Mensch-Maschine-Kommunikation basierend auf Natural Language Processing: Evaluierung von Möglichkeiten für den Einsatz eines humanoiden Roboters im Kontext einer Hochschulbibliothek

Masterarbeit

Zur Erlangung des akademischen Grades

Master of Engineering („M.Eng.“)

Technische Hochschule Wildau

Fachbereich Ingenieur- und Naturwissenschaften

Studiengang Telematik

Eingereicht am: 16. April 2018

Eingereicht von: Philipp Müller <pmueller@th-wildau.de>

Geboren am: 13. Dezember 1992

Erstbetreuer/in: Prof. Dr. rer. Nat. Janett Mohnke <jmohnke@th-wildau.de>

Zweitbetreuer/in: Dr. Frank Seeliger <fseeliger@th-wildau.de>

Themenstellender Betrieb: Technische Hochschule Wildau

Bibliographische Beschreibung

Autor: Philipp Müller

Thema: Mensch-Maschine-Kommunikation basierend auf Natural Language Processing: Evaluierung von Möglichkeiten für den Einsatz eines humanoiden Roboters im Kontext einer Hochschulbibliothek

Metadaten: Masterabschlussarbeit 2017, Version 3, Technische Hochschule Wildau, 172 Seiten, 74 Abbildungen, 16 Tabellen, 4 Anhänge

Ziel: Zielstellung der Arbeit ist die intelligente Sprachkommunikation zwischen einem humanoiden Roboter und einem Menschen. Der Roboter nimmt dabei die Rolle als Experte für kontextbezogene Themen ein und versucht bei der Beantwortung von Fragen bzw. Problemen zu helfen. Als Szenario dient die Verwendung des Roboters in einer Hochschulbibliothek, in welcher Studierende und Interessierte bibliotheksspezifische Fragen formulieren und sich im weiteren Verlauf der Kommunikation ein natürlich sprachlicher Dialog zwischen Mensch und Roboter entwickelt.

Inhalt: Thematisiert wird der Weg, von einer theoretischen Betrachtung computerlinguistischer Grundlagen der Sprachverarbeitung, bis hin zu einer prototypischen Umsetzung einer Software zur natürlich sprachlichen Dialogkommunikation zwischen Mensch und Roboter.

Die Arbeit widmet sich dabei zunächst der Bereitstellung theoretischer Grundlagen (Definition, Geschichte, Funktionsweise, Herausforderungen, Aktuelle Lage) von Mensch-Maschine-Kommunikation unter Zuhilfenahme des Natural Language Processings. Darauf basierend werden Anforderungen aufgestellt, die eine softwaremäßige Implementierung eines solchen Systems beschreiben. Dies geschieht unter der Beachtung von Bedingungen durch den themenstellenden Betrieb dieser Arbeit.

Es folgt eine ausgiebige Evaluierung verschiedener NLP-Systeme anhand des Abgleichs von Anforderungen und zusätzlicher Kennziffer-Messungen. In diesem Zusammenhang entsteht auch ein Testprogramm, welches ein automatisiertes Evaluierungsverfahren bereitstellt. Die beiden besten NLP-Systeme werden anschließend in ihrer Verwendung detailliert beschrieben. Eine prototypische Implementierung stellt die Verwendung eines NLP-Systems, zur Kommunikation von Mensch und Roboter, in einem konkreten Anwendungsfall unter Beweis.

Dem Anhang dieser Abschlussarbeit ist eine CD-ROM beigefügt, in welcher u.a. sämtlicher entstandener Programmcode sowie die Ergebnisdateien der Evaluierung enthalten sind.

Stichwörter: Mensch Maschine, Kommunikation, Interaktion, Natural Language Processing, NLP, Natural Language Understanding, NLU, Natural Language Generation, NLG, Computerlinguistik, Robotik, Robotics, Pepper, NAO, SoftBank, IBM, Watson Conversation, Watson Assistent, Rasa, Rasa Technologies GmbH, Framework, Library, Software

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Abschlussarbeit selbstständig angefertigt und nur die angegebenen Hilfsmittel und Quellen verwendet habe.

Wildau, den 16. April 2018

. . . . . .

Philipp Müller

Hinweise zum Lesen dieser Arbeit

Abkürzungen werden zunächst einmalig ausgeschrieben. Dahinter folgt in Klammern die dazugehörige Bedeutung. Im weiteren Verlauf der Arbeit wird anschließend lediglich die genannte Abkürzung weiterverwendet.

Zusätzlich sind in dieser Arbeit verwendete Abkürzungen im Abkürzungsverzeichnis, Abbildungen im Abbildungsverzeichnis und verwendete Quellen im Quellenverzeichnis aufgeführt. Die Quellenangaben im Text, bei Abbildungen, Tabellen und Auflistungen folgen dem ISO 690 Standard, um den Textfluss zu erhalten, und sieht im Text wie in folgendem Beispiel aus: [1].

Sämtliche Online-Links in den Quellen oder im Text haben das letzte Abrufdatum 11.04.2018. Die Angabe dieses letzten Abrufdatums erfolgt nur hier und wird nicht pro Online-Link erneut gesetzt.

Um auf Eigennamen, Fachtermini, Produktbezeichnungen, Fremdwörter sowie Schlüsselbegriffe hinzuweisen, werden solche *kursiv* geschrieben. Einen Sonderfall stellt Programmcode dar. Dieser wird durch einen separaten `eigenen Schriftstil` gekennzeichnet.

Nachtrag vom 13.06.2018: In dieser Arbeit wird vom Dienst IBM Watson Conversation gesprochen. Während des Bearbeitungszeitraums entschied sich IBM zur Umbenennung des Dienstes in IBM Watson Assistant[1]. Die hier vorliegende Arbeit verwendet dennoch den bis dato bekannten Namen. Die gewonnenen Erkenntnisse gelten für beide Bezeichnungen, da der zugrundeliegende Dienst identisch ist.

[1] Siehe Release Notes „15 March 2018“: https://console.bluemix.net/docs/services/conversation/release-notes.html#release-notes

Inhaltsverzeichnis

1 Einleitung

Nichts auf der Welt ist so kraftvoll

wie eine Idee, deren Zeit gekommen ist.

Victor Hugo[2]

Es gibt viele computergestützte Systeme, Produkte und Prototypen, die Verbesserungen und Fortschritt in der Lebens- und Arbeitssituation vieler Menschen gebracht haben. Man denke nur an die Möglichkeiten von Smart Home bzw. Building, Mobile Apps, Smartphones, Web-Dienste, Big Data, Künstliche Intelligenz und Robotik, die allesamt ihren Anteil zur digitalisierten Gesellschaft beitragen. Jedoch stehen die meisten dieser Techniken momentan für sich alleine. Es fehlt die Zusammenführung zu etwas Ganzheitlichen. Etwas, bei dem die digitalen Dienste und Techniken nur Rädchen eines großen Getriebes sind und die Kooperation aller Bestandteile den Antrieb ermöglicht.

In Wildau, einer brandenburgischen Stadt im südöstlichen Speckgürtel Berlins, hat die dort ortsansässige Technische Hochschule Wildau (kurz: TH Wildau) eine Vision. Gemeinsam mit der hochschuleigenen Bibliothek, wird der Versuch unternommen, die Innovationskraft neuester Digitaltechniken in praktischer Art und Weise zu kombinieren und in den studentischen Alltag zu integrieren.

Die Einbindung einer Bibliothek ist dabei besonders spannend. Denn sicherlich erwarten Menschen zwar Informationen über Ansätze, Techniken und Erklärungen bzgl. der Digitalisierung in Büchern zu finden, dass die Bibliothek aber selbst zum Ort der Umsetzung wird, könnte durchaus einige Menschen überraschen. Schließlich erwarten die meisten Menschen in einer Bibliothek eher dunkle Räumlichkeiten, lange Regalreihen, verstaubte Bücher, langwierige Suchvorgänge und diesen unverkennbaren „Geruch der Geschichte“.

Diese Sichtweise ist veraltet und die Bibliothek der TH Wildau ein Paradebeispiel für die gegenteilige Realität, wie sie in vielen Bibliotheken heute gelebt wird. Das helle,

[2] Der Franzose *Victor-Marie Hugo* (1802 – 1885) war ein berühmter Schriftsteller, der sowohl literarische als auch politische Publikationen verfasste. Zu seinen bekanntesten Werken gehören *Notre Dame de Paris* (Der Glöckner von Notre-Dame) und *Les Misérables* (Die Elenden). Das vorgestellte Zitat wurde über die Zeit eingedeutscht und lautet ursprünglich *„On résiste à l'invasion des armées; on ne résiste pas à l'invasion des idées“* (Man kann der Invasion von Armeen Widerstand leisten, aber keiner Invasion von Ideen). [118] [117]

offene und freundliche Gebäude zeigt auf, wie Tradition und Moderne miteinander sinnvoll verbunden werden können.

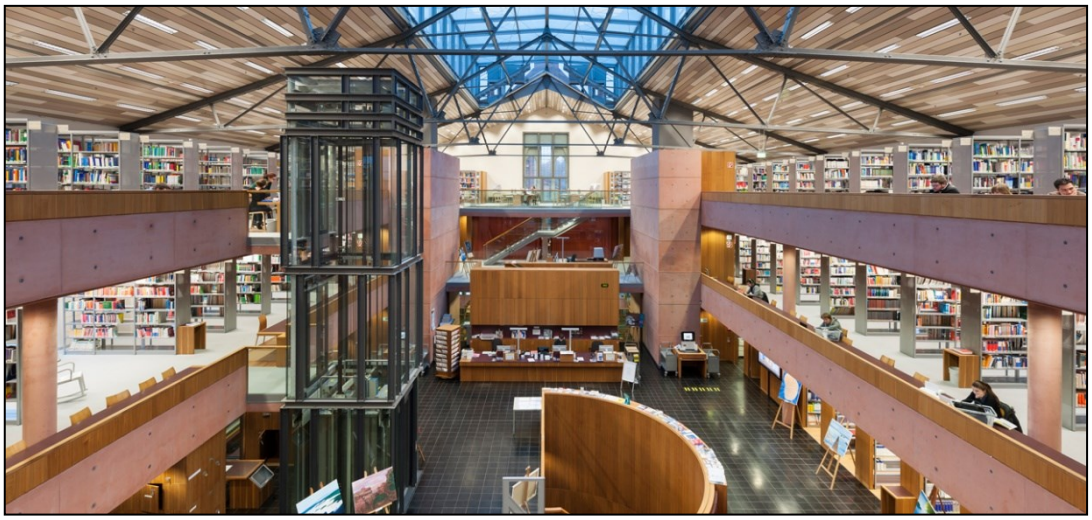


Abbildung 1: Blick auf den Bibliotheks-Eingangsbereich der TH Wildau. [Quelle: [3]]

In den über 3.000 Regalmetern der Bibliothek, stehen insgesamt über 100.000 gedruckte Medien wie Bücher, Hefte, Zeitschriften und Magazine zur fachspezifischen Kompetenzaneignung zur Verfügung. Neben der Ausleihe und Mitnahme dieser Medien, besteht auch die Möglichkeit an einem der ca. 200 Arbeitsplätze, die entweder als Sitzflächen im Hauptraum der Bibliothek oder auch als separate Ruhe- und Gruppenarbeitsräume zur Verfügung stehen, die eigenen Recherchen direkt im Bibliotheksumfeld zu erledigen. [1] [2]

Doch was ist nun die Idee, welche der TH Wildau vorschwebt und die einen verbindenden Bogen über die neuen Innovationen schlagen soll?

Die Digitalisierung in allen Bereichen der Gesellschaft bringt auch für Bibliotheken die Herausforderung einer zeitgemäßen Bereitstellung und Präsentation von Angeboten wie Medien, Dienstleistungen und Events. Die Technische Hochschule Wildau möchte dazu den Versuch einer Hybridbibliothek wagen, mit dem der aktuelle Medienbruch zwischen analoger und digitaler Welt versöhnt wird. Dazu heißt es:

> *„so soll vor allem deutlich werden, das es Ziel ist, den Brückenschlag beim Medienbruch oder der Hybridbibliothek zu wagen, die klassischen mit den neuen Aufgaben einer Informationseinrichtung zu verbinden, sowohl hinsichtlich der Dienstleistungen, als auch hinsichtlich der Digitalisierung und Automatisierung von Angeboten, Prozessen bis hin zur personellen Befähigung, gerade die digitalen Services eigenständig und mündig auszubauen.“ [3]*

Die Bibliothek hat dazu sechs Grundsäulen[3] definiert: Indoor-Ortung, humanoide Roboter, Digitales Wissen, Open Access, Web-Dienste und Showroom. Dabei werden entsprechende Projekte in Teilaufgaben zergliedert und mit großem Anteil von den Studierenden realisiert. Durch die Fluktuation der Studierenden, ist die Umsetzung der großen Ziele dieses Projektes eine langfristige Aufgabe, welche nicht nur Jahre andauern wird, sondern bereits seit Jahren läuft.

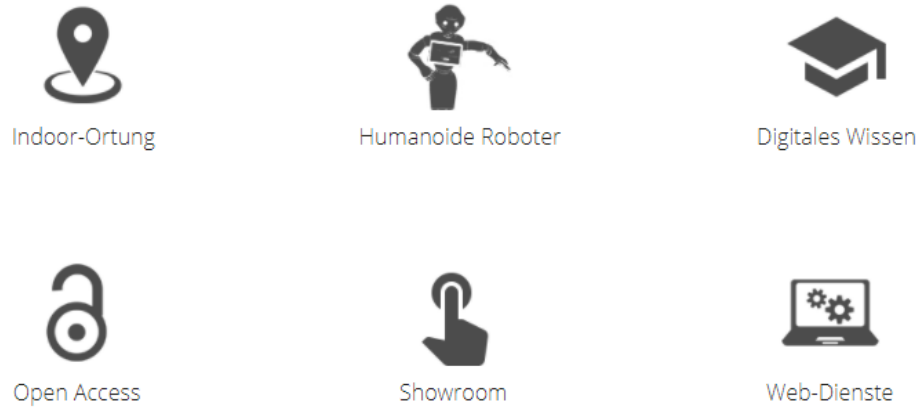


Abbildung 2: Grundkonzepte zur Digitalisierung der Hochschulbibliothek in Wildau. [Quelle: [3]]

So wurde das Angebot an klassischen Medien bereits durch digitalisierte Varianten wie E-Books, E-Journals und E-Magazines ergänzt. Durch Web-Dienste kann heute die Suche nach Medien bequem vom Computer aus erledigt und der Standort des betreffenden Mediums bis auf die Regalposition genau angezeigt werden. Bei Bedarf ist auch eine Reservierung von Arbeitsräumlichkeiten bequem Online möglich. Durch die Einführung von Unidos[4] Wildau, einer App für iOS und Android, kann die gesamte Angebotspalette auch auf mobilen Endgeräten wahrgenommen werden.

Die Menge an Projekten ist enorm und eine genauere Betrachtung empfehlenswert. Entsprechende Fußnoten dieser Einleitung sollen als Wegweiser dienen. Die Herangehensweise der Hochschule und Bibliothek führte im Jahr 2012 sogar zum Gewinn des Titels *„Bibliothek des Jahres“* [4]. Der Preis bestätigt die Sinnhaftigkeit der ursprünglichen Idee und ist sicherlich auch Fingerzeig und Motivator für die Fortsetzung der großen Vision.

Das Wintersemesters 2016 kann als Startpunkt für die nächste Evolutionsstufe der Hybridbibliothek angesehen werden. Mit Anschaffung des humanoiden Roboters „Pepper“ des japanischen Großunternehmens SoftBank Robotics Corp. bekommt nun auch die Robotik-Grundsäule des Konzeptes eine konkrete Form. Erklärtes Ziel ist es,

[3] Die sechs Grundsäulen im Detail: https://icampus.th-wildau.de/bewerbung-bdj-2018/index.html

[4] Unidos Wildau App: https://icampus.th-wildau.de/icampus/home/de/unidos-wildau

den Roboter als Schnittstelle zwischen Mensch und Bibliotheksangebot zu verwenden. Entsprechend soll sich der Roboter in der Bibliothek bewegen und Auskünfte über die angebotenen Dienstleistungen geben. Vereinfacht gesagt, dient Pepper als „sprechende Litfaßsäule auf Rädern“. Dieser Ansatz bietet der Bibliothek das Potential eines Alleinstellungsmerkmals und kann daher auch indirekt dem Ansatz des Showrooms zugeordnet werden.

Der Einsatz als Schnittstelle zwischen Mensch und Roboter dient dem Zweck eines Assistenzsystems im Kontext der Bibliothek. Somit wird eine künstliche Intelligenz geschaffen, die der Rolle eines Bibliothekars entspricht. Um dieses ambitionierte Ziel zu erreichen, muss quasi die gesamte Bibliothek digitalisiert werden, da der Roboter nur so die notwendigen Informationen zur konkreten Hilfestellung geben kann. Bibliothek und Roboter werden folglich kompatibel zueinander und erreichen dadurch neue Synergie-Effekte.

Der Status als künstliche Intelligenz ist an unglaublich viele Prozesse und Verarbeitungsschritte gebunden. Beispielsweise die Koordination im Raum während der Fortbewegung, die Steuerung des Fahrstuhls zur Erreichung der unterschiedlichen Stockwerke, die Auswertung von Daten angebrachter Temperatursensoren zur Regelung der Raumtemperatur oder auch die Suche nach Medien in Abhängigkeit zu Eingaben der Bibliotheksbesucher.

Es stellt sich in diesem Zusammenhang die Frage, wie Menschen mit dem Roboter kommunizieren sollen. Sicherlich wäre es ohne weiteres möglich einige Schaltflächen in die Unidos-App einzufügen, über die, beispielsweise eine Suche von Medien ausgelöst werden kann. Doch der Anspruch, den Roboter als Bibliothek-Assistenzsystem einzusetzen, führt zwangsläufig zum Wunsch die Kommunikation für den Menschen einfacher bzw. natürlicher zu gestalten. Die Roboter-Mensch-Kommunikation, allgemein auch als Mensch-Maschine-Kommunikation bekannt, soll schlichtweg „menschlich“ sein und dadurch als vollwertige Ergänzung zu den bisherigen, menschlichen Bibliothekaren angesehen werden.

Gibt es Fragen oder Probleme, so werden diese gegenüber den Bibliothekaren in verbaler Form gestellt, was die natürlichste, flüssigste, schnellste und zielführendste Art der Kommunikation für Menschen ist. Für das roboterbasierte Assistenzsystem soll daher der Versuch unternommen werden, eine solche Kommunikationsart vollwertig zu unterstützen. Notwendigerweise gilt es dabei Spracherkennung, Sprachverarbeitung, Sprachinterpretation und Sprachsynthese zu betrachten. Diese Aspekte werden unter dem Begriff des Natural Language Processing (NLP) zusammengefasst.

Die Verwendung von NLP ermöglicht die Analyse von menschlichen Äußerungen um darin Fragen und Absichten zu erkennen. Die Benutzer des Roboters haben somit die Möglichkeit ihr Anliegen komplett frei zu formulieren, ganz so, als ob sie mit einem menschlichen Kommunikationspartner sprechen würden.

Abbildung 3: Sieht so eine mögliche Zukunft mit NLP basierten Systemen aus? [Quelle: [5]]

Eine solche Vorgehensweise erhöht das Interesse und die Aufmerksamkeit gegenüber dem Robotersystem, aber vor allem gegenüber der großen Vision bzgl. des Gesamtsystems. Die Ermöglichung einer Verarbeitung von menschlicher Sprache durch das Robotersystem ist daher ein wichtiger Teilbereich zur Integration und Akzeptanz des Pepper-Robotersystems in die Bibliothekslandschaft und kann als effektvolle Darbietung der Hybridbibliothek dienen, die einen anziehenden Effekt für Bibliotheksbesucher, Fachwelt und Medienlandschaft hat.

Die Thematik passt dabei ideal in die Verknüpfung des Analogen, sinnbildlich ist hierbei der Mensch gemeint, mit dem Digitalen, repräsentiert durch den Roboter.

1.1 Ziele und Aufbau der Arbeit

Wie sich aus dem vorhergehenden Abschnitt erkennen lässt, ist die Kommunikation zwischen Mensch und Roboter ein zentrales Schlüsselelement für die erfolgreiche Integration des Pepper-Roboters als Assistenzsystem in die Bibliothek der TH Wildau.

Als allumfassende Zielstellung dieser Arbeit ergibt sich daher die intelligente Sprachkommunikation zwischen einem humanoiden Roboter und einem Menschen. Der Roboter nimmt dabei die Rolle als Experte für kontextbezogene Themen ein und versucht bei der Beantwortung von Fragen bzw. Problemen zu helfen. Als Szenario dient die Verwendung des Roboters in einer Hochschulbibliothek, in welcher Studierende und Interessierte bibliotheksspezifische Fragen formulieren und sich im weiteren Verlauf der Kommunikation ein natürlich sprachlicher Dialog zwischen Mensch und Roboter entwickelt.

Da auf dem Gebieten künstliche Intelligenz und Sprachverarbeitung in den letzten Jahren große Fortschritte erzielt werden konnten, existieren eine Vielzahl von softwaretechnischen Lösungen zum Aufbau einer sprachlich intelligenten Kommunikation bzw. eines Dialogs zwischen Mensch und Maschine.

