ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Chatbots zur Unterstützung von industriellen Serviceprozessen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Jens Reiner
jens.reiner@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Klaus Kammerer

2019

Fassung 5. März 2019

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Chatbots und andere digitale Assistenten haben in den letzten Jahren für Mensch-Maschinen Interaktionen immer mehr an Bedeutung gewonnen. Sie sind in der Lage, komplexe Aufgaben zu lösen, indem sie einen natürlichen Dialog mit Menschen führen. Ein Anwendungsgebiet für Chatbots ist die Durchführung von Wartungsprozessen, die als Konversationen mit unterschiedlichen Benutzern dargestellt werden können. Modellierte Wartungsprozesse können hierbei als Vorlage für Konversationen dienen. Ein Chatbot kann somit Maschinenwartungen vereinfachen, indem er Benutzern die Möglichkeit bietet, direkt mit Maschinen zu kommunizieren, um Informationen über deren aktuellen Status zu erhalten.

Ziel dieser Arbeit ist die Konzeption und Implementierung eines generischen Chatbots, um natürliche Konversationen mit Benutzern basierend auf Geschäftsprozessmodellen zu führen. In dieser Arbeit werden unterschiedliche Technologien für die Erstellung von Chatbots, ein Interaktionskonzept basierend auf verschiedenen Anwendungsfällen sowie eine prototypische Implementierung vorgestellt.

Inhaltsverzeichnis

1

Einleitung

Der Einsatz digitaler Assistenten findet immer mehr Gebrauch im alltäglichen Leben. Als digitalen Assistenten bezeichnet man eine Software, welche in der Lage ist, Dialoge mit einem Menschen zu führen, um Aufgaben zu erledigen. Unterhaltungen mit einem digitalen Assistenten finden dabei mit gesprochener Sprache statt. Fortschritte auf dem Gebiet der künstlichen Intelligenz und immer leistungsfähigerer Hardware ermöglichen einen immer breiteren Einsatz virtueller Helfer im privaten und kommerziellen Umfeld. Im Vordergrund steht dabei immer das Ziel mit einer Maschine einen natürlichen Dialog zu führen [1]. Eine besondere Art von digitalen Assistenten ist der Chatbot. Chatbots sind Applikationen, die selbständig textbasierte Konversationen mit Menschen führen können [2]. Chatbots finden immer häufiger Anwendung in der Automatisierung von Prozessen, welche typischerweise von einem Benutzer durchgeführt werden [2]. Die Anwendungsgebiete können dabei einfache Aufgaben, wie z.B. das Erstellen von Termin-Erinnerungen, bis hin zu komplexen Aufgaben umfassen, in denen ein Chatbot den Kundensupport übernimmt (vgl. Projekt Metis bei der schweizerischen Post AG [3]). Hierbei stellt sich die Frage, welche Vorteile ein Chatbot gegenüber einer normalen Software Anwendung mit sich bringt. Die Integration von Chatbots in unterschiedliche Chatplattformen und die Verfügbarkeit auf mehreren Endgeräten ermöglichen einen überall verfügbaren Zugriff auf Chatbots [2]. Die Bedienung eines Chatbots ist einfacher für Benutzer, weil diese die Software nicht kennen müssen, sondern das gewünschte Ergebnis in einem Dialog mit dem Chatbot erarbeiten kann [4]. Diese Arbeit beschäftigt sich mit dem Konzept der Implementierung eines Chatbots zur Unterstützung unterschiedlicher Geschäftsprozessen. Der hierfür erstellte Prototyp beschränkt sich dabei auf einen Assistenten für die Maschinenwartung am Beispiel einer Fabrik Simulation.

1.1 Problemstellung

In der Vergangenheit wurden bereits Chatbots eingesetzt, um unterschiedliche Anwendungsgebiete zu automatisieren. Das Problem war häufig, dass Benutzereingaben speziellen Mustern folgen mussten, um eine Funktion des Chatbots verwenden zu können und die Unterhaltungen natürlicher Sprache nicht ähnlich waren. Der Chatbot muss aus einer Benutzereingabe die Absicht des Benutzers ermitteln und basierend auf der Absicht eine passende Antwort erstellen. Für das Erstellen einer Antwort stehen dem Chatbot unterschiedliche Daten bereit aus denen dieser wählen kann. Zusätzlich müssen die angefragten externen Daten mit der Eingabe des Benutzers und dessen Absicht übereinstimmen. Weitere Probleme - speziell bei der Maschinenwartung - sind Rückfragen vom Benutzer an die gestellte Aufgabe, wobei der Chatbot dabei in der Lage sein sollte, dem Benutzer aktiv helfen zu können.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Erstellung eines Konzepts sowie die Implementierung eines webbasierten Chatbots zur Unterstützung von technischem Personal bei der Wartung von Maschinen. Der Chatbot soll dabei aktiv die Wartung unterstützen, indem Wartungsabläufe, Arbeitsschritte und der aktuelle Status von Maschinen als Frage dem Chatbot gestellt werden können und dieser dem technischen Personal antwortet. Um auch ungeschultes Personal die Möglichkeit anzubieten, mit einer Maschine zu interagieren, muss der Chatbot sich auf natürliche Art und Weise mit Benutzern unterhalten. Deshalb wurde für die prototypische Implementierung ein Bedienkonzept für die Umsetzung von Konversationen erarbeitet. Für eine praktische Veranschaulichung von industriellen Prozessen wird eine Fabrik Simulation eingesetzt. Aus diesem Grund benötigt der Chatbot zusätzliche Komponenten in Form eines Servers für die Kommunikation mit der Fabrik und der Verarbeitung ihrer Daten.

1.3 Struktur der Arbeit

In Kapitel 2 werden die Grundlagen für die Themen Chatbots, Prozesse und der Fabrik Simulation geklärt. Anschließend erfolgt eine Anforderungsanalyse in Kapitel 3 an das Konzept hinter dem Chatbot und den funktionalen sowie nicht-funktionalen Anforderungen an die Implementierung. Kapitel 4 stellt das Konzept für einen Wartungs-Chatbot vorgestellt. Kapitel 5 beschreibt die Implementierung zusammen mit den eingesetzten Technologien. Kapitel 6 gibt eine Zusammenfassung der Arbeit sowie eien Ausblick auf mögliche Erweiterungen und zukünftige Entwicklungen.

2

Grundlagen

2.1 Fabrik Simulation DBIS Factory

Die DBIS Factory ist ein Bausatz von Fischertechnik zur Simulation von Produktionsprozessen. Der Bausatz umfasst vier verschiedene Stationen, welche mit Sensoren und Aktuatoren ausgestattet sind (z.B. Lichtschranken und Motoren) [5]. Die vier einzelnen Stationen sind: *high rack*, *vacuum gripper*, *furnance* und *sorting line* (vgl. 2.1).

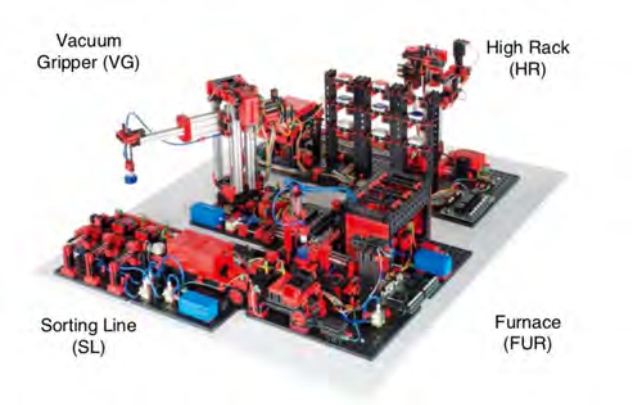


Abbildung 2.1: Aufbau der DBIS Factory [5]

Die Station *high rack* ist ein Hochregallager für Werkstücke, welche mithilfe eines Greifarms auf ein Förderband transportiert werden können. Das Förderband besitzt zwei Lichtschranken, um die Position eines Werkstücks genau zu bestimmen [5].

Die Station *vacuum gripper* ist ein Vakuumsauggreifer, welcher die Werkstücke vom Förderband der Station *high rack* nimmt und diese zur Station *furnance* transportiert. Die Station *furnance* simuliert anschließend das Brennen des Werkstücks in einem Ofen. Danach wird das fertige Werkstück auf ein Förderband gelegt und an die letzte Station *sorting line* transportiert. Die Station *sorting line* sortiert die Werkstücke nach drei verschiedenen Farben (Weiß, Rot oder Blau). Das Sortieren erfolgt dabei durch Auswerfen des Werkstücks in einen separaten Bereich für die jeweilige Farbe [5].

In Tabelle 2.1 sind die einzelnen Stationen mit ihren Sensoren sowie ihren Aufgaben beschrieben.

Station	**Sensor**	**Beschreibung**
high rack	2x Lichtschranke	Bestimmt die Position des Werkstücks auf dem Förderband
furnance	Lichtschranke	Prüfen ob Werkstück abgelegt wurde
sorting line	2x Lichtschranke	Bestimmt die Position des Werkstücks auf dem Förderband
	Farbsensor	Bestimmt die Farbe des Werkstücks

Tabelle 2.1: Sensoren der DBIS Factory [5]

Die DBIS Factory kann entweder in einem sequenziellen- oder parallelen Modus arbeiten. Im sequenziellen Modus befindet sich nur ein Werkstück innerhalb der Produktionskette. Beim parallelen Modus hingegen können sich mehrere Werkstücke innerhalb der Produktion befinden, wobei aber nur ein Werkstück pro Station zulässig ist. Im nachfolgenden ist der sequenzielle Modus beschrieben. Zuerst wird ein Werkstück aus dem Hochregal Lager auf ein Förderband gelegt, um dieses im Hochofen weiterzuverarbeiten. Der Vakuumgreifer nimmt sich das Werkstück vom Förderband und transportiert es zum Hochofen. Im Hochofen wird das Werkstück erhitzt und dessen Form verändert.

Anschließend kommt das fertige Werkstück in eine Sortieranlage, um nach seiner Farbe sortiert zu werden.

2.2 Business Process Management

Business Process Management umfasst die Themen für das Prozessmanagement und beschäftigt sich mit der Gestaltung, Ausführung, Dokumentation, Implementierung sowie dem Steuern und Verbessern von Prozessen [6]. Ein Prozess wird als verbundene festgelegte Abfolge einer oder mehrerer Aktivitäten definiert, mit dem Ziel eine Aufgabe zu lösen und können von einer Maschine oder Menschen ausgeführt werden [7] [6]. Das automatische Ausführen eines Prozesses kann dabei von einer sogenannten *Process Engine* gesteuert werden, welche anhand des technischen Prozessmodells Entscheidungen trifft und Prozessbeteiligte über anstehende Aufgaben informiert [6]. Das technische Prozessmodell ist dabei die Basis für eine standardisierte und grafische Prozessnotation, welche auch in der Prozessautomatisierung Anwendung findet [6]. Dieses Ziel verfolgt die Beschreibungssprache Business Process Modeling Notation (kurz BPMN). Textuelle Beschreibungssprachen (z.B. Business Process Execution Language BPEL) werden als XML definiert, beinhalten aber keine Symboldefinitonen. Die grafischen Beschreibungssprachen (z.B. BPMN) besitzen eine standardisierte Symboldefinitionen und können dadurch auch von Nicht-IT-Spezialisten eingesetzt werden [6].

Im Nachfolgenden werden die BPMN-Kernelemente vorgestellt und ihre Verwendung erläutert. Allgemein ist ein Prozess aus verschiedenen Aktivitäten aufgebaut, welche in einer bestimmten Reihenfolge abgearbeitet werden. Dabei können Aktivitäten von Ereignissen beeinflusst werden wie z.B. eine Bedingung, die zuerst zutreffen muss [6].

Aktivitäten werden als Aufgaben des Prozesses aufgefasst und beschreiben was getan werden muss. Ein Gateway ist eine Verzweigung innerhalb eines Prozesses und es kann automatisiert oder von Prozessbeteiligten entschieden werden, welche Aktivität als Nächstes ausgeführt werden soll. Das Start- und Endereignis deutet an, wann der Prozess gestartet werden soll und mit welchen Ereignis dieser endet. Der Sequenzfluss

beschreibt die zeitlich-logische Reihenfolge der BPMN-Elemente und wie diese zusammenstehen [6]. Die grafische Notation für BPMN-Kernelemente ist in 2.2 dargestellt.

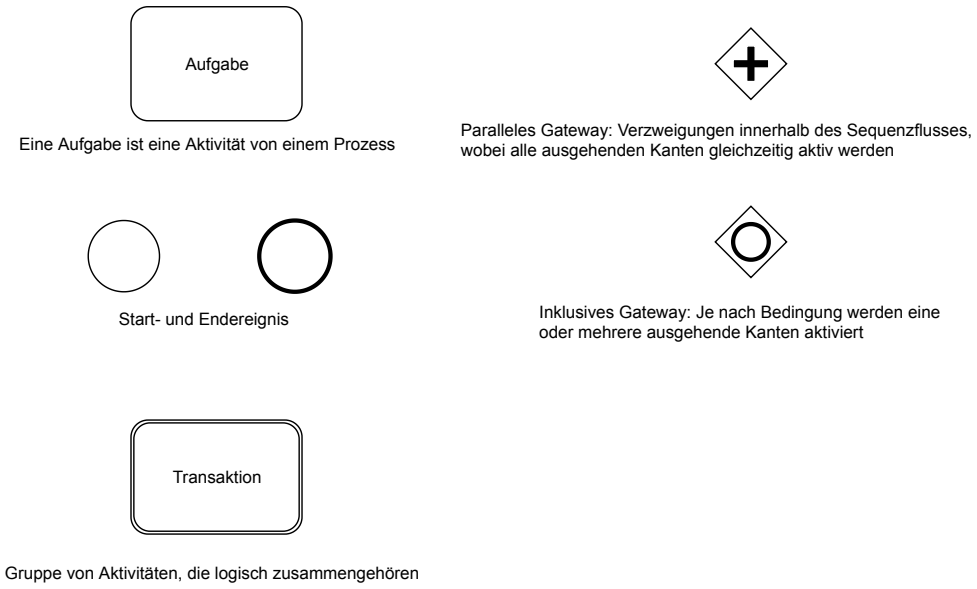


Abbildung 2.2: BPMN-Kernelemente im Überblick [6]

2.3 Chatbots

Als Chatbot bezeichnet man im Allgemeinen eine Anwendung, die in der Lage ist, mit einem Menschen einen natürlichen Dialog zu führen [1]. Die Hauptaufgabe eines Chatbots ist es, ein Ergebnis zu einer gestellten Aufgabe zu liefern. Gegenüber digitalen Assistenten wie z.B. Siri (vgl. Kapitel 2.4.1) sind diese in der Regel nicht an ein Gerät gebunden. Unterhaltungen mit einem Chatbot finden in einem Chatprogramm (z.B. *Messenger* [8]) statt und sind daher Chatplattformen spezifisch. Dadurch findet die Kommunikation eines Chatbots mit geschriebener anstelle von gesprochener Sprache statt.

Im Gegensatz zu digitalen Assistenten, welche auch als persönliche Assistenten bezeichnet werden und mit einer Person kommunizieren, werden Chatbots generischer eingesetzt und können sich mit mehreren unterschiedlichen Personen gleichzeitig unterhalten. Dabei können Chatbots nur spezifische Aufgaben für die Benutzer lösen und

sind entsprechend auf diese trainiert. (z.B. ein Chatbot im Bereich eCommerce kann keine Terminplanung ausführen)

Die technische Umsetzung eines Chatbots kann über definierte Regeln stattfinden bis hin zum Einsatz von künstlicher Intelligenz für das Verstehen von natürlicher Sprache. Daher bestehen Chatbots auch aus vielen, unterschiedlichen Technologien: Zum Einen benötigt man eine Technologie für das *Verstehen* von natürlicher Sprache, eine für das *Empfangen und Versenden* von Nachrichten und eine Business Logic, die entscheidet welche internen Funktionen aufgerufen werden sollen.

2.4 Natural Language Understanding

Das Verstehen von Eingaben kann bei Chatbots in zwei Kategorien unterschieden werden. Die erste Kategorie ist das *Abgleichen von Mustern*, welche dem Chatbot vorliegen. Mustererkennung eignet sich allerdings häufig nicht für den durchschnittlichen Benutzer, weil dieser nicht alle Eingabemuster kennt. Gleichzeitig ist der Chatbot-Entwickler nicht in der Lage alle Kombinationen von Eingaben zu einem bestimmten Problem zu definieren [3]. Die zweite Kategorie ist der Einsatz von künstlicher Intelligenz zur *Aufbereitung der Eingaben* und der Erkennung des Intents eines Benutzers. Die Aufbereitung findet dabei in mehreren Schritten statt, welche in Tabelle 2.2 näher erläutert werden.

Funktion	**Beschreibung**
Data cleaning	Korrekturen am Satz im Bezug auf Rechtschreibung und Grammatik.
Natural Language Processing	Struktur Analyse und Elemente klassifizieren Bedeutung der Elemente bestimmten
Intent Matching	Absicht des Benutzers aus der aufbereiteten Eingabe erkennen

Tabelle 2.2: Funktionen der Aufbereitung von Benutzereingaben [3]

Ist die Absicht des Benutzers bekannt, kann der Chatbot eine passende Antwort generieren und zurückgeben. Verschiedene IT-Dienstleister bieten fertige Komponenten zur

Einbindung von Natural Language Understanding (NLU) Komponenten in Chatbots an. Diese werden in Kapitel 2.4.1 näher betrachtet.

2.4.1 Technologieübersicht

Die Umsetzung eines Chatbots oder digitalen Assistenten kann mit einer Vielzahl unterschiedlicher Technologien realisiert werden. Neben den bereitgestellten Technologien der *Big Five* (Google, Apple, Amazon, Facebook und Microsoft) existiert eine große Anzahl an Open-Source Frameworks, welche eine Basis für Chatbot Applikationen bereitstellen [9]. Die angebotenen Technologien der Big Five sind zum Teil fertige Assistenten, welche in verschiedenen Produkten bereits zum Einsatz kommen. Darüber hinaus werden Integrationen angeboten, welche verschiedene Dienste der Technologie anbieten um mit einer eigenen Anwendung kombiniert werden zu können (vgl. Kapitel 2.4.1). Entwicklern wird dadurch ermöglicht ihren Chatbot mit einem Open Source Framework aufzubauen und die Verarbeitung von Daten, auf die der Chatbot zugreifen muss intern zu behalten, ohne diese mit externen Diensten zu teilen. Der Chatbot kann die Komplexität beim Verstehen von natürlicher Sprache an einen externen Dienst auslagern (vgl. Kapitel 2.4) während der Zugriff auf interne Daten geheim bleibt. Die Wahl der Technologie spielt eine entscheidende Rolle für den Erfolg eines Chatbots. Müssen Benutzer bei ihrer Eingabe ein bestimmtes Muster beachten, damit der Chatbot auf ihre Anfragen reagiert, sind sie abgeneigt den Chatbot weiterzuverwenden [10]. Für die Wahl der Technologie sind in 2.3 einige wichtige Merkmale aufgelistet, welche für die Umsetzung eines Chatbots in Betracht gezogen werden sollten. Im darauf folgenden Abschnitt wird eine Auswahl an Technologien näher erläutert.

