogflowService
WebSocketController
- webSocketService:WebSocketService
+ send(inputMessage: Message, header: SimpMessageHeaderAccessor): void
+ connectionListener(event: SessionEvent): void
+ disconnectListener(event: SessionEvent): void
webSocketService
handoverModule
0...*
HandoverModule
- name: String
+ HandoverModule(name: String)
+ getName(): String
+ setName(name: String): void
+ initHandoverProcess(): ArrayList<Context>
+ checkHandover(dialogflowModule: DialogflowModule): boolean
0..1
WebSocketService
- supportTickets: ArrayList<SupportTicket>
- instance: WebSocketService
- WebSocketService()
+ getInstance(): WebsocketService
+ addSupportTicket(ticketId: String, created: String, lastUpdated: String): void
+ removeSupportTicket(ticketId: String): void
+ getSupportTickets(): SupportTicket[]
SentimentModule
- sentimentAnalyzer: SentimentAnalyzer
+ SentimentModule()
+ checkHandover(): boolean
- isSentimentNegative(model: DialogflowModel)
SimilarityModule
- similarityAnalyzer: SimilarityAnalyzer
+ SimilarityModule()
+ checkHandover(): boolean
- isInputSimilar(model: DialogflowModel)
supportTickets
0...*
sentimentAnalyzer
similarityAnalyzer
SupportTicket
+ ticketId: String
+ label: String
+ created: String
+ lastUpdated: String
+ email: String
+ status String
+ SupportTicket(ticketId: String, label: String, created: String, lastUpdated: String, email: String, status: String)
SimilarityAnalyzer
- baseUrl: String
- token: String
- similarityCounter: Int
- similarityThreshold: double
- similarityModel: SimilarityModel
- model: DialogflowModel
- prevUserInput: String
- fallback: String
+ SimilarityAnalyzer()
+ getSimilarityCounter(): Int
+ setSimilarityCounter(counter: Int): void
+ getSimilarity(): double
+ calculateSimilarity(): void
- similarityCall(curUserInput: String, prevUserInput: String)
- isInputSimilar(model: DialogflowModel)
SentimentAnalyzer
- appID: String
- appKey: String
+ SentimentAnalyzer()
+ getSentiment(): Sentiment
+ setSentiment(sentiment: Sentiment): void
+ sendSentiment(model: DialogflowModel): void

Figure 5.2: UML diagram of the system's server.

The *DialogflowController* is the entry point for the NLP interface. The controller uses the *DialogflowService* for the handover detection and the creation of the response message for the *Dialogflow* API:

```java
@RestController
@RequestMapping("/api")
public class DialogFlowController {
    private DialogFlowService dialogFlowService;
    private WebSocketService websocketService;

    public DialogFlowController(TicketService service) {
        dialogFlowService = new DialogFlowService();
        dialogFlowService.initService(service);
        websocketService = WebSocketService.getInstance();
    }

    @PostMapping(path = "/getDialogflowResponse")
    @Timed
    public HashMap<String, Object> getDialogflowResponse(@RequestBody
    DialogFlowModel model) {
        dialogFlowService.checkHandoverModules(model);
        return dialogFlowService.getDialogFlowResponse();
    }
}
```

The server provides a webhook interface for the corresponding handover detection result, which the service transmits to the Dialogflow endpoint. The system communication is based on RESTful Web services. Therefore, the system provides a communication service the *getDialogflowResponse*. The JSON structure of each Dialogflow request to the server webhook:.

```json
{
  "responseId": "4fcc3a3b-de64-4c67-83f5-b9a72b1430d2-e6604cc1",
  "queryResult": {
    "queryText": "Ich mag keine Chatbots",
    "action": "HandoverIntent.HandoverIntent-yes",
    "intent": {
      "name": "3f61d509-7f75-4bfd-b41e-140ba057a5e1",
      "displayName": "Handover Intent",
    },
}
```

Important for the handover system is the user input message (*queryText*) the handover trigger (*action*) and the triggered intent. With the Dialogflow information, the handover server creates a response message. The JSON response message of the HTTP-post call is based on the input message from the user:

```json
{
  "outputContexts": [
    {
      "name": "/handoverintent-followup",
      "lifespanCount": 1,
    }
  ],
  "fulfillmentText": "Wollen Sie einen unserer Mitarbeiter kontaktieren?"
}
```

If the action attribute of the message is a Handover intent, the output message (fulfillmentText) questions the user if the system should begin the handover process also, the server responses with the followed intent (outputContexts) after the handover process. If the system detects a frustrating situation from the input message, the system detects the handover situation and calls the *HanodverModule*. The application receives a HTTP-post request from *Dialogflow* in the form of the response from the user message's request that matched the corresponding *Intent*.

The *DialogflowService* provides the *checkHandoverModules* function, which iterates over all possible handover cases and triggers the handover module if the system detects a frustrating user input:

```
public void checkHandoverModules(DialogFlowModel model) {
    try {
        if (!isHandoverModuleTriggered(model)) {
            buildDefaultResponse(model);
        }
    } catch (Exception e) { e.printStackTrace(); }
}

private boolean isHandoverModuleTriggered(DialogFlowModel model){
    for (HandoverModule module : handoverModules) {
        if (module.checkHandover(model)) {
            buildHandoverResponse(module);
            return true;
        }
    }
    return false;
}
```

Also the *isHandoverModuleTriggered* function iterates over all modules and builds the specific handover response for the *Dialogflow* API. For the detection of a handover process the *handoverModule* stores a list of *HandoverModule*, which illustrates all possible handover scenarios.

5.2.3 Handover Detection

The *HandoverModule* class is the base class of all handover modules:

```
public abstract class HandoverModule {
    private String name;

    HandoverModule(String name) {
        setName(name);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public abstract boolean checkHandover(DialogFlowModel model);
```

```
        public ArrayList<OutputContext> initHandoverProcess() {
            System.out.println(name + " module was triggered!");
            ArrayList<OutputContext> outputContexts = new ArrayList<>();
            outputContexts.add(new OutputContext(DIALOGFLOW_FOLLOWED_INTENT));
            return outputContexts;
        }
    }
```

So every module must extend from that abstract class this ensures that each module implements the required functions. Every module implements two base functions the *checkHandover* function and the *initHandoverProcess* function. The *checkHandover* function checks if the defined conditions, for a handover process, are fulfilled.