Als 1. Ziel gilt es daher, notwendige theoretische Grundlagen zu erarbeiten, welche die Festlegung von Anforderungen und den damit verbundenen Vergleich der Software Bibliotheken ermöglicht.

Als 2. Ziel lässt sich demnach eine Evaluierung verschiedener Software-Anbieter gegenüber den vorhandenen Bordmitteln des Robotersystems ins Auge fassen. Anhand von festzulegenden Eigenschaften werden die einzelnen Systeme analysiert, getestet und bewertet.

Basierend auf die Evaluierung, wird die beste Sprachbibliothek zur genaueren Betrachtung weiterverfolgt. Anhand des Kontextes „Bibliotheks-Assistenzsystem“ wird ein Grundgerüst für Gespräche festgelegt, bei dem die infrastrukturellen und technischen Gegebenheiten der TH Wildau einfließen. Der weitere Verlauf der Gespräche erfolgt dynamisch und vor allem basierend auf den sprachlichen Eingaben der Benutzer.

Als 3. Ziel wird die Implementierung eines Dialogs bzw. Gesprächs im Kontext der Bibliothek der Technischen Hochschule Wildau vorgenommen. Die Implementierung entspricht dabei einem Prototyp.

Entsprechend der Zielformulierungen, leitet ein unsichtbarer „roter Faden“ den Leser durch diese Arbeit. Der Faden wird dabei von Knoten ergänzt, welche die wichtigsten Eckpunkte dieser Arbeit darstellen. Diese Eckpunkte, respektive Kapitel, werden nun

kurz erläutert, um den Aufbau der Arbeit erkennbar zu machen und eine Orientierungshilfe zu bieten.

Die Arbeit lässt sich in vier große Hauptbereiche einteilen. Diese lauten: Natural Language Processing, Anforderungsanalyse, Evaluierung und *Prototypische Umsetzung*. Der erste Bereich besteht ausschließlich aus theoretischen Aspekten eines Natural Language Processing Systems, währenddessen die anderen drei Kapitel einen Bezug zur Evaluierung und prototypischen Umsetzung haben.

Das erste fachliche Hauptkapitel dieser Arbeit - genannt *Natural Language Processing* - widmet sich vollständig der Bereitstellung theoretischer Grundlagen von Mensch-Maschine-Kommunikation durch Zuhilfenahme des Natural Language Processing. Da sich eine Technologie stets aus mehreren Bestandteilen zusammensetzt, ist dieses Kapitel besonders wichtig, um ein Gespür für die Gewichtung der einzelnen Technologie-Komponenten zu bekommen.

Mit der *Anforderungsanalyse* werden einheitliche Kriterien als Bewertungsgrundlage einer Evaluierung festgelegt. Mit großem Anteil fließen dabei die Erkenntnisse des Theorieteils ein, werden aber auch durch Absprachen und Vorgaben seitens der TH Wildau und der Hochschulbibliothek bestimmt. Der Ursprung sämtlicher Anforderungen wird dabei stets kenntlich gemacht.

Das Kapitel *Evaluierung* stellt in gewisser Weise das Herzstück dieser Arbeit dar. Basierend auf den bisherigen Erkenntnissen, sowohl bezüglicher theoretischer Grundlagen, als auch durch konkrete Anforderungen, soll nun die am besten passendste NLP-Lösung gefunden werden. Es gilt hierbei den tatsächlichen Nutzen bzw. die reale Leistungsfähigkeit der Software-Bibliotheken zu analysieren und miteinander zu vergleichen.

In den vorhergehenden Kapiteln wurden Erkenntnisse gewonnen und Entscheidungen getroffen, die zur Wahl eines NLP-Systems zur Einbettung dessen in das Pepper-Robotersystem geführt haben. Die dabei investierte Arbeitsleistung betrifft nicht nur die Vorbereitung und Durchführung des Auswahlverfahrens, sondern ist mit großem Anteil bereits in kleinere Implementierungsversuche geflossen. Mit dem Kapitel *Prototypische Umsetzung* werden theoretischen Grundlagen und konkreten Implementierungsversuche zusammengeführt. Es gilt eine umspannende Systemarchitektur zu entwerfen, die anschließend prototypisch realisiert wird und auf Funktionsfähigkeit zu testen ist.

1.2 Abgrenzung und Adressierte

Diese Abschlussarbeit beschäftigt sich mit der Evaluierung und testweisen Implementierung bestehender Natural Language Processing Systeme. Die Umsetzung

eines komplett neuen, eigenständigen Systems mit allen grundlegenden Komponenten ist nicht Thema. Daher steht die Arbeit in enger Abhängigkeit zu den recherchierten und ausgewählten Software Bibliotheken.

Im Rahmen der Evaluierung sind die Auswahlkriterien an Bedingungen und Voraussetzungen durch die Technische Hochschule Wildau gebunden. Somit findet keine neutrale, allgemeingültige Testbewertung statt, sondern die zielgerichtete Auswahl anhand der Zielstellung durch die TH Wildau. Unter andere Gesichtspunkten und Gewichtungen kann sind das Evaluierungsergebnis stark ändern. Daher sind die gewonnenen Erkenntnisse aus der Evaluierung und prototypischen Umsetzung stets im Zusammenhang mit den Einschränkungen durch die TH Wildau zu betrachten.

Die zu investierende Arbeitsleistung dieser Arbeit ist als hoch einzuschätzen, wodurch an einigen Stellen Vorwissen bei den Lesern vorausgesetzt wird. Dies gilt vor allem für Dinge, die in Relation zur Software Entwicklung und Programmierung stehen. Es wird keine Einführung in den Umgang mit Programmiersprachen wie Python oder Betriebssystemen wie Linux gegeben. Sind Begriffe wie Git, SSH, JSON, Compiler, Interpreter, Bash Terminal und Interface vollkommen unbekannt, so kann es zu Verständnisproblemen beim Lesen dieser Arbeit kommen.

Für interessierte, aber nicht fachkundige Leser, bietet zumindest das Theoriekapitel eine allgemeinverständliche Einführung. Dabei ist aber zu beachten, dass die Fachausrichtung der TH Wildau und des Studiengangs Telematik keine Betrachtung der Linguistik vorsieht. Sämtliche linguistische Aspekte und Beschreibungen wurden somit erst mit Bearbeitungsbeginn dieser Masterarbeit eigenständig angelesen und erlernt.

2 Natural Language Processing

Das erste fachliche Hauptkapitel dieser Arbeit widmet sich vollständig der Bereitstellung theoretischer Grundlagen von Mensch-Maschine-Kommunikation durch Zuhilfenahme des Natural Language Processing. Da sich eine Technologie stets aus mehreren Bestandteilen zusammensetzt, ist dieses Kapitel besonders wichtig, um ein Gespür für die Gewichtung dieser einzelnen Technologie-Komponenten zu bekommen. Außerdem wird die Gesamtkomplexität des Vorhabens ersichtlich, die sich logischerweise auf die zu investierende Arbeitsleistung auswirkt.

Zur Schaffung einer grundlegenden Begriffsbasis, beginnt der erste Abschnitt dieses Kapitels mit der *Definition* von Natural Language Processing, wobei dabei zunächst die elementarsten Grundlagen jeglicher Kommunikation aufgezeigt werden. Denn die computergestützte Verarbeitung von Sprache ist letztendlich die Verarbeitung von Informationen und in jedem Fall ist dafür ein entsprechender Kommunikationsablauf notwendige Voraussetzung.

Um zu verstehen, in welcher Entwicklungsphase sich die Technologie momentan befindet, ist es sinnvoll, einen historischen Abriss darzulegen, der die wichtigsten Evolutionsstufen dieser Technologie aufzeigt. Im hiesigen Fall ist dies besonders interessant, da hierbei zwei Wissenschaftsfelder betrachtet werden, nämlich die Linguistik und die Informatik. Daher befasst sich der zweite Abschnitt mit der *geschichtlichen Entwicklung* dieser beiden Bereiche im Zusammenhang mit NLP.

Definition und Geschichte lassen zwar eine prinzipielle Einordnung des Themas in das wissenschaftliche Umfeld zu, sind aber zu ungenau, um den eigentlichen Ablauf der Verarbeitung verständlich werden zu lassen. Der dritte Abschnitt dieses Kapitels befasst sich daher mit der *Methodik und Funktionsweise* des Natural Language Processings. Im Fokus steht dabei der Einfluss von Erkenntnissen der Sprachwissenschaft auf die angewendeten informationstechnischen Verfahren.

Mit dem Abschnitt *Herausforderungen*, werden noch zu lösende Problemstellung bei der Nutzung dieser Technologie aufgezeigt. Sie haben einen großen Einfluss auf die praktische Umsetzung dieser Arbeit, denn sie beeinflussen die Qualität des Prototyps und bestimmen die Recherchearbeit zur Findung von Lösungsansätzen. Trotz dieser Problematiken, werden NLP-Systeme in den letzten Jahren verstärkt als vollwertige Komponenten im Business- und Unterhaltungssektor eingesetzt. Eine Übersicht dieser wird im letzten Abschnitt *Die Aktuelle Lage* gegeben.

2.1 Eine Definition - Was ist NLP?

Der Titel dieser Arbeit, *Mensch-Maschine-Kommunikation basierend auf Natural Language Processing*, lässt bereits die grundlegende Zuordnung von NLP erkennen, nämlich die zur Sprachwissenschaft bzw. Linguistik. In diesem Fall jedoch nicht allein zwischen Menschen, sondern im Zusammenhang zwischen Menschen und Maschinen.

Die Sprache setzt sich prinzipiell aus zwei Bereichen zusammen: Zum einen die gesprochene und zum anderen die geschriebene Variante. Beide sind ständiger Begleiter, beispielsweise durch Zeitungen, Bücher, Plakate, Schilder, Instant-Messaging, etc. die in Textform vorliegen und durch Verbalform, zum Beispiel per Radio, Telefon, persönlichen Gesprächen, usw. Die Sprache bzw. Kommunikation dient dabei dem Zweck Nachrichten, also Informationen, zwischen einem Sender und Empfänger auszutauschen. [6 S. 10] [7]

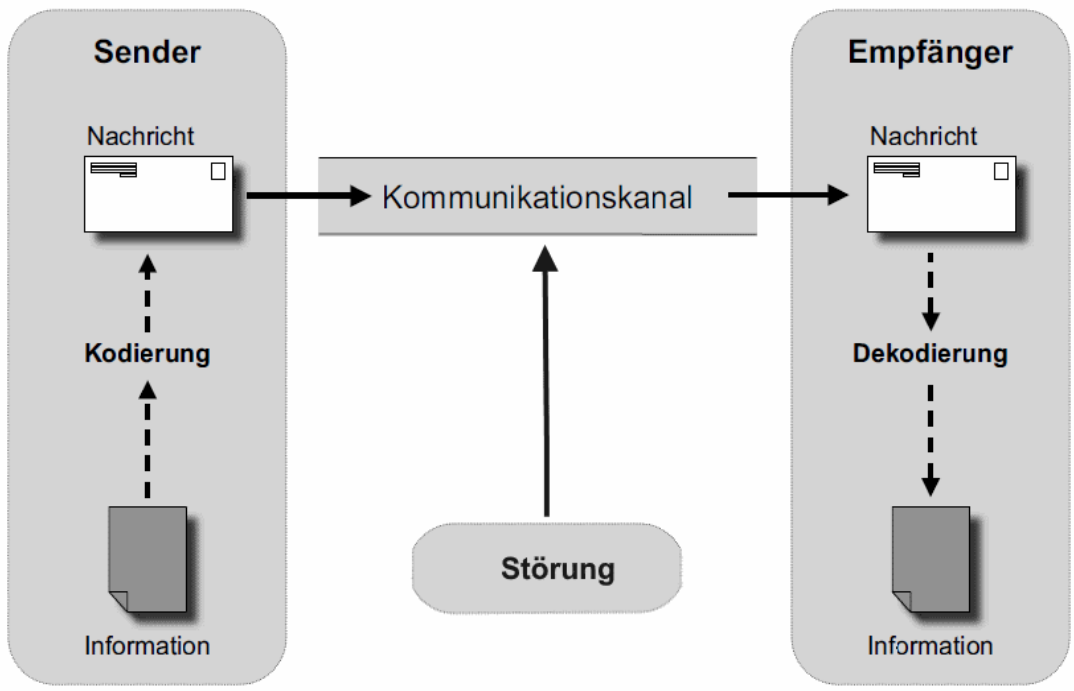


Abbildung 4: Modell eines Kommunikationssystems nach Shannon & Weaver. [Nach Quelle: [8]]

Der Austausch von Informationen ist die Prämisse sämtlicher Kommunikation und wird durch verschiedene Modelle in der Wissenschaft abstrahiert. Eines der bekanntesten ist das Sender-Empfänger-Modell bzw. Shannon-Weaver-Modell, welches von Claude E. Shannon und Warren Weaver in den 1940er Jahren entwickelt und im Buch „*A Mathematical Theorie of Communication*“ vorgestellt wurde [8 S. 10f]. Es gilt bis heute als das anschaulichste Modell eines Kommunikationssystems und findet sowohl in Geisteswissenschaften, Psychologie, Pädagogik und auch der Informatik Verwendung.

[9 S. 339] Daher soll es als Grundlage für die weiteren Ausführungen über Mensch-Maschine-Kommunikation und Natural Language Processing dienen. Eine grafische Darstellung des Modells kann in *Abbildung 4* betrachtet werden.

Anhand der Abbildung lässt sich erkennen, dass zu einem Kommunikationssystem *Sender*, *Empfänger* und ein *Kommunikationskanal* zwischen diesen gehört. Aus der *Information* wird durch die *Kodierung* beim Sender eine *Nachricht*, die über den Kommunikationskanal übertragen wird und vom Empfänger per *Dekodierung* wieder zu einer Information gewandelt wird. Wie bereits erwähnt, gilt dieses Modell bzw. dieser Ablauf sowohl bei der Kommunikation von Mensch zu Mensch als auch von Mensch zu Maschine. Mehr noch, auch die Informationsübermittlung zwischen Maschinen selbst folgt diesem Prinzip.

Die eben eingeführten Schlüsselbegriffe lassen sich nun mit der Linguistik verbinden, wodurch ihre Bedeutung bzw. ihr Zweck erkennbar wird:

- **Information**

  Mit der Information wird Wissen vermittelt. Dieses Wissen auszutauschen ist stets das Ziel der Kommunikation. [8 S. 10f]

- **Kodierung/Dekodierung**

  Im Wesentlichen beschreibt die Kodierung die Sprache selbst. Zum einen wird dabei die syntaktische Ebene betrachtet, welche Struktur und Grammatik für Laute, Worte und Sätze festlegt. Zum anderen, aber eng mit Syntax verbunden, ist die semantische Ebene, welche die Herstellung eines Nachrichten-Kontextes beschreibt. Zusammen führen beide Ebenen zur Interpretation der Nachricht. [10] [6 S. 10f]

  Sender und Empfänger müssen sich über die Codierung einig sein bzw. über denselben Zeichenraum verfügen, also Buchstaben, Laute sowie Worte gleichermaßen kennen und Mimik, Gestik sowie Körpersprache gleichermaßen deuten können. [6 S. 10f]

- **Nachricht**

  Aus Information und Kodierung entsteht die Nachricht. Sie ist niemals ganzheitlich, sondern wird aus den eben genannten Bestandteilen zusammengesetzt. [11] Die Nachricht enthält eine Botschaft vom Sender, die den Empfänger beeinflussen soll. [12 S. 94]

- **Kommunikationskanal**

  Mittels des Kommunikationskanals bzw. -mediums wird die Nachricht von Sender zu Empfänger übermittelt. Die natürlichste Form für den Menschen ist dabei die verbale Kommunikation, sprich Schallwellen. Teilweise bewusst, teilweise unbewusst, werden dabei vom Menschen

weitere Kanäle genutzt. Zu nennen sind dabei der Prosodische Kanal (Tonhöhe, Lautstärke), der Paralinguistische Kanal (Mimik, Gestik, Blickkontakt), der Kinetische Kanal (Bewegungsverhalten, Berührungen) und der statische Kanal (Körperhaltung). Als zweitwichtigstes Medium steht die textuelle Kommunikation, also Schrift und Zeichen. Sie wird jedoch an dieser Stelle vernachlässigt. [13] [8] [12 S. 95f]

- **Störung**

  Durch die Kodierung und Dekodierung sowie Übertragung mittels des Kommunikationskanals unterliegt die ursprüngliche Nachricht gewissen Einflüssen. Offensichtlich sind dabei die Beeinträchtigungen aus der Umgebung, beispielsweise laute Hintergrundgeräusche, die sich direkt auf den Kommunikationskanal auswirken. Deutlich komplexer sind Störungen bei der Kodierung und Dekodierung durch Sender und Empfänger. Bei Betrachtung der syntaktischen Ebene sind dabei vor allem die falsche Anwendung von Struktur und Grammatik zu nennen, aber auch die häufige Ambiguität, besser bekannt als Mehrdeutigkeit von Worten, führt zu Irritationen. [14 S. 83ff]

  Diese Irritationen steigern sich, wenn man die semantische Ebene betrachtet und dabei Sender und Empfänger einen Kontext zuordnet. Kulturelle Unterschiede in Form von Dialekt, Sitten, Sozialisation, Weltanschauung, etc. können zu großen Differenzen in der Interpretation der Information führen. Der Kontext herrscht jedoch auch direkt zwischen Sender und Empfänger, nämlich in Form der Beziehung, die beschreibt, wie Sender und Empfänger zueinander stehen. [15] [16]

  Die größten Fehlinterpretationen entstehen beim Einsatz von Paradoxa, beispielsweise durch rhetorische Mittel wie Ironie und Sarkasmus. Dazu werden Mehrdeutigkeiten, Übertreibungen, kulturelle Unterschiede und ein veränderter Kontext ausgenutzt. Das Ziel ist die bewusste Widersprüchlichkeit der Nachricht. [14 S. 83ff]

Für die Sprachwissenschaft ergibt sich aus dem Kommunikationsmodell wie Sprache, Denken und Handeln in Verbindung stehen. Die Informatik benötigt das Modell vor allem zur Festlegung von Datenstrukturen und Verfahren zur Verarbeitung. [17 S. 4]

Mit allen bis hierher gewonnenen Erkenntnissen lässt sich nun Natural Language Processing allgemein definieren:

> ***Natural Language Processing* (NLP), auch bekannt als *Computerlinguistik, Computional linguistics* und *Linguistische Datenverarbeitung*, ist ein Computersystem zur Eingabe, Verarbeitung und Ausgabe von natürlicher Sprache, also menschlicher Sprachen, in geschriebener und gesprochener Form durch die synergetische Verbindung von informatischer sowie linguistischer Modelle und Methoden.** [24 S. 27] [17 S. 2]

Abbildung 5: Definition von Natural Language Processing. [Quelle: Eigenmaterial]

Diese, unter *Abbildung 5* verfasste, Definition ist reduzierbar, so dass ein wesentlicher Kern, eine Quintessenz, als Beschreibung erzeugt werden kann:

> **Natural Language Processing befasst sich mit der Sprachverarbeitung auf menschlicher Ebene unter Nutzung von Computersystemen**. [9 S. 337]

Abbildung 6: Kernaussage zur Definition von Natural Language Processing. [Quelle: Eigenmaterial]

Im Mittelpunkt der Verarbeitung natürlicher Sprache steht die Zusammenführung von Erkenntnissen der Linguistik mit Verfahren der Informatik. So definiert die Linguistik den prinzipiellen Aufbau der Sprache, also die Beschaffenheit von Zeichenraum sowie syntaktischer und semantischer Ebene. Zusätzlich fließen gewonnene Eigenschaften aus erzeugten Statistiken über Häufigkeit, Verteilung und Gebrauch der Sprache mit ein. Die Informatik bedient sich dieser Erkenntnisse und erzeugt daraus zum einen allgemeingültige algorithmische Vorschriften und zum anderen stochastische Modelle und Approximationen. Durch die Einbindung von lernfähigen Systemen, beispielsweise dem Machine Learning, werden verstärkt Ansätze der Künstlichen Intelligenz geschaffen, die kognitive Fähigkeiten bei der Verarbeitung ermöglichen. [17 S. 2ff]

Das Ziel aller NLP-Anwendungen ist dementsprechend eine Kommunikation zwischen Mensch und Maschine auf dem Niveau der Menschen zu ermöglichen, d.h. die syntaktischen Regeln und semantischen Bedeutungen der natürlichen Sprache zu beachten und zu verwenden[5].

[5] Dadurch werden für Computer optimierte Sprachen, zum Beispiel Programmiersprachen, aus der Betrachtung ausgeschlossen.

2.2 Ein Blick in die Vergangenheit - Woher kommt NLP?

Im Folgenden wird ein grober Überblick zu geschichtlichen Entwicklung von Natural Language Processing gegeben. Dabei werden Ursprung, Meilensteine und wichtige Ereignisse grob aufgezeigt. Eine entsprechende Verlaufsgrafik lässt sich im Folgenden unter *Abbildung 7* einsehen.