Siri und SiriKit

Für die Betriebssysteme iOS, macOS, watchOS und tvOS stellt Apple ihren digitalen Assistenten Siri bereit. Siri ist eine Software für das Verstehen von gesprochener Sprache und ermöglicht dem Benutzer, ein Produkt von Apple nur mit Sprachsteuerung zu bedienen. Siri ist bereits seit 2011 für das mobile Betriebssystem von Apple verfügbar

Bezeichnung	Beschreibung
Intent Matching	Die Fähigkeit aus verschiedenen Eingaben die Absicht des Benutzers zu erkennen. Ermöglicht es verschiedene Phrasen, Ausdrücke und Sätze zu verwenden, um zum gewünschten Ergebnis zu gelangen.
Konversationen	Chatbots besitzen oft nur ein Frage-und Antwortmechanismus. Konversationen können eingesetzt werden, um längere und komplexere Unterhaltungen zu führen mit dem gleichen Kontext
Plattformunterstützung	Es gibt eine Vielzahl an Plattformen, welche eine Chatumgebung bereitstellen. Die ausgewählte Technologie sollte deshalb eine Unterstützung für die Chatumgebung bereitstellen.
Implementierung	Häufig bieten Frameworks bereits eine Möglichkeit den Chatbot zu konfigurieren ohne eine Zeile Code zu schreiben.
Anbindungen	Chatbots erledigen häufig Aufgaben, welche auf einer anderen Plattform bereit gestellt werden. Der Chatbot muss deshalb in der Lage sein, mit anderen Diensten kommunizieren zu können und Daten auszutauschen.
Sprachen	Chatbots können in unterschiedlichen Ländern eingesetzt werden und müssen aus diesem Grund mehrere Sprachen unterstützen.

Tabelle 2.3: Evaluierte Funktionen der Technologieauswahl, basierend auf [11]

und wurde über die Jahre auf weiteren Plattformen von Apple angeboten [3]. Apple bewirbt Siri als *hands free access* Software, denn der Benutzer hat die Möglichkeit mit Siri zu interagieren ohne sein Gerät dabei in der Hand zu halten [12]. SiriKit wird von Apple bereitgestellt, um es App Entwicklern zu ermöglichen Siri in ihre Apps zu integrieren.

Alexa und Amazon Lex

Im Jahr 2015 stellt Amazon ihren digitalen Assistenten Alexa vor, welcher zu Beginn nur in Amazon Geräten (z.B. Amazon Echo) zu finden war. In den darauf folgenden Monaten stellte Amazon auf ihrer AWS Plattform Amazon Lex bereit, als Service für eine Konversationsschnittstelle für Sprache und Text ohne an ein Gerät gebunden zu sein [13] [14]. Amazon Lex stellt eine Plattform für die Aufgaben einer NLU (vgl. 2.4) bereit und ermöglicht es Chatbot Entwicklern einen komplexen Chatbot zu entwickeln, welcher in der Lage ist, Konversationen zu führen [13]. Die Serviceplattform AWS skaliert dynamisch nach Anzahl ihrer Benutzer, wodurch der Chatbot beliebig viele Benutzer gleichzeitig unterstützen kann.

Google Assistant und Dialogflow

Der Google Assistant ist der digitale Assistent von Google und entstand aus dem früheren Projekt Google Now [3]. Die Hauptaufgaben sind die gleichen wie bei Siri und Alexa. Dem Benutzer die Möglichkeit geben mit gesprochener Sprache verschiedene Software zu bedienen. Einen Vorteil gegenüber Alexa und Siri besitzt der Google Assistant bei der Unterstützung mehrerer unterschiedlicher Plattformen. Google Assistant ist auf mobilen Geräten, Smart Speakern, Smart Watches und im Automobilbereich verfügbar [15].

Dialogflow (auch bekannt als API.AI) ist ein Service zur Erstellung von Konversationsschnittstellen. Es stellt eine Natural Language Understanding Engine bereit, um natürliche Spracheingaben zu verarbeiten [16]. In Dialogflow werden sogenannte *Agents* erstellt, welche die Aufgaben der Umwandlung von natürlicher Sprache in ein strukturiertes Daten Format übernehmen und auf den Anwendungen später zurückgreifen können. Agenten können eine Vielzahl an Intents unterstützen, welche vom Benutzer vorher definiert werden können. Ein Intent in Dialogflow ist aufgebaut aus einer *Trainings Phase*, *Actions and Parameters* und *Responses* [16]. Die *Trainings Phase* wird als Training verwendet auf welche Phrasen und Sätze der Agent (bzw. Intent) reagieren soll. In *Actions and Parameters* können zusätzliche Informationen aus den Eingaben extrahiert werden und sind in Form von einer *Entitiy* später abrufbar. Für die Vollständig-

keit einer Konversation benötigt man noch eine Antwort, welche der Agent zurückgeben soll. Diese kann in *Responses* (deutsch: *Antworten*) definiert werden, wo man auf die extrahierten Informationen aus *Actions and Parameters* zurückgreifen kann. Der Ablauf der Funktionsweise eines Agents ist in Abbildung 2.3

Abbildung 2.3: Dialogflow Agent: Intent Matching und Ablauf [17]

Microsoft Bot Framework

In 2016 startete der Dienst Microsoft Bot Framework (heute auch als Azure Bot Service [18] bekannt) welches eine REST und SDK API für die Chatbot Entwicklung anbietet [19]. Der Azure Bot Service von Microsoft bietet ein Grundgerüst für die Chatbot Ent-

wicklung an, indem es low-level I/O Operationen übernimmt und verschiedene Chat Features out-of-the-box bereitstellt. Ein Vorteil für Chatbot Entwickler ist es, dass sie selbst entscheiden können, welche Funktionalität und Komplexität ihr Chatbot später besitzen soll. So bietet das Microsoft Bot Framework die Möglichkeit an, entwickelte Chatbots mit Language Understanding (*LUIS* [20]) zu erweitern, um den Gebrauch von natürlicher Sprache zu ermöglichen. Das Microsoft Bot Framework kann entweder auf einer Webseite verwendet werden oder in einen externen Messenger-Dienst (z.B. Slack, Skype und Facebook Messenger) eingebunden werden [19].

Rasa

Rasa ist ein Framework zur Erstellung von Konversationssystemen und ist eine Brücke zwischen der aktuellen Forschung von KI gestützten Konversationen und Anwendern [21]. Das Framework bietet Anwendern zwei Lösungen an, welche entweder getrennt voneinander oder in Kombination eingesetzt werden können.

Rasa Core ist ein Dialog Management System, dass anstelle von State Machine Konversationen auf Machine Learning Konversationen konzipiert ist [22]. Dem Chatbot kann mit interaktiven Lernen beigebracht werden, welche Antworten der Benutzer auf seine unterschiedlichen Aktionen erwartet. Hierfür werden zuerst alle Aktionen definiert, die der Chatbot ausführen kann. Falls sich der Chatbot für eine falsche Aktion entschieden haben sollte, so kann der Benutzer diese direkt korrigieren [22]. Interaktives Training ermöglicht es Entwicklern, keine komplexeren State Machines mit Randbedienungen definieren zu müssen.

Rasa NLU ist ein Natural Language Understanding Dienst und kann als externe Komponente in verschiedene Chatbot Frameworks eingebaut werden. Der Vorteil gegenüber Dialogflow und anderen NLU Diensten ist, dass *Rasa NLU* auf eigenen Plattformen bereitgestellt werden kann und der Dienst nicht über einen externen Anbieter laufen muss. Verwaltet der Chatbot z.B. sensible Daten, welche nicht an Dritte übertragen werden dürfen, kann die *Rasa NLU* Komponente die Aufgaben einer NLU übernehmen.

Chatfuel

Chatfuel ist die größte Plattform zur Entwicklung von Chatbots für den Facebook Messenger [8] [23]. Über 40% der aktiven Chatbots auf der Messenger Plattform werden mit Chatfuel betrieben [23]. Das Besondere an Chatfuel ist die Möglichkeit der Chatbot Entwicklung ohne selbst Code Zeilen zu implementieren [2]. Der Entwickler kann den Chatbot über die Weboberfläche und verschiedenen Modulen per Maus zusammenstellen. Chatfuel bietet out-of-the-box die Verarbeitung von natürlicher Sprache an und der Entwickler kann ähnlich zu Dialogflow (vgl. Kapitel 2.4.1) die Künstliche Intelligenz von Chatfuel auf Sätze und Phrasen verschiedener Benutzereingaben trainieren [24].

Botkit

Botkit ist ein auf NodeJS basierendes Open Source Framework zur Entwicklung von Chatbots mit Anbindung zu den wichtigsten Messenger-Plattformen [25] [26]. Das Grundgerüst für das Versenden und empfangen von Nachrichten der verschiedenen Messenger-Plattformen stellt Botkit out-of-the-box bereit. Neben den out-of-the-box Funktionen von Botkit, bietet das Framework die Möglichkeit an, eigene Funktionalitäten und externe Dienste zu Botkit hinzuzufügen. Das Erweitern von Botkit wird über eine Middleware realisiert und ermöglicht es, sogenannte Plugins für den Chatbot zu erstellen [27]. Die am häufigsten verwendeten Middlewares sind Anbindungen an externe Natural Language Understanding Dienste. Botkit wird in der späteren Implementierung als Basis für den Chatbot verwendet (vgl. Kapitel 5).

3

Anforderungsanalyse

Das Schreiben von Textnachrichten hat sich in den letzten Jahren als eine der am meist genutzten Methoden für Konversationen zwischen verschiedenen Menschen durchgesetzt [10]. Personen im Alter zwischen 18 und 24 stellen Textnachrichten gleich mit einem Telefonat und Jugendliche, die aktiv ein Smartphone benutzen, unterhalten sich intensiv mit Textnachrichten [10]. Textnachrichten sind in der Regel kurze Nachrichten, die nicht immer alle nötigen Informationen beinhalten. Für den Fall fehlender Informationen, kann die andere Person eine Rückfrage stellen, um sich so fehlende Informationen anzueignen.

Angewandt auf einen Chatbot kann so ein Frage- und Antwort-System entworfen werden, welches in der Lage ist, schnell und präzise benötigte Informationen zu liefern. Ist ein Chatbot nicht in der Lage eine Frage zu beantworten, weil ihm zusätzliche Informationen fehlen, können Chatbots die fehlenden Informationen einfach bei dem Benutzer anfragen. Eine Webseite ist in vielen Fällen überladen mit Informationen und ein Benutzer benötigt Zeit gewünschte Antworten zu finden [10]. Gegenüber Webseiten hat ein Chatbot den Vorteil nur Informationen anzuzeigen, die vom Benutzer angefragt worden sind. Zusätzlich kann ein Chatbot aktiv dem Benutzer helfen, um an seine gewünschten Informationen zu gelangen, indem dieser mögliche Fragestellungen bereits im Voraus beantwortet und den Benutzer durch einen Prozess leitet.

Betrachtet man die Produktion der DBIS Factory vorgestellt in 2.1 und möchte Informationen über verschiedene Sensor Daten oder Wartungsprozesse erhalten kann anstelle von einer Weboberfläche auch ein Chatbot eingesetzt werden, welcher mit dem Benutzer direkt kommuniziert. Die unterschiedlichen Anwendungen sind im folgenden Kapitel genauer erläutert.

3.1 Betrachtete Anwendungsfälle

3.1.1 Maschinenwartung und Wartungsprozesse

Produktionsstätten mit Maschinen können auf unterschiedliche Arten gewartet werden. Zum einen gibt es Maschinen, welche in einem festgelegten Intervall, eine Wartung benötigen. Darüber hinaus gibt es auch Maschinen, die keine Regelmäßige Wartungen benötigen, sondern nur im Fehlerfall gewartet werden müssen. Wartungen können dabei als Anleitungen oder Checklisten definiert werden und beinhalten die auszuführenden Arbeitsschritte. Das Problem an Anleitungen und auch Checklisten ist, dass es keine einheitliche Notation gibt. Aus technischer Sicht ist eine standardisierte Notation aber notwendig, um Wartungen immer nach demselben Schema ausführen zu können.

Die Maschinen der DBIS Factory (vgl. Kapitel 2.1) haben einen Zyklus, in welchem sie gewartet werden müssen. Wartungen sind notwendig, um eine durchgehende Produktion zu gewährleisten. Im Allgemeinen sind Wartungen anspruchsvolle Prozesse, welche von geschultem Personal ausgeführt werden. Um einen Wartungsprozess für die DBIS Factory abzubilden, wird in BPMN (vgl. Kapitel 2.2) ein Wartungsprozess definiert. Der abgebildete Wartungsprozess in BPMN, wird als Vorlage für die Chatbot Konversation eingesetzt. Der Wartungsprozess beinhaltet alle Informationen, welche notwendig sind, um eine Wartung erfolgreich durchzuführen. Dazu gehört auch eine Dokumentation der einzelnen Arbeitsschritte, um einem Benutzer später bei Unklarheiten helfen zu können. Benutzer sind alle Personen, welche mit der Factory zu tun haben z.B. ein Maschinentechniker.

Ein Chatbot kann diesen Prozess unterstützen, indem dieser den Wartungsablauf kennt und den Benutzer durch die Wartung führt. Der Benutzer kann den Chatbot nach den nächsten Schritten fragen und bei Unklarheiten den Chatbot nach zusätzlicher Hilfe fragen. Zusätzlich kann der Benutzer den Chatbot fragen, wann die nächste Wartung stattfinden muss und wann die letzte Wartung durchgeführt worden ist. Für diese Aufgaben muss der Chatbot auf unterschiedliche Benutzer Absichten reagieren können. Die erste Aufgabe ist das Anzeigen einer Liste mit allen verfügbaren Wartungsprozessen, aus denen der Benutzer wählen kann. Ist dem Benutzer der Wartungsprozess bekannt

soll dieser die Wartung sofort durchführen können, ohne vorher die gewünschte Wartung speziell auszuwählen. Während einer Wartung kann es passieren, dass der Benutzer eine andere Funktionalität des Chatbots benutzt. Aus diesem Grund muss der Chatbot in der Lage sein, den aktuellen Wartungsprozess und dessen Fortschritt zu speichern, um dem Benutzer zu ermöglichen an dieser Stelle wieder fortzufahren. Ist eine Wartung bei einer Maschine notwendig, soll der Chatbot den Benutzer die Frage stellen, ob dieser mit der Wartung sofort beginnen möchte und den entsprechenden Wartungsprozess vorschlagen.

Aus diesen Anforderungen an einen Chatbot lässt sich das folgende Anwendungsdiagramm ableiten, welches in 3.1 dargestellt ist. Beim Übermitteln des Wartungsprozesses interagiert der Benutzer mit einer externen Schnittstelle (vgl. Kapitel 3.2), andernfalls mit dem Chatbot. Das Durchführen eines Wartungsprozesses ist nur möglich, wenn der Chatbot die Fähigkeit besitzt eine Konversation über mehrere Eingaben hinweg aufrechtzuerhalten. Außerdem gibt es in einem Wartungsprozess verschiedene Auswahlmöglichkeiten, aus denen der Benutzer auswählen kann, welche unterschiedliche Konversationszweige zur Folge haben, in welchen der Chatbot navigieren muss.

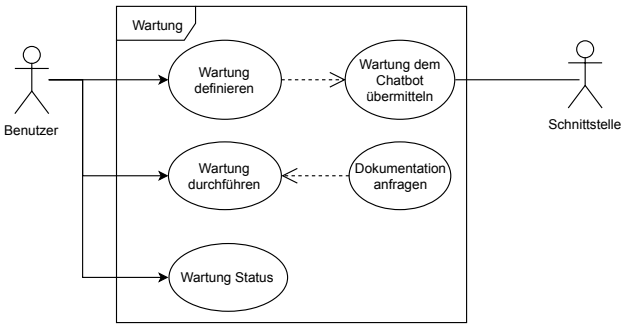
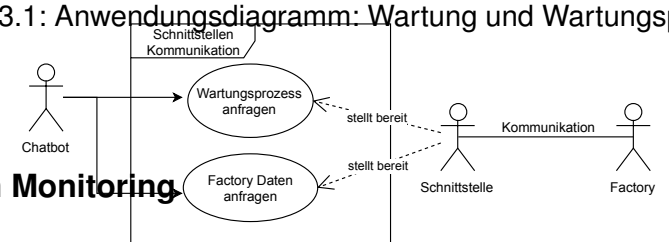
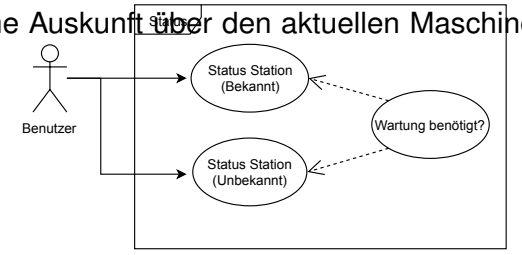


Abbildung 3.1: Anwendungsdiagramm: Wartung und Wartungsprozesse

3.1.2 Condition Monitoring

Condition Monitoring ist das überwachen eines Parameters innerhalb einer Maschine mit dem Ziel, eine Auskunft über den aktuellen Maschinenzustand zu erhalten. Eine





Maschine kann z.B. mit mehreren Temperatur Sensoren ausgestattet sein, welche in einem festen Intervall Messungen vornehmen. Treten innerhalb der Messdaten Unregelmäßigkeiten auf, kann die Maschine oder ein Mensch entsprechend reagieren und z.B. die Maschine abschalten. Condition Monitoring kann eingesetzt werden, um Wartungsintervalle zu minimieren. Wartungen werden dann ausgeführt, wenn die Messdaten entsprechende Unregelmäßigkeiten aufzeigen. Anstatt für feste Wartungsintervalle die Maschine abzuschalten, kann die Maschine weiter laufen und muss nur im Einzelfall abgeschaltet werden [28].