Afterwards, the *initHandoverProcess* function initializes the output context (part of the response for the Dialogflow NLP) and returns the *Followed Intent*, which is the intent for the Dialogflow API to hand the conversation over to a human agent.

The next sections describes the different kinds of handover modules and there functionaries.

Sentiment Module

Figure 5.3 illustrates the handover process detection with the sentiment analysis integration and gives an overview of the used components and how they are related to each other. As seen in the figure, the initial situation is user input. So if the user types a message the *DialogMessageService* will transfer it with an HTTP-post request to the DialogFlow system. There the *Hanodver Agent* will classify which intent should respond to the message. After the intent classification *Dialogflow* makes a request (webhook) to the Project with an HTTP-post request. The *DialogFlowRessource* manages the given request and send it to the *SentimentService*, which communicates with the *Aylien* API. The API responses with a sentiment data structure:

```
    {
        "polarity" : "positive",
        "subjectivity" : "subjective",
        "text" : "I love chatbots",
        "polarity_confidence" : "0.9886510372",
        "subjectivity_confidence" : "1"
    }
    {
        "polarity" : "negative",
        "subjectivity" : "subjective",
        "text" : "This chatbot is stupid",
        "polarity_confidence" : "0.9997755885",
        "subjectivity_confidence" : "1"
    }
    {
        "polarity" : "neutral",
        "subjectivity" : "subjective",
        "text" : "Neutral message",
        "polarity_confidence" : "0.5926589369",
        "subjectivity_confidence" : "0.6941109833"
    }
```

The *SentimentService* extracts the polarity (positive, negative or neutral) forms the sentiment response and sends it back to the *DialogFlowRessource*. The *DialogFlowRes-*

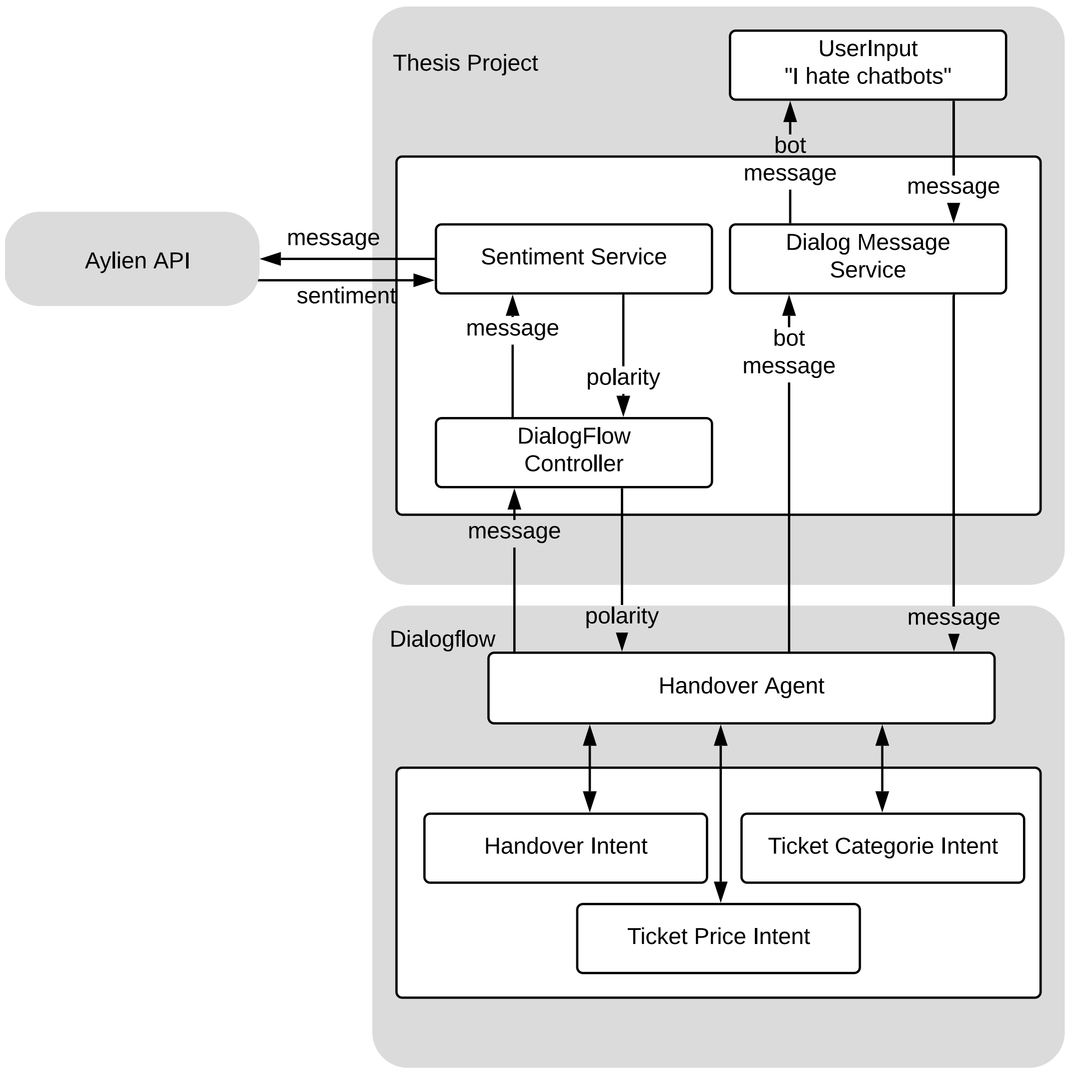


Figure 5.3: Sentiment Analysis Hanodover Detection.

source builds the response with the emotional information and send it to the *Handover Agent*. With the sentiment analysis, the agent can decide which intent should provide the response message for the user input and sends it back to the *DialogMessageService*. For the emotion detection/sentiment analysis, the *Aylien* API is used. The *sendMessage* functions calls the API:

```java
public void sendMessage(DialogFlowModel model) {
    String message = model.getQueryResult().getQueryText();
    try {
        builder.setText(message);
        setSentiment(client.sentiment(builder.build()));
    } catch (TextAPIException e) { e.printStackTrace(); }
}
```

Therefore the function sets the sentiment based on the response from the API. Every response has polarity information from the message. The polarity is calculated with the checkHandover function with the use of the polarity confidence value:

```
@Override
public boolean checkHandover(DialogFlowModel model) {
    return isSentimentNegative(model);
}

private boolean isSentimentNegative(DialogFlowModel model) {
    String polarity = sentimentAnalyzer.getSentiment().getPolarity();
    System.out.println("Polarity: " + polarity);
    sentimentAnalyzer.sendMessage(model);
    return polarity.equals("negative");
}
```

Is the absolute value closer to 1, the higher the probability is that the given message is either positive (1) or negative (-1). Additionally, every message can categorize in subjective (meaning it is reflecting the user's opinion) or objective (without personal opinion).