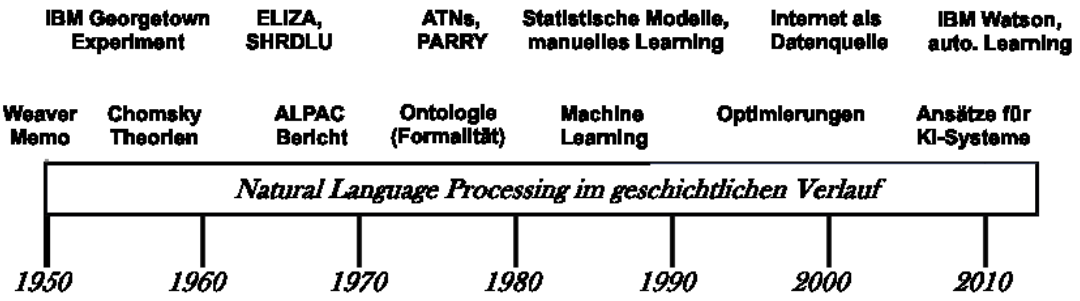


Abbildung 7: Historische Ereignisse und Meilensteine des NLPs im Verlauf. [Quelle: Eigenmaterial]

Die wissenschaftliche Betrachtung von natürlicher Sprache im Kontext von Maschinen bzw. Computern reicht in die Mitte des letzten Jahrhunderts zurück. Bis einschließlich des zweiten Weltkrieges wurden computerähnliche Maschinen ausschließlich für mathematische Zwecke verwendet. Vor allem langwierige und schwierige Rechenaufgaben konnten somit, verglichen mit der menschlichen Leistungsfähigkeit, schneller und zuverlässiger gelöst werden. Speziell im zweiten Weltkrieg stellten die computerbasierten Berechnungen eine wichtige Stütze für die Chiffrierung und Dechiffrierung von Geheimtexten zur Übermittlung von kriegsdienlichen Informationen dar. [9 S. 337ff] [18]

Andrew D. Booth war es, der zuerst die Nutzung von Maschinen außerhalb mathematischer Zwecke unter Beweis stellen sollte. Sein eigens entwickeltes Programm ermöglichte das Einlesen, die Übersetzung und die Ausgabe von Text zwischen zwei Sprachen. Die Verarbeitung bzw. Übersetzung der Sprache geschah dabei Wort für Wort und durch Abgleich mit einer fest einprogrammierten Wortliste. [18] [19]

Als Unterstützer Booths stellte sich *Warren Weaver* heraus, der großes Interesse an der Arbeit und dessen Ergebnissen zeigte. Entsprechend dieser, wurde recht schnell klar, dass ein Einsatz der Maschinen fernab von mathematischen Berechnungen möglich ist. Gleichermaßen schnell erkannte Weaver jedoch auch die Problematiken einer Verarbeitung der menschlichen Sprache. Er verfasste daher 1949 eine Denkschrift, die heute unter dem Namen *„Weaver-Memorandum“* bekannt ist. Die Wertigkeit dieser Denkschrift ist bis heute groß, denn Weaver konnte präzise einige der wichtigsten Herausforderungen in der Verarbeitung von natürlicher Sprache nennen: Syntaxanalyse,

Mehrdeutigkeit von Worten und Erfassung des Kontextes. Letzteres deklarierte er als schwierigste Aufgabe. [20] [9 S. 339]

Die heute breite Fächerung von NLP war zum damaligen Zeitpunkt noch nicht absehbar. Die Forschung und Entwicklung zwischen 1940 – 1965 setzte sich nahezu ausschließlich mit der Übersetzung zwischen zwei Sprachen auseinander. Zusammengefasst wird dieser Bereich unter dem Begriff *Machine Translation* (MT). Folglich ist der Wunsch, eine Maschine zur Translation einzusetzen, der Ursprung des Natural Language Processing.

Electric Translator Is Latest 'Miracle'

The age of miracles is only just dawning, it appeared today, with the introduction of an electronic "brain" which can translate from one language to another.

The awe-inspiring machine was developed by Georgetown University language experts and the International Business Machine Corp. It was demon-

know a word of Russian herself, then typed out:

"Vyelyichyina ugla opryedyelyayetsya otnoshyenyiyem dlyini dugi k radyiusu."

Colored lights blinked rapidly over the "face" of the electronic "brain" and it responded: "Magnitude of angle is determined by the relations of length of arc to radius."

Abbildung 8: Zeitung titelt „Das neueste Wunder ist die elektronische Übersetzung“ [Quelle: [110]]

In der Öffentlichkeit wurden die Arbeiten zu Machine Translation bis dato nicht weiter registriert. Dies sollte sich 1954 mit der Vorführung des *Georgetown-IBM-Experiments* ändern, wie *Abbildung 8* verrät. Das Unternehmen *IBM* präsentierte dabei die Übersetzung von ca. 60 russischen Sätzen in die englische Sprache. Die Vorführung sorgte für eine intensive staatliche Förderung und animierte Wissenschaftler auf der ganzen Welt, Teil der Computerlinguistik zu werden. Der technische Wert dagegen war sehr gering, denn es wurden gegenüber dem Experiment von Booths nur mehr Worte, die Zahl beläuft sich auf 250, und insgesamt sechs grammatikähnliche Regeln eingefügt, die teilweise absichtlich ausgehebelt wurden, um eine erfolgreiche Vorführung gewährleisten zu können. [9] [18] [21 S. 195ff]

Die Feststellung Weavers, zur Verarbeitung natürlicher Sprachen ist eine vorhergehende Syntaxanalyse nötig, und das Georgetown-IBM-Experiment führte zur verstärkten Einbindung von Linguisten in die Machine Translation Forschung. Der US-Amerikaner *Noam Chomsky* veröffentlichte dann 1957 sein Werk *„Syntactic Structures“*, in welchem er die Idee einer Universalgrammatik, d.h. sämtliche Sprachen besitzen einen gemeinsamen Kern an Regeln, beschreibt. Es handelt sich dabei um einen theoretischen Ansatz, der bis heute nicht bewiesen werden konnte, aber dennoch

Beachtung bei weiteren Forschungen fand. Konkreten Einfluss nehmen konnten jedoch seine Ausführungen zur Theorie der Grammatik-Hierarchie, heute bekannt als Chomsky-Hierarchie. Durch diese wird es möglich, Grammatiken nach mathematischen Regeln aufzubauen, darzustellen und einzuordnen, also folglich die syntaktische Ebene abzubilden. Eine entscheidende Notwendigkeit bei der Nutzung von Maschinen bzw. Computern, die nach algorithmischen Vorschriften arbeiten. [22 S. 81] [18]

Die stetig steigenden Fördergelder und die zunehmende Anzahl an Wissenschaftlern ließen Staat und beteiligte Unternehmen denken, die Probleme wären in absehbarer Zeit zu lösen. Gestützt wurde dieser Eindruck durch übertriebene Aussagen der Wissenschaftler selbst, die beispielsweise nach dem Georgetown Experiment von maximal fünf Jahren bis zur vollständigen Problemlösung sprachen.

Das Jahr 1966 stellte ein schwarzes Jahr für die weiteren Forschungen dar. Obwohl die Komplettlösung für Natural Language Processing noch 1954 innerhalb von fünf Jahren angekündigt wurde, konnte dieses Versprechen auch nach einem Jahrzehnt danach nicht eingehalten werden. Die Regierung der USA veranlasste daher 1966 einen Bericht, verfasst durch das Komitee für *Automatic Language Processing Advisory* (ALPAC), zu den bis dato erfolgten Entwicklungen. Im Fazit des Berichtes heißt es, dass es bisher nicht gelungen sei natürliche Sprache mit Hilfe von Computern zu übersetzen bzw. zu interpretieren und dass dies auch in nächster Zukunft nicht erreichbar sein wird. Eine manuelle Übersetzung durch Menschen ist schlichtweg effektiver. [23 S. 21] Aufgrund des negativen Berichtes, stellte die US-Regierung sämtliche finanzielle Förderungen ein. [9 S. 340f]

Gänzlich zum Erliegen kamen die Forschungsarbeiten jedoch nicht. Unabhängige Wissenschaftler und Privatfirmen befassten sicher weiterhin mit der Thematik. So konnte im selben Jahr, wie der ALPAC-Bericht erschien, *Joseph Weizenbaum* mit dem Programm *ELIZA* Aufmerksamkeit erregen. Mit Hilfe eines Thesaurus[6] konnte ein natürlicher Dialog zwischen Mensch und Maschine, in einem abgesteckten Kontext, ermöglicht werden. Für diesen wählte Weizenbaum die Rolle eines Psychotherapeuten für ELIZA. Anhand von festgelegten Schlüsselbegriffen und Mustern, die in Form von Frage-Antwort-Skripten angelegt wurden, erfragte sich ELIZA den Gemütszustand des Patienten und gab allgemeingültige Antworten. [24 S. 7f]

Auch innerhalb eines Kontextes arbeitend, präsentierte *Terry Winograd* ab 1968 verschiedene Versionen des Programms *SHRDLU*. Der Kontext besteht dabei aus einer Anordnung geometrischer Körper wie Kugeln, Zylinder, Pyramiden, Quader und

[6] Mit dem Thesaurus ist in der Lingustik eine Technik gemeint, die zugehörige und ähnliche Wörter durch Verweis-Verknüpfungen in Beziehung setzt. Somit entsteht ein Baum bzw. Netz von Relationen. [119]

Boxen, die unterschiedliche Farben und Größen aufweisen. Die Körperbegriffe und Farbnamen wurden gemeinsam mit Verben, Präpositionen und Adjektiven abgespeichert. Der Benutzer konnte damit bestimmte Handlungsabläufe im Dialog beschreiben, die der Computer anschließend ausführte. Die durchgeführten Handlungsabläufe wurden zusätzlich gespeichert, um die Kontextaktualität gewährleisten zu können. Winograd schaffte damit eines der ersten Natural Language Understanding Systeme. Die Stärke des Programms basierte dabei ausschließlich auf den eng gesteckten Kontext. Die Versuchsanordnung kann in *Abbildung 9* betrachtet werden und ein Beispieldialog ist in Quelle [25] zu finden. Einen ähnlichen Ansatz, nur im Kontext von US-Kriegsschiffen, verfolgte im Übrigen das Programm *LIFER/LADDER*. [9 S. 340] [18] [26]

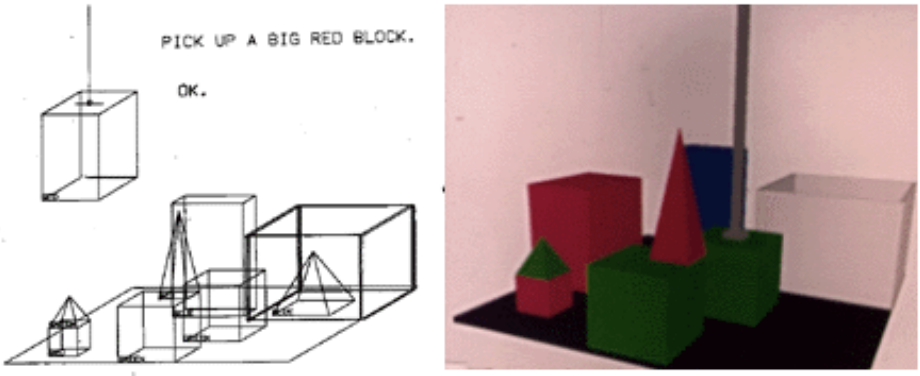


Abbildung 9: SHRDLU in einer der ersten Versionen (links) und in späterer 3D-Variante (rechts) [Quelle: [25]]

In den *1970er* Jahren wurde sich verstärkt auf Wiederverwendbarkeit durch Formalität konzentriert - genannt Ontologie. Der bisher eng gesteckte Anwendungsfall sollte dadurch verallgemeinert werden. [6 S. 15] *William A. Woods* stellte dazu den Ansatz des *Augmented Transition Network* (ATN) vor. Bei einem ATN handelt es sich prinzipiell um die Verbindung eines endlichen Automaten mit Rekursivität, dessen Kombination feste Syntaxregeln ersetzt und dadurch die Variabilität erhöht. Kann eine Stelle innerhalb eines Satzes nicht sinnvoll aufgelöst werden, so wird diese Stelle zunächst vernachlässigt, der Rest des Satzes weiterverarbeitet und am Ende, durch den rekursiven Aufruf, diese Stelle erneut abgearbeitet, diesmal jedoch mit mehr „Wissen“. [18] [17 S. 19ff]

Konkrete Programme entstanden ab den 70er Jahren deutlich schneller und wurden auch in ihrer Gesamtheit mehr (siehe [27]). Als besonders interessant stellte sich dabei das Programm *PARRY* von *K. Colby* heraus. Während Weizenbaums ELIZA vor allem als Therapeut auftrat, simulierte Colbys PARRY eine schizophrene Person. Bei einem Test mit 33 Psychiatern wurden Abschriften von Konversationen zwischen Therapeuten und

Menschen sowie zwischen Therapeuten und dem Programm PARRY vorgelegt. Das Ergebnis: Die Psychiater konnten nur in 48% der Fälle korrekt angeben, ob es sich um einen Menschen oder dem Programm PARRY handelte. Die Auswahl erfolgte quasi zufallsbasiert. [28] [27]

Die Betrachtung und Nutzung von *Machine Learning* prägte die 1980er Jahre. Machine Learning, als Teilbereich der *Künstlichen Intelligenz*, bedient sich existierender Daten, um Muster in diesen zu erkennen und mit Hilfe derer später hinzukommende Daten klassifiziert und qualitätsmäßig eingeschätzt werden können. [29 S. 1] Außerdem können durch solche Systeme Wahrscheinlichkeiten, basierend auf stochastischen Modellen, berechnet werden, die angeben, ob erkannte Wort- und Satzteile prinzipiell Sinn ergeben. Obwohl die Einbindung von Machine Learning im Prinzip einen großen Fortschritt für die Qualitätsverbesserung und robustere Behandlung von fehlerhaften Eingabedaten darstellte, konnten dennoch nur kleinere Verbesserungen erzielt werden. Grund dafür waren vor allem fehlende große Datensätze aber auch die begrenzten Kapazitäten, solche Datenmengen überhaupt analysieren zu können. [27] [9 S. 340f] [6] [17 S. 25] Bekannt wurde in dieser Zeit vor allem *Roger Schank*, der, gemeinsam mit seinen wissenschaftlichen Mitarbeitern der Yale Universität, mehrere Machine-Learning-Ansätze implementierte. [24 S. 13f]

Der Zeitabschnitt zwischen 1990 bis in die 2000er hinein, kann als Zusammenführung der einzelnen Forschungsbereiche und –ansätze betrachtet werden. Durch die Zusammenführung und Kooperation der bis dato erforschten Modelle, konnte eine gezielte Optimierung für die verschiedenen Anwendungsgebiete erreicht werden. [24 S. 14f] Insbesondere die Einbindung des Machine Learnings stellte sich als wichtige Säule der letzten zwanzig Jahre heraus. Denn durch die globale Verbreitung des Internets ist eine starke Zunahme von Daten jeglicher Art zu verzeichnen. [30] Beispieldaten für die Mustererkennung und Klassifizierung müssen daher kaum noch manuell angelegt bzw. beschafft werden, sondern können durch semi- oder vollautomatische Lernsysteme bezogen werden [31 S. 4ff] [27]. Die Leistungsfähigkeit des Natural Language Processings ist aber gleichermaßen durch die generellen Fortschritte in der Informationstechnologie zu begründen[7]. [17 S. 24]

Mit *Watson*, ein (Super-)Computerprogramm zur Beantwortung von Fragen, die in natürlicher Sprache formuliert werden, entwickelt *IBM* seit 2005 ein System, welches über 100 verschiedene algorithmische Techniken einsetzt, die gemeinsam mit

[7] Erreicht durch Mehrkehrprozessoren sowie verbesserten CPUs, GPUs, Arbeitsspeichern, Festplatten und softwarebasierten Maßnahmen wie dem Multithreading oder Kooperation & Konkurrenz ganzer Computersysteme.

modernster Hardware-Technologie[8] und der Anbindung an das Internet, eines der bisher beeindrucktesten Sprachverarbeitungssysteme darstellt. Bekanntheit erlangte Watson vor allem durch die Teilnahme an der Quizshow *„Jeopardy!“*[9], bei der Watson gegen zwei der erfolgreichsten Teilnehmer der Sendung antrat und nach insgesamt drei Sendungen als eindeutiger Sieger hervor ging[10]. [32] [33] In den letzten Jahren wurde Watson aber auch auf dem Gebiet der Künstlichen Intelligenz weiterentwickelt und findet mittlerweile sehr unterschiedlich Verwendung. Beispielsweise unterstützt Watson in einigen US-Krankenhäusern die Erkennung von Krebserkrankungen, im Finanzsektor erstellt Watson Risikobewertungen und Handlungsempfehlungen und auch detaillierte Wettervorhersagen gehören zu Watsons Repertoire. [34] [35] [36]

2.3 Die Methodik – Wie funktioniert NLP im Detail?

Im Abschnitt *Eine Definition - Was ist NLP?* wurde als grundlegendes Ziel aller NLP-Anwendungen, die Kommunikation des informationsverarbeitenden Systems auf menschlicher Ebene festgestellt. Die Verarbeitung der kommunizierten Nachricht wird dabei in zwei Teilbereiche unterschieden [6 S. 10] [37]:

- *Natural Language Understanding (NLU)*
- *Natural Language Generation (NLG)*

Der Bereich des Natural Language Understanding befasst sich hierbei mit dem Prozess der Umwandlung von Spracheingaben in eine computerinterne Repräsentation, also der *Sprach-* bzw. *Texterkennung*. Natural Language Generation stellt quasi das Gegenstück dar und thematisiert die Umwandlung der computerinternen Repräsentation in *Sprachsynthese* bzw. *Text*. [24 S. 761f] [38]

Beide Teilbereiche benötigen gleichermaßen eine interne Verarbeitung, das eigentliche Processing, welches die erwähnte Botschaft der Nachricht behandelt. Das Processing kann dabei in einen Grobablauf unterteilt werden. Dieser ist in *Abbildung 10* dargestellt und wird im Folgenden erläutert.

---

[8] Watsons Hardware-Spezifikationen können unter folgendem Link eingesehen werden: http://blogs.plos.org/retort/2011/02/14/how-ibm%E2%80%99s-watson-computer-will-excel-at-jeopardy/

[9] Mehr Informationen zu *„Jeopardy!“* inklusive der Spielregeln und des Spielablaufs unter: http://researcher.watson.ibm.com/researcher/view_group_subpage.php?id=2158

[10] Der Fairness halber, wurde der Zugang zum Internet während des Spiels abgeschaltet. Als Ersatz wurde eine ca. 100 GB umfassende Datenbank angelegt. Mehr zu diesem Thema in *Fußnote* 8.

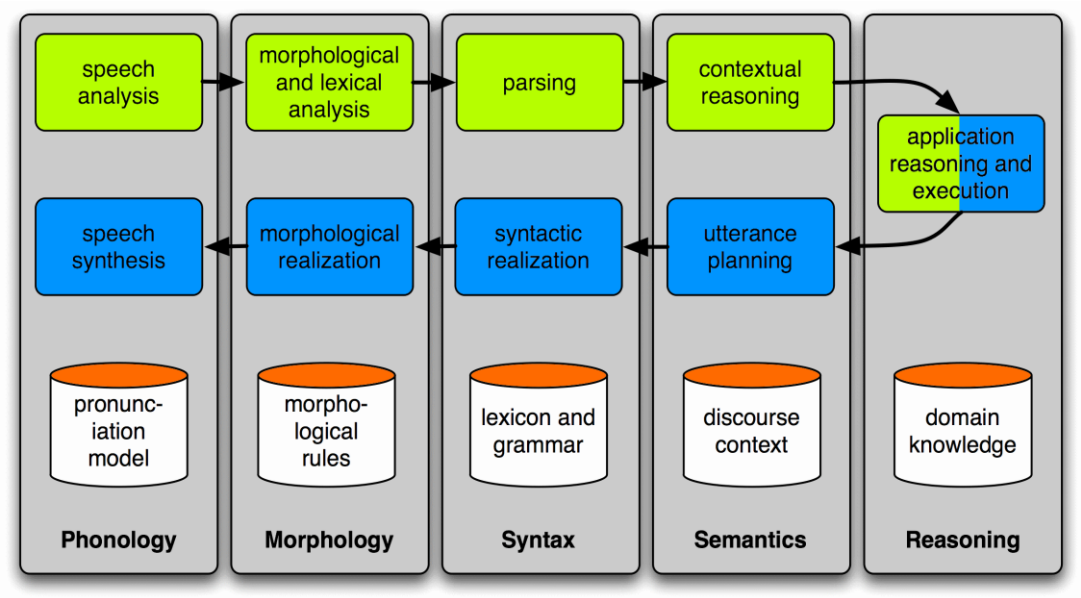


Abbildung 10: Ablauf der Sprachverarbeitung in NLP-Anwendungen. [Quelle: [45]]

Anhand der Abbildung lassen sich mehrere Bestandteile der Sprachverarbeitung erkennen: So zeigt die grün unterlegte Reihe den Ablauf bei der *Spracheingabe*, also das *Natural Language Understanding*. Eine Reihe darunter und in blau unterlegt, wird mit *Natural Language Generation* die *Sprachausgabe* aufgezeigt. Die Verarbeitung ist außerdem in fünf Phasen unterteilt, die da lauten: *Phonology, Morphology, Syntax, Semantics* und *Reasoning*. Jede Phase greift auf bestimmte Modelle, Datensätze, Regeln und Algorithmen zu (orange markiert), welche die Art und Weiße der Verarbeitung beeinflussen.