Die Stationen der Factory (2.1) liefern unterschiedliche Sensordaten, welche abgefragt und ausgewertet werden sollen. Die Abfrage nach den Sensordaten soll dabei an einen Chatbot gerichtet werden. Dieser muss selbstständig in der Lage sein die aktuellen Informationen der DBIS Factory abzurufen und die Sensordaten an den Benutzer zurückzugeben. Die DBIS Factory besitzt unterschiedliche Stationen und Sensordaten. Der Benutzer muss also die Möglichkeit haben zwischen den unterschiedlichen Stationen wählen zu können. Dafür muss der Chatbot die gewünschte Station aus der Benutzereingabe extrahieren. Ist keine Station vom Benutzer spezifiziert, kann der Chatbot eine Liste der verfügbaren Stationen liefern und diese dem Benutzer anzeigen. Anders als bei einem Wartungsprozess folgt der Chatbot einem Frage-und Antwortschema. Der Benutzer stellt eine Anfrage an eine Station und der Chatbot liefert das gewünschte Resultat. Wenn der Benutzer den Status einer Station abruft und die Maschine eine Wartung benötigt, soll der Chatbot zusätzlich den Benutzer darüber informieren.

Die Anforderungen sind im Anwendungsdiagramm 3.2 abgebildet. Die Bezeichnungen *bekannt* und *unbekannt* stehen für den beschriebenen Fall, dass der Benutzer keine spezifische Station angibt. Der Benutzer interagiert in allen Fällen mit dem Chatbot.

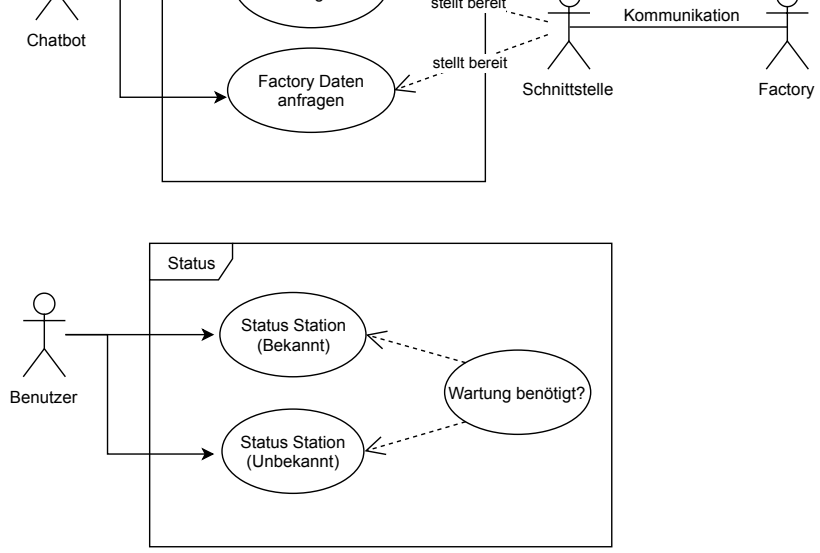
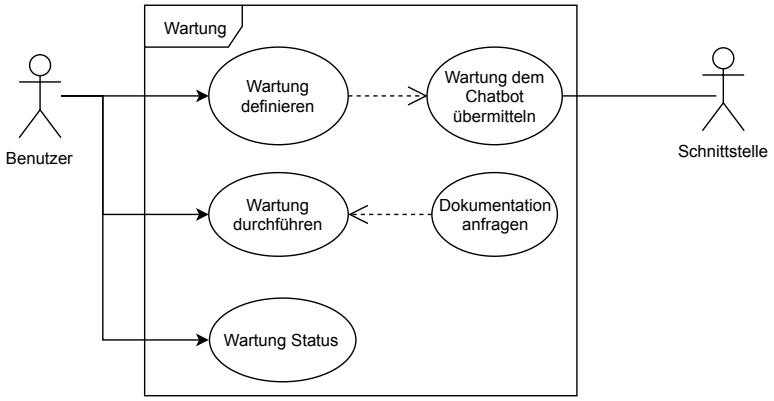


Abbildung 3.2: Anwendungsdiagramm: Condition Monitoring

3.2 Funktionale Anforderungen

Aus d
analy
Syste

Die e
um d
Ein e
der C
folger



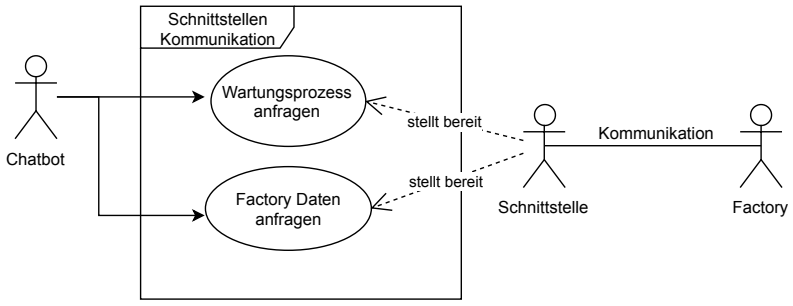
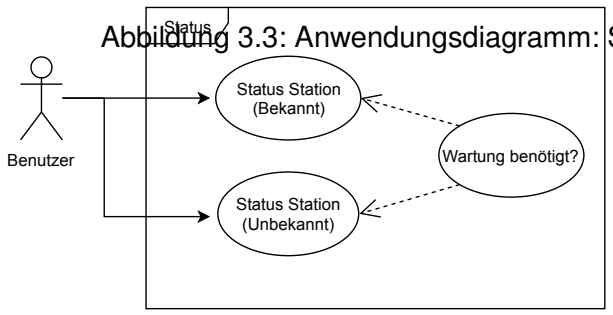


Abbildung 3.3: Anwendungsdiagramm: Schnittstelle



Die Abkürzungen *S* und *C* stehen dabei für die Schnittstellen Komponente bzw. Chatbot Komponente.

FA01-S Wartungsprozess hochladen *[Erforderlich]*

Benutzer können definierte Wartungsprozessen als BPMN Datei an der Schnittstelle hochladen.

FA02-S Wartungsprozess konvertieren *[Erforderlich]*

Der Chatbot kann BPMN Dateien nicht ausführen. Aus diesem Grund müssen sie zunächst in ein passendes Format konvertiert werden. Die Schnittstelle übernimmt die Konvertierung, nachdem ein Wartungsprozess hochgeladen worden ist.

FA03-S Wartungsprozess umwandeln *[Erforderlich]*

BPMN Dateien haben zusätzliche Informationen über die grafische Darstellung, welcher der Chatbot nicht benötigt. Aus diesem Grund soll das zuvor konvertierte Format umgewandelt werden in eine interne Struktur die nur benötigte Daten beinhaltet.

FA04-S API bereitstellen *[Erforderlich]*

Die Schnittstelle muss eine API bereitstellen auf die der Chatbot zugreifen kann für den Austausch von Daten.

FA05-S Factory Anbindung *[Erforderlich]*

Die Factory stellt verschiedene Daten bereit. Die zu implementierende Schnittstelle muss auf diese Daten zugreifen, diese ggf. aufbereiten und bereitstellen.

Das zweite System ist die Implementierung von einem Chatbot.

FA01-C Nachrichten versenden *[Erforderlich]*

Der Chatbot muss Nachrichten über eine Chatplattform versenden können.

FA02-C Nachrichten empfangen *[Erforderlich]*

Der Chatbot muss Nachrichten von einem oder mehreren Benutzern empfangen können.

FA03-C Konversation: Erstellen *[Erforderlich]*

Der Chatbot kann Konversationen mit einem oder mehren Benutzern führen. Eine

Konversation ist eine Abfolge von Nachrichten zwischen dem Chatbot und dem Benutzer, welche im selben Kontext stattfinden. Der Chatbot stellt dem Benutzer verschiedene Fragen und verarbeitet die Antworten des Benutzers.

FA04-C Konversation: Beenden *[Erforderlich]*

Der Benutzer hat die Möglichkeit eine aktuelle Konversation (z.b. Wartungsprozess) jederzeit zu beenden.

FA05-C Wartungsprozesse abfragen *[Erforderlich]*

Der Chatbot muss beim Start eine Liste mit allen Wartungsprozessen von der Schnittstelle anfragen. Wenn der Benutzer einen Wartungsprozess auswählt, wird der gewählte Prozess vom Chatbot angefragt und geladen.

FA06-C Wartungsprozess: Durchführen *[Erforderlich]*

Der Chatbot kann aus einem gewählten Wartungsprozess eine Konversation erstellen, welche die einzelnen Schritte der Wartung mit dem Benutzer durchführen.

FA07-C Wartungsprozess: Hilfe *[Erforderlich]*

Während eine Wartung mit dem Chatbot durchgeführt wird, bietet der Chatbot die Möglichkeit an, zusätzliche Informationen in Form von einer Dokumentation anzuzeigen. Ist keine zusätzliche Dokumentation für eine Aufgabe definiert, wird eine entsprechende Nachricht dem Benutzer mitgeteilt.

FA08-C Wartungsprozess: Speichern *[Optional]*

Wird ein aktueller Wartungsprozess unterbrochen, ist der Chatbot in der Lage den aktuellen Stand zu speichern. Bei der Wahl der Wartung kann der Benutzer danach angeben, ob er mit der letzten Wartung fortfahren möchte.

FA09-C Status: Stationen anzeigen *[Erforderlich]*

Der Chatbot kann alle verfügbaren Stationen anzeigen.

FA10-C Status: Station Sensordaten *[Erforderlich]*

Der Chatbot zeigt aktuelle Sensordaten der unterschiedlichen Stationen an. Spezifiziert der Benutzer eine Station wird nur die ausgewählte Station und ihre Informationen angezeigt.

FA11-C Status: Wartung informieren *[Erforderlich]*

Steht bei einer Station eine Wartung bevor und der Benutzer ruft die Sensordaten der Station ab, informiert der Chatbot selbständig über die bevorstehende Wartung.

FA12-C Status: Nächste und vorherige Wartung *[Erforderlich]*

Der Benutzer kann sich die Information der nächsten Wartungen aller Stationen anzeigen lassen. Der Benutzer kann sich die Informationen der letzten Wartungen aller Stationen anzeigen lassen.

FA13-C Natürliche Sprache *[Erforderlich]*

Der Chatbot kann natürliche Sprache verstehen und nicht nur auf definierte Schlüsselwörter reagieren.

FA14-C Absicht des Benutzers *[Erforderlich]*

Der Chatbot versucht die Absicht des Benutzers zu ermitteln. Ist der Chatbot nicht in der Lage, die Absicht des Benutzers herauszufinden, gibt er eine Nachricht an den Benutzer zurück.

FA15-C Parameter aus Eingaben extrahieren *[Erforderlich]*

Der Chatbot kann aus Benutzereingaben Parameter extrahieren. Parameter sind z.B. Stationen oder die direkte Auswahl von einem Wartungsprozess.

FA16-C Mehrere Benutzer gleichzeitig *Optional*

Der Chatbot kann sich gleichzeitig mit mehreren Benutzern unterhalten und auf Eingaben reagieren.

3.3 Nicht-funktionale Anforderungen

Die zuvor beschriebenen funktionalen Anforderungen 3.2 bilden das zu entwickelte System. Neben funktionalen Anforderungen gibt es noch die nicht-funktionalen Anforderungen an das System, welche im Folgenden aufgelistet sind. Die nicht-funktionalen Anforderungen dienen hauptsächlich der Nutzbarkeit der Software [29].

NFA01 Korrektheit [29]

Die entstandene Software (Chatbot und Schnittstelle) stimmt mit den definierten Spezifikationen überein.

NFA02 Sicherheit [29]

Die Verarbeitung von sensiblen Daten (z.b. Textnachrichten zwischen Benutzer und Chatbot) werden nicht an Dritte übermittelt und Außenstehende erhalten keinen Zugriff auf diese Daten.

NFA04 Verfügbarkeit [29]

Chatbot und Schnittstelle arbeiten rund um die Uhr.

NFA05 Robustheit und Fehlertoleranz [29]

Unterläuft einem der Systeme ein Fehler, wird der Benutzer darüber informiert. Die Systeme müssen nach einem Fehler weiter zur Verfügung stehen.

NFA06 Skalierbarkeit

Alle Systeme skalieren automatisch mit der Anzahl an Benutzern.

4

Konzeption

4.1 Konversationsdesign

Die User Experience mit einem Chatbot ist stark abhängig eines guten Designs während der Konversationen mit Benutzern [30]. Aus diesem Grund werden im Folgenden Grundlagen zum Design von Konversationen vorgestellt, welche in der späteren Implementierung 5 angewandt werden.

Abwechselnd sprechen

In normalen Unterhaltungen zwischen zwei Menschen gibt es eine Person, die am Sprechen ist und eine andere, die nur zuhört [30]. Konversationen mit einem Chatbot sind nach dem selben Prinzip aufgebaut. Der Chatbot hört im Normalfall auf die Eingaben des Benutzers und antwortet, nachdem dieser seine Frage gestellt hat. Dabei unterbricht der Chatbot den Benutzer nicht bei seinen Eingaben. Hat der Chatbot die Benutzereingabe verarbeitet, antwortet dieser und wartet auf die nächste Benutzereingabe.

Konversationsfluss

Interaktive System, die mit Menschen zusammenarbeiten, bieten häufig vordefinierte Optionen an, aus denen ein Benutzer wählen kann [30]. Solche Designentscheidungen haben häufig die Folge, dass der Benutzer abgeneigt ist mit dem System zu interagieren, weil es nicht intuitiv und langweilig erscheint [30]. Der Chatbot hingegen soll direkt mit dem Benutzer interagieren und die gewünschte Absicht des Benutzers anhand von einer Unterhaltung herausfinden. Als Absicht bezeichnet man den Gedanken des Benutzers hinter seiner Eingabe und was dieser damit erreichen möchte. Dafür können dem Benutzer Fragen gestellt werden, welche der

angebotenen Dienste dieser gerne verwenden möchte, anstatt direkt eine Liste mit allen Diensten anzuzeigen [30]. Ein weiterer Vorteil ist, dass dem Benutzer nur die Dienste angezeigt werden, die auf seine Eingabe passen und irrelevante ausgeblendet bleiben.

Kontext

Kontext in Konversationen bedeutet, sich auf Informationen zu beziehen, welche dem Chatbot bereits vorliegen [31]. Diese Art der Informationen können aus vorherigen Konversationen stammen oder sich auf das Umfeld beziehen, indem der Chatbot eingesetzt wird. Als einfaches Beispiel betrachtet man einen Chatbot, welcher Informationen über das aktuelle Wetter liefert. Wird die Frage nach dem aktuellen Wetter gestellt, möchte man nicht zusätzlich den aktuellen Ort angeben. Der Chatbot kann direkt mit vorhandenen GPS-Daten den aktuellen Standort bestimmen und die Wettervorhersage zurückgeben. Stellt der Benutzer dann eine weitere Frage, ob dieser einen Regenschirm benötigt, kann der Chatbot - basierend auf der vorherigen Antwort - eine Antwort liefern [30]. Kontext ermöglicht es dem Chatbot Verknüpfungen zwischen früheren Fragen und Antworten zu erstellen. Zusätzlich muss eine Konversation nicht immer von vorne beginnen, wenn der Chatbot im gleichen Kontext bleibt.

Bestätigungen

Chatbots erledigen viele Aufgaben im Hintergrund, von denen ein Benutzer nichts mitbekommt. In bestimmten Konversationen ist es aber wichtig dem Benutzer Rückmeldung zu geben, dass der Chatbot die Eingabe verstanden hat [30]. Möchte der Benutzer Daten löschen sollte der Chatbot noch einmal die Eingabe des Benutzers wiederholen und fragen, ob sich der Benutzer sicher ist die ausgewählten Daten zu löschen. Um nicht ständig nach Bestätigungen zu fragen, kann man auf die *Confidence* Eigenschaft einer NLU (vgl. Kapitel 2.4) Antwort zurückgreifen und Rückfragen überspringen, wenn ein bestimmter Wert zutrifft [30]. Die *Confidence* Eigenschaft ist eine Auskunft, wie sicher sich die NLU in ihrer Antwort ist und kann als Referenzwert benutzt werden, um falsche Antworten der NLU zu ignorieren.

Fehlerbehandlungen

In Konversationen mit einem Chatbot können Fehler auftreten. Entweder bricht der Benutzer die Unterhaltung unerwartet ab oder der Chatbot hat ein Problem bei der Verarbeitung von Daten [30]. Antwortet der Benutzer dem Chatbot in einem festgelegten Zeitintervall nicht, kann der Chatbot selbständig die Konversation beenden und ist für neue Benutzereingaben bereit. Bei Fehlern, die dem Chatbot unterlaufen, muss der Chatbot den Benutzer darüber in Kenntnis setzen. Ist es z.B. nicht möglich die Absicht des Benutzers herauszufinden, kann der Chatbot noch einmal nachfragen [30]. Wichtig dabei ist es, die Rückfrage, die der Chatbot stellt, zu spezifizieren und nicht die Schuld des Fehlers dem Benutzer zuzuschreiben.

Hilfe anbieten

Benutzer können bei einer Konversation ein Problem haben und benötigen Hilfe, weil sie etwas nicht verstanden haben [30]. Für den Chatbot gibt es unterschiedliche Möglichkeiten dem Benutzer zu helfen. Zum einen kann er seine Frage erneut stellen, wenn der Benutzer die Frage nicht verstanden hat oder sie neu formulieren. Der Kontext, in dem sich die Konversation aktuell befindet, spielt auch eine Rolle hinsichtlich der Frage, welche Hilfe vom Chatbot angeboten wird. In dem Wartungsprozess der DBIS Factory könnte der Benutzer eine Frage zu einem aktuellen Arbeitsschritt haben und benötigt im Kontext der Wartung Hilfe. Der Benutzer kann aber auch generell nach Hilfe fragen, um z.B. die verschiedene Funktionen des Chatbots kennenzulernen [30].

4.1.1 Konversationsfluss

Die in Kapitel 4.1 vorgestellten Eigenschaften einer Konversation mit einem Benutzer spielen bei dem Design eines Konversationsflusses eine wichtige Rolle. In der nachfolgend vorgestellten Implementierung finden diese vor allem bei dem Design der Wartungsprozesse ihre Anwendung (vgl. Kapitel 5).

Aufgaben, die ein Chatbot mit einem Benutzer durchführt, sind im Vorfeld bereits definiert (vgl. Kapitel 3.1.1). Aus den Aufgaben wird dann eine Konversation mit einem Benutzer gebildet und diese Art der Konversationen werden als *prozedural* bezeichnet

[32]. Prozedural geführte Konversationen zeichnen sich dadurch aus, dass der Chatbot dem Benutzer Fragen stellt und nicht der Benutzer dem Chatbot [32]. Bei einem Wartungsprozess kennt der Chatbot die einzelnen Arbeitsschritte, teilt diese dem Benutzer mit und wartet, bis der Benutzer mit der Aufgabe fertig ist und den nachfolgenden Schritt durchführt. Die Arbeitsschritte haben eine definierte Reihenfolge und werden vom Chatbot sequentiell ausgeführt.