Similarity Module

The next handover module is the *SimilarityModule*. This module uses the *Dandelion* API. The function uses the current and the previous user message and compares them:

```
private boolean similarityCheck(String curUserInput, String prevUserInput) {
    HttpClient httpClient = HttpClientBuilder.create().build();
    HttpPost httpPost = new HttpPost(baseUrl);
    try {
        addParameter(prevUserInput, curUserInput, httpPost);
        HttpResponse response = httpClient.execute(httpPost);
        similarityModel = generateSimilarityObject(response);
        return true;
    } catch (IOException e) { e.printStackTrace(); }
    return false;
}
```

Therefore the HTTP call to the API endpoint responds with the similarity value:

```
{
  "timestamp": "2019-05-21T16:29:37",
  "lang": "en",
  "langConfidence": 1,
  "text1": "When is the next game?",
  "text2": "When will the next game be played?",
  "similarity": 0.7655
}
```

Is the similarity value greater than 0.5 the *similarityCounter* will be increased. The *SimilarityModule* calls the *similarityAnalyzer.getSimilarityCounter* function, which calculates the similarity with the *similarityCheck* function. The function *checkHandover* triggers the handover module when the *similarityCounter* is greater or equals two as shown in Program 5.1. So, if the user types in the same sentence or phrase three times in a row the handover module will be triggered.

Program 5.1: *SimilarityModule's* function to check the similarity handover module.

```java
@Override
public boolean checkHandover(DialogFlowModel dialogFlowDTO) {
    return isInputSimilar(dialogFlowDTO);
}

private boolean isInputSimilar(DialogFlowModel dialogFlowDTO) {
    similarityAnalyzer.setDialogFlowModel(dialogFlowDTO);
    similarityAnalyzer.calculateSimilarity();
    int similarityCounter = similarityAnalyzer.getSimilarityCounter();
    System.out.println("SimilarityCounter: " + similarityCounter);
    return similarityCounter >= 3;
}
```

5.2.4 Handover Reaction

After the handover detection, the system responds and prepares the handover to a human agent. Therefore, the chat dialog displays the current transition status and gives the user a visual feedback on how long the process lasts. In this phase the system connects the frustrated user with a human agent. After the agent joined the conversation the chatbot will be disconnect and the human agent takes over the chat dialogue. For the communication process between user and the human agent the system provides a *WebSocketController*.

WebSocketController

The *WebSocketController* manages the connection between users and human agents and the users can connect to the websocket with the *connectListener* function:

```java
@EventListener(SessionConnectEvent.class)
@Timed
public void connectListener(SessionConnectEvent event) {
    String dateString = simpleDateFormat.format(new Date());
    MessageHeaders headers = event.getMessage().getHeaders();
    String sessionId = SimpMessageHeaderAccessor.getSessionId(headers);
    webSocketService.addSupportTicket(sessionId, dateString, dateString);
}
```

And can disconnect from the websocket with the *disconnectListener* function:

```java
@EventListener(SessionDisconnectEvent.class)
public void disconnectListener(SessionDisconnectEvent event) {
    String dateString = simpleDateFormat.format(new Date());
    MessageHeaders headers = event.getMessage().getHeaders();
    String sessionId = SimpMessageHeaderAccessor.getSessionId(headers);
    webSocketService.removeSupportTicket(sessionId);
}
```

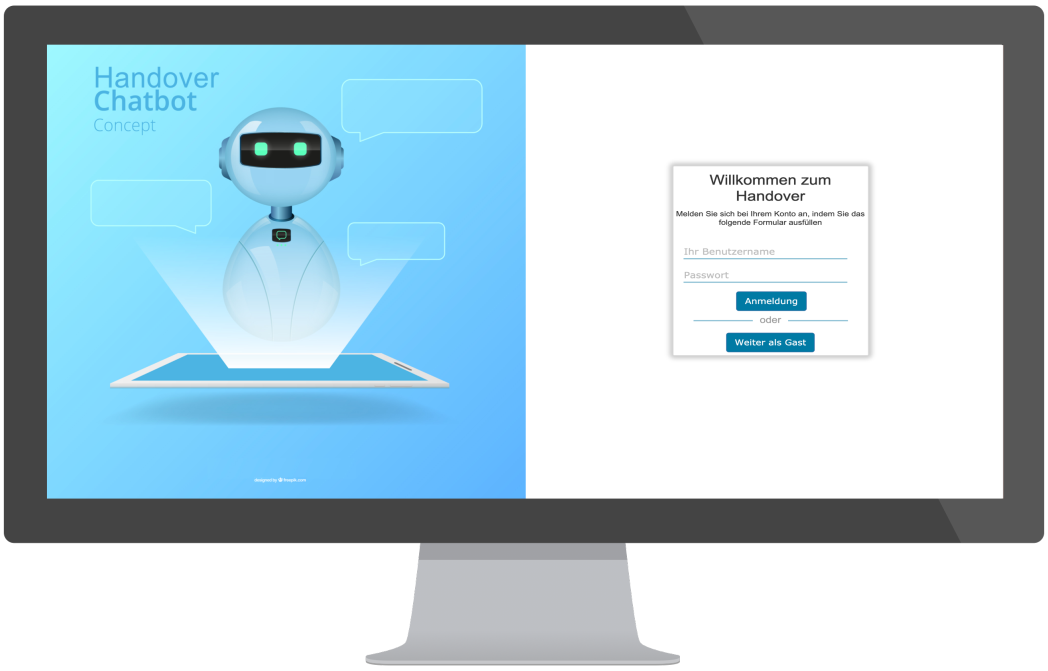
If a user connects to the websocket server the *addSupportTicket* function creates a new support ticket, which the client displays on the human agent's page. After the connection is established the communication partner can exchange messages via the *sendMessage* function:

```
@MessageMapping("/chat")
public void sendMessage(InputMessage inputMessage, SimpMessageHeaderAccessor
    headerAccessor) throws Exception {
    String timestamp = new SimpleDateFormat("HH:mm").format(new Date());
    OutputMessage outputMessage = new OutputMessage(headerAccessor.getSessionId(),
    inputMessage.getText(), timestamp);
    simpMessagingTemplate.convertAndSend("/queue/chats-" + inputMessage.getReceiver
    (), outputMessage);
}
```