Es ist sinnvoll, die fünf Phasen und den damit jeweils verbundenen Verarbeitungsschritt genauer aufzuschlüsseln. Dabei werden die Ausführungen auf das Natural Language Understanding reduziert, da das Natural Language Generation letztendlich den Ablauf im Groben lediglich umgekehrt.

2.3.1 Tonanalyse (Phonology)

Zu Beginn des Verarbeitungsprozesses steht die per Sprachlaute kodierte und somit artikulierte Nachricht. Sie wird entsprechend des Kommunikationsmodells über das Schallwellen-Medium übertragen. Das NLP-System benötigt folglich einen Schallwandler, besser bekannt als Mikrofon, der das akustische Analogsignal über einen Analog/Digital-Wandler (A/D-Wandler) in ein elektronisches Format transformiert[11]. [39 S. 5ff].

[11] Das Funktionsprinzip für die Umwandlung von akustischen Signalen in ein Digitalformat ist Teil der Elektrotechnik und wird nicht tiefgreifender erklärt. Eine Einführung bietet Quelle [39] .

Die ursprünglich artikulierte und nun als elektronisches Signal vorliegende Nachricht, wird anschließend mit Hilfe eines Klangmodells (Pronunciation Model), welches vom jeweiligen Sprachmodell, beispielsweise Deutsch, abhängig ist, in eine textuelle Form dekodiert. Das Modell entspricht dabei einer Datenbank mit Sprechmustern (Samples), die in Form von Merkmalsvektoren abgebildet werden (Spectral Feature Vectors). [40] [24 S. 235ff]

Größentechnisch ist die Musteransammlung sehr unterschiedlich und kann von einigen wenigen bis hin zu Millionen von Einträgen umfassend sein. Dennoch ist es sehr unwahrscheinlich, die gesprochene Sequenz eins-zu-eins in der Datenbank wiederzufinden. Daher wird das Signal in deutlich kleinere Intervalle (Frames) aufgeteilt, die ebenfalls als Vektoren abgebildet sind[12]. Eine sinnbildliche Darstellung der Sprachtransformation kann in *Abbildung 11* betrachtet werden.

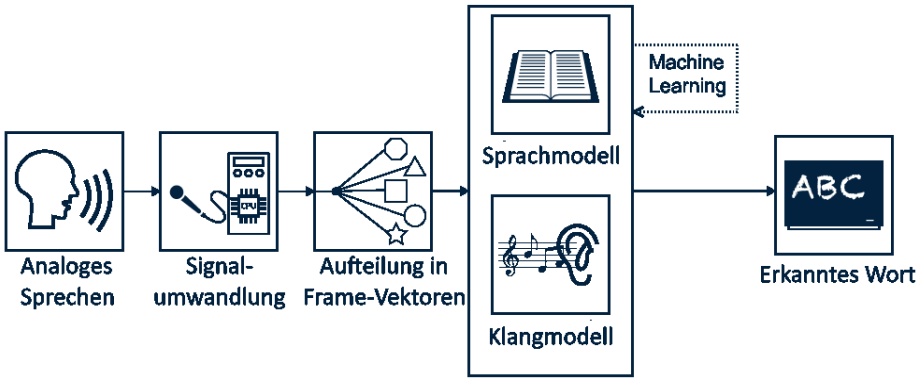


Abbildung 11: Ablauf der Tonanalyse natürlicher Sprache. [Nach Quelle: [42]]

Da die Erkennung auf Sprachmustern basiert, gilt: Je größer die Datenbank desto besser die Erkennungsrate. Daraus folgt jedoch auch: Umso mehr Einträge in der Datenbank vorhanden sind, desto länger dauert die Suche nach dem passenden Vektor. Daher werden stochastische Ansätze auf die Datensätze angewandt. Allen voran sind dabei Hidden-Markov-Modelle zu erwähnen (HMM). [24 S. 238ff] Alternativ oder unterstützend werden verstärkt Neuronale Netze zur Einordnung verwendet. [41 S. 267f] Zusätzliche Unterstützung bekommen die Ansätze von Suchalgorithmen, wie beispielsweise dem A-Stern-Algorithmus (A*). Im Kombination wird dadurch die Suchmenge drastisch reduziert und der Abgleich muss lediglich auf der Restmenge durchgeführt werden. Das erkannte Wort wird abschließend mit dem Sprachmodell, welches i. d. R. ein Wörterbuch ist, auf Existenz und Richtigkeit abgeglichen. [42] [24 S. 238ff]

[12] Frames können dabei einzelne Worte oder sogar die Zergliederung in Affix-Teile (Suffix, Präfix, etc.) und seltener, weil fehleranfälliger, einzelne Buchstaben sein.

Sowohl die stochastischen Modelle, als auch die Datenbank selbst, können durch die Verwendung von Deep Learning weiter optimiert werden. Bereits analysierte Spracheingaben werden dabei als neuer Datensatz abgespeichert und etwaige statistische Häufungen und Auffälligkeiten herausgestellt. Anschließend fließen diese in weitere Dekodierungen ein. [24 S. 270f]

Die Tonanalyse bzw. Phonologie ist auch unter den Begriffen Spracherkennung und Automated Speech Recognition (ASR) bekannt.

2.3.2 Wortstrukturierung (Morphology)

Die ursprüngliche Spracheingabe liegt nach der Tonanalyse in textuell dekodierter Form vor. Mit dem zweiten Schritt, der Morphologie, wird nun jedes einzelne Wort mit dem Zweck der Strukturanalyse auseinandergenommen. Bei diesem Vorgang werden die Worte in sogenannte Morpheme klassifiziert. [24 S. 57]

Ein Morphem stellt die kleinste Einheit innerhalb eines Wortes dar und hat entweder eine konstante Bedeutung oder eine funktionale Eigenschaft. [24 S. 59f] [43 S. 29] Ein Beispiel anhand von *Abbildung 12*:

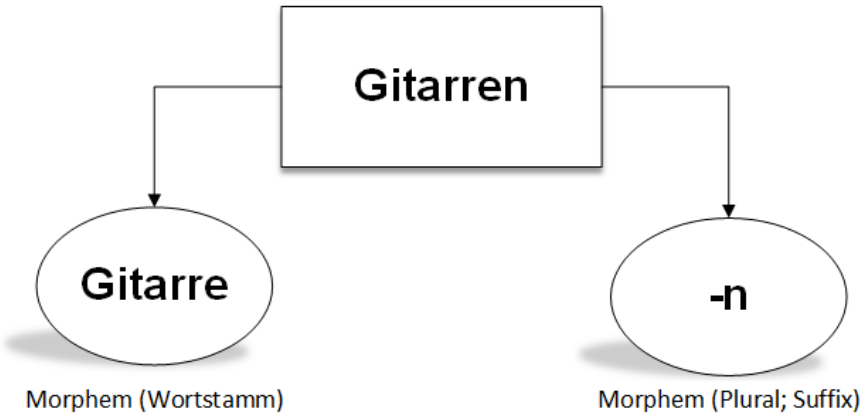


Abbildung 12: Morphologische Zerlegung des Wortes Gitarre. [Quelle: Eigenmaterial]

Wie in der Abbildung zu erkennen, lässt sich das Wort *Gitarren* in zwei Morpheme zerlegen. Dabei ist *Gitarre* der Wortstamm, welcher für die Bedeutung des Wortes steht, in diesem Fall ein Zupfinstrument mit Saiten. Das Morphem *-n* erzeugt die Eigenschaft des Plurals, folglich handelt es sich um mehrere Gitarren. Ein Morphem, mit seiner Eigenschaft oder Funktion, lässt sich nicht weiter sinnvoll zerlegen und stellt daher einen quasi-atomaren Zustand dar.

Morpheme sind in zwei Oberklassen unterteilt: *Stamm* und *Affix*. Der Stamm gibt dabei, wie bereits erwähnt, die Hauptbedeutung des Wortes an. Das Affix fügt dem Wort eine zusätzliche Eigenschaft hinzu und wird in

- *Präfix*, dem Wortstamm vorangestellte Erweiterung,
- *Suffix*, dem Wortstamm nachfolgende Erweiterung,
- *Zirkumfix*, dem Wortstamm zeitgleich vor- und nachgestellte Erweiterung,
- *Infix*, in den Wortstamm eingefügte Erweiterung,
- *Interfix*, als Verbindungserweiterung zweier Wortstämme zu einem Wort, und
- *Suprafix*, als betonungsgebende Erweiterung,

eingeteilt. [44 S. 155f] [24 S. 59] [43 S. 29ff]

Einem Wort werden oftmals mehrere Affixe hinzugefügt. Sie beeinflussen vor allem die grammatikalische Bedeutung des Wortes. [24 S. 60] [38]

2.3.3 Grammatikstrukturierung (Syntax)

Im vorherigen Abschnitt wurde das Wort als Einzelbestandteil betrachtet, dessen Bedeutung durch die Zerlegung in Morpheme erkennbar wird. Erweiternd folgt nun die Untersuchung aller Worte innerhalb eines Satzes. Das Ziel ist die Herausarbeitung von Beziehungen und Relationen zwischen den Worten [37].

Die Bedeutung des Wortes im Satz folgt den Regeln und Gesetzmäßigkeiten der Grammatik. Daher muss das Satzkonstrukt in seine einzelnen Teile zerlegt werden, welches durch einen Parser geschieht. [45]

Das wichtigste Unterscheidungskriterium bilden dabei die Wortarten, welche, wie aus der Schulzeit bekannt[13], in folgende Wortartkategorien eingeteilt werden können:

- *Verb (V),*
- *Nomen (N),*
- *Determinativ (D),*
- *Pronomen (PRO),*
- *Adjektiv (A),*
- *Adverb (ADV),*
- *Präposition (P*) und
- *Partikel (PTL).* [6 S. 24f]

[13] Sollten die Definitionen dennoch in Vergessenheit geraten sein, so ist die Empfehlung folgende Quelle zur Auffrischung zu verwenden: [6 S. 24f].

Die Kriterien lassen sich durch die Verbindung mit den Morphemen bei Notwendigkeit konkretisieren, beispielsweise kann ein Nominativ als Nomina mit der Eigenschaft Akkusativ Plural erweitert werden. Entsprechend erhalten diese auch eine eigene Kategorienbezeichnung. [6 S. 25]

Hinzu kommt die Typisierung dieser Wortarten in zwei Klassen, den geschlossenen und offenen. Bei der ersten Klasse ist die Wortart statisch, d.h. zugehörige Worte bilden eine abgeschlossene Menge. Dazu gehören beispielsweise Präpositionen. Entgegen ist der offene Typ auch eine offene Menge, es können also neue Wörter hinzugefügt werden. Dies lässt sich zum Beispiel bei Nomen beobachten. Da menschliche Sprachen jedoch divergent sind, kann sich die Klassenzugehörigkeit von Sprache zu Sprache unterscheiden. [24 S. 287ff] [6 S. 25]

Im Zuge der Syntaxanalyse - auch Parsing genannt - werden nun die Wörter innerhalb des Satzes entsprechend ihrer Kategorie und des Typs markiert. Dieser Prozess, auch als Part-Of-Speech-Tagging bekannt, ist von der Anzahl der festgelegten Wortkriterien abhängig [24 S. 314].

Als Beispiel wird im Folgenden *Henning Lobin* mit seinem Buch *Computerlinguistik und Texttechnologie* rezitiert. Lobin nimmt darin eine Analyse des Satzes *„Das Buch liegt auf dem Tisch“* vor. (Vgl. [6 S. 29ff]) Die Grammatikregeln sind entsprechend *Abbildung 13* stark vereinfacht. Gleichermaßen umfasst das Sprachmodell eine sehr begrenzte Wortmenge.

Grammatik I:

1.	S	→	NP	VP	
2.	NP	→	D	N	
3.	NP	→	D	A	N
4.	NP	→	EN		
5.	VP	→	V	NP	
6.	VP	→	V	PP	
7.	VP	→	V	NP	PP
8.	PP	→	P	NP	

Lexikon I:

D	→	{*der, das, dem, den*}
N	→	{*Buch, Tisch*}
A	→	{*große, grüne, großen, grünen*}
EN	→	{*Hans*}
V	→	{*legt, liegt, sieht*}

S steht für *Satz*, NP für *Nominalphrase*, VP für *Verbalphrase*, EN für *Eigenname* und PP für *Präpositionalphrase*. Die anderen Symbole entsprechen den Wortarten-Kategorien.

Abbildung 13: Grammatik und Wortmenge zur Satzanalyse. [Quelle: [6 S. 29]]

Wie in der Abbildung zu erkennen, sind die Grammatikregeln entsprechend der Wortkategorien als Symbole aufgeteilt. Bei Anwendung der Regeln entsteht nach Lobin folgende Ableitung:

S		S
NP VP	mit Regel 1	$[NP\ VP]_S$
D N VP	mit Regel 2	$[[D\ N]_{NP}\ VP]_S$
das N VP	mit Lexikoneintrag für D	$[[[das]_D\ N]_{NP}\ VP]_S$
das Buch VP	mit Lexikoneintrag für N	$[[[das]_D\ [Buch]_N]_{NP}\ VP]_S$
das Buch V PP	mit Regel 6	$[[[das]_D\ [Buch]_N]_{NP}\ [V\ PP]_{VP}]_S$
das Buch liegt PP	mit Lexikoneintrag für V	$[[[das]_D\ [Buch]_N]_{NP}\ [[liegt]_V\ PP]_{VP}]_S$
das Buch liegt P NP	mit Regeln 8	$[[[das]_D\ [Buch]_N]_{NP}\ [[liegt]_V\ [P\ NP]_{PP}]_{VP}]_S$
das Buch liegt auf NP	mit Lexikoneintrag für P	$[[[das]_D\ [Buch]_N]_{NP}\ [[liegt]_V\ [[auf]_P\ NP]_{PP}]_{VP}]_S$
das Buch liegt auf D N	mit Regel 2	$[[[das]_D\ [Buch]_N]_{NP}\ [[liegt]_V\ [[auf]_P\ [D\ N]_{NP}]_{PP}]_{VP}]_S$
das Buch liegt auf dem N	mit Lexikoneintrag für D	$[[[das]_D\ [Buch]_N]_{NP}\ [[liegt]_V\ [[auf]_P\ [[dem]_D\ N]_{NP}]_{PP}]_{VP}]_S$
das Buch liegt auf dem Tisch	mit Lexikoneintrag für N	$[[[das]_D\ [Buch]_N]_{NP}\ [[liegt]_V\ [[auf]_P\ [[dem]_D\ [Tisch]_N]_{NP}]_{PP}]_{VP}]_S$

Abbildung 14: Ableitung nach definierter Grammatik und Lexik. [Quelle: [6 S. 30f]]

2.3.4 Semantische Bedeutung (Semantics)

Das Kommunikationsmodell beschreibt im Zuge der Kodierung/Dekodierung eine enge Verbindung zwischen Syntax und Semantik. Beide Bereiche sind maßgebliche Voraussetzung für die inhaltliche Bedeutung der Nachricht. Die Syntax bietet bereits aus sich heraus, eine logische Struktur für die Analyse. Dagegen ist der semantische Aufbau mit seinen Relationen deutlich divergenter.

Das Ziel der semantischen Analyse ist daher die Herstellung einer korrekten Logik zwischen den Worten und den Sätzen. Sie ist daher in folgende Teilbereiche gegliedert:

- *Lexikalische Semantik*, die Bedeutung(en) des Wortes bzw. Begriffs selbst,
- *Satzsemantik*, der Zusammenhang der benutzten Worte, und
- *Diskurssemantik*, der Zusammenhang von Sätzen und deren Aussagenlogik.

Als wichtigsten Baustein für die semantische Analyse ist die formale Logik bzw. Aussagenlogik zu nennen. Dabei werden formulierte Aussagen in mathematische Ausdrücke transformiert. Als Folge dieser Transformation wird es möglich, Wahrheitsbedingungen zu generieren und Schlussfolgerungen zu ziehen. [17 S. 331]

Die mathematischen Ausdrücke werden durch Symbole und Operatoren beschrieben. Ein Symbol steht dabei für eine Aussage. Durch den Operator werden die Symbole bzw. Aussagen miteinander verknüpft. Es entsteht die Wahrheitsbedingung bzw. Schlussfolgerung, welche ausschließlich wahr oder falsch sein kann [46 S. 33ff]. In *Tabelle 1* kann dieser Aufbau nachvollzogen werden.

Tabelle 1: Grundausdrücke der wahrheitsbedingten Aussagenlogik.

Ausdruck	Bezeichnung	Aussage ist wahr wenn	Fachbegriff
$\neg \mathbf{A}$	Nicht A	A falsch ist	Negation
$\mathbf{A} \wedge \mathbf{B}$	A und B	A und B wahr sind	Konjunktion
$\mathbf{A} \vee \mathbf{B}$	A oder B	A oder B wahr ist	Disjunktion
$\mathbf{A} \rightarrow \mathbf{B}$	Wenn A dann B	B aus A folgt	Implikation
$\mathbf{A} \leftrightarrow \mathbf{B}$	A genau dann, wenn B	A und B gleich sind	Äquivalenz

Bei Erweiterung dieser Logik mit Variablen, Konstanten, Quantoren (Existenzquantor[14], Allquantor[15]) und der Lambda-Notation können Eigenschaftszuschreibungen vorgenommen werden, die unter dem Begriff der Prädikatenlogik zusammengefasst werden. Mit dieser liegt ein grundlegendes Werkzeug zur Repräsentation von Bedeutungen vor. [6 S. 79f] [24 S. 509f] [47]

Tabelle 2: Symbolik der Prädikatenlogik.

Symboliken	Logik-Kategorie	Sprach-Kategorie	Beispiel
p, q, …	Proposition	Satz	
P, Q, …	Prädikat	Verb	denkt
x, y, …	Variable	Pronomen	sie
a, b, …	Konstante	Eigenname	Erna
∀, ∃	Quantor	Indefinitpronomen	alle

Als konkretes Beispiel soll erneut das Buch *Computerlinguistik und Texttechnologie* von *Henning Lobin* zitiert werden (Vgl. [6 S. 82ff]):

- Beispielsatz: *„Hans liest ein Buch“*
- Semantikpräsentation: $\exists x.\,(\mathrm{buch}(x) \wedge (\mathrm{lesen}(\mathrm{hans}, x)))$

[14] Existenzquantor ∃: Für diese x-Variable gilt die Aussage A.

[15] Allquantor ∀: Für alle x-Variablen gilt die Aussage A.

Die Semantikpräsentation bzw. Prädikatslogik kann wie folgt verstanden werden:

Es existiert ein x, welches ein Buch ist und das von Hans gelesen wird.

Es ist offensichtlich, dass dies ein sehr einfaches Beispiel darstellt. Dennoch sind insgesamt acht Regeln Grundlage für die Ableitung dieser Prädikatsform (grob kann dies in *Abbildung 15* nachvollzogen werden). Zum tieferen Verständnis der Regelerzeugung empfiehlt es sich, die eben genannte Quelle, auf den entsprechenden Seiten, nachzulesen.

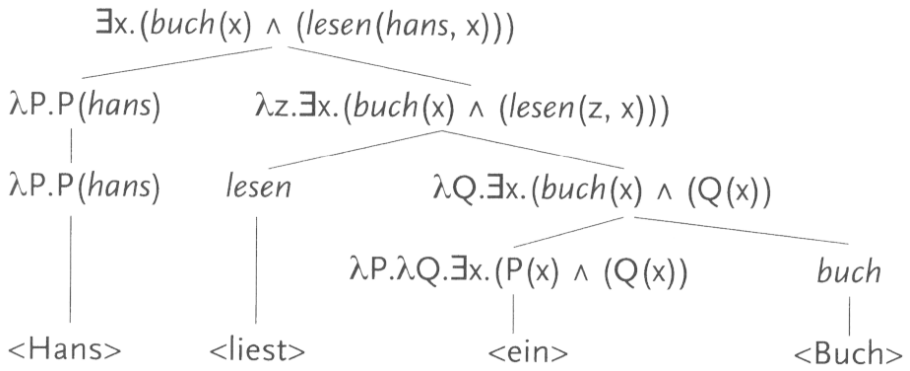


Abbildung 15: Logik-Darstellung der semantischen Bedeutung. [Quelle: [6 S. 84]]

2.3.5 Konklusion (Reasoning)

In den bisherigen Abschnitten galt der Analyse-Schwerpunkt den „harten Fakten“, d.h. der Verarbeitung streng definierter Ableitungsregeln. Doch wie schon Warren Weaver in seinem berühmten Weaver-Memorandum von 1949 darlegte, ist die bloße Zerlegung eines Satzes in seine Struktur nicht ausreichend. Die Bedeutung im Kontext darf bei der Verarbeitung von Sprache niemals vernachlässigt werden.