Prozedural geführte Konversationen können Unterbrechungen enthalten, auf welche der Chatbot keinen Einfluss hat. Es kann somit nicht gewährleistet werden, dass Benutzer die Aufgaben zusammenhängend und in korrekter Reihenfolge erledigen, denn Menschen arbeiten oft nicht sequentiell und ändern ihre Entscheidungen während einer Konversation [32].

Im folgenden Beispiel führt der Chatbot eine Konversation und wartet auf die Antwort vom Benutzer. Der Benutzer ändert hierbei in der Konversation seine Absicht und möchte etwas anderes von Chatbot wissen.

Chatbot Bitte wechseln Sie den Filter an der Position 23.

Benutzer Was ist der aktuelle Status beim Hochregallager?

In diesen Fällen muss das Konzept hinter den Konversationen eine Entscheidung treffen, wie der Chatbot mit Unterbrechungen im Konversationsfluss umgehen soll. Die einfachste Möglichkeit ist, dass der Chatbot in einer Konversation nur auf eine passende Antwort reagieren kann und entsprechend die ursprüngliche Frage wiederholt [32]. Eine andere Lösung ist der Abbruch der aktuellen Konversation und das damit verbundene Zurücksetzen auf eine neue Konversation mit dem Benutzer [32]. Idealerweise kann der Chatbot auf solche Unterbrechungen reagieren und den aktuellen Stand der Konversation speichern, die Frage vom Benutzer verarbeiten und beantworten und dann zur ursprünglichen Konversation mit dem letzten Stand zurückkehren [32].

4.1.2 Konversationsfluss bei Wartungsprozessen

Für die Ausführung der Wartungsprozesse befolgt der Chatbot die sequentielle Abfolge der verschiedenen Aufgaben aus dem Geschäftsprozess (BPMN-Datei). Als Beispiel

für die Umsetzung eines BPMN-Geschäftsprozesses in eine Chatbot Konversation betrachtet man zunächst das Beispiel aus Abbildung 4.1. Dieses bildet eine generische Wartung für eine Maschine ab mit der Aufgabe unterschiedliche Ventile zu prüfen. Die erste Aufgabe ist die Maschine auszuschalten. Anschließend muss der Benutzer die Ventile prüfen und entscheiden, ob der Zustand in Ordnung ist oder nicht. Ist der Zustand nicht in Ordnung, werden die Ventile ausgetauscht. Andernfalls kann die Maschine wieder eingeschaltet werden und die Wartung ist beendet.

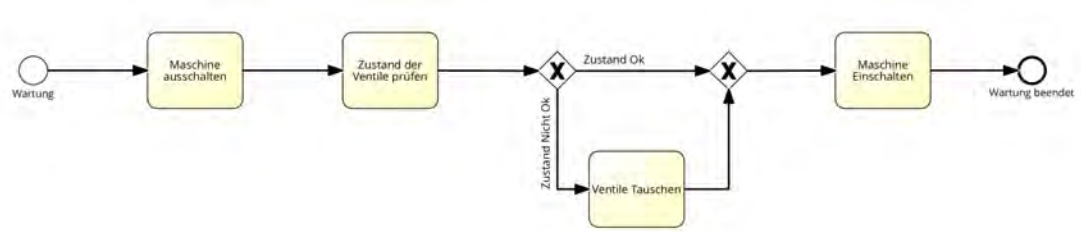


Abbildung 4.1: Beispiel Wartungsprozess

Das Übersetzen eines Geschäftsprozesses kann nun folgendermaßen ablaufen: Die einzelnen Aufgaben-Objekte werden als Frage vom Chatbot wiedergegeben; ihre Beschreibung ist die Ausgabe des Chatbots. Zu jeder Aufgabe gibt es noch zusätzliche Informationen in Form von einer Dokumentation, welche der Chatbot nach Rückmeldung vom Benutzer anzeigt und anschließend die ursprüngliche Frage wiederholt. Für die Navigation zwischen den einzelnen Aufgaben reagiert der Chatbot auf die Eingabe der Intents *weiter* und *zurück*. Der Intent *weiter* wechselt zur nächsten Aufgabe und intent *zurück* geht eine Aufgabe zurück und wiederholt diese. Weitere mögliche Intents sind *Abbrechen* und *Hilfe*. Mit *Abbrechen* hat der Benutzer jederzeit die Möglichkeit den Wartungsprozess zu unterbrechen.

Für die Verzweigung, welche eine Entscheidung vom Benutzer benötigt, gibt der Chatbot beide Auswahlmöglichkeiten aus und dieser kann dann aus beiden eine auswählen. Nach einer Verzweigung, ist es nicht mehr möglich einzelne Aufgaben zurückzugehen, sondern die Navigation beschränkt sich auf die Aufgaben nach der Verzweigung. Ist der Wartungsprozess am Ende angelangt, informiert der Chatbot den Benutzer über die abgeschlossene Wartung und wartet auf neue Eingaben.

Der Wartungsprozess aus Abbildung 4.1 abgebildet in einem generischen Chatbot mit einem Benutzer kann wie folgt aussehen:

Benutzer Ich würde gerne mit der Wartung beginnen.

Chatbot Okay.

Chatbot Maschine auschalten.

Chatbot (Weiter) (Zurück) (Hilfe) (Abbrechen)

Benutzer weiter

Chatbot Zustand der Ventile prüfen.

Chatbot (Weiter) (Zurück) (Hilfe) (Abbrechen)

Benutzer Wie macht man das?

Chatbot Das Ventil ausschrauben und auf Abnutzung Untersuchen.

Chatbot Zustand der Ventile prüfen.

Benutzer Okay erledigt was nun?

Chatbot Wie ist der Zustand der Ventile?

Chatbot (Zustand Ok) (Zustand Nicht Ok)

Benutzer Zustand ist in Ordnung.

Chatbot Maschine einschalten

Benutzer Stop

Aus diesem Konzept ist die Interaktion mit dem Benutzer umgesetzt, um Wartungsprozesse durchzuführen.

4.2 Architektur

Die Architektur des Chatbots setzt sich aus drei logisch eng zusammenhängenden Modulen zusammen [29]. Die Module sind getrennt voneinander implementiert und können auch als unabhängige Module in andere System Architekturen eingesetzt werden.

Aus dem Kapitel der funktionalen Anforderungen 3.2 lassen sich zwei Module ableiten, welche für die Chatbot Architektur relevant sind. Das erste Modul ist ein *Datenservice* zur Bereitstellung von unterschiedlichen Daten [31]. Die weiteren Module können mit dem Datenservice kommunizieren, um auf externe Informationen zuzugreifen. Externe Informationen können z.B. Wetter Daten sein, welche das Datenservice Modul von einer API lädt und den anderen Modulen zur Verfügung stellt. Das zweite Modul ist ein *Dialog Manager*, der die Mensch-Maschine Interaktionen abbildet [31]. Das *Dialog Manager* Modul stellt eine Chatplattform zum Schreiben und Empfangen von Nachrichten zwischen einem oder mehrerer Benutzer mit einem Chatbot bereit. Zusätzlich beinhaltet es die Logik für die Verarbeitung von Benutzereingaben, um eine passende Antwort zu erstellen [31].

Damit das *Dialog Manager* Modul die richtige Antwort für die Benutzereingabe erstellen kann, muss zuerst der Intent des Benutzers ermittelt werden. Dafür gibt es ein drittes Modul, welches die Aufgaben des *Natural Language Understanding* übernimmt und in Kapitel 2.4 vorgestellt wurde. Das Zusammenspiel zwischen den einzelnen Modulen ist in Abbildung 4.2 abgebildet. *Dialog Manager* baut eine Chatumgebung mit einem Benutzer auf und bekommt verschiedene Nachrichten übermittelt. Diese Nachrichten werden nach empfangen an das *Natural Language Understanding* Modul übergeben, welches den Intent des Benutzers ermittelt und dann als Antwort an das *Dialog Manager* Modul zurückgibt. Die verschiedenen Intents sind zusätzlich im *Dialog Manager* definiert und dienen als Einstiegspunkt um eine passende Antwort zu generieren. Sind bei einer Antwort zusätzliche externen Daten notwendig, kann das *Dialog Manager* Modul eine Anfrage an das *Datenservice* Modul senden.

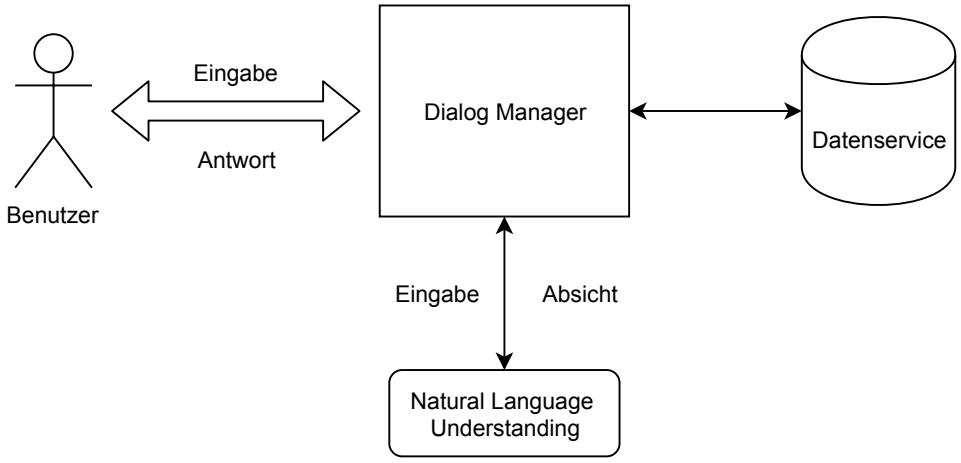


Abbildung 4.2: Architektur des Chatbots

4.3 Komponentenaufbau

Abgeleitet aus der Architektur in Kapitel 4.2 kann der folgende Komponentenaufbau definiert werden. Der Chatbot setzt sich aus vier einzelnen Komponenten zusammen, welche das System bilden. Die Unterteilung in verschiedene voneinander getrennte Komponenten ermöglicht es, leicht einzelne Elemente des Systems zu wechseln. Abbildung 4.3 zeigt die Zusammensetzung der vier Komponenten sowie deren Abhängigkeiten.

Die *dbisfactorybotserver* Komponente stellt Daten für den Chatbot bereit und spiegelt den *Datenservice* aus der Architektur wider (vgl. Kapitel 4.3.1). Daten können aus unterschiedlichen externen Diensten oder Systemen stammen und werden hier bereitgestellt und gegebenenfalls aufbereitet.

Die Aufgaben eines *Dialog Manager* übernimmt die Komponente *dbisfactorybot* (vgl. Kapitel 4.3.4). Es wird eine Chatumgebung mit den benötigten I/O Operationen realisiert, um Nachrichten zwischen Benutzer und System zu ermöglichen. Die Sprachaufbereitung und das Natural Language Understanding sind in einem externen Dienst ausgelagert und als Komponente *nlu-engine* dargestellt (vgl. Kapitel 4.3.2).

Als Erweiterung zur Architektur gibt es noch eine vierte Komponente für das Verwalten von Daten auf der *dbisfactorybotserver* Komponente (vgl. Kapitel 4.3.1). Die Verwaltung

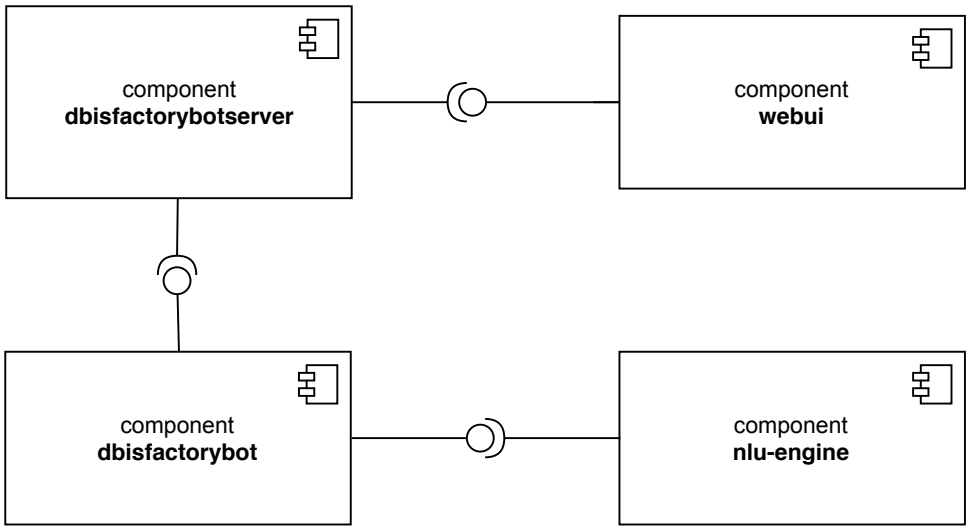
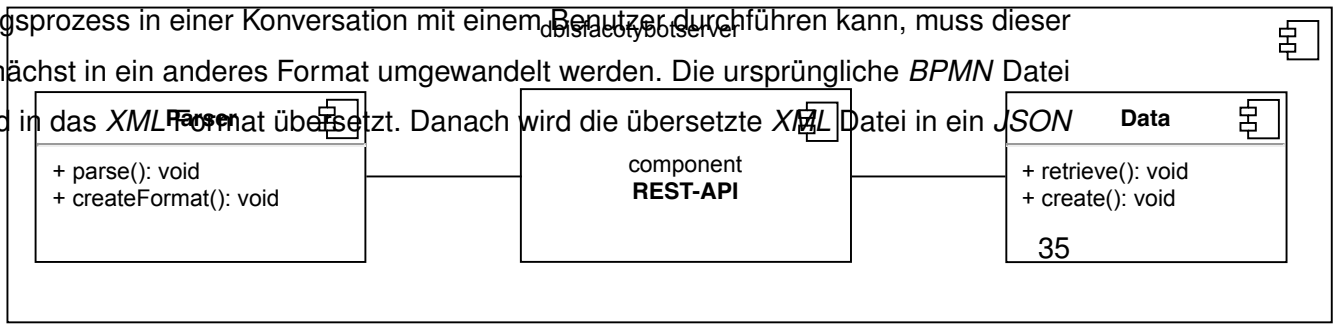


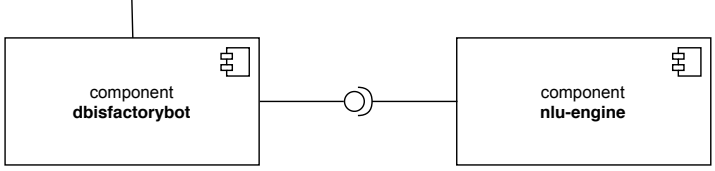
Abbildung 4.3: Komponentendarstellung des Chatbots

4.3.1 Komponente: dbisfactorybotserver

Die Schnittstellen- und Serverkomponente *dbisfactorybotserver* stellt dem Chatbot während seiner Laufzeit mittels API Daten über alle verfügbaren Prozesse, Sensordaten und den Status von einzelnen Maschinen der DBIS Factory (vgl. Kapitel 2.1) bereit. Darüber hinaus ist die Komponente verantwortlich für das Umwandeln von Prozessen auf ein neues internes JSON Format, welches der Chatbot anstelle eines BPMN-Formats benutzt.

Der Benutzer erstellt einen Wartungsprozess in einer Prozessmodellierungs-Umgebung und kann danach den erstellten Prozess über die *webui* (4.3.3) Komponente an die *dbisfactorybotserver* Komponente hochladen. Damit der Chatbot den erstellten Wartungsprozess in einer Konversation mit einem Benutzer durchführen kann, muss dieser zunächst in ein anderes Format umgewandelt werden. Die ursprüngliche *BPMN* Datei wird in das *XML* Format übersetzt. Danach wird die übersetzte *XML* Datei in ein *JSON*





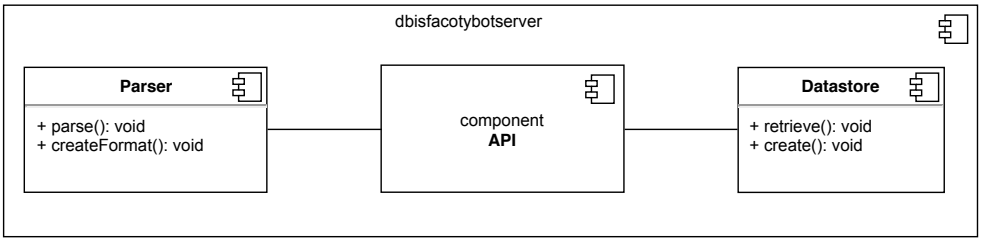


Abbildung 4.4: Innerer Aufbau der dbisfactorybotserver Komponente

4.3.2 Komponente: nlu-engine

Die *nlu-engine* Komponente ist eine Schnittstelle zum externen Dienst Dialogflow (vgl. Kapitel 2.4.1) und wird für die Verarbeitung von natürlicher Sprache eingesetzt. Ohne die *nlu-engine* Komponente könnte der Chatbot später keine natürlichen Eingaben verarbeiten, sondern nur passende Muster, welche zuvor definiert wurden. Das Trainieren von Sätzen und Phrasen, die der Chatbot später unterstützen soll, erfolgt dabei auf dem externen Dienst und nicht in der Chatbot Komponente selbst. Die Chatbot Komponente übermittelt alle Eingaben für das Intent-Matching an die *nlu-engine*, welche eine Antwort mit unterschiedlichen Informationen in Form eines *JSON* Dokumentes zurückgibt. Neben dem Intent können auch Parameter aus den Eingaben extrahiert werden und sind in der Antwort als *Entities* enthalten.

Um unabhängig von einem bestimmten Anbieter arbeiten zu können, kann der gewählte Dienst zu einem anderen Anbieter gewechselt werden. Die einzige Einschränkung dazu ist, dass die spätere Middleware in der Implementierung 5 angepasst werden muss.

4.3.3 Komponente: webui

Damit Benutzer ihre Prozesse an die *dbisfacotybotserver* Komponente hochladen können, wird eine Weboberfläche bereitgestellt. Die Weboberfläche ist in der Komponente *webui* umgesetzt und verfügt neben der Möglichkeit für das Hochladen auch über die Funktionen für das Umwandeln von den Prozessen. Dadurch das die Weboberfläche nur API Schnittstellen der *dbisfactorybotserver* Komponente anspricht, ist die Möglichkeit gegeben, Funktionen der Weboberfläche komplett in den Chatbot zu integrieren. Dadurch kann ein zusätzlicher Dienst gespart werden und der komplette Prozess von Hochladen und Konvertieren wird vom Chatbot übernommen.

4.3.4 Komponente: dbisfactorybot

Die Umsetzung des *Dialog Manager* Moduls findet in der Komponenten *dbisfactorybot* statt. Als Vorlage dient ein Chatbot Framework, aus welchem sich die folgenden Komponenten abbilden lassen und in 4.5 dargestellt sind.