The method creates an output message and forwards it to the stored user id. This method guarantees that the human agent communicates with the correct user. Additionally the user can quit the session or logout from the system. Therefore, if a user disconnects from the server the *removeSupportTicket* function clears the ticket with the user's session id and the human agent can choose the next support ticket.

5.3 Result

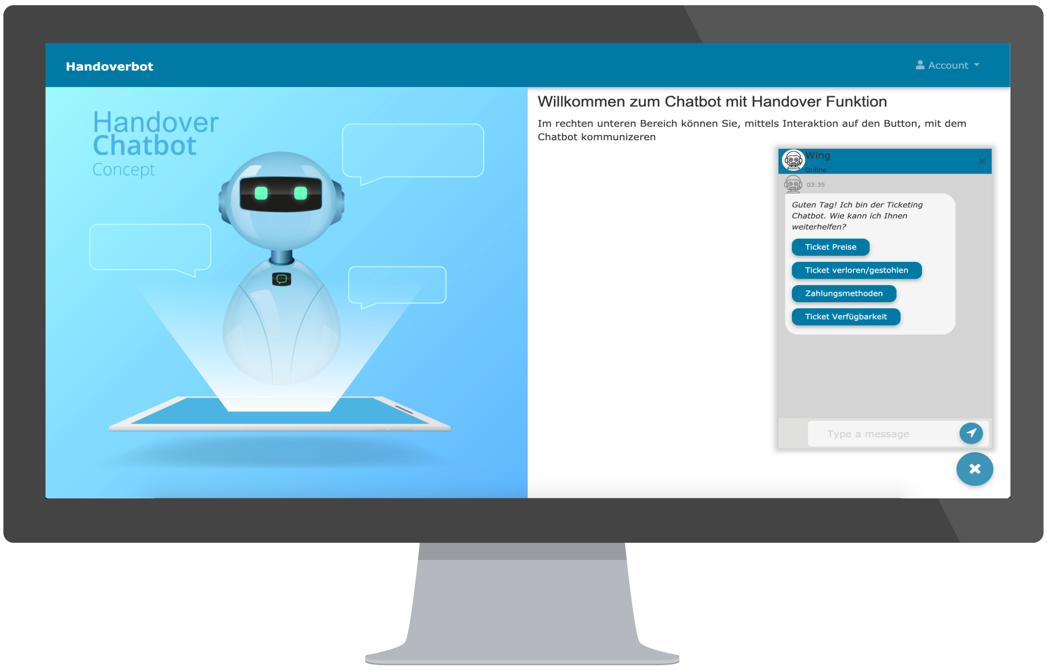
The implemented result of the system and how it looks like is seen in Figure 5.4, which illustrates the user authentication page. Figure 5.5 shows the user page and Figure 5.6 represents the *Human Agent Page*. The human agent has the option to select a user with the user table on the administration page. With this solution, the human agent can choose which user he wants to communicate. The concept, which has been defined at the planning state of the system in chapter 4, has been implemented in terms of functionality and design. The design was developed based on the conceptional project Design 4.5 template and hardly deviates from it.



Willkommen zum Handover

Melden Sie sich bei Ihrem Konto an, indem Sie das folgende Formular ausfüllen

Figure 5.4: Result of the thesis project: The *User Authentication Page*.

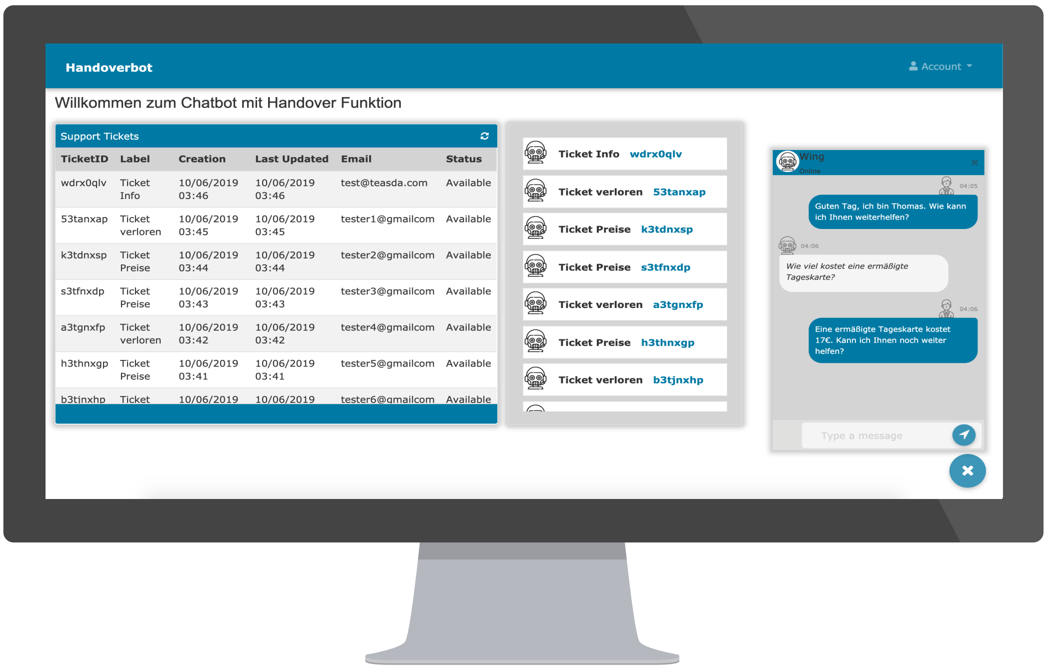


Willkommen zum Chatbot mit Handover Funktion

Im rechten unteren Bereich können Sie, mittels Interaktion auf den Button, mit dem Chatbot kommunizeren

Wing
Online
03:35
Guten Tag! Ich bin der Ticketing Chatbot. Wie kann ich Ihnen weiterhelfen?