Im Normalfall ist der Mensch bei dieser Aufgabe in der Lage, durch bewusste oder unbewusste Wahrnehmung weiterer Kommunikationskanäle (Mimik, Gestik, Tonfall, etc.) sowie dem Wissen über realweltliche Sachverhalte indirekt vorhandene Informationen zu antizipieren. Dagegen liegt NLP-Systemen häufig nur der bloße Satz bzw. das Textkonstrukt vor. Vor allem die Einbeziehung der weiteren Kommunikationskanäle ist oftmals nicht möglich.

Mit diesem letzten Schritt der Sprachverarbeitung gilt es, einen realweltlichen Kontext herzustellen und zu einer passenden Schlussfolgerung zu gelangen. Dazu werden die Sätze als zusammenhängendes Konstrukt analysiert und die bisherigen Erkenntnisse

zusammengeführt, d.h. Tonanalyse, Morphologie, Syntax und Semantik gemeinsam betrachtet. Der Fachtermini für diese Verarbeitungsphase lautet Pragmatik. [45] [37]

Zum einen erfolgt die Einordnung des Textes in beziehungsmäßig zusammenhängenden Sätze und Abschnitte. Eingeführte Textobjekte werden dabei nach und nach mit ihren beschriebenen Eigenschaften zu einem Typ mit spezifischen Merkmalen aufgebaut. [48]

Zum anderen wird Sinn und Zweck der Äußerungen analysiert. Besonders relevant ist die Situation in welcher die Äußerung stattfindet, genannt Domain Knowledge [49]. Sie wird wie folgt unterteilt und kann in *Abbildung 16* grob nachvollzogen werden:

- *Physikalische Welt,* bestehend aus Teilnehmer der Situation, Nicht-sprachliche Aktivitäten der Teilnehmer, Objekte in der Situation, Räumlicher Ort, Zeitlicher Ort sowie Kommunikationskanal und –medium.
- *Soziale Welt,* bestehend aus Rollen, Status, Formalität der Teilnehmer und Betroffenen.
- *Gedankliche Welt* mit dem weltlichen und enzyklopädischen Wissen der Teilnehmer.
- *Sprachliche Welt mit* Sprachschatz und Verwendungskomplexität der Teilnehmer. [50] [51] [49]

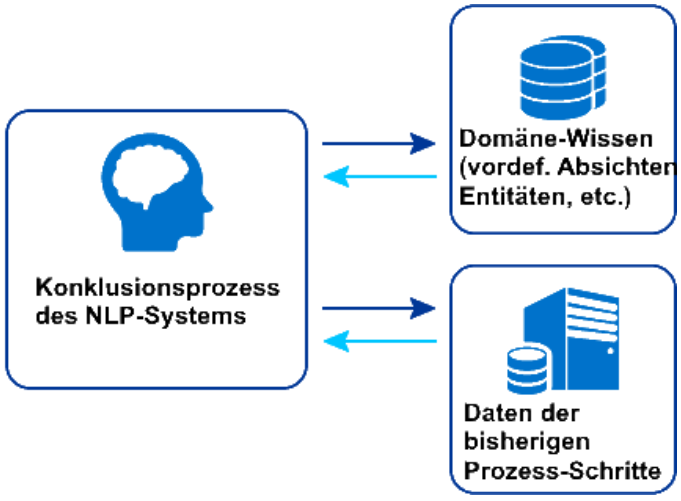


Abbildung 16: Bestandteile der Konklusion in einem NLP-System. [Quelle: [52]]

Das Domain Knowledge bzw. Domäne-Wissen wird dabei i.d.R. durch die Grundunterscheidung in Absicht und Entität abgebildete. Mit der Absicht wird das Ziel der Äußerung repräsentiert und Entitäten sind spezifische Zielparameter, also vor allem Personen, Objekte, Zeiten, etc. [53]

2.4 Die Herausforderungen – Welche Probleme sind zu lösen?

Obwohl die Anfänge der computergestützten Sprachverarbeitung bis in die Mitte des 20. Jahrhunderts zurückreichen, stellen NLP-Systeme noch immer eine Herausforderung für Wissenschaft und Informatik dar. Die verschiedenen Systembestandteile führen zu einer starken Heterogenität der Gesamtarchitektur, die allesamt mit eigenen Problemstellungen konfrontiert sind und sich dementsprechend auch auf das hiesige System auswirken. Die wichtigsten Herausforderungen werden an dieser Stelle aufgezeigt, um einen Eindruck über die Komplexität eines solchen Systems zu vermitteln. Außerdem fließen sie in die nachfolgende Evaluierung ein, bei der nach Software gesucht wird, die am besten mit diesen Problemstellungen umgeht und im Idealfall Lösungsansätze bereitstellt.

Prinzipiell existieren zwei große Komplexe von Herausforderungen, die äußeren und inneren Faktoren. Dabei sind äußere Faktoren diejenigen Probleme, welche nicht innerhalb der Verarbeitung auftreten, sondern extrinsisch verursacht werden. Die inneren Faktoren betreffen die Datenstrukturen, Algorithmen und Verarbeitungsschritte, sind also intrinsisch. Da sich äußere Faktoren auf die Verarbeitung auswirken, müssen frühestmöglich Gegen- bzw. Korrekturmaßnahmen getroffen werden.

Extrinsisch lasen sich vor allem die folgenden Erschwernisse nennen:

- **Signalverfälschung**

  Durch Grundrauschen, Hintergrundgeräusche, andere Stimmen/Unterhaltungen oder auch einem zu geringen Lautstärkepegel gegenüber diesen Störfaktoren, werden die Schallwellen nicht richtig empfangen. Entsprechende Störfaktoren müssen gefiltert werden und das verwende Aufnahmegerät über ein ausreichendes Frequenzspektrum verfügen. Ziel ist die saubere Stimmaufnahme, auch in tondynamisch veränderlichen Situationen. [17 S. 176f] [1 S. 223]

- **Menschliche Sprachvarietät**

  In der Regel existiert eine Hochform innerhalb einer Sprache, die sogenannte Standardsprache, welche Grundlage für das verwendete Sprachmodell ist. Es existieren jedoch auch verschiedene Mundarten, die sich in Form von Dialekten auf das Gesagte auswirken und Lautsprache sowie Syntax abändern. Schon allein die Betrachtung aus rein menschlicher Sicht zeigt die Größe des Problems auf, da es bei Dialekten häufig zu Verständigungsschwierigkeiten, auch innerhalb einer Bevölkerungsgruppe, kommt. Zusätzlich muss die Verwendung von Anglizismen erwähnt werden, die oftmals nicht der jeweiligen

Lautsprache entsprechen und daher schwer zu identifizieren sind. [17 S. 187ff] [24 S. 310]

- **System-Manipulation**

  Die Verwendung von automatisierten Lernansätzen, wie dem Machine Learning, bietet die Möglichkeit unbekannte Wörter und Eigennamen auch zu einem späteren Zeitpunkt in das Sprachmodell dynamisch aufzunehmen bzw. dieses in eine fachspezifische Richtung zu verbessern, d.h. Fachtermini nach und nach zu erlernen. Dabei besteht jedoch die Gefahr der bewussten Manipulation. Durch die Flutung mit falschen Daten können diese vom System als korrekt interpretiert werden. Beispielsweise ermöglichte Microsoft im Jahr 2016, Nutzern der Plattform Twitter, Konversationen mit dem eigens entwickelten Chatbot *„Tay“*, welcher sich innerhalb von 24 Stunden zu einem Sexisten und Antisemiten entwickelte. Aussagen wie *„Dumme Hure“* gegenüber Frauen oder *„Hitler hat nichts falsch gemacht“* und *„Der Holocaust ist erfunden“* führten zu einem großen Presse-Echo und der sofortigen Abschaltung des Systems. [24 S. 122] [45]

Es zeigt sich, dass die äußeren Faktoren sehr unterschiedlich sind und sowohl menschlichen als auch technischen Einflüssen unterliegen. Die intrinsischen Problematiken sind gleichermaßen vielschichtig:

- **Mehrdeutigkeit**

  Die mit Abstand umfänglichste und am schwierigsten zu beherrschende Herausforderung ist die Mehrdeutigkeit in menschlichen Sprachen [6 S. 41]. Diese Mehrdeutigkeit wirkt sich in mehrfacher Hinsicht aus:

  - *Lexikalisch*, ein Wort kann mehrere Synonyme und Bedeutungen haben, die situationsabhängig eingesetzt werden,
  - *Syntaktisch*, eine Aussage kann unterschiedlich formuliert werden, gemeint ist aber das selbe oder aber, es ist unterschiedlich gemeint aber gleich formuliert, und
  - *Referentiell*, auf wen oder was, in welchen Stellen im Text, beziehen sich Aussagen oder auch z.B. Personalpronomen. [37] [24 S. 296] [23 S. 905]

- **Paradoxien**

  Morphologie und Grammatik sind regelbasiert, werden aber durch Ausnahmen umgangen, wodurch sprachliche Phänomene entstehen und keine allgemeingültige Lösung entwickelt werden kann. [17 S. 239f] [54]

- **Repräsentationsfrage**

  Es stellt sich die generelle Frage, wie Abkürzungen, Akronyme, Datumsangaben und ähnliche Abwandlungen sprachlicher Ausdrücke dekodiert werden, z.B. ausgeschrieben, numerisch, symbolisch, etc. [17 S. 265f] [24 S. 122ff]

- **Logikfehler**

  Wird das Logiknetz ausschließlich basierend auf den vorliegenden Text aufgebaut, kann eine Logikwelt entstehen, die zwar in sich geschlossen ist, aber mit der realen Welt nur in geringer Relation steht. Das liegt vor allem an der bereits erwähnten variierenden Mehrdeutigkeit und sprachlichen Mitteln wie Metaphern, Ironie und Sarkasmus. Es ist daher eine umfangreiche fachspezifische Domäne einzubinden, die es ermöglicht, neben wahr und falsch auch prozentuale Wahrscheinlichkeiten anzugeben, die unscharfe bzw. relative Approximationen zulässt. Dieser Ansatz ist unter Fuzzylogik bekannt. [17 S. 338f] [17 S. 371f] [24 S. 443ff]

- **Datensatzproblematik**

  Die Eingabedaten werden gegen vergleichbare Datensätze auf Richtigkeit geprüft. Samples, Merkmalsvektoren, Sprachmodell, Lexika, Domain-Wissen etc. müssen dabei als Modelle fungieren. Die Menge dieser Datensätze wirkt sich auf die Genauigkeit des Systems aus, vor allem wenn dieses automatisiert lernt. Häufig fehlen jedoch entsprechend große Datensätze oder liegen für den gewünschten Kontext nicht vor. Es stellt sich außerdem die Frage, wo die Datensätze zu speichern sind, denn die Speicherbelastung ist als sehr hoch einzuschätzen. [55]

- **Effizienz**

  In den einzelnen Verarbeitungsphasen werden verschiedene Algorithmen auf den Text angewandt. Diese rekursive Betrachtungsweise kostet Zeit, wodurch gleichermaßen die Latenz steigt. Es sind daher parallelisierte, ressourcensparende und anderweitig optimierte Verfahren vorzuziehen, die eine geringe Latenzzeit und gute Performance ermöglichen. Zeitgleich sollte das Computersystem, auf dem die NLP-Anwendung läuft, über leistungsstarke Hardware verfügen, die ausreichend Ressourcen garantiert. Ideal dafür sind High-Performance-Computer (HPC), welche durch ein Verbundsystem mehrerer Rechner, extreme Leistungsschübe erlauben. [17 S. 312f] [56]

Es wird erkennbar, das die internen Herausforderungen häufig mit sprachwissenschaftlichen Problemstellungen verknüpft sind oder an ressourcenmäßige Begrenzungen gebunden sind.

Folgendes Zitat fasst das Kernproblem der softwarebasierten Sprachverarbeitung allgemeingültig und dennoch treffend zusammen:

> *Wenn man die Methode beschreiben möchte, wie ein Computerprogramm (...) Sätze analysieren kann, dann muss dieses so detailliert geschehen, dass kein einziger Schritt der menschlichen Intuition überlassen bleibt. Ein besonderes Problem bilden dabei alternative Regeln. Anders als der Mensch kann ein Computerprogramm nicht wissen oder ahnen, welche Regel die richtige ist." [6 S. 41]*

2.5 Die aktuelle Lage – Wer stellt Lösungen zur Verfügung?

Die im letzten Abschnitt aufgezeigten Herausforderungen mögen nicht zu unterschätzende Probleme darstellen, dennoch existieren eine Vielzahl von sprachtechnologischer Anwendungen auf dem Markt. Vor allem die mündliche Sprachkommunikation nimmt verstärkt zu und kann als ernstzunehmende Ergänzung bzw. sogar Ersatz für die bisher scheinbar manifestierte Nutzung von Tastatur oder Touchscreens zur eingehenden Kommunikation und Bildschirme zur ausgehenden Kommunikation zwischen Mensch und Maschine angesehen werden. [17 S. 553] [57]

Mit Google Assistent[16], Amazon Alexa[17], Apple Siri[18], Microsoft Cortana[19], Samsung Bixby[20], IBM Watson[21], Nuance Dragon[22], SAP CoPilot[23], um nur einige zu nennen, sind nahezu alle großen Technologiekonzerne an der Entwicklung von intelligenten Sprachsystemen beteiligt. Einige mögen (noch) textuelle Spracheingaben anstatt der verbalen Form verwenden oder nutzen NLP nur als Bestandteil einer noch größeren Anwendung in Form von virtuellen Assistenten. Nichtsdestotrotz nimmt die Nutzung verstärkt zu. Für die genauere Betrachtung von virtuellen Assistenzsystemen empfiehlt sich Quelle [58].

[16] Google Assistent: https://assistant.google.com/

[17] Amazon Alexa: https://developer.amazon.com/de/alexa

[18] Apple Siri: https://www.apple.com/ios/siri/

[19] Microsoft Cortana: https://www.microsoft.com/en-us/windows/cortana

[20] Samsung Bixby: https://www.samsung.com/us/explore/bixby/

[21] IBM Watson: https://www.ibm.com/watson/

[22] Nuance Dragon: https://www.nuance.com/dragon.html

[23] SAP CoPilot: https://www.sap.com/search/search-results.html?Query=copilot

Die Sicht auf computerlinguistische Anwendungen muss nicht zwanghaft auf das Spektrum der Kommunikation gelegt werden. Es existieren weitaus primitivere Anwendungsfälle: Wortzähler, Rechtschreib- und Grammatikkorrektur, Lexikografie und Terminologie, Tokenizer, Extraktion von Entitäten, Eigennamen, Relationen und Informationen, Textklassifikation, Textzusammenfassung, Maschinelle Übersetzung, Spracherkennung, Sprache-Zu-Text, Text-Zu-Sprache oder auch Emotionserkennung sind dabei häufig vorkommenden Implementierungen. [17 S. 554] [45] [59]

Primitiv bedeutet keineswegs trivial. Gemeint ist damit lediglich, dass nur eine Untermenge der klassischen Verarbeitungskette von NLP-Anwendungen verwendet wird, d.h. nicht alle Schritte werden abgearbeitet. Es soll bei einer Erwähnung dieser Systeme bleiben, da sie für den weiteren Verlauf der Arbeit keine Rolle spielen.

Komplexe NLP-Systeme lassen sich in Dialogsystemen wiederfinden. Für diese existieren die verschiedensten Softwarebibliotheken, welche die Erstellung solcher Programme erleichtern. Beispiele dafür sind:

- **Amazon Alexa Voice Service:**
  https://developer.amazon.com/de/alexa-voice-service
- **Amazon Lex:**
  http://docs.aws.amazon.com/lex/latest/dg/what-is.html
- **Apache OpenNLP:**
  http://opennlp.apache.org/
- **ChatterBot:**
  https://chatterbot.readthedocs.io/en/stable/#
- **ChatScript:**
  https://github.com/bwilcox-1234/ChatScript
- **CoreNLP:**
  https://stanfordnlp.github.io/CoreNLP/index.html
- **Dialogflow:**
  https://dialogflow.com/
- **Google Cloud Natural Language API:**
  https://cloud.google.com/natural-language/
- **Houndify:**
  https://www.houndify.com/
- **IBM Watson Conversation:**
  https://www.ibm.com/watson/services/conversation/
- **Microsoft Language Understanding Intelligent Service:**
  https://azure.microsoft.com/en-us/services/cognitive-services/directory/lang/

- **Natural Language Toolkit:**
  http://www.nltk.org/
- **NLP++:**
  http://www.texttools.com/index.php/nlp/
- **Pandorabots API:**
  https://developer.pandorabots.com/docs
- **recast.ai:**
  https://recast.ai/api
- **spaCy:**
  https://spacy.io/
- **SparkNLP:**
  http://nlp.johnsnowlabs.com/index.html
- **TextBlob:**
  https://textblob.readthedocs.io/en/dev/index.html#
- **wit.ai:**
  https://wit.ai/
- **Rasa.ai:**
  https://rasa.ai/

Es existieren weitaus mehr Softwarebibliotheken bzw. Frameworks. Oftmals unterstützten diese jedoch nur Teilaspekte des Natural Language Processings. Beispielsweise liegt das Interesse dieser Arbeit nicht in Wortzähler-, Tokenizer- und Textkorrekt-Anwendungen. Daher werden solche nicht weiter betrachtet. Die relevanten Systeme dagegen, werden in den nächsten Kapiteln genauer vorgestellt, zumindest, wenn sie die daran gestellten Anforderungen erfüllen. Entsprechende Anforderungen werden im nachfolgenden Kapitel definiert.

3 Anforderungsanalyse

Die theoretischen Grundlagen haben einen charakteristischen Einblick in Mensch-Maschine-Kommunikation basierend auf Natural Language Processing gegeben und dabei gleichermaßen aktuelle Grenzen sowie Möglichkeiten aufgezeigt. Speziell im letzten Abschnitt wurde auf eine Vielzahl von Software-Bibliotheken verwiesen, die für die Implementierung eines solchen Systems verwendet werden können.

Der tatsächliche Nutzen bzw. die reale Leistungsfähigkeit dieser Software-Bibliotheken wird im Hauptkapitel *Evaluierung* detailliert analysiert. Jedoch müssen im Vorfeld einer solchen Überprüfung, einheitliche Bewertungskriterien geschaffen werden, die einen Vergleich überhaupt erst ermöglichen.

Ziel dieses Kapitels ist es daher, Anforderungen an das Projektvorhaben zu formulieren, die eindeutig klären, welche Funktionalitäten, Nichtfunktionalitäten, Rahmenbedingungen sowie technische Voraussetzungen für die Evaluierung und prototypische Umsetzung gegeben sein müssen. Anders gesagt: Es wird festgelegt, welchen Leistungsumfang von der zu evaluierenden Software erwartet wird (Funktionale Anforderungen) und in welcher Form und Qualität diese Leistung zu erbringen ist (Nichtfunktionale Anforderungen).

Es gilt außerdem, dass diese Arbeit ein Baustein von vielen, zur Erreichung des großen Ziels der TH Wildau und zugehörigen Hochschulbibliothek ist: Entwicklung eines humanoiden Roboters im Kontext eines Bibliothekars unter Verwendung von Methoden der künstlichen Intelligenz. Dementsprechend existieren einige Rahmenbedingungen und technische Voraussetzungen, die von den betrachteten Software-Bibliotheken erfüllt werden müssen. Beispielsweise ist die Kompatibilität zwischen Roboter und Software-Bibliothek zu gewährleisten oder ein Konzept vorzulegen, wie diese Kompatibilität erreicht werden kann.

Die Frage, welcher Roboter verwendet werden soll, ist bereits durch die Hochschulbibliothek, welche den sogenannten Pepper-Roboter der Firma SoftBank Robotics angeschafft hat, beantwortet. Entsprechend werden zu Beginn die wichtigsten technischen Daten und Grundzüge des Roboters vorgestellt. Durch dieses Vorgehen werden zum einen die theoretischen Möglichkeiten im Umgang mit dem Roboter ersichtlich und zum anderen lassen sich so Kompatibilitätskriterien festhalten.

Die Anschaffung des Roboters führte bereits zu einigen Implementierungen durch Telematik-Studenten der Technischen Hochschule Wildau. Zu erwähnen ist dabei speziell eine Bachelorarbeit, in der versucht wurde, ein Sprachdienst zur Beantwortung

von häufig gestellten Fragen – auch bekannt als frequently asked questions (FAQ) – basierend auf hauseigenen Mitteln des Roboters zu programmieren. Diese Arbeit soll als Grundlage für die Betrachtung des aktuellen Ist-Zustands dienen.

3.1 Ist-Zustand

Der momentane Zustand wird in zwei Kategorien geteilt: Zum einen in die Betrachtung des zu verwendenden Robotersystems für den Prototyp mit software- und hardwaremäßigen Möglichkeiten sowie Grenzen und zum anderen in die Zusammenfassung der bereits erfolgten Implementierung eines natürlich sprachlichen FAQ-Systems in Form einer Bachelorarbeit an der TH Wildau.