In der *Skills* Komponente werden die Funktionalitäten (Skill) des Chatbots definiert. Für jede Funktionalität wird ein Intent bzw. Pattern deklariert auf das der Chatbot später reagieren soll. Passt die Benutzereingabe auf einen definierten Skill, kann ein internes Event gefeuert werden. Die Logik für eine vordefinierte Konversation kann daraufhin auf das interne Event reagieren und eine Konversation mit dem Benutzer starten.

Die *Chatplattform* Komponente stellt eine webbasierte Chatumgebung bereit, um mit dem Chatbot kommunizieren zu können. Die *Chatplattform* kommuniziert mit dem Chatbot über eine WebSocket-Schnittstelle, um Nachrichten zwischen Benutzer und Chatbot zu ermöglichen. Möchte man einen externen Nachrichten Dienst verwenden (z.B. Slack), kann die *Chattplattform* Komponente mit dem gewählten Nachrichten Dienst ersetzt werden.

Die *Datenbank* Komponente ist ein interner Speicher, auf den der Chatbot zurückgreifen kann, um vergangene Unterhaltungen zu speichern oder Benutzereingaben zwischenzuspeichern. Ohne zusätzliche Konfiguration ist die *Datenbank* Komponente nur ein *Key - Value* Speicher. Die Komponente kann aber um eine SQL-basierte Datenbank erweitert werden.

Eine weitere Komponente ist die *Bot* Komponente. In dieser kann der Chatbot konfiguriert werden z.B., welche NLU-Engine verwendet werden soll und ob der Chatbot in einer externen Chatplattform integriert ist.

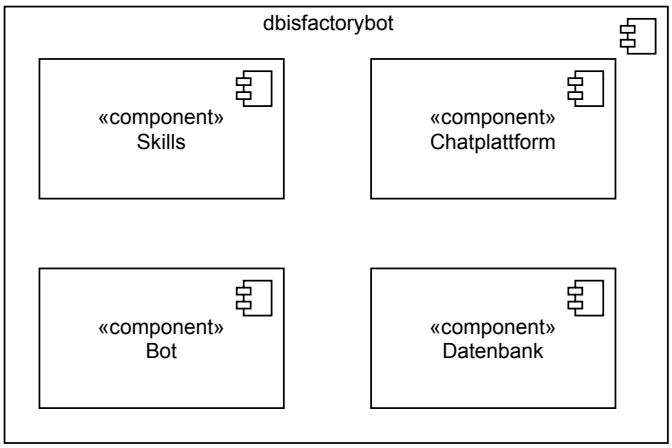


Abbildung 4.5: Innerer Aufbau der dbisfactorybot Komponente

5

Implementierung

In diesem Kapitel wird der Prototyp eines Chatbots vorgestellt. Als Basis für den Prototypen wurde das Kapitel 3 und 4 verwendet. Der allgemeine technische Ablauf einer Unterhaltung und die Kommunikation zwischen den Komponenten ist im Sequenzdiagramm in Abbildung 5.1 abgebildet. Das Sequenzdiagramm dient als Grundlage für die vorgestellten Implementierungen und zeigt einen vereinfachten Ablauf einer möglichen Konversation zwischen Chatbot und Benutzer. Die Nachrichten zwischen Benutzer und Chatbot mit der Bezeichnung *conversation* und *answer* verallgemeinern das Austauschen von Chat Nachrichten im entsprechenden Kontext der Unterhaltung. Im abgebildeten Diagramm besitzt die Konversation außerdem kein Ende der aktuellen Unterhaltung. Entsprechend werden die letzten Nachrichten *conversation*, *answer*, *intent matching* und *return intent* für die gesamte Dauer der Unterhaltung wiederholt. Zusätzlich ist der Ablauf für das Hochladen eines Geschäftsprozesses im ersten Schritt dargestellt.

Für die Implementierung wurden unterschiedliche Technologien eingesetzt, welche im nachfolgenden Unterkapitel 5.1 genauer beschrieben werden. Die eingesetzte Programmiersprache für alle entwickelten Komponenten ist JavaScript.

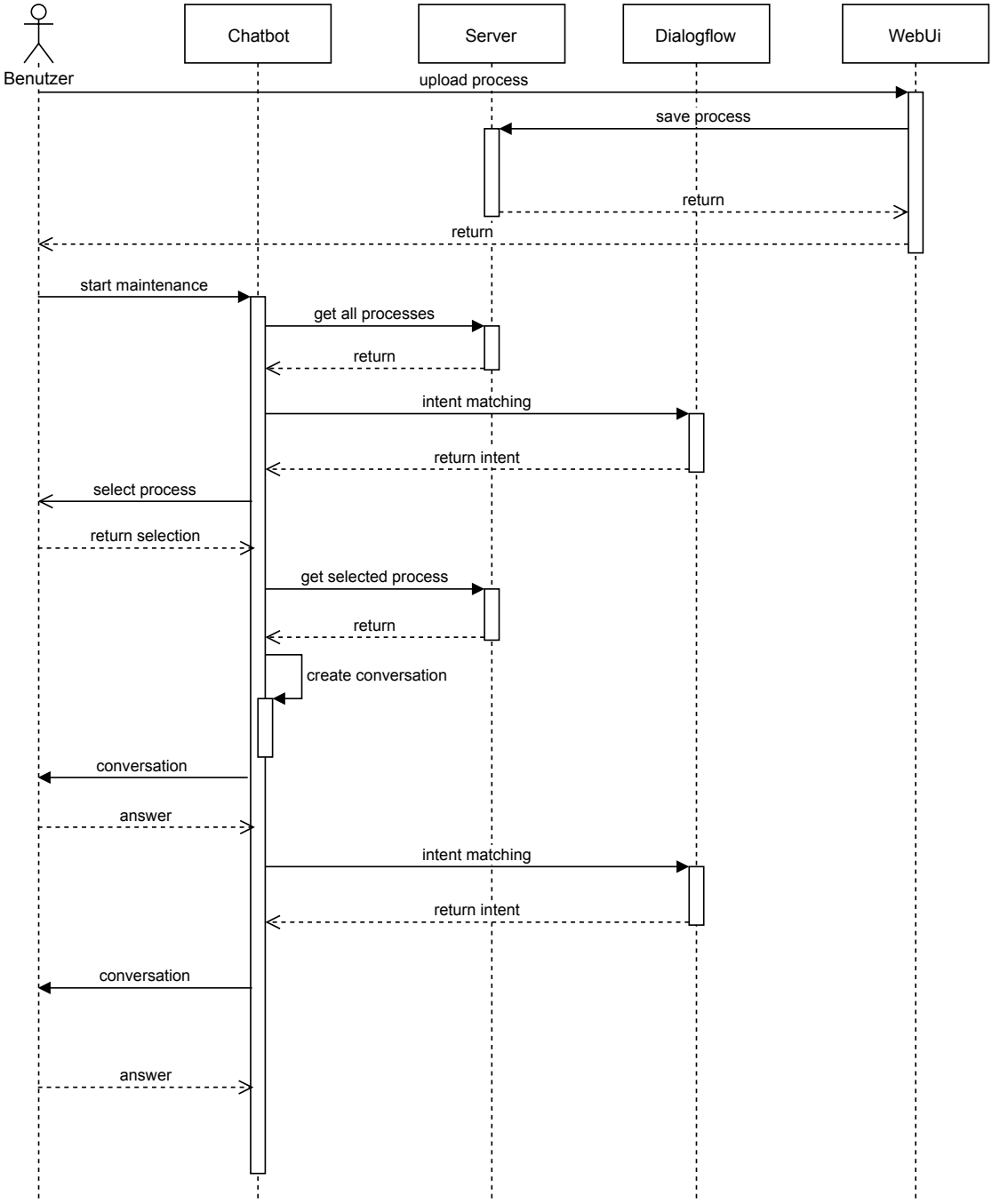


Abbildung 5.1: Sequenzdiagramm: Technischer Ablauf der Komponenten

5.1 Verwendete Technologien

5.1.1 JSON

JavaScript Object Notation (JSON) ist ein programmiersprachen-unabhängiges Dateiformat zum Austausch von Daten [33]. JSON ist auf Text Basis ausgelegt und kann dadurch leicht von einem Menschen gelesen bzw. von einer Maschine interpretiert werden. Am häufigsten eingesetzt wird es im Web, bei der Kommunikation zwischen Server und Client.

Für den Austausch von Daten zwischen den Komponenten (Chatbot, NLU Engine, Server) wird das JSON Format verwendet.

5.1.2 NodeJS

NodeJS ist eine „asynchrone, ereignisgesteuerte JavaScript Laufzeitumgebung“ [34] und wurde speziell für Netzwerk-anwendungen konzipiert. Mit *NodeJS* werden alle benötigten Module bereitgestellt, um JavaScript-anwendungen ausführen zu können. Das Besondere an *NodeJS* ist die Skalierbarkeit, ohne dass andere Prozesse die Anwendung blockieren. *NodeJS* verwendet deshalb komplett asynchrone Operationen um z.B. Daten aus einer Datenbank zu laden oder auf das Dateisystem zuzugreifen [35]. Das Abarbeiten von Aufgaben passiert in *NodeJS* nacheinander basierend auf einer *ready-to-execute* Liste. Ist eine Aufgabe darunter, welche eine I/O Operation ausführen möchte wird die aktuelle Aufgabe abgebrochen und eine neue Aufgabe für die Antwort erstellt. Die neue Aufgabe, welche auf das Ergebnis der I/O Operation wartet, wird erst dann in die *ready-to-execute* Liste eingetragen, wenn die Operation fertig ist [35]. Als Entwickler muss man deshalb *callback* Funktionen definieren, welche ausgeführt werden, wenn das Ergebnis bereitsteht.

Eine *NodeJS* Umgebung kann mit zusätzlichem Software Paketen erweitert werden. Diese Pakete werden über *Node Packaged Modules* (Oder kurz *npm*) zur Verfügung gestellt. Externe Software Pakete oder Module können dann mittels *npm install* installiert werden und im Code über *require* eingebunden werden [35].

Für die Umsetzung der Schnittstelle aus Kapitel 4.3.1 wird ein solches externes Modul eingesetzt und im nächsten Abschnitt vorgestellt.

5.1.3 ExpressJS

Die Kommunikation zwischen den einzelnen Komponenten (vgl. Kapitel 4.3) ist über eine REST-Schnittstelle realisiert. Das Bereitstellen von einer REST-Schnittstelle ist häufig mit dem folgenden Prozess verbunden [36]:

- Übersetzen des HTTP-Requests.
- Lesen von cookies.
- Mehrere aktive Sitzungen verwalten.
- Verschiedene Routen über die Ressourcen angefragt werden können verwalten.
- HTTP-Response Header bereitstellen basierend auf dem angefragten Datentyp.

Dieser gesamte Prozess wird von *Express* übernommen und darüber hinaus wird eine MVC Struktur für Webanwendungen bereitgestellt [36].

In einer *Entry* (ähnlich der Main Methode in Java) Datei wird das *Express* Modul eingebunden und in dieser Datei können dann Middlewares und Routen definiert werden. Die einfachste und schnellste Möglichkeit das *Express* Modul einzusetzen ist in 5.1 dargestellt. Auf dem Loaklen Port *3000* wird eine Server Instanz erzeugt und eine REST-Schnittstelle mit einer *GET* Anfrage für die Route / angeboten, welche einen einfachen Text zurückliefert.

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => res.send('Hello World!'))
6
7  app.listen(port, () => console.log(`Server is online`))
```

Listing 5.1: Express Framework 'Hello World' Beispiel [37]

Ressourcen auf dem Server können dann nach folgenden Muster [38] in *Express* erstellt werden, auf die später der Client über einen HTTP Request zugreifen kann. *App* ist eine Instanz von Express. *Method* ist die angefragte HTTP-Methode. *Path* Pfad auf dem Server, wo die Resource zu finden ist. *Handler* Funktion, welche ausgeführt wird wenn die Route zutrifft.

```
app.METHOD(PATH, HANDLER)
```

Listing 5.2: Ressourcen bereitstellen mittels Express [37]

5.1.4 Botkit

Botkit [25] ist das Framework, mit dem die Chatbot Anwendung implementiert ist. *Botkit* bietet Entwicklern ein einfaches Grundgerüst für die Chatbot Entwicklung und ermöglicht es selbst Code zu implementieren, wodurch die Funktionalität vom Chatbot individuell angepasst und erweitert werden kann. In diesem Unterkapitel werden die verschiedenen Funktionalitäten vorgestellt, welche von Botkit out-of-the-box bereitgestellt werden.

Für die Chatumgebung gibt es unterschiedliche Chatplattformen welche von *Botkit* unterstützt werden. Im Prototyp wurde die Webplattform von *Botkit* ausgewählt. Dadurch kann der Chatbot einfach über eine Webseite aufgerufen und verwendet werden, dass dazugehörige Chatfenster ist in 5.2 abgebildet.

Im nachfolgenden sind die unterschiedlichen Funktionalitäten von Botkit aufgelistet und genauer erläutert.

Abbildung 5.2: Botkit Chatfenster

Events

Botkit reagiert auf eine Vielzahl an unterschiedlichen Events und kann zusätzlich mit eigenen Events erweitert werden [39]. Zusätzliche Events können entweder interne, welche von der Anwendung ausgelöst werden, oder Chatplattformen spezifische Events sein. Mit Events kann auf Eingaben von Benutzern reagiert werden und in 5.3 ist dargestellt, wie Botkit auf das Event *message_received* reagieren kann. Das *Controller* Objekt wird von Botkit bereitgestellt und die Funktion *on* erwartet das Event, auf welches reagiert werden soll zusammen mit einer *callback* Funktion. Die *Callback* Funktion bekommt die beiden Objekte *Bot* und *Message*. *Bot* ist die Bot instanz und *Message* die Nachricht des Benutzers.

```
1     controller.on('message_received',
2                   function(bot, message) {
3       // Do something
4     });
```

Listing 5.3: Verarbeitung eines Events in Botkit

Im Beispiel aus 5.3 wird das Event aber jetzt bei jeder Benutzereingabe ausgelöst. Möchte man auf eine bestimmte Benutzereingabe reagieren kann die *Controller* Funktion *hears* benutzt werden.

Hears

Botkit kann ohne Erweiterungen nur auf definierte Schlüsselwörter oder Patterns antworten. Kann die Benutzereingabe nicht zugeordnert werden, reagiert der Chatbot nicht. Dies führt besonders zu Problemen, wenn der Intent eines Benutzers nicht genau einem definierten Pattern entspricht. Bei der Auswertung von Benutzereingaben auf ein bestimmtes Pattern arbeitet Botkit nach dem top-down Prinzip. Das Pattern, welches zuerst mit der Benutzereingabe übereinstimmt, wird von Botkit ausgewertet. Dies kann zur Folge haben, dass Eingaben eine unerwünschte Aktion auslösen können. Nimmt man als Benutzereingabe das Wort *another* und hat ein Pattern definiert, welches auf *no* reagiert kann es passieren, dass *Botkit* die Funktionalität zu dem Pattern *no* ausführt.

Die *hears* Funktion bekommt als Argument entweder einen einzelnen String oder eine Liste von Strings auf welche reagiert werden soll. Die beiden Möglichkeiten sind in 5.4 abgebildet.

```
1     controller.hears('Hello', 'message_received',
2                      function(bot, message) {
3       // Do something
4     });
5
6     controller.hears(['hi','hello'],['message_received'],
7                      function(bot,message) {
8       // Do something
9     });
```

Listing 5.4: Verarbeitung einer Benutzeranfrage in Botkit

Bis jetzt kann der Chatbot auf Events reagieren und Benutzereingaben, welche zu einem Schlüsselwort oder Pattern passen erkennen, aber noch nicht dem Benutzer antworten.

Replies

Für das Erstellen von Antworten kann das *Bot* Objekt verwendet werden mit der Funktion *reply* (vgl. Listing 5.5). Als Parameter erhält die Funktion das *message*

Objekt und einen String mit der Ausgabe. Die *reply* Funktion kann überall im Quellcode verwendet werden, solange man Zugriff auf das *Bot* und *message* Objekt hat.

```
bot.reply(message, 'Example response');
```

Listing 5.5: Antworterstellung in Botkit

Hears und *reply* sind die Grundlagen für den Chatbot zur Interaktion mit Benutzern. Mit den aktuellen Funktionalitäten von unserem Chatbot können aber nur direkte Benutzereingaben verarbeitet werden. Es gibt noch keine Möglichkeit, dass der Chatbot den Benutzer eine Frage stellt und mit der Antwort vom Benutzer weiterarbeitet. In *Botkit* gibt es deshalb Konversationen, welche erstellt werden können und der Chatbot den Benutzer eine oder mehrere Fragen stellen kann und die Antworten vom Benutzer extrahiert und für die Weiterverarbeitung bereitstellt.

Conversations

Im ersten Schritt wird eine *hears* Funktion erstellt zusammen mit einem Schlüsselwort, dass die Konversation auslösen soll. *Botkit* hat zwei unterschiedliche Konversationsfunktionen, um unterschiedliche Anwendungsfälle abbilden zu können.

Die erste Funktion ist *startConversation* und wird hauptsächlich für kurze Frage und Antworten benutzt, welche keine tiefen Verzweigungen beinhalten. Innerhalb der *startConversation* gibt es dann eine *callback* Funktion mit einem neuen Objekt *convo*. *Convo* ermöglicht es nicht nur Antworten zu erstellen, sondern auch Fragen an den Benutzer zu richten und mit der Antwort weiterzuarbeiten. Der gesamte Konversationsablauf ist in 5.6 dargestellt. Die Funktion *next()* signalisiert Botkit das nächste Konversationsobjekt auszuführen.

```
1     bot.startConversation(message, function(err,convo) {
2       convo.addQuestion('How are you?',
3                         function(response, convo) {
4         convo.say('Your input was: ' + response.text);
5         convo.next();
6       }, {}, 'default');
7     });
```

Listing 5.6: Erstellung einer Konversation in Botkit

Die zweite Funktion ist *createConversation* und kann verwendet werden, wenn viele unterschiedliche Pfade in einer Konversation auftreten.