Figure 5.5: Result of the thesis project: The *User Page*.



Willkommen zum Chatbot mit Handover Funktion

Support Tickets

TicketID	Label	Creation	Last Updated	Email	Status
wdrx0qlv	Ticket Info	10/06/2019 03:46	10/06/2019 03:46	test@teasda.com	Available
53tanxap	Ticket verloren	10/06/2019 03:45	10/06/2019 03:45	tester1@gmailcom	Available
k3tdnxsp	Ticket Preise	10/06/2019 03:44	10/06/2019 03:44	tester2@gmailcom	Available
s3tfnxdp	Ticket Preise	10/06/2019 03:43	10/06/2019 03:43	tester3@gmailcom	Available
a3tgnxfp	Ticket verloren	10/06/2019 03:42	10/06/2019 03:42	tester4@gmailcom	Available
h3thnxgp	Ticket Preise	10/06/2019 03:41	10/06/2019 03:41	tester5@gmailcom	Available
b3tjnxhp	Ticket	10/06/2019	10/06/2019	tester6@gmailcom	Available

Ticket Info	**wdrx0qlv**
Ticket verloren	**53tanxap**
Ticket Preise	**k3tdnxsp**
Ticket Preise	**s3tfnxdp**
Ticket verloren	**a3tgnxfp**
Ticket Preise	**h3thnxgp**
Ticket verloren	**b3tjnxhp**

Wing
Online
04:05
Guten Tag, ich bin Thomas. Wie kann ich Ihnen weiterhelfen?
04:06
Wie viel kostet eine ermäßigte Tageskarte?
04:06
Eine ermäßigte Tageskarte kostet 17€. Kann ich Ihnen noch weiter helfen?

Figure 5.6: Result of the thesis project: The *Human Agent Page*.

Chapter 6

Evaluation

This chapter focuses on the evaluation of the project. The evaluation consists of the fulfilment of the requirements, the usability analysis and the performance analysis.

6.1 Accomplishment of the Requirements

The following Section give a more detailed description of the requirements defined in chapter 4.1 and how these have been accomplished.

6.1.1 Web-Application and Chat Interface

The first requirement was the creation of the web application and the integration of a chat messaging interface. As seen in Section 4.5, the project provides a web-based chat messenger for the interaction with the chatbot and afterwards with the human agent. Because of the separation of two different user roles, the system provides authentication, and the chatbot can also be used if the user is not logged in. The chat interface provides a list of messages as well as an input field for new messages. The system currently processes responses immediately after receiving a message request from the user.

6.1.2 Communication of the Web-Application and Dialogflow

The next requirement was the integration and communication with a chatbot based on Natural Language Processing. As seen in the Section 4.5.3, for this task *Dialogflow* was integrated in the project. The communication between *Dialogflow* and the chatbot was accomplished with a webhook, Section 4.5.3, and the messages are sent via REST. Because the protocol is a simple HTTP request, it can easily be implemented and integrated into any client technology.

6.1.3 Handover Detection

In the mentioned Section 4.3 scenarios have been implemented for handover detection. This approaches define the triggers for a human handover and initiate the handover process. To evaluate whether the system triggers the handover process correctly and promptly, a Usability Test was performed seen in Section 6.2. Overall the detection works for different users as long as the communication language is German, and the user does not write sarcastic, in dialect or a word which has more than one meaning. The handover detection has a modular structure so that besides a transfer module can be easily integrated into the system. To achieve this feature, the *HandoverModule* class represents the base class of all handover detection modules, as mentioned in the Section 5.2.3.

6.1.4 Handover Reaction

The handover reaction requirement was fulfilled by the handover reaction phases, as illustrated in Section 4.4. The three phases were integrated into the project to provide an easy and frustration-free handover process. For the requirement, the first step after the system detects a frustrating situation was to inform the user that it is possible to talk with a human agent. If the user no longer intends to talk to the chatbot, the system then connects to the human agent. Therefore, the system should inform the user about the current process as well as the steps for transferring the call to a human agent. Each step gives the user a visual overview of the process and the estimated waiting time.

6.1.5 Human Agent Live Chat

The last project requirement was that the system should build a connection between the user and the human agent. Therefore, a live chat feature was implemented, and the connection was built with a websocket. Their advantage for a real-time capability because of the bidirectional communication technology makes them a useful technology for communication. With the websocket, the human agent can send messages to a specific user at any time and offers a fast and reliable solution for the live chat interface.

6.2 Usability Testing

Good user experience is an important aspect when it comes to communicating with a machine. Therefore a qualitative usability testing was necessary to get a useful evaluation for this project. To do this testing there where different approaches (record every communication with the chatbot, create a live stream session schedule all participants), or do a usability test. For this evaluation method, the *Usability Testing* [25] was chosen because it is a lean and agile approach and has good and simple steps to get the maximum result out of the system. Six volunteers (three female, three male) were picked either from the local university or family.

6.2.1 Scenarios

At first, the task was to find a scenario which will trigger the handover process. Therefore for every participant, two different scenarios are created.

Scenario I

Therefore the user had to ask for a FAQ Question, about how much a Ticket for the category “Reduced” is. Because this kind of ticket category is not implemented in the system, the chatbot could not provide the correct answer. That is the point where the system should detect that the user is frustrated because it is an unsolvable scenario. After that, the handover reaction process will be activated, and the system connects the user with the human agent. For the final task, the *Human Agent Chat*, a agent communicates with the participant after the handover process.

1. Login: The participant should open the user page using the login form.
2. Open Messenger: The participant should start the conversation with the chatbot.
3. Ticket Price: The participant should get the information about a specific ticket price for the category "Reduced".
4. Handover Trigger: The system should trigger the handover process and prepare the connection with the human agent.
5. Human Agent Chat: The participant should chat with the human agent.