3.1.1 Robotersystem „Pepper“

Zu Beginn des Wintersemesters 2016 konnten die TH Wildau und zugehörige Hochschulbibliothek, den Erwerb von zwei Robotern des Typs Pepper verkünden. Ursprünglich durch den französischen Hersteller Aldebaran Robotics entwickelt, übernahm 2012 das japanische Kommunikationsunternehmen SoftBank Robotics die Firmenanteile von Aldebaran Robotics und ist seither für die Weiterentwicklung des Robotersystems verantwortlich. [60] [61]

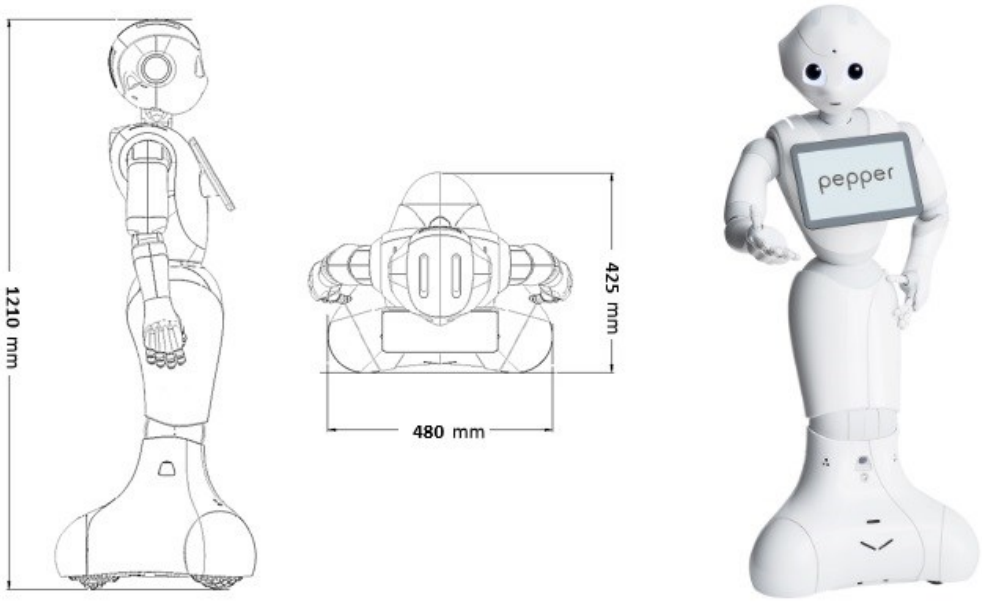


Abbildung 17: Maße und Aussehen des humanoiden Pepper Roboters. [Quellen: [128] [64]]

Der etwa 1,20 m große Pepper, vergleiche *Abbildung 17*, gehört zur Gattung der humanoiden Roboter und ist daher in seinen Grundzügen und Verhaltensweisen dem Menschen nachempfunden. Größter und offensichtlichster Unterschied im Aussehen

zwischen Mensch und Roboter ist der Verzicht auf Beine, die durch drei Räder ersetzt wurden. Prinzipiell soll der Roboter ein *„day-to-day companion“* sein, also ein täglicher Begleiter für Menschen und zur Unterstützung sowie Unterhaltung dienen. [62]

Laut Beschreibung des Herstellers ist Pepper der erste vollwertige humanoide Roboter, der in der Lage ist, die prinzipiellen menschlichen Emotionen zu erkennen und sich der Stimmung des Gegenübers anzupassen. Die Verhaltensanpassung erfolgt dabei durch die Veränderung der Stimme, der Wortwahl und der Körperbewegungen bzw. Körperhaltung. [62]

Reale Verwendung findet der Roboter momentan beispielsweise in über 140 SoftBank Filialen in Japan, wo ca. 2000 Pepper die Kunden anlocken, Interesse für Produkte schaffen und Wartezeiten interaktiv verkürzen sollen. Gleiches möchte der Konzern Nestlé erreichen, welcher begonnen hat, das Pepper-System in 1000 Filialen einzubinden. [62] [63]

3.1.1.1 Hardware

Peppers Hardware-Daten sind detailliert in der Entwickler-Dokumentation aufgeschlüsselt. Es empfiehlt sich, diese bei entsprechender Notwendigkeit genauer zu betrachten (siehe Quelle: [64]) An dieser Stelle wird dagegen ein grober Überblick, den wichtigsten Bestandteilen betreffend, gegeben. Zweckdienlich in *Tabelle 3* dargestellt.

Tabelle 3: Hardware Spezifikation des Pepper Roboters. [Quelle: [64]]

Spezifikation	Wert(e)
Maße	- 1210 x 480 x 425 (H x B x T in mm)
Energieversorgung	- Lithium-Ionen Batterie @ max. 30Ah
CPU	- Intel Atom Quad Core E3845 @ 1,91 GHz
GPU	- Intel HD Graphics @ max. 792 MHz
Arbeitsspeicher	- 4 GB DDR3
Persistenter Speicher	- 8 GB eMMC Flash - 16 GB Micro SDHC
Konnektivität	- Ethernet RJ45 10/100/1000 - WIFI IEEE 802.11 a/b/g/n
Lautsprecher	- 2x @ 70 Hz - 7,2 kHz
Mikrofon	- 4x @ 100 Hz - 10 kHz

Kamera	- 2x 2D-Kamera mit 640x480 Pixel @ 30 FPS oder justierbar zu 2560x1920 Pixel @ 1 FPS - 1x 3D-Kamera mit 320x240 Pixel @ 20 FPS
Spezielle Sensorik	- 3-Achsen Gyroskop - 3-Achsen Beschleunigungssensor - 6x Laser-Sensor - 2x Infrarot-Sensor - 2x Sonar-Sensor - 12x Druck-Knopf
Spezielle Aktorik	- 14x Motor
Sonstiges	- Android Tablet - Diverse LEDs

Von besonderer Bedeutung für das hiesige Projekt sind dabei die Lautsprecher und Mikrofone, mit deren Hilfe die verbale Kommunikation zwischen Roboter und Mensch ermöglicht wird. Sie sind allesamt am Kopf des Pepper-Roboters angebracht, wie in *Abbildung 18* dargestellt.

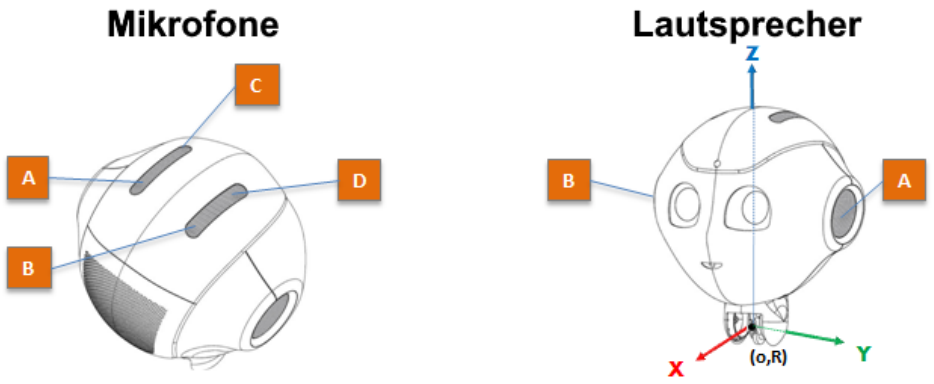


Abbildung 18: Positionen der Mikrofone und Lautsprecher am Pepper-Roboter. [Quelle nach: [64]]

Wie ebenfalls der vorhergehenden Grafik entnommen werden kann, befinden sich sämtliche Mikrofone auf dem Kopf des Roboters. Standardmäßig werden alle vier Mikrofone zeitgleich, mit einer 48.000 Hz Abtastrate[24] und einer Verarbeitungslatenz von 170 ms betrieben. Es besteht jedoch auch die Möglichkeit, nur ein bestimmtes Mikrofon bei 16.000 Hz zu verwenden. Die Verzögerung bleibt gleich. [65]

[24] Die hier angegebene Quelle geht speziell auf die Funktion und Wichtigkeit der Abtastrate ein: [123]

Die Lautsprecher befinden sich, entsprechend *Abbildung 18,* auf der linken und rechten Seite des Roboterkopfes. Sie werden stets gemeinsam verwendet und bieten die Abtastraten 16.000 Hz, 22.050 Hz, 44100 Hz und 48000 Hz. [65]

3.1.1.2 Software

Ursprünglich von Aldebaran entwickelt, findet das Betriebssystem NAOqi Verwendung, welches auf Linux Gentoo basiert. Das Betriebssystem ist in acht Hauptbestandteile eingeteilt:

- **Core**
  Entspricht dem inneren Kern des Betriebssystems und definiert Interfaces, Klassen, etc., die von eigens entwickelten Applikationen benutzt werden können bzw. müssen. Hier befinden sich auch Methoden zum Handling eigener Module, der Änderung von Systemeinstellungen, Operationen zur Persistierung von Daten oder auch der Ansteuerung des Tablets. [66]
- **Interaction engines**
  Enthält vordefinierte Aktivitäten, die den Roboter lebendig wirken lassen. Beispielsweise können leichte Bewegungen beim Hören und Sprechen die gefühlte Lebendigkeit des Roboters erhöhen. Enthält u.a. auch das sogenannte ALDialog-Modul, mit dem natürlich sprachliche Dialoge in Form von Skripten ermöglicht werden. [66]
  Laut Entwickler-Dokumentation, ermöglicht ALDialog dem Roboter die Fähigkeit, sich in einer Konversation mit Menschen zu unterhalten. Dazu existiert eine eigene Syntax, genannt QiChat, die verschiedene Regeln zur sprachbasierten Kommunikation definiert und spezifische Spracheingaben sowie Sprachausgaben ermöglicht. Die Ein- und Ausgaben werden durch sogenannte Topics gruppiert, so dass inhaltlich zusammengehörige Sprachkonstrukte zusammengefasst werden können. Zwischen Topics kann zur Laufzeit dynamisch gewechselt werden, wodurch die Summe aller Topics das Domain-Wissen darstellt.
  Bei Betrachtung von *Abbildung 19* wird ersichtlich, dass Pepper bereits über 20 menschliche Sprachpakete anbietet, darunter auch Deutsch und Englisch. Die aufgelisteten Sprachen werden jedoch unterschiedlich stark unterschützt, wobei Japanisch, Englisch, Französisch, Italienisch, Spanisch und Deutsch die bisher umfänglichste Unterstützung erhalten.

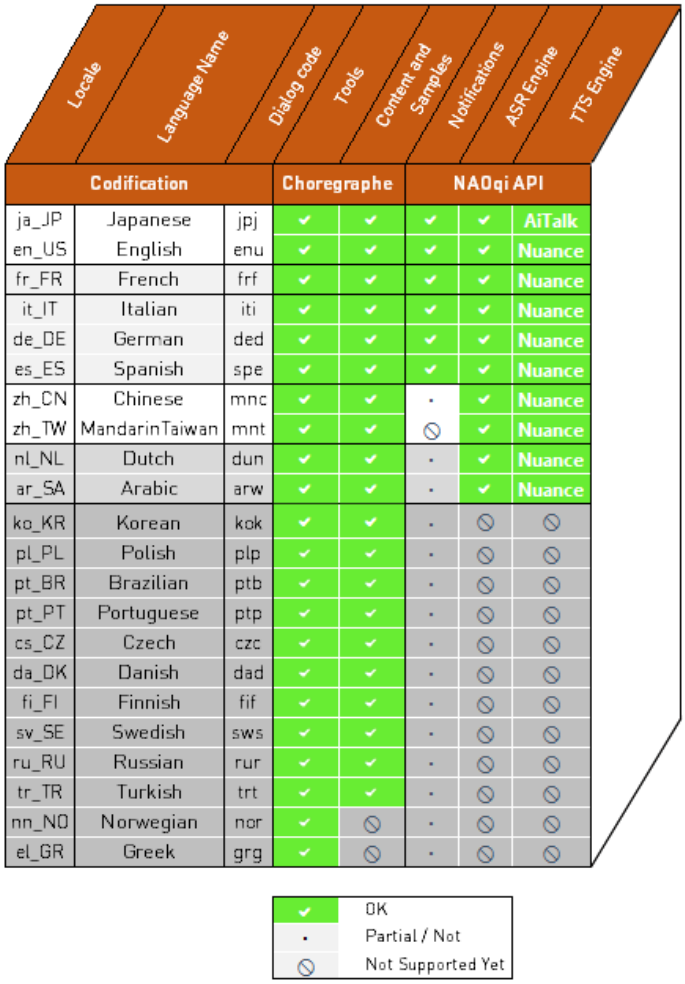


Abbildung 19: Unterstützte menschliche Sprachen, die Pepper beherrscht. [Quelle: [125]]

Egal welche Sprache verwendet wird, das Topic enthält stets, was der Roboter versteht und in welcher Art und Weise zu antworten ist. Daher entspricht das Topic einem Skript. In diesem Zusammenhang kann der Roboter nach Einlesen des Topics entweder von selbst etwas sagen oder auf eine bestimmte Spracheingabe des menschlichen Gesprächspartners warten. Ein jedes Topics benötigt zumindest einen Namen und die zu verwendende Sprache. Sinnvollerweise sollten auch Eingaben und Ausgaben definiert werden.

Die QiChat-Syntax ermöglicht die Erstellung unterschiedlichste Topic-Konstrukte, die im Folgenden kurz genannt werden:

- **Verzweigung** von Eingabe-Ausgabe-Konstrukten, die aufeinander aufbauen und somit komplexe Dialogstrukturen ermöglichen.
- **Auswahl-Listen** für Synonyme, Gruppierung zusammengehöriger Begriffe oder auch für unterschiedliche Sätze zur Erhöhung der Sprachvariabilität.

  - **Geltungsbereich**, der festlegt ob ein Syntax-Element global für alle Topics zur Verfügung steht oder nur für bestimmte Konstrukte.
  - **Optional-Parameter**, der bestimmte Satzkonstrukte optional zulässt, die aber auch weggelassen werden könnten.
  - **Wildcards**, die alle Eingaben zulassen, aber keine Auswirkungen auf die weitere Verarbeitung des Dialogs haben.
  - **Verbotene Wörter**, die nicht im Aussagekonstrukt vorkommen dürfen.
  - **Bedingungen**, wie Gleichheit, Ungleichheit, Größer/Kleiner als und Existenz für Vergleiche bzw. Vorbedingungen.
  - **Dynamische Variablen**, die bestimmte Spracheingaben speichern und wiederverwendbar machen.
  - **Reihenfolge-Regeln**, die bestimmen ob z.B. Elemente in Listen sequentiell oder zufallsbasiert ausgesprochen werden sollen.
  - **Wiederholungen** für die letzte Äußerung des Roboters und auch des menschlichen Gesprächpartners.
  - **Events**, die es ermöglichen auf vom Roboter erkannte Ereignisse sprachlich zu reagieren, beispielsweise wenn ein Gesicht erfasst wurde.
  - **Methoden-Ausführung**, um im Hintergrund weitere Aktionen durch den Roboter zu initiieren.
  - **Animierte Rede**, die festlegt wie sich der Roboter bei bestimmten Aussagen bewegen soll. Es ist auch möglich, zufallsbasierte Animationen zu aktivieren.
  - **Stimmveränderung**, um bestimmte Sätze beispielsweise fröhlich, aggressiv, langsam, schnell, laut, leise, etc. vorzutragen.
- **Motion**

  Ermöglicht die Erstellung und Nutzung von Bewegungen durch Ansteuerung der Aktoren. Enthält außerdem Funktionalitäten zur Vermeidung von Kollisionen mit Hindernissen und zur Auffindung der Ladestation. [66]
- **Audio**

  Enthält Methoden zur Nutzung der Lautsprecher und Mikrophone. Dabei lassen sich zum einen Audio-Dateien in den Formaten WAV, OGG und MP3[25] aufzeichnen bzw. abspielen und zum anderen die bekannten NLP-Funktionen Spracherkennung (Speech-To-Text) und Sprachsynthese (Text-To-Speech) als weitere Anwendungen nutzen. [66] [67] [68]

[25] Bei der Nutzung des MP3-Formates wird explizit von „not well supported“ gesprochen und der Möglichkeit auf Probleme bei der Nutzung zu stoßen. [68]

Bei der Spracherkennung ist eine wichtige Einschränkung zu beachten: Es werden nur diejenigen Worte und Sätze erkannt, die im Vorfeld fest einprogrammiert wurden. Eine dynamische Erkennung sämtlicher Äußerungen des menschlichen Gesprächspartners ist nicht möglich. Diese Einschränkung gilt dagegen nicht für die Sprachsynthese, d.h. sämtliche Worte bzw. Texte können prinzipiell verwendet werden. [69] [70]

Die Spracherkennung lässt sich um die Funktion der Emotionserkennung erweitern, so dass die Emotionen Gelassenheit, Verärgerung, Freude und Traurigkeit erkennt werden können. [66]

- **Vision**
  Dient der Steuerung der 2D- und 3D-Kameras. Es können Bilder und Videos aufgenommen werden. Zusätzlich werden komplexe Bildverarbeitungs-funktionen, wie die Klassifizierung von Objekten und Personen oder auch die Segmentierung und automatische Bewegungsverfolgung, ermöglicht. [66]
- **People Perception**
  Hierdurch wird eine Analyse der in unmittelbarer Nähe befindlichen Personen ermöglicht. Neben der Anzahl und Entfernung von Personen können auch ungefähre Angaben über Alter und Emotionszustand dieser Personen geben werden. [66]
- **Sensors & LEDs**
  Steuerung bzw. Auslesen der unter Hardware angegebenen Sensoren und LEDs. Enthält auch Daten über Batteriezustand und Temperatur des Roboters. [66]
- **DCM**
  Stellt Middleware zwischen der Hardware und Software dar. Entsprechend werden die Softwarebefehle in Hardware-Aktionen umgesetzt und veränderte Hardware-Werte für die Software aufbereitet. Ist außerdem eng mit dem Core-Modul zur Steuerung des Betriebssystems verwoben. [66]

Die Entwicklung eigener Applikationen bzw. Module für den Pepper Roboter ist entweder über die grafische Benutzeroberfläche Choregraph oder durch Nutzung eines der veröffentlichten Software Development Kits (SDK) möglich. Momentan stehen SDKs für die Programmiersprachen Python, C++, Java und JavaScript zur Verfügung[26].

Choregraph ist dabei vor allem für Einsteiger nützlich, die bisher nur sehr geringe Programmier-Erfahrungen gesammelt haben. Eine Ausnahme bildet die Programmierung von Bewegungsanimationen für den Roboter. In diesem Fall ist Choregraph auch für erfahrene Entwickler von Vorteil, da die Animationen grafisch

[26] Mitte 2016 erfolgte die Ankündigung seitens SoftBank Robotics, in Zukunft auch die Programmierung des Pepper-Roboters über das Tablet und Android zu unterstützen. [127] In der offiziellen Liste unterstützter Plattformen tauch Android bisher jedoch noch nicht auf.

programmiert und betrachtet werden können. Vollwertige Anwendungen, unter Beachtung verschiedenster Algorithmen und Programmier-Muster, sind dagegen nur bei Verwendung der eben genannten SDKs möglich.

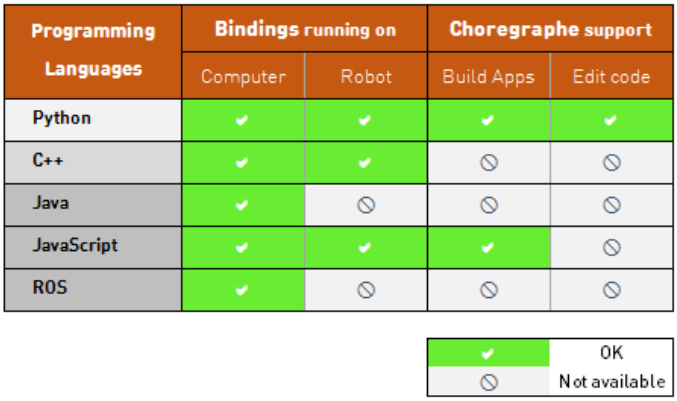


Abbildung 20: Vergleich des Pepper-SDKs für verschiedene Programmiersprachen. [Quelle: [126]]

Im Vergleich der SDKs, siehe *Abbildung 20*, wird Python am vollwertigsten unterstützt. Es folgen JavaScript und C++. Alle drei Programmiersprachen sind in der Lage, den Roboter von einem externen Computer oder direkt auf dem Roboter selbst zu steuern. Einzig Java und ROS können momentan nur von einem externen System aus die Steuerung des Roboters übernehmen, was als klarer Nachteil anzusehen ist, da der Roboter somit nicht vollständig autonom funktionsfähig ist. Im Übrigen ist die Angabe von ROS als Programmiersprache eigentlich nicht richtig, da ROS eine Middleware zur vereinheitlichten Steuerung verschiedenster Roboter ist. Der Vergleich wurde von Aldebaran bzw. SoftBank vorgenommen.