```
1     bot.createConversation(message, function(err,convo) {
2       convo.activate();
3     });
```

Listing 5.7: Erstellung einer Konversation in Botkit - alternativer Ansatz

Wichtig bei der Funktion *createConversation* gegenüber von *startConversation* ist, dass die Konversation zusätzlich mit der Funktion *convo.activate()* initialisiert werden muss. Betrachtet man als Beispiel eine Konversation in der mehrere Fragen vom Chatbot gestellt werden.

Als erstes definiert man mit der Funktion *addQuestion* die einzelnen Frageblöcke und fügt eine Bezeichnung hinzu (vgl. im oberen Beispiel 5.6 die Bezeichnung *'default'*). Diese Bezeichnung wird intern in *Botkit* als sogenannte *thread_id* gehandhabt und ermöglicht die Navigation zwischen verschiedenen Konversationsfunktionen (z.B. *addQuestion*) innerhalb einer *createConversation* Funktion. Die Navigation wird mit der Funktion *gotoThread* aufgerufen und bekommt als Parameter die *thread_id* als String. Eine besondere *thread_id* ist *'default'* und wird von *Botkit* als Startpunkt für die Konversation genutzt. Im Quellcode lässt sich diese Funktionalität folgendermaßen abbilden 5.8, wobei die Funktion *addMessage* dazu dient, eine Nachricht in eine Konversation einzufügen (ähnlich wie *bot.reply*):

```
1      convo.addMessage({text: "Hello World",}, 'my_thread');
2
3      convo.addQuestion('Thread change',
4                          function(response,convo) {
5        convo.gotoThread('my_thread);
6        }, {}, 'default');
7      });
```

Listing 5.8: Navigation innerhalb einer Botkit Konversation

Der Chatbot kann mehrere Fragen stellen und zwischen unterschiedlichen Nachrichten und Fragen innerhalb der Konversation navigieren. Für eine Frage vom Chatbot kann es unterschiedliche Antworten vom Benutzer geben. Um zwischen diesen unterscheiden zu können, kann ein Array von Objekten mit unterschiedlichen Patterns der *addQuestion* Funktion übergeben werden (vgl. Listing 5.9). Die Objekte erhalten einen String mit einem Schlüsselwort und eine *callback* Funktion, welche aufgerufen wird, wenn das entsprechende Pattern zutrifft.

```
1      convo.addQuestion('Yes or no?', [
2        {
3          pattern: 'yes',
4          callback: function(response,convo) {
5            // do something
6          },
7        },
8      ], {}, 'default');
```

Listing 5.9: Erstellung von Antwort-Patterns in Botkit

Mit Konversationen ermöglicht *Botkit* sowohl einfache als auch komplexe Dialoge mit Benutzern zu führen.

Allerdings kann es zu Problemen führen, als Benutzer das definierte Pattern auf eine Frage korrekt einzugeben. Aus diesem Grund bietet *Botkit* die Möglichkeit mit

Nachrichten eine Liste an passenden Antworten oder Begriffen anzuzeigen, aus denen der Benutzer wählen kann.

Quick Replies

Quick Replies sind ein einfaches Mittel dem Benutzer während der Unterhaltung mit dem Chatbot aktiv zu unterstützen. So können häufig genutzte Funktionen direkt angezeigt werden oder Auswahlmöglichkeiten dargestellt werden, ohne dass der Benutzer alle Möglichkeiten im Vorfeld kennen muss. *Quick Replies* können als zusätzliches Objekt einer *reply* oder *addQuestion* Funktion als Parameter übergeben werden. Das Objekt besteht aus einer Liste von *quick_replies* Objekten, welche einen *title*- und *payload* Attribut besitzt. Der *title* wird dem Benutzer angezeigt und *payload* wird als Antwort dem Chatbot übergeben. Die genaue Implementierung ist in Listing 5.10 zu finden, die spätere Darstellung auf der Chatoberfläche wird in Abbildung 5.3 dargestellt.

```
1     bot.reply(message, {
2         text: "Would you like to start with the maintenance?",
3         quick_replies: [
4           {
5             title: "Yes",
6             payload: "yes",
7           },
8           {
9             title: "No",
10            payload: "no",
11          }
12        ]});
```

Listing 5.10: Erstellung von ‘Quick Replies’ in Botkit

Die vorgestellten Implementierungen sind die grundlegenden Funktionalitäten, welche von *Botkit* out-of-the-box angeboten werden, um einen Chatbot umzusetzen. Basierend auf diesen ist die Chatbot Applikation in 5.3 umgesetzt und dienen als Grundlagen für spätere Designentscheidungen.

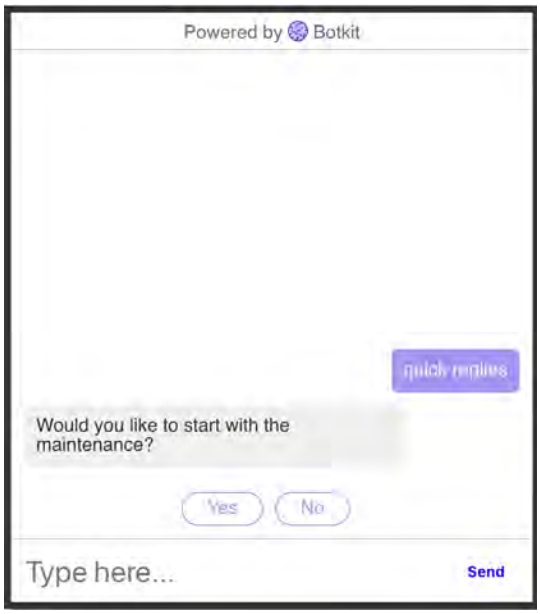


Abbildung 5.3: Botkit Chatfenster: Quick Responses Darstellung

5.2 Server Applikation

Die Server Applikation ist die technische Umsetzung der *dbisfactorybotserver* Komponente aus Kapitel 4.3.1. Im Folgenden sind die Details der einzelnen umgesetzten Komponenten genauer erläutert.

5.2.1 Parser Implementierung

Die Geschäftsprozesse werden im BPMN Format erstellt und müssen erst in ein internes Format umgewandelt werden, damit der Chatbot die Prozesse als Vorlage für die Konversation verwenden kann. Der Server übernimmt den gesamten Prozess der Umwandlung für den Chatbot. Der Ablauf ist im Sequenzdiagramm 5.4 dargestellt.

Die erste Funktion *parseProcess* übersetzt die ursprüngliche BPMN Datei von XML nach JSON mithilfe der externen Bibliothek *xml-js* und speichert die neue JSON Datei. Auf der Basis der neuen JSON Datei werden nun die Informationen aus dem ursprünglichen Prozess gelesen und in ein eigenes Format umgewandelt. Das interne Format für den

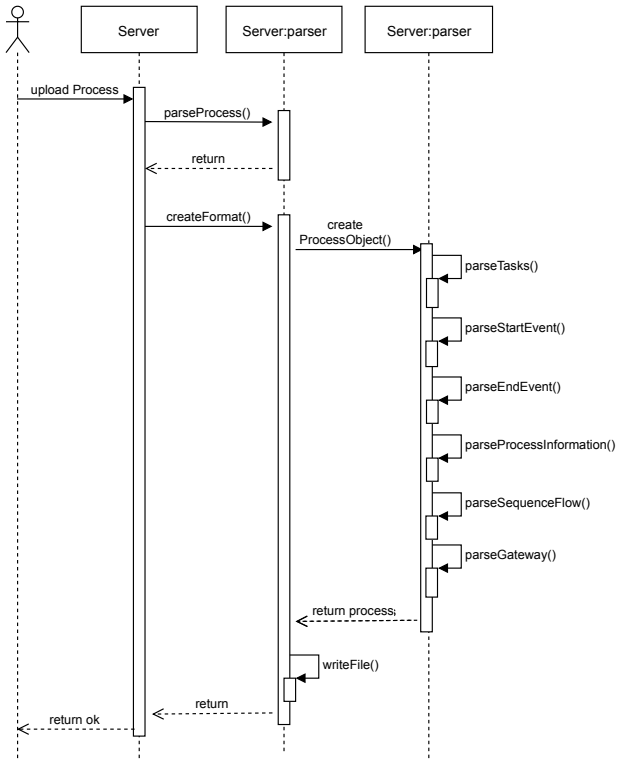


Abbildung 5.4: Sequenzdiagramm: Server Parser Ablauf

Chatbot beinhaltet die einzelnen Prozesse aus der BPMN Datei mit den folgenden Informationen: *processInformation*, *startEvent*, *endEvent*, *sequenceFlow*, *tasks* und *gateways*. Die Funktion *createFormat* liest das JSON Dokument und ruft die Funktion *createProcessObject* auf, welche für jede benötigte Information einen Parser aufruft, welcher aus dem Dokument die Daten extrahiert und in ein neues JSON Objekt schreibt. Sind alle benötigten Informationen gelesen, wird abschließend die Funktion *writeFile* aufgerufen und das neue JSON Objekt wird als String in einer separaten Datei gespeichert.

5.2.2 REST API

Betrachtet man den Ablauf der Umwandlung aus 5.2.1 benötigt der Server eine Verbindung zwischen Chatbot, für das Bereitstellen der Geschäftsprozesse und einem

Benutzer, um das Hochladen von einer BPMN Datei zu ermöglichen. Diese Verbindung wird über eine REST API (vgl. *FA04-S*) und der externen Bibliothek *Express* (vgl. Kapitel 5.1.3) realisiert. Mithilfe der *Express* Bibliothek werden in Tabelle 5.1 die implementierten REST-Endpunkte für Chatbot und Benutzer beschrieben.

URL	HTTP Methode	Beschreibung
/upload	POST	Endpunkt für das Hochladen von Dateien
/parse	POST	Übersetzt alle XML Dateien auf dem Server in JSON Dateien
/convert	POST	Konvertiert alle JSON Dateien in das interne Format
/processes/	GET	Liefert eine Liste mit allen hochgeladenen und konvertierten Prozessen zurück
/data	GET	Liefert die Daten der DBIS Factory zurück
/stations	GET	Liefert ein Objekt mit den Abkürzungen der Stationen zurück
/process	GET	Gibt einen ausgewählten Prozess zurück

Tabelle 5.1: Verfügbare REST-Endpunkte

Die Interaktion mit dem Server zwischen Benutzer und Chatbot ist im Sequenzdiagramm 5.5 abgebildet. Das parallele Arbeiten ist aufgrund der Komplexität nicht im Sequenzdiagramm dargestellt, aber der Zugriff auf die Endpunkte kann zu jeder Zeit stattfinden und auch mehr als ein Benutzer oder Chatbot kann Daten hochladen bzw. abrufen. Der Benutzer benötigt die Endpunkte für das Umwandeln der Geschäftsprozesse in ein Chatbot internes Format. Der Chatbot hingegen bedient sich nur an den bereitgestellten Daten und stellt eine Anfrage, wenn diese benötigt werden. Für den Fall, dass noch keine Daten vom Benutzer hochgeladen sind, kann der Server dem Chatbot keine Informationen zurückgeben und der Chatbot ist nicht in der Lage vollständig zu arbeiten.

Die Struktur der zurückgelieferten Daten soll anhand von folgenden Beispielen genauer betrachtet werden.

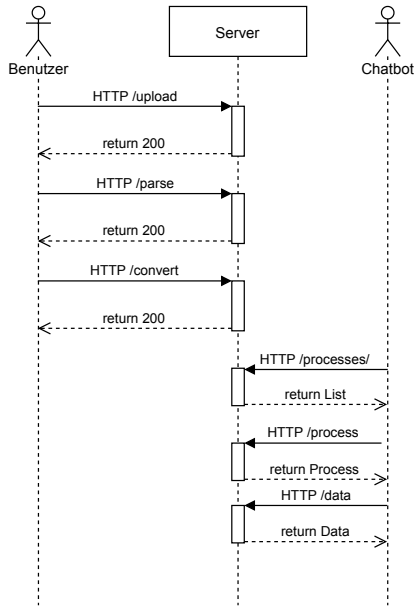


Abbildung 5.5: Sequenzdiagramm: Kommunikation zwischen Server, Benutzer und Chatbot

Der Endpunkt */processes/* liefert eine Liste im Dateiformat JSON mit den Bezeichnungen der einzelnen verfügbaren Prozessen als Strings zurück. Dabei werden nur die bereits konvertierten Prozesse, welche im internen Chatbot Format vorliegen, zurückgegeben.

```
[ "Furnance",
  "HighRack",
  "SortingLine",
  "VacuumGripper" ]
```

Listing 5.11: Datenmodell: */processes/*

Der DBIS Factory Endpunkt */data* gibt eine Liste mit den unterschiedlichen Stationen als einzelne Objekte zurück. Ein *Stations-Objekt* (vgl. Listing 5.12) hat ein Feld für den Namen der Station und zwei weitere Objekte mit Informationen für die nächste Wartung und den Status der aktuellen Station.

```
{ "station": "FUR",
  "maintenance": { ... },
  "status": { ... }, }
```

Listing 5.12: Datenmodell: *Stations-Objekt*

Das *Wartungs-Objekt* (vgl. Listing 5.13) wird vom Server selbst erstellt und ist für die Simulation einer größeren Anlage konzipiert, um darzustellen, dass unterschiedliche Informationen angeboten werden können. Für die Simulation gibt es deshalb zwei Zeitstempel über die nächste und letzte Wartung, ein Verweis auf die Dokumentation und ob die aktuelle Station eine Wartung benötigt.

```
{ "required": false,
  "last": "2019-2-6",
  "next": "2019-2-6",
  "documentation": "FUR Maintennace", }
```

Listing 5.13: Datenmodell: *Wartungs-Objekt*

Das *Status-Objekt* (vgl. Listing 5.14) verwaltet die Daten, welche von der DBIS Factory erstellt werden. Es beinhaltet einen Zeitstempel, Stations Bezeichnung, eine Variable, die eine Sensor Art repräsentiert und einen Wert für den entsprechenden Sensor.

```
{ "ts": "2019-2-6",
  "station": "FUR",
  "variable": "SensorD",
  "value": 87 }
```

Listing 5.14: Datenmodell: *Status-Objekt*

Informationen zu den Abkürzungen der Stationen können mit dem Endpunkt */stations* erhalten werden. Diese Daten werden im Server festgelegt und können nicht von außen geändert werden. Als Rückgabe bekommt der Chatbot ein Objekt mit den folgenden Informationen:

```
{ "FUR": "FUR" , "SL": "SL" , "VG": "VG" , "HR": "HR" }
```

Listing 5.15: Datenmodell: */stations*

Aktuell kann der Chatbot sich alle verfügbaren Prozesse, die Daten der DBIS Factory und Bezeichnungen der Stationen als API Anfrage an den Server stellen. Im letzten Endpunkt der REST Schnittstelle */process* kann über den Parameter *name* ein ausgewählter Prozess und dessen Informationen abgefragt werden. Der Server antwortet mit einer Liste von unterschiedlichen Prozessen, die zuvor in der Parser Komponente 5.2.1 erstellt worden sind. Eine Liste von Prozessen kann zustande kommen, wenn sich der Geschäftsprozess aus unterschiedlichen Prozessen zusammensetzt. In den meisten Fällen beschränkt sich aber die Liste auf einen einzelnen Prozess. Das Datenmodell der Antwort ist in 5.16 dargestellt.

```
[{ processInformation: {...},
   startEvent: {...},
   endEvent: {...},
   sequenceFlow: [{...}],
   tasks: [{...}], }]
```

Listing 5.16: Datenmodell: */station?name=*

Alle Objekte innerhalb von einem Prozess sind zusätzlich mit einer *ID* versehen, um später über die Liste *sequenceFlow* die richtigen *Tasks* identifizieren und auswählen zu können. Das Objekt von einem *sequenceFlow* Eintrag (vgl. Listing 5.17) hat Informationen über den vorherigen *Task*, dem nächsten *Task* und ein *Name* Feld für die Bezeichnung bei *Gateways*. Ist in einem Prozess kein Gateway vorhanden oder besitzt das Gateway keine Bezeichnung, enthält das Feld einen leeren String.

```
{ id: "sid-534F4A99-8195-404F-AB0A-F85091C7BE42"
  sourceRef:  "sid-B0B2F389-3368-4097-A854-2548F2FDF443"
  targetRef:  "sid-BBC452D2-DBA5-414E-923D-4EAC4795B7CA"
  name:       "" }
```

Listing 5.17: Datenmodell: *sequenceFlow*

Ein *Task* Objekt bildet eine Aufgabe für den Benutzer ab und wird vom Chatbot als Frage dem Benutzer angezeigt. Das Datenmodell (vgl. Listing 5.18) besteht aus einem *Namen*, einem Feld für den vorherigen und nachfolgenden *Task* und der *Dokumentation*. Zusätzlich gibt es noch interne Felder, welche aber in den meisten Fällen nicht verwendet werden und deshalb nicht erwähnt sind.

```
{
  id: "sid-04C55345-9FC2-42D2-B970-2E8C05FDF19B",
  name:       "Maschine vorbereiten",
  documentation: {
    id        "sid-28ac228b-1f45-4776-9a99-4eeb4116e99b",
    text      "Maschine leerfahren, Abdeckungen entfernen",
  },
  incoming:   "sid-762E2227-053B-4FCD-A39F-62998DB5D550"
  outgoing:   "sid-713FF9F7-7F31-4713-B59D-1E247219E743"
}
```

Listing 5.18: Datenmodell: *task*

Für ein vollständiges Datenmodell zu einem Prozess wird auf den Quellcode verwiesen.

5.3 Chatbot Applikation

In diesem Abschnitt geht es um die Implementierung der Komponente *dbisfactorybot* (vgl. Kapitel 4.3.4) und der Umsetzung von einem Chatbot. Das Grundgerüst für die Kommunikation und der Chatumgebung bildet das externe Framework *Botkit* (vgl. Kapitel 5.1.4).

Im ersten Schritt werden die sogenannten *skills* definiert, also die Funktionalitäten vom Chatbot. *Botkit* gibt dabei die Struktur vor, dass sich alle Funktionen vom Chatbot in diesem Ordner befinden müssen.

5.3.1 Intent Matching

Aus dem Kapitel der funktionalen Anforderungen (vgl. Kapitel 3.2) und Anforderungsanalyse (vgl. Kapitel 3.1) muss der Chatbot Wartungen und das Condition Monitoring unterstützen. Aus diesen Vorlagen werden die folgenden Benutzerabsichten definiert: *Maintenance-Start*, *Maintenance-Select*, *Maintenance-Resume*, *Machine-Status* und *Machine-Maintenance*. *Botkit* ist aber nur in der Lage auf Patterns und Schlüsselwörter zu reagieren (vgl. Kapitel 5.1.4). Ein allgemeines Pattern für diese Benutzerabsichten zu finden ist nahezu unmöglich und aus diesem Grund, wird eine NLU Komponente für das Intent Matching benötigt.

Deshalb wird die externe *Middleware* für *Dialogflow* in *Botkit* eingebunden und die interne *hears* Funktion mit der *middleware hears* Funktion überschrieben. Das Besondere an *Botkit* und *Middlewares* ist der einfache Tausch der entsprechenden Erweiterung. Für das Einbinden von *Dialogflow* benötigt es nur die folgenden Codezeilen 5.19. Die *keyFilename* ist eine Datei mit Informationen über einen Google Service Account , welcher benötigt wird, um auf den Dialogflow Dienst zuzugreifen [40].