Scenario II

The second scenario was to reclaim the money from a lost ticket. Therefore, the user should ask the system if it is possible to refund the money paid for the ticket. Since the system can not reimburse the money and can not provide the correct answer, the system should initiate the handover process and the system connects the user to the human agent. Also, for this scenario, a agent communicates with the participant after the handover process.

1. Login: The participant should open the user page using the login form.
2. Open Messenger: The participant should start the conversation with the chatbot.
3. Ticket Price: The participant should get the information about refund the money for a lost ticket.
4. Handover Trigger: The system should trigger the handover process and prepare the connection with the human agent.
5. Human Agent Chat: The participant should chat with the human agent.

6.2.2 Task Completion

The next step was to capture the task completion for each of the participants. The evaluation uses the following designation:

- If a user can perform the task quickly and with no trouble, the task gets a mark 1.
- If a user can perform the task but has some problems, the task get a mark 2.
- If a user could not perform a task, the task gets a mark 3.

Table 6.1 shows that every participant could solve the first (login) and second task (open messenger) without any problem. As expected nobody could solve the third task (Ticket Price for the category "Reduced"), because the system does not provide this category. After every participant tried to solve the task, the system started the handover detection process. The handover trigger value indicates how long it took the system to trigger the handover process after the chatbot was unable to answer the question. Because every user triggered a different handover module, seen in Table 6.3, the result value for the handover trigger differs. The last task was the handover reaction after the system detects and initialises the handover process. Therefore every participant, expected from *Tester#2*, answered the question if the system should connect the user with a human agent with yes. As the Table 6.1 shows, *Tester#2* could not solve the usability test because the system could not trigger the handover process. The reason was that the participant tried to communicate with the chatbot, and the handover trigger could not detect the frustration of the conversation.

	Login	*Open Messenger*	*Ticket Price*	*Handover Trigger*	*Human Agent Chat*
Tester#1	1	1	3	2	2
Tester#2	1	1	3	3	3
Tester#3	1	1	3	2	2
Tester#4	1	1	3	1	2
Tester#5	1	1	3	2	2
Tester#6	1	1	3	1	2

Table 6.1: Result of the usability test for the Scenario I

Table 6.2 shows similar to the first scenario that every participant could solve the first (login) and second task (open messenger) without any problem. The third task was also an unsolvable challenge, and every participant failed to solve this talk in order to trigger the handover process. The system starts with the handover detection after every participant tries to solve the problem in vain. The last challenge for the participants was similar to the first scenario. Therefore the handover reaction live chat connects the user with the human agent. As Table 6.2 illustrates that up to *Tester#6* everyone could solve the last task. In this case, the reason was that the system failed because it did not find the negative sentiment of the participant's messages.

	Login	*Open Messenger*	*Ticket Lost*	*Handover Trigger*	*Human Agent Chat*
Tester#1	1	1	3	1	2
Tester#2	1	1	3	1	2
Tester#3	1	1	3	2	2
Tester#4	1	1	3	1	2
Tester#5	1	1	3	1	2
Tester#6	1	1	3	3	3

Table 6.2: Result of the usability test for the Scenario II

Table 6.3 shows the used time for the handover detection and the triggered handover module. Taken from the table the *Sentiment Module* was the most triggered handover module, which suggests that the frustration level was highly correlated with the user's emotional input messages. On the other hand, the most time-consuming module was the *Keyword Module* because the participants did not consider that there was the possibility of this module. Except for one participant, every other one triggered a different module as the *Keyword Module*. Finally, the tasks marked with *X* in the table failed to trigger the handover process. Overall, two of the twelve processes fail to trigger a handover module, and the overall usability test has an achievement rate of 83.34%.

	Scenario I	*Scenario II*	*Handover Module I*	*Handover Module II*
Tester#1	44s	12s	Sentiment	Sentiment
Tester#2	X	35s	X	Sentiment
Tester#3	65s	44s	Keyword	Similarity
Tester#4	38s	20s	Similarity	Sentiment
Tester#5	51s	27s	Sentiment	Sentiment
Tester#6	34s	X	Sentiment	X

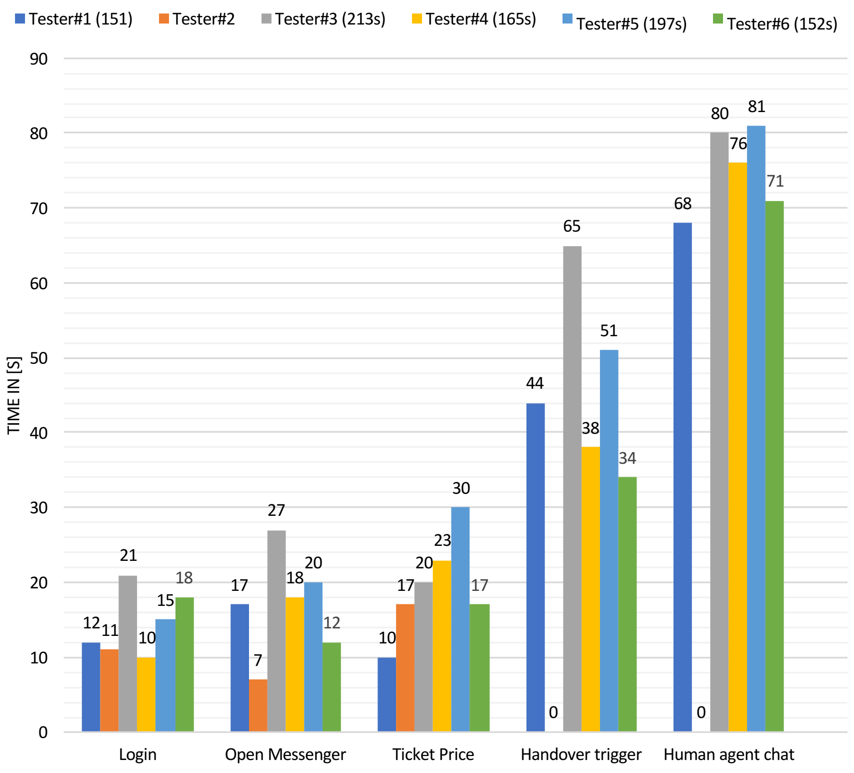
Table 6.3: Time in [s] for the handover detection