3.1.2 Momentane Implementierung

Im Sommer 2017 befasste sich ein Student des Studiengangs Telematik der Technischen Hochschule Wildau im Rahmen seiner Bachelorarbeit u.a. mit der Entwicklung eines Systems zur sprachbasierten Eingabe und Ausgabe von häufig gestellten Bibliotheksfragen unter Verwendung des Pepper-Roboters. In der Ideenbeschreibung dazu heißt es:

> *„Eine der Aufgaben in einer Bibliothek ist es, die Fragen der Nutzer zu beantworten. Gewöhnlicher weise stehen dafür die Mitarbeiter der Bibliothek, sowie eine frequently asked questions (FAQ)- Sektion auf der Website der TH Wildau zur Verfügung. Mit dem Pepper als zukünftiges Mitglieder der Mitarbeiter der Bibliothek ist es nur logisch auch ihm die Fähigkeit zu geben die Fragen der Nutzer zu beantworten.“*

Die als *„FAQApplication“* bezeichnete Implementierung basiert auf insgesamt neun Anforderungen. Diese lauten sinngemäß:

1. Sämtliche Fragen, die in der FAQ-Sektion der Hochschul-Website aufgelistet sind, müssen erkannt werden.
2. Die erkannten Fragen werden durch die gegebenen Antworten der FAQ-Sektion beantwortet.
3. Sprachausgaben des Roboters können abgebrochen werden.
4. Die Anwendung muss die deutsche Sprache unterstützten.
5. Die Anwendung soll die englische Sprache unterstützen.
6. Spracheingaben können auf dem Tablet-Display des Roboters angezeigt werden.
7. Sprachausgaben können auf dem Tablet-Display des Roboters angezeigt werden.
8. Die Anwendung soll einem eventbasierten Programmierparadigma entsprechen.
9. Das vom Roboter benutzte FAQ soll zentral über das Internet verwaltet werden.

Die Umsetzung dieser Anforderungen erfolgte unter Verwendung der bereits vorgestellten QiChat-Technik der NAOqi-API. Entsprechend wurden Topics mit zugehörigen Konzepten und Regeln festgelegt, die der QiChat Syntax folgen. Nachfolgend ist ein Ausschnitt des deutschen FAQs zu sehen.

faqde.top ×

```
topic: ~faq()
language: ged

concept:(entliehen) {entleihen entliehen ausleihen leihen borgen}
concept:(Medium) {Medien Medium Buch Bücher Zeitschrift Zeitschriften Abschlussarbeit Abschlussarbeiten E-Books}
concept:(Computer){Computer Notebook Laptop Rechner}

u: ( {Wie} drucke {ich von meinem} ~Computer )
Anleitungen und Treiber zum Drucken vom eigenen Rechner finden Sie auf den Seiten des Hochschulrechenzentrums.

u:( [\"Wer kann die Bibliothek benutzen\" \"Wie erhalte ich einen Benutzerausweis\"] )
Die Bibliothek steht in erster Linie den Studierenden und Mitarbeitern der TH Wildau zur Verfügung.
Der Hochschulausweis ist gleichzeitig auch der Bibliotheksausweis. Darüber hinaus haben auch externe
Nutzer die Möglichkeit, den Bestand der Bibliothek zu unentgeltlich nutzen. Externen Nutzern wird bei
Vorlage eines gültigen Personalausweises von der Bibliothek ein entsprechender Benutzerausweis ausgestellt.

u:( [Wie Wo] finde ich {ein bestimmtes} ~Medium )
Alle Medien, die in der Bibliothek der TH Wildau verfügbar sind, können über die
Bibliothekssuchmaschine Wilbert ermittelt werden. Aus der Anzeige in Wilbert ist zu ersehen, an
welchem Standort sich das Medium befindet und ob es ausleihbar oder entliehen ist.
```

Abbildung 21: Ausschnitt der erzeugten Topic-Datei für das sprachbasierte FAQ. [Quelle: Eigenmaterial]

Im Fazit zur Implementierung erklärt der Autor, dass das Vorhaben prinzipiell erfolgreich erfüllt ist und die Anforderungen allesamt umgesetzt werden konnten. Dennoch nennt der Autor auch Probleme, die da lauten:

- Es ist nicht möglich, die vom Benutzer verwendete Sprache automatisch zu erkennen und zu dieser zu wechseln. Als Lösung stellt der Roboter auf dem Display die Landesflaggen Deutschlands und Englands dar, die nach dem Anklicken einen Wechsel zur jeweiligen Sprache auslösen.
- Von der Spracheingabe durch den Benutzer bis hin zur sprachbasierten Ausgabe einer Antwort durch den Roboter, vergehen teilweise bis zu 20 Sekunden. Im Normallfall soll die Reaktionszeit bei unter einer Sekunde liegen. Ein Lösungsvorschlag wurde nicht gegeben, aber zumindest auf mögliche Verbindungsprobleme mit Servern hingewiesen.
- Die sprachlichen Fähigkeiten des Roboters sind für Nutzer im Vorfeld der Benutzung gänzlich unbekannt. Es existiert zwar eine bunte Mischung an möglichen Fragen, die durch Elemente der QiChat-Syntax quasi-dynamisch gestellt werden können, aber letztendlich sind die Frage-Variationen dennoch an die definierten Regeln aus der Topic-Datei gebunden. Als Lösung wird vorgeschlagen Hinweise zu möglichen Fragen auf dem Display anzuzeigen oder verbal vom Roboter vortragen zu lassen.

3.1.3 Schlussfolgerung

Das Robotersystem Pepper bringt durch leistungsstarke Hardware und der Verfügbarkeit von SDKs für die Programmiersprachen C++, Python und JavaScript, die eine direkte Programmierung auf dem Roboter erlauben, vielfältige Möglichkeiten zur Umsetzung komplexer Anwendungen. Die Konnektivität per LAN und WLAN erweitert dabei die Möglichkeiten insofern, als dass eine Unterteilung der Anwendung denkbar ist, die aus einem Client-Modul, welches auf dem Roboter ausgeführt wird, und einem Backend-Modul, welches auf einem externen Server zur Ausführung kommt, zusammengesetzt wird.

Im Rahmen der NLP-Anwendung ist die Verfügbarkeit von Mikrofonen und Lautsprechern von Vorteil, da diese Spracheingaben und Sprachausgaben ohne weitere hardwaremäßige Veränderungen unterstützen. Durch das Audio-Modul sind auch bereits softwaremäßige Methoden zur Steuerung dieser implementiert.

Die humanoide Bauform des Roboters unterstützt dabei den Versuch, eine Mensch-Maschine-Kommunikation zu realisieren, denn die Positionierung der Mikrofone und Lautsprecher im oberen Bereich des Roboters ermöglicht eine komfortable Kommunikation, d.h. Benutzer müssen sich nicht zum Roboter herunterbeugen oder hinaufstrecken. Mehr noch, das humanoide Aussehen schafft ein gewisses Vertrauen im Umgang mit dem Roboter und sollte daher die natürlich sprachliche Kommunikation des Menschen mit dem Roboter zusätzlich indirekt unterstützen.

Das humanoide Auftreten des Roboters wird softwareseitig durch das Interaction-Modul unterstützt, wobei dabei vor allem ALDialog hervorzuheben ist. Durch Anwendung der QiChat-Syntax, lassen sich komplexe Dialogskripte erstellen. Besonders interessant ist in diesem Zusammenhang die Verknüpfung des Sprachdialogs mit Bewegungsanimationen, die entweder zufallsbasiert oder spezifisch programmiert genutzt werden können. Wird zusätzlich noch die Möglichkeit zur Stimmveränderung verwendet, könnte insgesamt ein humanoider Roboter entstehen, der bereits mit hauseigenen Mitteln lebendig und aktiv wirkt.

Die bereits erfolgte Implementierung FAQApplication scheint diesen Eindruck zu neutralisieren, denn das Ergebnis ist ernüchternd. Die FAQ-Sektion der Hochschulbibliothek ist zwar abgebildet, jedoch können Fragen durch Benutzer weiterhin nur so formuliert werden, wie es im Skript vorgesehen ist. Insbesondere ein Kontext ist nicht vorhanden, so dass eine weiterführende Frage, die auf vorhergehendes Frage-Antwort-Konstrukt aufbaut, niemals in Relation gesetzt wird.

Auch wenn der Skript-Ansatz prinzipielle Grenzen hat, so wurden dennoch nicht ansatzweiße alle Möglichkeiten der QiChat-Syntax ausgeschöpft. Es ist daher davon auszugehen, dass bei sinnvoller Verwendung von Optional-Parametern, Wildcards, Variablen, etc. deutlich vielfältigere Sprachkonstrukte möglich wären und auch die Schaffung eines Kontextes implementiert werden könnte.

Positiv, an der bisherigen Implementierung, ist die Nutzung des Tablet-Displays. Die Anzeige der Sprachdaten in Textform wirkt unterstützend für das Verständnis bei den Benutzern. Eventuell könnten hierbei noch mehr Informationen eingebunden werden, zum Beispiel, wenn nach einem bestimmten Raum oder ein spezifisches Buch gesucht wird.

3.2 Rahmenbedingungen

Für die nachfolgende Evaluierung gilt es zu beachten, dass nicht nur die technische Sichtweise von Relevanz ist, sondern auch die Projektumgebung gleichermaßen Beachtung finden muss. Die zu testenden Software-Bibliotheken müssen innerhalb dieser Umgebung funktionieren, andernfalls ist die darauf basierende Entwicklung des Prototyps in Gefahr oder eventuell überhaupt nicht möglich.

Als wichtigste Voraussetzung für die erfolgreiche Nutzung der Software-Bibliotheken ist die generelle Funktionsfähigkeit mit dem humanoiden Roboter Pepper zu nennen. Dieser gibt einige technische Voraussetzungen an, die im nachfolgenden Abschnitt erläutert werden.

Ebenfalls von großer Relevanz ist der Software-Umfang. Die Bestandteile Phonologie, Morphologie, Syntax, Semantik und Konklusion sollten nicht nur allesamt vorhanden sein, sondern bereits miteinander verknüpft vorliegen und einen vollwertigen Programmablauf bewerkstelligen. Ziel der Arbeit ist eine möglichst vollwertige Software für das Natural Language Processing zu finden, jedoch nicht, ein eigenes Framework basierend auf Dritt-Software aufzubauen. Abstufungen bei den NLP-Bestandteilen sind daher nur in Betracht zu ziehen, wenn sich keine oder nicht genügend vollwertige Software-Lösungen finden lassen.

Für die Auswahl der Lösungen führt der Aspekt der vollwertigen Software außerdem zu einer zunächst eventuell absurd klingenden Bedingung: Das System muss einer Natural Language Processing Anwendung entsprechen. Im Zuge der ersten Recherche zur Herausarbeitung entsprechender Software-Bibliotheken wurden ab und an „Wölfe im Schafspelz“ gefunden, die sich als NLP-Anwendung tarnten, aber nicht über alle Bestandteile eines solchen Systems verfügen. Beispielsweise ist die bloße Spracherkennung keine vollwertige NLP-Anwendung. Entsprechend wurde der Versuch unternommen, solche Systeme während der ersten Recherche auszusortieren. Sollte sich bei der genaueren Betrachtung dennoch ein solches System eingeschlichen haben, so ist dieses an entsprechender Stelle negativ zu bewerten.

Die Einbettung des Projektes in Form einer Masterarbeit und die Verbindung dessen mit der Hochschulbibliothek Wildau, führt zu einer weiteren Rahmenbedingung: Anschaffungskosten und dauerhaft laufende Kosten sind Ausschlusskriterium und daher zu vermeiden. In diesem Zusammenhang ist auch von Systemen abzusehen, die eine Obergrenze für die Anzahl von Verarbeitungsprozessen festlegen (z.B. 1/Sekunde, 1000/Monat, etc.). Solche Systeme sind oftmals extern gehostet, d.h. die Anwendung läuft auf Servern des Anbieters und nicht lokal.

Ebenfalls im Zusammenhang mit der Einbettung des Projektes in die Hochschulbibliothek, ist die Verfügbarkeit von Sprach- und Klangmodellen zu sehen. Historisch begründet, verfügen die meisten NLP-Anwendungen über einen englischen Sprachkorpus. Dagegen lässt sich Deutsch seltener finden. Bei Betrachtung der studentischen Zusammensetzung der TH Wildau, zeigt sich, dass etwa 24,9%, also knapp ¼, der Studenten aus dem Ausland stammen. Entsprechend sind ca. 75% einheimische Studenten ansässig. Aufgrund der Schwierigkeit, alle an der TH Wildau gesprochenen Muttersprachen innerhalb des zeitlichen Rahmens einzubeziehen, sollte zumindest Englisch zur Verfügung stehen, da es weltweit i.d.R. als erste Fremdsprache unterrichtet wird und sowieso quasi stets als erstes softwareseitig Einzug findet. Mit der gewaltigen Mehrheit von ¾ aller Studenten, ist jedoch auch ein deutscher Sprachkorpus von besonderem Vorteil und wird daher prinzipiell als Notwendigkeit angesehen.

Zusammenfassend ergeben sich folgenden Rahmenbedingungen:

- Gewährleistung der Kompatibilität zwischen Roboter und NLP-Anwendung
- Verfügbarkeit und Kooperation sämtlicher NLP-Bestandteile im Ganzen
- Vermeidung von Kosten, insbesondere laufende (Lizenz-)Kosten
- Verfügbarkeit eines englischen und eines deutschen Sprachkorpus

3.3 Technische Voraussetzungen

Das bereits vorgestellte Robotersystem Pepper schränkt die Möglichkeiten der Technikwahl für die NLP-Anwendung im Vorfeld um einige Punkte ein. In diesem Fall sind damit vor allem die softwaremäßigen Kriterien gemeint.

Die Vorstellung des Roboters im Abschnitt *Robotersystem „Pepper“* zeigte auf, dass als Programmiersprache Python, C++ oder JavaScript vorliegen sollte, damit das Client-Modul direkt auf dem Roboter ausgeführt werden kann. Da JavaScript nicht nativ auf dem Pepper-System läuft, sondern innerhalb eines Browsers ausgeführt werden muss, wird es für die weiteren Betrachtungen vernachlässigt.

Mit der Wahl der Programmiersprache einhergehend, ist die Verfügbarkeit einer Schnittstelle in dieser, genannt Application Programming Interface (API). Durch diese können alle Funktionen des Client-Moduls der NLP-Anwendungen nativ verwendet werden. Die API und auch die gesamte NLP-Anwendung sollten außerdem dringlichst über eine ausführliche Entwickler-Dokumentation verfügen, um ein zügiges und korrektes Voranschreiten bei der Nutzung des NLP-Systems gewährleisten zu können.

Obwohl der Pepper-Roboter bereits über eine Spracherkennung verfügt, scheint diese nicht für das hiesige Szenario verwendbar zu sein, denn es werden nur diejenigen Wörter und Sätze erkannt, die im Vorfeld dem System manuell bekannt gemacht wurden. In einem natürlich sprachlichen Dialog ist es jedoch unmöglich alle Sprachvariationen im Vorfeld abzudecken. Das ALDialog-System scheint sich dieser Problematik anzunehmen, scheint zeitgleich jedoch schnell an Grenzen zu stoßen, den Erkenntnissen aus dem Ist-Zustand folgend. Die Abdeckung der Spracherkennung durch die NLP-Anwendung ist daher als wichtiger Vorteil anzusehen[27].

Im vorherigen Abschnitt wurde ein mögliches Hosting des gesamten NLP-Systems auf externer Seite, also bei einem Drittanbieter, erwähnt, welches mit Kosten verbunden sein kann. Es ergibt sich in diesem Zusammenhang auch ein technischer Aspekt: Das System ist in diesem Fall komplett vom Drittanbieter abhängig und gestörte

[27] Prinzipiell sollte eine NLP-Anwendung sowieso stets über Spracherkennung oder Texteingabe verfügen, wobei der Einfachheit halber i.d.R. häufiger die textbasierte Variante verwendet wird.

Datenverbindungen würden zwangsläufig zum Ausfall des Systems führen. Es ist daher von Vorteil, wenn das System entweder gänzlich oder zumindest in eingeschränkter Weise auch bei fehlender Internetverbindung einen Dialog aufrecht halten kann. Bestenfalls lässt sich die gesamte NLP-Anwendung lokal aufsetzen und betreiben.

Zusammenfassend ergeben sich die folgenden zu beachtenden Aspekte:

- Verfügbarkeit einer API zur nativen Verwendung der NLP-Anwendung
- Client-API in C++ oder Python
- Ausführliche und komplette Entwickler-Dokumentation
- Verfügbarkeit der Spracherkennung durch die NLP-Anwendung
- Offline Verfügbarkeit der NLP-Anwendung (bestenfalls durch lokalen Betrieb)

3.4 Funktionale Anforderungen

Im Folgenden sollen vom System bereitzustellende Funktionen und Services festgelegt werden. Es geht dabei um die Frage „Was soll das System hinsichtlich Logik, Dynamik und Statik leisten?“ [71 S. 455ff]. Die Antwort auf diese Frage ergibt sich aus den theoretischen Grundlagen, gemachten Absprachen mit den Betreuern dieser Arbeit und zum Teil aus der bereits existierenden Implementierung, vorgestellt in der Ist-Analyse.

Zur besseren Übersicht der Frage nach dem *„was“*, wird die Antwort darauf in Form einer Tabelle gegeben. Die *Identifikationsnummer* (*ID*) dient dabei der eindeutigen Abgrenzung bzw. Zuordnung der einzelnen Funktionen. Die nachfolgende Spalte Anforderung beschreibt die Funktion selbst und die abschließende Spalte Relevanz macht die Gewichtung kenntlich. Sie kann mit „*muss*“, für eine zwingend notwendige Funktion, ohne die das System nicht vollständig funktionsfähig ist, oder mit „*kann*“ für eine optionale Anforderung, die zwar wünschenswert und sinnvoll erscheint, aber nicht zwingend notwendig für die Funktionsfähigkeit ist, belegt werden.

Sämtliche Anforderungen sind zusätzlich in der Datei *Anforderungen.xlsx* im Ordner-Verzeichnis *~/Anf/* auf der beiliegenden CD mit den Spalten ID, Typ, Beschreibung, Relevanz, Verfassungsdatum, Änderungsdatum, Änderung, Referenz und Bemerkung eingefügt.

Tabelle 4: Funktionale Anforderungen des NLP-Systems.

ID	Anforderung	Relevanz
/SW0010/	Die Eingabe von Sprache ist verbal möglich.	muss
/SW0020/	Bei der verbalen Eingabe von Sprachdaten erfolgt eine Filterung von Störgeräuschen.	muss
/SW0030/	Die Eingabe von Sprache ist schriftlich möglich.	muss
/SW0040/	Eingegebene Sprache wird in Echtzeit verarbeitet.	muss
/SW0050/	Die Ausgabe von Sprache ist verbal möglich.	muss
/SW0060/	Die Ausgabe von Sprache ist schriftlich möglich.	muss
/SW0070/	Das System ermöglicht eine Dialogkommunikation.	muss
/SW0080/	Das System ermöglicht eine multimodale Dialogkommunikation.	kann
/SW0130/	Daten werden hinsichtlich phonologischer Eigenschaften analysiert.	muss
/SW0140/	Das phonologische Modell ist konfigurierbar.	muss
/SW0150/	Daten werden hinsichtlich morphologischer Eigenschaften analysiert.	muss
/SW0160/	Das morphologische Modell ist konfigurierbar.	muss
/SW0170/	Morphologisch fehlerhafte Daten werden korrigiert.	kann
/SW0180/	Daten werden hinsichtlich syntaktischer Eigenschaften analysiert.	muss
/SW0190/	Das syntaktische Modell ist konfigurierbar.	muss
/SW0200/	Syntaktisch fehlerhafte Daten werden korrigiert.	kann
/SW0210/	Daten werden hinsichtlich semantischer Eigenschaften analysiert.	muss
/SW0220/	Das semantische Modell ist konfigurierbar.	muss
/SW0230/	Daten werden hinsichtlich situationsgebenden Eigenschaften analysiert.	muss
/SW0240/	Die kontextuelle Situation ist konfigurierbar.	muss
/SW0250/	Mehrdeutigkeiten werden aufgelöst.	muss

/SW0260/	Paradoxien werden aufgelöst.	muss
/SW0270/	Repräsentationsfragen werden aufgelöst.	muss
/SW0280/	Basierend auf analysierten Eigenschaften der Daten erfolgt eine Domain-Zuordnung.	muss
/SW0310/	Es besteht die Möglichkeit weitere Kommunikationskanäle in den Verarbeitungsprozess einzubinden.	kann
/SW0360/	Sämtliche Spracheingaben werden persistent gespeichert.	muss
/SW0400/	Das System ist in der Lage zu lernen, d.h. Machine-Learning-Ansätze einzubinden.	muss
/SW0410/	Das System ist manuell trainierbar.	muss
/SW0420/	Das System ist semiautomatisch trainierbar.	muss
/SW0430/	Das System ist vollautomatisch trainierbar.	kann

3.5 Nichtfunktionale Anforderungen

Entgegen funktionaler Anforderungen, behandeln die Anforderungen nichtfunktionalen Typs die Frage „Wie und mit welcher Qualität soll das System Leistungen erbringen?“ [71 S. 463ff]. Die Antwort auf diese Frage ergibt sich durch die selben Punkte, wie im letzten Abschnitt beschrieben.

Sämtliche nichtfunktionale Anforderungen sind zusätzlich in der Datei *Anforderungen.xlsx* im Ordner-Verzeichnis *~/Anf/* auf der beiliegenden CD mit den Spalten ID, Typ, Beschreibung, Relevanz, Verfassungsdatum, Änderungsdatum, Änderung, Referenz und Bemerkung eingefügt.

Tabelle 5: Nichtfunktionale Anforderungen des NLP-Systems.