```
1  const dialogflowMiddleware =
2  require('botkit-middleware-dialogflow')({
3    keyFilename: './mybot-service-key.json',
4  });
5
6  controller.middleware.receive.use(
7    dialogflowMiddleware.receive
8  );
```

Listing 5.19: Einbindung der Dialogflow Middleware

Zusammen mit der eingebunden NLU Komponente können die *hears* Funktionen definiert werden, um den Intent des Benutzers dem Chatbot mitzuteilen. Dafür muss die ursprüngliche *hears* Funktion um ein Array erweitert werden, welches den definierten Intent von Dialogflow beinhaltet und als zusätzlichen Parameter das *dialogflow-hears* Objekt.

```
1  controller.hears(["Maintenance-Start"],
2                   "message_received",
3                   dialogflowMiddleware.hears,
4                   (bot,message) => {
5    // do something
6  });
```

Listing 5.20: *hears* Funktion mit NLU Integration

Die *Middleware* Integration erledigt nun die folgenden Aufgaben. Jede Benutzereingabe wird zuerst von *Botkit* an den *Dialogflow* Dienst übertragen und dort verarbeitet. Als Rückgabe wird das *Botkit* Objekt *message* mit zusätzlichen Informationen von *Dialogflow* überschrieben. Das *message* Objekt beinhaltet nun z.B. *message.intent*, *message.entities* und *message.fulfillment* als zusätzliche Informationen. (Weitere Informationen siehe [41])

Die Erweiterung mit *Dialogflow* ermöglicht es den Intent über eine Wartung, als Benutzer zu äußern und der aktuelle Stand der Implementierung ist in Abbildung 5.6 dargestellt.

Alle Intents der Benutzer werden in einer *skills* Datei festgehalten und mit den entsprechenden *hears* Funktionen deklariert. Im nächsten Schritt werden die eigentlichen Funktionalitäten hinter den möglichen Intents deklariert und im nachfolgenden Kapitel 5.3.2 erläutert.

5.3.2 Intents ausführen

Für die eigentliche Logik hinter dem jeweiligen Intent, werden in der Chatbot Anwendung eigene *Botkit Events* definiert. Am Beispiel für das Condition Monitoring existiert der Intent *Machine-Status*, welche das interne *Botkit Event machine-status* auslöst und die aufbereitete Benutzereingabe, als Parameter übergibt.

Innerhalb der *skills* gibt es einen Listener auf das Event *machine-status* und dieser führt die interne Logik aus. Die interne Logik ist im folgenden Sequenzdiagramm aus Abbildung 5.7 vereinfacht abgebildet. Der Chatbot:machine-status *skill* überprüft, ob in

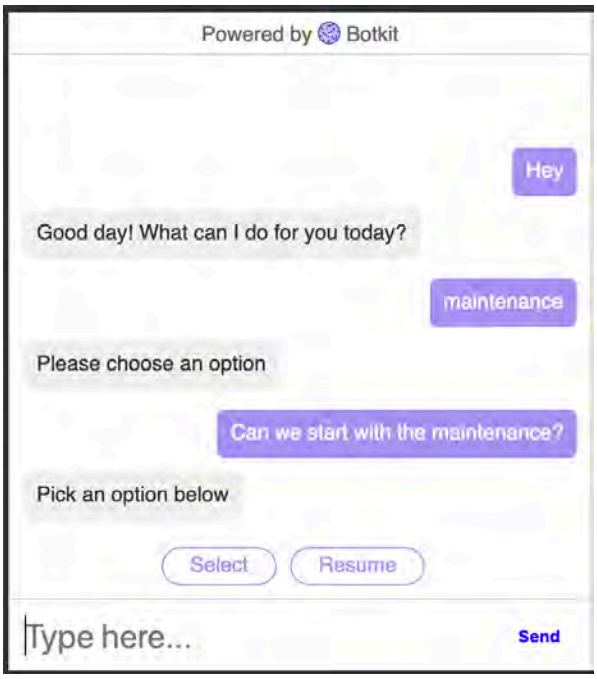


Abbildung 5.6: Chatfenster mit Intent Matching

der Anfrage für den Status der Maschine bereits eine Station angegeben worden ist oder nicht. Ist eine Station angegeben, wird ein Konversationsobjekt mit den aktuellen Informationen der gewählten Station erstellt. Wenn der Benutzer keine Station explizit angibt, erstellt der Chatbot ein Konversationsobjekt mit den verfügbaren Stationen. In beiden Fällen ist der Chatbot nach der Rückgabe des Konversationsobjektes wieder bereit für eine neue kontextunabhängige Eingabe

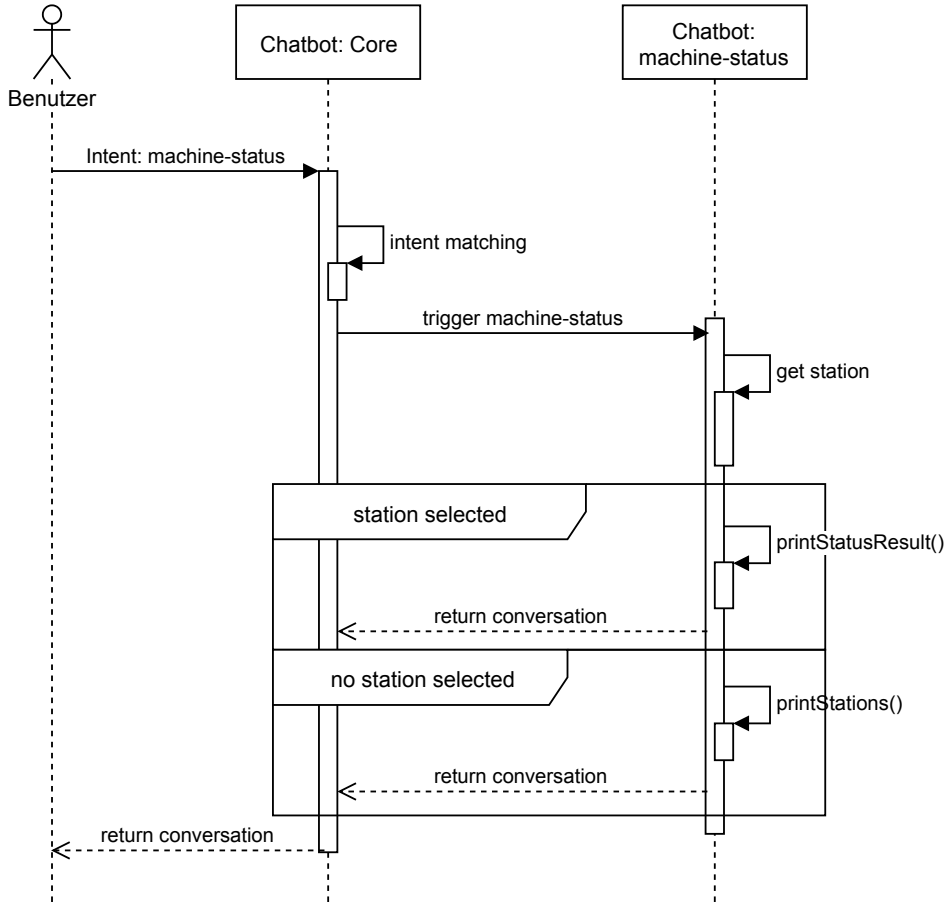


Abbildung 5.7: Sequenzdiagramm: Interner Ablauf für den Status einer Station

Der Ablauf für den Intent, die nächste Maschinenwartung abzufragen, verläuft fast identisch zu dem vorherigen Intent. Im ersten Schritt wird wieder ein eigenes Event ausgelöst *machine-maintenance* und von einem Listener verarbeitet.

Der gesamte Ablauf ist im Sequenzdiagramm 5.8 abgebildet und die Schritte für das Intent-Matching wurden aufgrund der Komplexität weggelassen. Der Chatbot kann drei Fälle unterscheiden anhand der Informationen aus der Benutzereingabe. Wenn der Benutzer keine genauen Angaben macht werden einfach alle Stationen ausgegeben, die eine Wartung benötigten. Im zweiten Fall gibt der Benutzer einen Stationsnamen an und der Chatbot überprüft, ob eine Wartung für die gewählte Station notwendig ist. Der letzte Fall ist die spezifische Frage, wann für eine gewählte Station die nächste Wartung ansteht oder die letzte Wartung stattgefunden hat. Ist der Chatbot nicht in der Lage, die Eingabe einen der drei Fällen zuzuordnen, wird der Benutzer darüber in Kenntnis gesetzt.

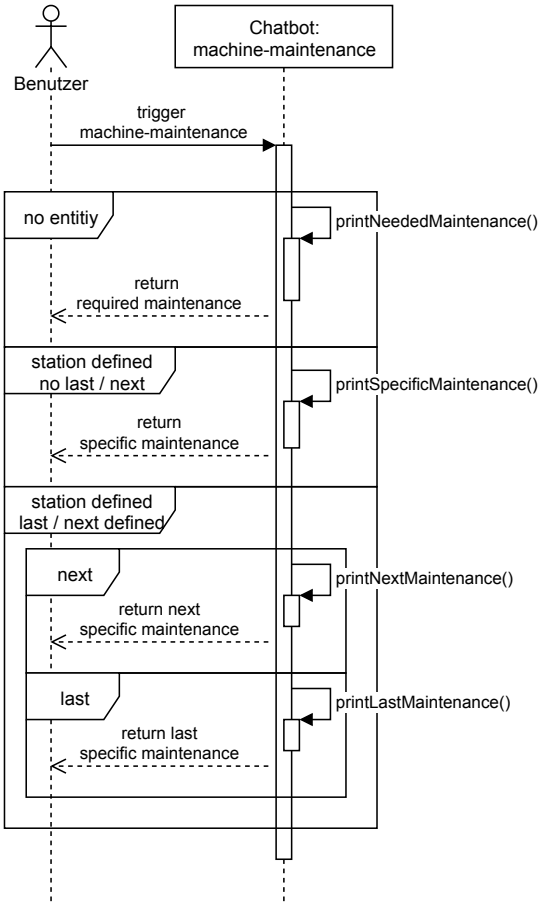


Abbildung 5.8: Sequenzdiagramm: Interner Ablauf für den Status von Wartungen

In beiden Events war die Konversation mit dem Chatbot aber sofort abgeschlossen. Der Chatbot musste keine Rückfragen an den Benutzer stellen und eine längere Unterhaltung durchführen. Für die Wartung ist dies aber notwendig, denn der Chatbot leitet den Benutzer anhand der Vorlage durch die Wartung. Zusätzlich sind die Wartungsprozesse im Vorfeld nicht spezifiziert und müssen dynamisch vom Chatbot erstellt werden.

5.3.3 Dynamische Konversationen erstellen

Für das Ausführen von einem Wartungsprozess muss der Chatbot dynamisch ein Konversationsobjekt erstellen. Das Konversationsobjekt setzt sich dabei aus den Aufgaben, Gateways, Dokumentation und sonstigen Nachrichten des Wartungsprozesses zusammen. Die Konversationsfunktionalität von *Botkit* (vgl. Kapitel 5.1.4) wird für die Umsetzung verwendet. Konversationen in *Botkit* ermöglichen es, mehrere Fragen hinzuzufügen und mit *thread_ids* zwischen diesen und anderen Objekten zu navigieren.

Betrachtet wird das Beispiel aus Abbildung 4.1. Der Chatbot hat ein Event *create-maintenance-conversation*, dass die gesamte Logik zum Erstellen und Ausführen der Konversation beinhaltet. Als erstes wird vom Server (vgl. Kapitel 5.2) der ausgewählte Prozess geladen. Anschließend wird das Konversationsobjekt erstellt.

In einem ersten Schritt werden dem Konversationsobjekt zwei Nachrichten hinzugefügt, eine für den Abbruch und das erfolgreiche Absolvieren der Wartung. Dann wird die Liste mit den Prozessen gelesen und für jedes Objekt innerhalb der *tasks* Liste wird eine Frage dem Konversationsobjekt hinzugefügt. Die Fragen werden dabei nach dem folgenden Prinzip erstellt:

Frage

Beinhaltet den Text der Aufgabe und ein *Quick_replies* Objekt mit den Optionen: *Next*, *Previous*, *Abort* und *Documentation* (vgl. Kapitel 4.1.2).

Pattern

Das Pattern für die Benutzereingaben während der Konversation. Jedes Pattern besteht aus den Intents für die erstellten *Quick_replies* und wenn der Benutzer z.B. zu der nächsten Aufgabe wechseln möchte, sucht die *callback* Funktion des

Patterns die nächste Aufgabe und wechselt den *thread*. Ist der Chatbot nicht in der Lage, die Benutzereingabe einem Pattern zuzuordnen, wird die ursprüngliche Aufgabe wiederholt.

Thread_ID

Alle erstellten Fragen aus dem *task* Objekt werden mit einer entsprechenden *thread_id* gekennzeichnet, um später die Navigation zwischen ihnen zu ermöglichen.

Dieselbe Funktionalität wird auch für das Erstellen der Gateways verwendet. Bei den Gateways gibt es noch den Unterschied, dass der Benutzer innerhalb des Prozesses nicht mehr zurückgehen kann. Auch Gateways ohne Bezeichnungen werden ausgegeben und der Benutzer muss diese überspringen mit dem Intent *Next*. Dieses Problem kommt von der Umsetzung des Konversationsflusses und der Chatbot kann nicht mehrere Sequenzen überspringen.

Die Dokumentation der einzelnen Aufgaben sind mehrere unabhängige Nachrichten Objekte innerhalb des Konversationsobjekts und wechseln automatisch nach dem Anzeigen direkt wieder zur ursprünglichen Aufgabe zurück und geben diese erneut aus. Sind alle Aufgaben, Gateways und Nachrichten für die Dokumentationen erstellt, sucht der Chatbot die *thread_id* der ersten Aufgabe und wechselt in diesen *thread*.

Ist ein Wartungsprozess mit mehreren internen Prozessen ausgewählt, erkennt der Chatbot diesen anhand der Größe der Liste mit Prozessen und stellt dem Benutzer die Frage, welchen Prozess er starten möchte.

Unterbricht der Benutzer eine aktuelle Wartung und möchte z.B. eine andere Funktionalität benutzen, muss zuerst die Wartung über den Intent *abort* abgebrochen werden. Der Chatbot kann aufgrund seiner Implementierung innerhalb einer Wartung nicht das Event wechseln (vgl. Kapitel 4.1.1).

Der komplette Ablauf für das Erstellen von einem Wartungsprozess als Konversation ist im Sequenzdiagramm in Abbildung 5.9 vereinfacht dargestellt.

Für die Auswahl des gewünschten Wartungsprozesses besitzt der Chatbot das Event *maintenance-select*. Die Eingabe vom Benutzer wird von *Dialogflow* analysiert und ist

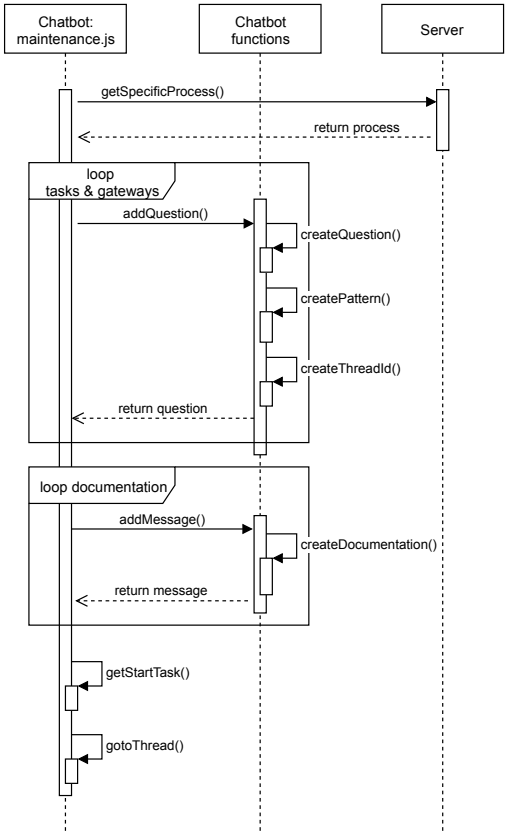


Abbildung 5.9: Sequenzdiagramm: Erstellen von einer Konversation

bereits eine Station in der Eingabe enthalten, wird automatisch der Wartungsprozess der ausgewählten Station gestartet. Befindet sich keine Angabe von einer Station in der Benutzereingabe, erstellt der Chatbot automatisch eine Liste mit allen verfügbaren Wartungsprozessen und bietet diese als *Quick_replies* (vgl. Abbildung 5.10) für den Benutzer an.

Am Anfang vom Unterkapitel 5.3 wurde der Dienst *Dialogflow* eingeführt und in der Implementierung für den Chatbot als NLU Komponente eingesetzt. Im letzten Abschnitt wird der Dienst *Dialogflow* noch einmal genauer vorgestellt zusammen mit seiner grafischen Oberfläche und die Definitionen der unterschiedlichen Intents.

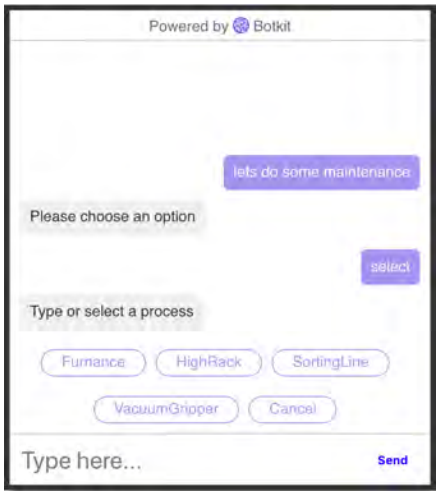


Abbildung 5.10: Chatfenster: Auswählen von dem gewünschten Wartungsprozesses

5.4 Integration von Dialogflow

Die Konfiguration von Dialogflow passiert über die Dialogflow-Konsole, welche als Weboberfläche verfügbar ist. Für die Umsetzung von einem Projekt zur sprachlichen Aufbereitung von Benutzereingaben muss ein sogenannter *Agent* erstellt werden. *Agents* sind Natural Language Understanding Module für die Übersetzung von Eingaben (Text oder gesprochen) in ein aufbereitetes Datenformat zur Weiterverarbeitung in Apps oder Webseiten [42]. Nach der Erstellung eines *Agents* kann dieser Konfiguriert werden. Als Erstes werden die *Intents* deklariert auf die der *Agent* reagieren soll. Dialogflow bietet bereits fertige Module für unterschiedliche *Intents* an. Die *dbisfactorybot* Komponente

greift aber nicht auf fertige Module zurück, sondern benutzt stattdessen eigene definierte *Intents*.

Das Erstellen eines *Intent* läuft dabei nach den folgenden Schritten ab. Zuerst wird dem *Intent* eine Bezeichnung gegeben (vgl. Abbildung 5.11 - 1). Diese ist wichtig für die Zuordnung und wird in der *Botkit* Implementierung in der *hears* Funktion benötigt (vgl. Kapitel 5.3.1). Für das Zuordnen von Benutzereingaben zu einem *Intent* muss der *Agent* mit Beispieleingaben trainiert werden (vgl. Abbildung 5.11 - 2). Anschließend können mögliche Antworten für das Eintreffen des *Intents* definiert werden. Um zu testen, ob der *Agent* auf die Eingaben den richtigen *Intent* zuordnet und auch die Antwortmöglichkeiten genutzt werden, bietet die Dialogflow-Konsole direkt ein Chatfenster an (vgl. Abbildung 5.11 - 3). Dies sind die Grundlagen für die Definition von einem *Intent* in Dialogflow. Zusätzliche erweiterte Möglichkeiten, *Intents* auf eine höhere Komplexität zu heben, sind in der Dialogflow Dokumentation vorgestellt (siehe Literatur [42]).