6.3 Performance Analysis

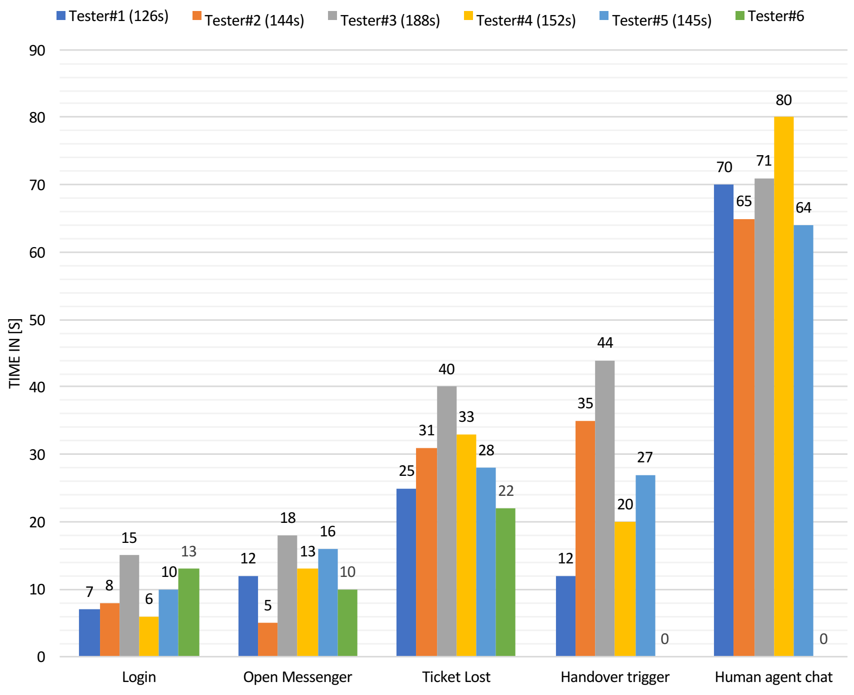
The performance analysis has been done on a desktop computer with *MacOS* and *Google Chrome* as a browser. The hardware of the computer can be considered average and the server is running on the *Java* environment. The whole test was running locally. The following Figure 6.1 shows the performance analysis of the system. The performance has been measured by acquiring the timestamps at specific tasks of the usability test from the *Login* to the *Human Agent Chat.* The total amount of time is also visible, which gives an overview value of how long the whole process took. As seen in the chart, the most time-consuming task was the handover reaction (Human agent chat). The participant had to wait until the support agent had time. Therefore a default waiting time of at least 60 seconds were integrated to the usability test.

6.4 Possible Extensions

Based on the requirements, the system is finished but it can be extended with additional handover modules for example a sarcasm detection. This is possible because of the generic program structure of the handover module. Another possible extension can be the expand of the FAQ, therefore the chatbot can be extended with additionally functions e.g. ticket purchase. Apart from this, the chatbot could also support internationalization.



(a) Scenario I



(b) Scenario II

Figure 6.1: Performance analysis of the system

Chapter 7

Conclusion

At a time where digitisation plays such a major role, and human interaction is becoming ever more important, the further development and integration of a chatbot system is becoming increasingly important. Meanwhile, there are a variety of chatbot systems, as mentioned in Section 2.2.5, and each of this system is developed for a specific type of use case, as illustrated in Section 2.2.6. One of this area is the customer support with the FAQ service. Large companies pay a huge amount of money for customers analysis, e-commerce, customer satisfaction and customer help services, that is because customer satisfaction is more important than ever. The companies always want to offer permanent support, where these chatbot systems come into play. Chatbots have gained more and more popularity also because of their outstanding advantages compared to human agents, e.g., provide 24/7 customer support.

Nevertheless, many of these systems still have problems when it comes to complexity, context of the conversation or recognition of frustrating situations. To tackle this problem, the hybrid chatbot was developed, which can hand the conversation over to a human agent in case of an unsolvable problem. The human agent can then take over the conversation and solve the task, which the machine could not solve. Therefore, the system should recognize these situations and react accordingly to the problem. Additionally the system should offer the user the opportunity to explicitly hand over the conversation to a human employee. An excellent chatbot should always be able to fall back to a human agent in the case of an unsolvable problem.

This thesis tried to improve customer support as much as possible to increase customer satisfaction. A hybrid version of a chatbot was developed in order to pass on the task, which the machine could not solve, to a human agent. Therefore, the system fulfilled the defined requirements.

During the evaluation process some challenges occurred. The system did not deliver the desired result in two out of twelve cases. That is partly because of the used Sentiment Analysis APIs, which has many problems with the German language and partly because of the tester's input message, especially a sarcasm message. The handover was not triggered by the following words or sentences:

- ernüchtert (sobered),
- jämmerlich (miserable),
- ratlos (helpless).

- Nerviges Produkt hätte gerne eine Antwort (Annoying product would like an answer).
- Dieses nervige Produkt, ist traumhaft schlecht. (This annoying product is fantastically bad) (sarcasm).
- Dieses tolle Produkt, ist sehr frustrierend (This great product is very frustrating) (sarcasm).

The system fulfills the defined requirements and Figure 5.3 illustrates that the desired concept has been implemented. Based on the requirements the system is complete, but in order to make a productive release, further FAQ functions would be needed to expand the conversation between user and chatbot.

The knowledge gained during this work is that, at present, a hybrid chat bot offers a perfect solution in terms of customer satisfaction and customer support. Altogether chatbots are a very active field with new inventions especially in the ML (Machine Learning), AI (Artificial Intelligent) and HCI (Human-Computer Interaction) areas, which gets popularity since over 50 years. Every functional B2C (business to customer) relation is a use case where a chatbot can be integrated, and for excellent customer satisfaction, a hybrid chatbot solution can save much frustration. Therefore only the future will show how important hybrid chatbots systems will become in the field of customer support.