Die Chatbot Anwendung (vgl. Kapitel 5.3) arbeitet mit extrahierten Informationen einer Benutzereingabe. Dialogflow kann dafür Entities definieren. Ein *Entity* ist aus einem Typ und einem Eintrag aufgebaut. Der *Entity Typ* gibt an, um was für eine Art von *Entity* es sich handelt. *Entity Einträge* sind die eigentlichen *Entities* und können sich aus einem Wort oder einer Phrase zusammensetzen [42]. Für die Chatbot Anwendung gibt es einen *Entity Typ Stations* mit den Abkürzungen der Stationen als Einträge. Die Zuordnung zu einem Eintrag kann aus einer Benutzereingabe durch das Definieren von *Synonymen* zu jedem Eintrag vereinfacht werden. Das Extrahieren von *Entities* kann dann über eine Erweiterung der *Trainings Phase* in einem *Intent* passieren. Die Trainings Daten enthalten die gewünschten *Entities* und Dialogflow erkennt diese automatisch und gibt die extrahierten Informationen in dem Feld *Actions and parameters* an (vgl. Abbildung 5.12). Die *Entity Stations* ermöglicht es dem Chatbot direkt die gewünschte Station aus einer Eingabe zu erhalten.

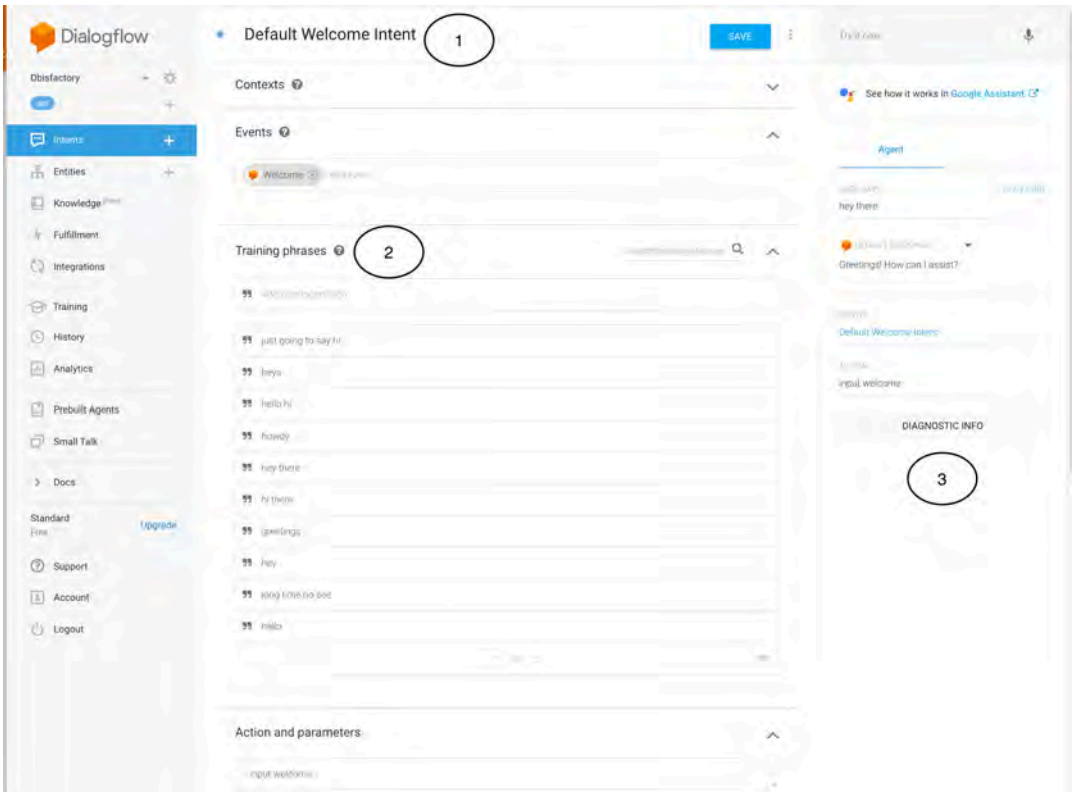


Abbildung 5.11: Übersicht der Dialogflow Konsole

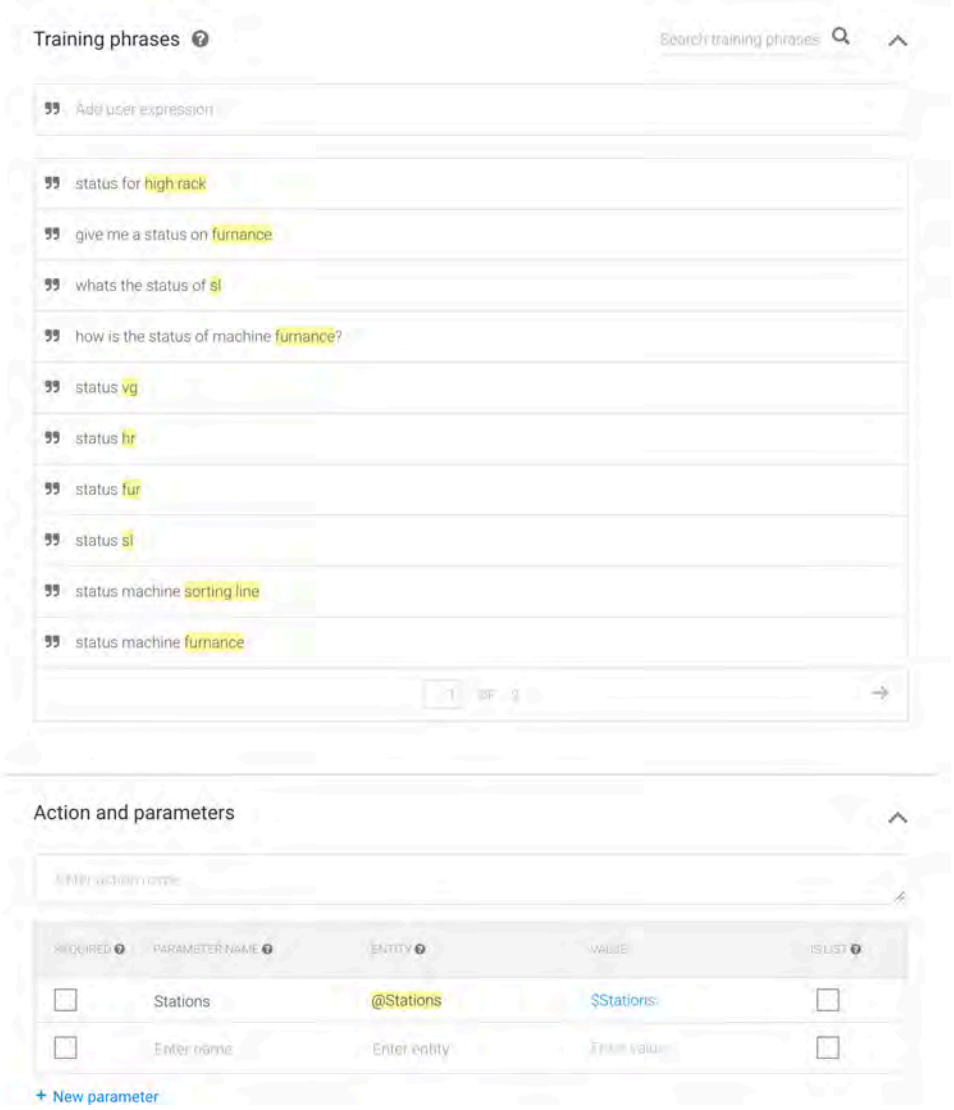


Abbildung 5.12: Extrahieren von zusätzlichen Informationen mittels Entities in Dialogflow

6

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, den Prototypen eines Chatbots für die Unterstützung einer Maschinenwartung zu entwickeln. Die Problemstellung beinhaltete dabei die Frage nach der technischen Umsetzung für das verstehen von natürlichen Eingaben. Als Vorlage für die Maschinenwartung wurden Geschäftsprozesse am Beispiel einer Fabrik Simulation definiert. Die Anwendung basiert dabei auf einem ausgewählten Chatbot Framework zusammen mit einem externen Natural Language Understanding Dienst.

Die Arbeit begann mit dem Grundlagen Kapitel zur Einführung in die Thematik Chatbots, Business Process Model Notation, Factory, Natural Language Understanding und es wurde ein Überblick an Technologien für die Chatbot Entwicklung gegeben. Die Themen Chatbots, Geschäftsprozesse und das Vorstellen der Factory wurden dabei generell betrachtet. Detaillierter wurden die Grundlagen für das Verstehen von natürlicher Sprache und die Vorstellung verschiedener Technologien betrachtet. Die Basis für das später gewählte Framework Botkit war der indirekte Technologien Vergleich. Anschließend folgte eine Anforderungsanalyse für die Konzeption und die unterschiedlichen Anwendungsfälle der Chatbot Anwendung wurden vorgestellt. In Anlehnung an die Anwendungsfälle konnten funktionale und nicht-funktionale Anforderungen definiert werden, welche im späteren Kapitel der Implementierung umgesetzt wurden und für die Konzeption relevant waren. Das Kapitel Konzeption beschäftigte sich mit einem allgemeinen Konzept für die Umsetzung eines Chatbots, ohne sich dabei aufgrund der späteren Implementierung einzuschränken. Der Fokus lag vor allem am Design des Konversationsflusses und einer generellen Architektur, welche Änderungen ermöglicht, um später den Prototypen schnell auf einen anderen Anwendungsfall übertragen zu können. Bei der Implementierung wurden zuerst die eingesetzten Technologien vorgestellt, welche es ermöglichten,

den Chatbot Prototypen mit geringem Aufwand umzusetzen. Darauffolgend wurden die beiden Komponenten Server und Chatbot vorgestellt. Bei der Server Komponente lag der Fokus auf der Bereitstellung der Daten. In der Chatbot Komponente wurde die Einbindung von natürlicher Sprache thematisiert und wie man Geschäftsprozesse als Vorlage für eine Konversation benutzen kann. Es konnten im Rahmen dieser Arbeit zwar nicht alle gewünschten Funktionen umgesetzt werden, dennoch bietet der Chatbot Prototyp eine gute Basis für spätere Weiterentwicklungen und die Konzepte lassen sich leicht auf andere Anwendungsfälle übertragen.

Das Wachstum an digitalen Assistenten wird in den nächsten Jahren weiter ansteigen und so können immer mehr Anwendungsgebiete für Chatbots ermöglicht werden. Grund für diesen Anstieg an digitalen Assistenten ist der technische Fortschritt auf dem Gebiet der natürlichen Spracherkennung. Dadurch können auch weniger erfahrene Entwickler auf Technologien für Natural Language Understanding zurückgreifen und ihre Anwendungen dahingehend erweitern. Das entwickelte Konzept und Implementierung erfüllen zwar noch nicht alle gewünschten Anforderungen an einen vollständigen Chatbot, aber die vorgestellte Implementation und Architektur erlauben, eine einfache Erweiterung der Funktionalitäten. Darüber hinaus ist der Chatbot auch in der Lage Geschäftsprozesse, welche keine Wartungen darstellen, als Vorlage zu benutzen, da der implementierte Parser an die BPMN Notation gebunden ist. Denkbare Erweiterungen sind z.B. ein Small Talk Modul für den Chatbot, um noch natürlichere Unterhaltungen mit Benutzern führen zu können oder auch das Speichern der aktuellen Konversationen und später die Unterhaltung an derselben Stelle fortzusetzen.

Literaturverzeichnis

[1] Dale, R.: The Return of the Chatbots. Natural Language Engineering **22** (2016) 811–817

[2] Kothari, A., Hoover, J., Zyane, R.: Chatbots for eCommerce: Learn How to Build a Virtual Shopping Assistant. Bleeding Edge Press (2017)

[3] Stucki, T., D'Onofrio, S., Portmann, E.: Chatbot – Der digitale Helfer im Unternehmen: Praxisbeispiele der Schweizerischen Post. HMD Praxis der Wirtschaftsinformatik **55** (2018) 725–747

[4] Henrich, O.: Chatbots auf dem Vormarsch: Der künstlich-intelligente Buchhalter kommt. Wirtschaftsinformatik & Management **9** (2017) 72–75

[5] Göster, M.: Concept and Implementation of a Factory Simulation. (2017)

[6] Jabok Freund, Bernd Rücker, T.H.: Praxishandbuch BPMN. Carl Hanser Verlag (2010)

[7] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer (2012)

[8] Messenger: Messenger Website. `https://www.messenger.com/` (2019) [Online; Zugriff 06-januar-2019].

[9] Weber, S.: Roboterjournalismus, Chatbots & Co.: Wie Algorithmen Inhalte produzieren und unser Denken beeinflussen. Telepolis. Heise Verlag (2018)

[10] Batish, R.: Voicebot and Chatbot Design. Packt Publishing (2018)

[11] Intelligence, D.A.: Chatbots Point of View. `https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/deloitte-analytics/deloitte-nl-chatbots-moving-beyond-the-hype.pdf` (2018)

[12] Apple: Apple Siri Website. `https://www.apple.com/siri/` (2019) [Online; Zugriff: 20-januar-2019].

[13] Amazon: Amazon Lex Website. https://aws.amazon.com/de/lex/ (2019) [Online; Zugriff: 20-januar-2019].

[14] Amazon: Amazon AWS Website. https://aws.amazon.com/de/ (2019) [Online; Zugriff: 20-januar-2019].

[15] Google: Google Assistant Website. https://assistant.google.com/intl/de_de/ (2019) [Online; Zugriff 20-januar-2019].

[16] Dialogflow: Dialogflow Website. https://dialogflow.com/ (2018) [Online; Zugriff 28-dezember-2018].

[17] Dialogflow: Dialogflow: How Agents work Abbildung. https://dialogflow.com/docs/intro/agents (2019) [Online; Zugriff: 20-januar-2019].

[18] Microsoft: Azure Bot Service. https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-3.0 (2019) [Online; Zugriff: 20-januar-2019].

[19] Microsoft: Microsoft Bot Framework. https://dev.botframework.com/ (2019) [Online; Zugriff: 20-januar-2019].

[20] Microsoft: Language Understanding (LUIS). https://azure.microsoft.com/en-us/services/cognitive-services/language-understanding-intelligent-service/ (2019) [Online; Zugriff: 20-januar-2019].

[21] Bocklisch, T., Faulkner, J., Pawlowski, N., Nichol, A.: Rasa: Open Source Language Understanding and Dialogue Management. CoRR **abs/1712.05181** (2017)

[22] Nichol, A.: A new approach to Conversational Software. https://medium.com/rasa-blog/a-new-approach-to-conversational-software-2e64a5d05f2a (2017) [Online; Zugriff 31-januar-2019].

[23] Chatfuel: Chatfuel Website. https://chatfuel.com/ (2019) [Online; Zugriff: 20-januar-2019].

[24] Hall, P., Venigalla, V., Janarthanam, S.: Hands-On Chatbots and Conversational UI Development: Build chatbots and voice user interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills. Packt Publishing (2017)

[25] Botkit: Botkit Website. `https://botkit.ai/` (2019) [Online; Zugriff 04-januar-2019].

[26] NodeJS: NodeJS Website. `https://nodejs.org/en/` (2019) [Online; Zugriff 08-januar-2019].

[27] Botkit: Botkit Middleware Documentation. `https://botkit.ai/docs/middleware.html` (2019) [Online; Zugriff 08-januar-2019].

[28] Rao, B.: Handbook of Condition Monitoring. Elsevier Advanced Technology (1996)

[29] Kleuker, S.: Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten. Springer Fachmedien Wiesbaden (2018)

[30] Lee, H.: Voice User Interface Projects: Build voice-enabled applications using Dialogflow for Google Home and Alexa Skills Kit for Amazon Echo. Packt Publishing (2018)

[31] Ciaburro, G., Ayyadevara, K., Perrier, A.: Hands-On Machine Learning on Google Cloud Platform: Implementing smart and efficient analytics using Cloud ML Engine. Packt Publishing (2018)

[32] Microsoft: Entwerfen und Steuern des Konversationsflusses. `https://docs.microsoft.com/de-de/azure/bot-service/bot-service-design-conversation-flow?view=azure-bot-service-4.0` (2019) [Online; Zugriff 27-januar-2019].

[33] T. Bray, E.: The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, RFC Editor (2017)

[34] NodeJS: NodeJS About. `https://nodejs.org/en/about/` (2019) [Online; Zugriff 31-januar-2019].

[35] Prediger, R., Winzinger, R.: Node.js: Professionell hochperformante Software entwickeln. Carl Hanser Verlag GmbH & Company KG (2015)

[36] Mardan, A.: Express.js Guide: The Comprehensive Book on Express.js. Createspace (2014)

[37] Express: Hello world example. https://expressjs.com/en/starter/hello-world.html (2019) [Online; Zugriff: 02-februar-2019].

[38] Express: Basic Routing. https://expressjs.com/en/starter/basic-routing.html (2019) [Online; Zugriff: 02-februar-2019].

[39] Botkit: Botkit Core Documentation. https://github.com/howdyai/botkit-docs/blob/master/docs/core.md (2019) [Online; Zugriff 08-januar-2019].

[40] Google: Service accounts . https://cloud.google.com/compute/docs/access/service-accounts (2019) [Online; Zugriff: 06-februar-2019].

[41] Schnurr, J.: Botkit-middleware-dialgflow. https://github.com/jschnurr/botkit-middleware-dialogflow (2019) [Online; Zugriff: 06-februar-2019].

[42] Dialogflow: Dialogflow Documentation. https://dialogflow.com/docs (2019) [Online; Zugriff: 08-januar-2019].

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Jens Reiner Matrikelnummer: 908294

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Jens Reiner