Appendix A

Content of the CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 PDF-Files

Path: /

MasterThesis.pdf	master thesis

A.2 Project Data

Path: /implementation/project/web_application

src/main/webapp/ . . .	folder of the client
src/main/java/	folder of the server
src/main/webapp/**.ts	TypeScript source files
src/main/webapp/**.html	HTML files of the Web-Application
src/main/java/**.java .	Java source files

A.3 Dialogflow Agent

Path: /implementation/project

dialogflow_agent/ . . .	folder of the dialogflow agent

A.4 Literature

These are the files used as references. The file names are correspond to the reference title in the literature.

Path: references

[reference_title].pdf . .	file of the reference

A.5 Online Literature

These files are copies of the webpages used as references. The file names are numbers which correspond to the reference title in the literature.

Path: references/online_literature

[reference_title].pdf . .	printout of the webpages

A.6 Miscellaneous

Path: /images

*.pdf	vectorized images

References

Literature

[1] Sameera A. Abdul-Kader and John Woods. "Survey on Chatbot Design Techniques in Speech Conversation Systems". *International Journal of Advanced Computer Science and Applications (IARJSET)* 5 (Nov. 2015), pp. 37–46 (cit. on p. 3).

[2] Charu C. Aggarwal and Cheng Xiang Zhai. *Mining Text Data.* Heidelberg: Springer-Verlag, 2012 (cit. on p. 11).

[3] Robert Epstein, Gary Roberts, and Grace. Beber. *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer.* Heidelberg: Springer-Verlag, 2008 (cit. on p. 4).

[4] Janarthanam, Srini. *Hands on chatbots and conversational UI development.* Birmingham: Packt Publishing, Dec. 2017 (cit. on p. 5).

[5] Rashid Khan and Anik Das. *Build Better Chatbots.* Apress, 2018 (cit. on p. 9).

[6] Kyusong Lee et al. "Conversational Knowledge Teaching Agent that Uses a Knowledge Base". In: *Proceedings of the SIGDIAL 2015 Conference.* (Prague). Association for Computational Linguistics (ACL), Sept. 2015, pp. 139–143 (cit. on p. 7).

[7] Bing Liu. "Sentiment Analysis and Subjectivity". In: *Handbook of Natural Language Processing.* Ed. by Nitin Indurkhya and Fred J Damerau. Chapman and Hall Ltd, Jan. 2010, pp. 627–666 (cit. on p. 11).

[8] Bing Liu and Lei Zhang. "A Survey of Opinion Mining and Sentiment Analysis". In: *Mining Text Data.* Ed. by Charu C. Aggarwal and Cheng Xiang Zhai. Heidelberg: Springer-Verlag, 2012, pp. 415–463 (cit. on p. 11).

[9] Marco Lui, Jey Han Lau, and Timothy Baldwin. "Automatic Detection and Language Identification of Multilingual Documents". *Transactions of the Association for Computational Linguistics* 2 (Dec. 2018), pp. 27–40 (cit. on p. 5).

[10] Lian Meng and Minlie Huang. "Dialogue Intent Classification with Long Short-Term Memory Networks". In: *Natural Language Processing and Chinese Computing.* Springer International Publishing, 2018, pp. 42–50 (cit. on p. 5).

[11] Kiran Ramesh et al. "A Survey of Design Techniques for Conversational Agents". In: *Communications in Computer and Information Science.* Vol. 750. Springer Verlag, 2017, pp. 336–350 (cit. on p. 7).

[12] Iulian V. Serban et al. “Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models”. In: *Proceeding of the Thirtieth AAAI Conference on Artificial Intelligence.* (Phoenix). Association for the Advancement of Artificial Intelligence (AAAI), Apr. 2016, pp. 3776–3783 (cit. on p. 11).

[13] Amir Shevat. *Designing Bots: Creating Conversational Experiences.* O’Reilly UK Ltd, 2017 (cit. on p. 8).

[14] Ion Smeureanu and Cristian Bucur. “Applying Supervised Opinion Mining Techniques on Online User Reviews”. *Informatica Economic Journal* 16.2 (2012), pp. 81–91 (cit. on p. 10).

[15] Dieu Thu Le, Cam-Tu Nguyen, and Kim Anh Nguyen. “Dave the Debater: A Retrieval-Based and Generative Argumentative Dialogue Agent”. In: *Proceedings of the 5th Workshop on Argument Mining.* (Brussels). Association for Computational Linguistics, Nov. 2018, pp. 121–130 (cit. on p. 7).

[16] Alan Mathison Turing. “Computing Machinery and Intelligence”. *Mind 49* (1950), pp. 433–460 (cit. on p. 3).

[17] J Weizenbaum. “ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine”. *Communications of the ACM 9* (1966), pp. 36–45 (cit. on p. 4).

[18] Hao Zhou et al. “Emotional Chatting Machine: Emotional Conversation Generation with Internal and External Memory”. In: *The Thirty-Second AAAI Conference on Artificial Intelligence.* AAAI, Feb. 2018, pp. 730–739 (cit. on p. 11).

Online sources

[19] *Dialogflow Agents Overview.* 2019. URL: https://dialogflow.com/docs/agents (visited on 04/12/2019) (cit. on p. 27).

[20] *Dialogflow Create Fulfillment Using Webhook.* 2019. URL: https://dialogflow.com/docs/tutorial-build-an-agent/create-fulfillment-using-webhook (visited on 04/12/2019) (cit. on p. 28).

[21] *Dialogflow Entities Overview.* 2019. URL: https://dialogflow.com/docs/entities (visited on 04/11/2019) (cit. on p. 27).

[22] *Dialogflow Entity Matching.* 2019. URL: https://dialogflow.com/docs/intents (visited on 04/10/2019) (cit. on p. 27).

[23] MonkeyLearn. *Sentiment Analysis nearly everything you need to know.* 2018. URL: https://monkeylearn.com/sentiment-analysis/ (visited on 04/10/2019) (cit. on pp. 11, 12).

[24] Olga Davydova. *25 Chatbot Platforms: A Comparative Table.* 2017. URL: https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff (visited on 03/12/2019) (cit. on p. 15).

[25] Tom Hall. *How to choose a user research method.* 2017. URL: https://uxplanet.org/how-to-choose-a-user-research-method-985112051d84 (visited on 06/02/2019) (cit. on p. 46).