ID	Anforderung	Relevanz
/SW0090/	Als Sprachmodell steht die englische Sprache zur Verfügung.	muss
/SW0100/	Als Sprachmodell steht die deutsche Sprache zur Verfügung.	muss
/SW0110/	Als Klangmodell steht die englische Sprache zur Verfügung.	muss

/SW0120/	Als Klangmodell steht die deutsche Sprache zur Verfügung.	muss
/SW0370/	Sämtliche persistent gespeicherte Verbal-Spracheingaben liegen in einem Audioformat vor.	kann
/SW0380/	Sämtliche persistent gespeicherte Spracheingaben liegen in Textformat vor.	muss
/SW0390/	Sämtliche Spracheingaben sind in Form von Logs einsehbar.	muss
/SW0440/	Es fallen keine Kosten für die Beschaffung des Systems an.	muss
/SW0450/	Es fallen keine laufenden Kosten für die Nutzung des Systems an.	muss
/SW0460/	Die Benutzung des Systems ist unlimitiert, d.h. es gibt keine Grenze für die Anzahl der Verarbeitungs-Anfragen.	muss
/SW0470/	Das System stellt eine Application Programming Interface (API) für die Benutzung zur Verfügung.	muss
/SW0480/	Die API-Schnittstelle ist mit der Programmiersprache C++ oder Python nutzbar.	muss
/SW0490/	Der Quellcode der Implementierung ist einsehbar.	kann
/SW0500/	Für die Benutzung der Implementierung steht eine Entwickler-Dokumentation zur Verfügung.	muss
/SW0510/	Die Benutzung der Implementierung ist an eine "Open Source"-Lizenz gebunden.	kann
/SW0520/	Der Verarbeitungsprozess ist auch bei Ausfall der Internetverbindung möglich.	muss

3.6 Szenario

Der Einsatzzweck des NLP-Systems, als humanoider Roboterbibliothekar im Kontext einer Hochschulbibliothek, stellt zwar bereits eine Eingrenzung des Anwendungsfalls dar, kann aber im Zuge der einheitlichen Evaluierung enger betrachtet werden. Im Vordergrund steht der Erkenntnisgewinn über die zu evaluierenden NLP-Systeme hinsichtlich der prinzipiellen Funktionsweise, Qualität der Verarbeitung und generelle Vor- und Nachteile in der praktischen Benutzung.

Insbesondere für die einheitliche Evaluierung gilt es daher ein konkretes Szenario zu formulieren, aus dem sich spezifische Sprachkonstrukte für die Überprüfung bilden lassen. Als Ausgangspunkt dient die bereits im Unterkapitel Ist-Analyse vorgestellte FAQ-Anwendung. Im hiesigen Fall wird das Szenario wie folgt inszeniert:

Auch zum aktuellen Wintersemester begrüßt die TH Wildau wieder viele neue Studenten, die sich spätestens mit Beginn der Vorlesungsphase Gedanken machen müssen, wie sie an Quellen für die Ausarbeitung von Vorträgen, vertiefende Literatur zur Vor- und Nachbereitung der Vorlesungsveranstaltungen oder auch an generelle Materialien für Seminare gelangen. In den Einführungsveranstaltungen empfehlen die Dozenten dazu einen Blick in die Hochschulbibliothek zu verwerfen, die ein großes Angebot stellt.

Diesen Tipp haben auch Till und Sofia gehört, die beide neu an der TH Wildau eingeschrieben sind und gemeinsam Telematik im Bachelor studieren. In der Mittagspause überlegen sie sich, die Bibliothek in Halle 10 zu besuchen, um mehr Informationen über das Angebot der Bibliothek zu erhalten. Da die Vorlesungszeit gerade wieder begonnen hat, ist es sehr voll und alle Mitarbeiter am Empfang haben außerordentlich viel zu tun. Die beiden wollen schon fast wieder gehen, als ihnen der Pepper-Roboter im Eingangsbereich auffällt. Neugierig treten die beiden näher.

Weder Till noch Sofia wissen, wie man den Roboter bedient und welche Fähigkeiten dieser besitzt. Vom Aussehen des Roboters angetan, spricht Till den Roboter an. Dieser grüßt prompt freundlich zurück und erklärt allgemeine Fragen zur Benutzung der Hochschulbibliothek beantworten zu können. Dazu gehören:

- *Angebotene Dienstleistungen der Bibliothek und welche Voraussetzungen zur Nutzung dieser gelten.*
- *Verwendung von Geräten wie Drucker, Tablets, Computer, etc.*
- *Nutzung und Ausleihe von Medien und deren Fristen.*
- *Verfügbarkeit und Buchung von speziellen Arbeits- und Vortragsräumen.*
- *Möglichkeit zur Einbindung eigener Endgeräte ins Campus-Internet sowohl auf dem Gelände der Hochschule als auch von zu Hause.*

Till und Sofia sind erstaunt von den sprachlichen Fähigkeiten des Roboters und so stellt Sofia die erste Frage, in der sie wissen will, ob es Voraussetzungen für Benutzung der Bibliothek gibt. Der Pepper-Roboter kann diese Frage gewissenhaft beantworten, da es zu seinen Kernaufgaben gehört, solche Fragen zu beantworten.

Till fällt nun ein, dass Sofia und er noch den Antrag zum Anschluss ihrer Studentenwohnungen an das Campusnetz ausdrucken wollten. Daher fragt er Pepper,

ob man hier drucken kann. Pepper kann diese Frage positiv beantworten und erklärt die wichtigsten Schritte.

Da Sofia aus Kolumbien stammt und nicht jedes einzelne Wort versteht, unterbricht sie den Roboter und bittet Till um Wiederholung. Gerade als Till mit einer kurzen Zusammenfassung beginnen will, setzt der Roboter von selbst ein und beginnt die letzte Aussage zu wiederholen. Auch diese Fähigkeit besitzt er, stellen Till und Sofia erstaunt lachend fest.

Sofia und Till konnten ihre wichtigsten Fragen mit Peppers Hilfe klären und verabschieden sich nun von diesem. Die beiden haben ihre Anliegen während der Konversation ganz spontan gestellt und sich im Vorfeld keine spezielle Formulierung überlegt oder einen Befehlston angewandt. Die Fragen stellten sie ganz natürlich, als ob sie von Mensch zu Mensch sprechen würden.

Das hier formulierte Szenario ist auch in allgemein gefasster Form in der Datei *UserStories.xlsx* im CD-Ordner *~/Anf/* als sogenannte User Stories erstellt worden. Die User Stories beschreiben durchführbare Aktionen und Ergebnisse aus Benutzersicht. Ein Ausschnitt des Dokuments ist nachfolgend gegeben.

User Stories				
Als <Akteur>	**möchte ich <Beschreibung>**	**um <Ergebnis>**	**Datum**	**Kommentare**
Benutzer	den Roboter begrüßen	ein Gespräch zu beginnen.	12.12.2017	
Benutzer	die Funktionen des Roboters erklärt bekommen	diese zu nutzen.	12.12.2017	
Benutzer	den Roboter unterbrechen	seine Antwort zu beenden wenn ich meine Information schon erhalten habe oder die Frage falsch interpretiert wurde, etc.	12.12.2017	
Benutzer	die letzte Antwort nochmal hören	mich zu vergewissern alles richtig verstanden zu haben	12.12.2017	

Abbildung 22: Formulierte User Stories für das hiesige Szenario. [Quelle: Eigenmaterial]

4 Evaluierung

Nachdem im letzten Kapitel einheitliche Bewertungskriterien für einen Vergleich der recherchierten Software-Bibliotheken definiert wurden, gilt es nun, den tatsächlichen Nutzen bzw. die reale Leistungsfähigkeit dieser Software-Bibliotheken zu analysieren und miteinander zu vergleichen.

Die Evaluierung stellt in gewisser Weise das Herzstück dieser Arbeit dar. Basierend auf den bisherigen Erkenntnissen, sowohl bezüglicher theoretischer Grundlagen, als auch durch konkrete Anforderungen, soll nun die am besten passendste NLP-Lösung gefunden werden, mit der sich ein realer Prototyp implementieren lässt.

Mit der Auflistung möglicher Software-Lösungen, am Ende des Theorie-Kapitels, wurde erstmals deutlich, dass der Markt über eine Vielzahl von Lösungsansätzen verfügt. Die einzelnen Softwarehersteller haben dabei unterschiedliche Prioritäten gesetzt, wodurch sowohl Komplettlösungen, als auch einem Baukasten entsprechende Toolkits existieren, die lediglich einzelne Komponenten darstellen und selbstständig zu einer vollwertigen Anwendung programmiert werden müssen. Im ersten Schritt dieses Kapitels wird daher eine *Vorauswahl* vorgenommen, mit dem Ziel, die Menge der zu bewertenden NLP-Systeme, hinsichtlich der Rahmenbedingungen, einzuschränken.

Die detaillierte Betrachtung aller Systemkomponenten und das Zusammenspiel dieser, ist einer oberflächlichen Betrachtung vieler Lösungen vorzuziehen. Dies gilt auch deshalb, weil der Abgleich aller Anforderungen eine langwierige und intensive Arbeitsleistung darstellt. Dabei soll dieser Abgleich für Leser transparent geschehen, die Software-Lösungen daher zunächst vorgestellt werden, inklusive der groben Darstellung ihrer Funktionalitäten und Besonderheiten. Das Unterkapitel *Vorstellung der Software-Lösungen* übernimmt diese Aufgabe.

Sind die Software-Lösungen grob umrissen, kann anschließend der *Anforderungsabgleich* im gleichnamigen Unterkapitel vorgenommen werden. Eine Endauswahl, einzig basierend auf diesen Abgleich, ist aber wenig sinnig, da Fragen hinsichtlich Einrichtungsaufwand, Trainingsaufwand, Performance und Latenz nicht geklärt werden.

Nur eine testweise Verwendung, kann diese Punkte klären. Daher werden die aussichtsreichsten NLP-Lösungen im Unterkapitel *Performance und Latenz* getestet. Die Gesamtheit aller Bewertungsmaßnahmen dieses Kapitels ermöglicht abschließend die Auswahl einer Software-Lösung für den Prototyp.

4.1 Vorauswahl

Im Zuge der Recherche zur Auffindung von NLP-Software, entstand eine Liste mit über 20 Einträgen, die potentiell zur genaueren Betrachtung in Frage kommen. Die vollumfängliche Evaluierung aller Listeneinträge ist hierbei nicht möglich, da die Arbeit zum einen nicht den entsprechenden zeitlichen Spielraum bietet und zum anderen ein qualitativ hochwertiges Ergebnis angestrebt wird, welches auch bereits Fragen zur Implementierung, in Form eines Prototyps, klärt.

Diese Gründe machen es nötig, den zahlenmäßigen Umfang der betrachteten Softwarebibliotheken einzugrenzen. Folglich ist eine Vorauswahl durchzuführen. Als Kriterien dieser Vorauswahl bieten sich die im Kapitel *Anforderungsanalyse* definierten Rahmenbedingungen und technischen Voraussetzungen an. Somit ergeben sich insgesamt neun Kriterien für die Vorauswahl:

- Gewährleistung der Kompatibilität zwischen Roboter und NLP-Anwendung
- Verfügbarkeit und Kooperation sämtlicher NLP-Bestandteile im Ganzen
- Vermeidung von Kosten, insbesondere laufende (Lizenz-) Kosten
- Verfügbarkeit eines englischen und eines deutschen Sprachkorpus
- Verfügbarkeit einer API zur nativen Verwendung der NLP-Anwendung
- Client-API in C++ oder Python
- Ausführliche und komplette Entwickler-Dokumentation
- Verfügbarkeit einer Spracherkennung durch die NLP-Anwendung
- Offline Verfügbarkeit der NLP-Anwendung (bestenfalls durch lokalen Betrieb)

Diese neun Kriterien stellen wichtige Säulen für das auszuwählende NLP-System dar, denn sie kombinieren Einschränkungen durch die Hochschule bzw. Bibliothek mit Grenzen, die das Robotersystem setzt. Die Bewertung der zu erfüllenden Software-Eigenschaften wird durch folgende Bewertungsparameter bestimmt:

- *Ja* → Für ein erfülltes Kriterium.
- *Nein* → Für ein nicht erfülltes Kriterium.
- *Teilweise* → Wenn nur eine Untermenge des Kriteriums erfüllt wird.

Der erste Abgleich der Kriterien erfolgte dabei nach dem Prinzip eines Ausschlussverfahrens, d.h. ein nicht erfülltes Kriterium führte zur Disqualifizierung der Software-Bibliothek. Wie sich herausstellte, ist diese Vorgehensweise nicht möglich, da nach Beendigung der Überprüfung keine Software-Lösungen für die Endauswahl verblieben. Es existiert folglich keine Lösungsvariante, die alle Rahmenbedingungen und technische Voraussetzungen erfüllt.

Entsprechend des ersten gescheiterten Abgleiches der Vorauswahlkriterien, erfolgte der Entschluss zum Verzicht auf ein Ausschlussverfahren. Stattdessen wird ein

Punktesystem verwendet, bei dem zunächst alle Software-Lösungen keine Punkte besitzen, sich diese aber wie folgt verdienen können:

- *Ja* → Gibt einen Punkt (+1).
- *Teilweise* → Gibt einen halben Punkt (+0.5).
- *Nein* → Führt zu keiner Veränderung (+0).

Mit dem Kriterium *Entspricht NLP Anwendung* existiert ein Sonderfall, bei dem die Punktzahl doppelt gewertet wird. Durch dieses Vorgehen wird garantiert, dass vollständige NLP-Anwendungen eine höhere Gewichtung erhalten als Toolkits, die lediglich Software-Werkzeuge für die Erstellung eigener NLP-Anwendungen bereitstellen.

Zusammenfassend ergibt sich folgende Konstellation, die in *Abbildung 23* betrachtet werden kann. Die vollständige Tabelle lässt sich zusätzlich auf der beiliegenden CD, im Verzeichnis *~/Eval/Vorauswah.xlsx*, einsehen.

	Merkmal / Software Lösung	Sprache Englisch	Sprache Deutsch	Keine Kosten	Keine Verarbeitungs-Begrenzung	API Verfügbarkeit	API in C++ oder Python	Entwickler-Doku	Offline Nutzbarkeit	Enthält Speech Recognition	"Entspricht NLP Anwendung" (2 x Pkte.)	Bewertung
1	Amazon Alexa Voice Service	ja	ja	teilweise	teilweise	ja	ja	ja	nein	ja	nein	6
2	Amazon Lex	ja	ja	teilweise	teilweise	ja	ja	ja	nein	nein	ja	7
3	Apache OpenNLP	ja	ja	ja	ja	ja	teilweise	ja	ja	nein	nein	6,5
4	ChatterBot	ja	ja	ja	ja	ja	ja	ja	ja	nein	nein	7
5	ChatScript	ja	ja	ja	ja	ja	ja	ja	ja	nein	nein	7
6	CoreNLP	ja	teilweise	ja	ja	ja	teilweise	ja	ja	nein	nein	6
7	Dialogflow	ja	ja	teilweise	teilweise	ja	ja	ja	nein	nein	ja	7
8	Google Cloud Natural Language API	ja	ja	teilweise	teilweise	ja	ja	ja	nein	nein	nein	5
9	Houndify	ja	nein	teilweise	teilweise	ja	ja	ja	nein	ja	ja	7
10	IBM Watson Conversation	ja	ja	teilweise	teilweise	ja	ja	ja	nein	ja	ja	8
11	Microsoft Language Understanding	ja	ja	teilweise	teilweise	ja	ja	ja	nein	ja	ja	8
12	Natural Language Toolkit	ja	teilweise	ja	ja	ja	ja	ja	ja	nein	nein	6,5
13	NLP++	ja	nein	ja	ja	ja	nein	ja	ja	nein	nein	5
14	Pandorabots API	ja	ja	nein	nein	ja	ja	ja	nein	nein	ja	6
15	Recast.ai	ja	ja	teilweise	teilweise	ja	ja	ja	nein	nein	ja	7
16	spaCy	ja	ja	ja	ja	ja	ja	ja	ja	nein	nein	7
17	SparkNLP	ja	nein	ja	ja	ja	ja	ja	ja	nein	nein	6
18	TextBlob	ja	nein	ja	ja	ja	ja	ja	ja	nein	nein	6
19	wit.ai	ja	ja	ja	ja	ja	ja	ja	nein	ja	nein	7
20	Rasa.ai	ja	ja	ja	ja	ja	ja	ja	ja	nein	ja	9

Abbildung 23: Vorauswahl der NLP-Lösungen inklusive Bewertung. [Quelle: Eigenmaterial]

Zunächst lässt sich erkennen, dass keine Software-Lösung die vollen zehn Punkte erreicht hat. Vor allem der Wunsch nach Verwendung der Anwendung ohne aktive Internetverbindung (*Offline Nutzbarkeit)* und dem Vorhandensein einer Spracherkennungseinheit *(Enthält Speech Recognition*), sind nur sehr selten vorzufinden und bemerkenswerterweise nie gemeinsam.

Mehr als 50% der recherchierten Systeme entsprechen außerdem keiner vollwertigen NLP-Anwendung, sondern bieten häufig eher Toolkits an oder benutzen den Begriff des Natural Language Processings als Schlüsselwort für mehr Aufmerksamkeit bei potentiellen Kunden.

Positiv zu erwähnen, ist die generelle Verfügbarkeit einer API für alle Software-Bibliotheken, deren jeweilige Verwendung durch eine zugehörige Entwickler-Dokumentation ebenfalls garantiert wird[28]. Dabei verfügen die als NLP-Anwendung geltenden Systeme auch stets über eine API speziell für Python oder C++.

Die Verteilung der Kriterien *Keine Kosten und Keine Verarbeitungsbegrenzung* ist äußerst heterogen und lässt sich häufig nicht eindeutig in *ja/nein* einteilen, sondern ist vor allem durch *teilweise* gekennzeichnet. Der Begriff *teilweise* steht dabei i.d.R. für ein komplexes Zahlungssystem, bei dem die Verwendung der jeweiligen Software-Bibliothek durch verschiedene Lizenzen bestimmt wird. Beispielsweise sind solche Angebote nur 14 Tage kostenlos verwendbar und verlangen danach ein Abonnement, welches monatlich zu zahlen ist. Innerhalb des Abonnements darf nur eine bestimmte Anzahl von Verarbeitungsanfragen gestellt werden (dies entspricht der Verarbeitungsbegrenzung). Die Erhöhung der Anzahl von Verarbeitungsanfragen oder die Einbindung zusätzlicher Dienstleistungen, beispielsweise eine Stimmenanalyse, ist mit teureren Preisstufen verbunden.

Bei Betrachtung, für welche Software-Bibliotheken Abo-Modelle vorzufinden sind, fällt auf: Es sind entweder großer Unternehmen, wie Amazon, Google, IBM und Microsoft oder Firmen, die sich auf NLP spezialisiert haben (Houndify, Pandorabots API, Recast.ai[29]). Sie können auch häufig zu den vollwertigen NLP-Anwendungen gezählt werden. Dagegen tendieren kostenfreie Angebote eher zu Toolkits (Apache OpenNLP, CoreNLP, Natural Language Toolkit, spaCy, TextBlob) oder decken nur Teilbereiche ab (ChatterBot, ChatScript, NLP++, SparkNLP, wit.ai).

Der abschließende Blick auf die Sprachmodelle (*Sprache Englisch, Sprache Deutsch*) offenbart die durchgängige Unterstützung von Englisch. Modelle für die deutsche Sprache lassen sich bei kostengebundenen Angeboten durchweg finden, mit Ausnahme

[28] Über die Qualität der jeweiligen Dokumentation kann dabei noch keine Aussage getroffen werden.

[29] Recast.ai wurde Ende Januar 2018 durch das IT-Unternehmen SAP erworben. [111]

von Houndify. Bei den kostenfreien Angeboten gilt die Konzentration Englisch, es lassen sich in der Regel aber eigene Modelle, in einer belieben anderen Sprache, trainieren. Ein entsprechend wohldefinierter Sprachkorpus ist dafür jedoch zwingend Voraussetzung und die zusätzlich zu leistende Arbeit ist als sehr hoch einzuschätzen.

Zusammenfassend wird ein sehr heterogenes Bild hinsichtlich der erfüllten Rahmenbedingungen und technischen Voraussetzungen erkennbar. Dies lässt sich sowohl für die kostenfreien als auch kostengebundenen Software-Bibliotheken feststellen. Der Großteil aller Systeme bewegt sich bei sechs bis sieben von zehn möglichen Punkten in der Gesamtbewertung. Lediglich drei NLP-Systeme können sich absetzen, die da lauten: IBM Watson, Microsoft Language Understanding Service (LUIS) und Rasa.ai.

Prinzipiell könnte die vollumfängliche Evaluierung nun zwischen diesen drei Angeboten stattfinden. IBM Watson und Microsoft LUIS sind aber hinsichtlich der Vorauswahlkriterien komplett identisch. Auch der jeweils gewählte Systemaufbau ähnelt sich stark. Interessanterweise haben IBM und Softbank Robotics in 2016 eine Kooperation verkündet, die zum Ziel hat, Watson mit dem Pepper Roboter zu kombinieren [72]. Aus diesem Grund wird auf die weitere Betrachtung von Microsoft LUIS verzichtet. Somit findet die Evaluierung zwischen Rasa.ai und IBM Watson statt. Es existiert damit ein interessanter Kontrast zwischen der kostenfreien, Open-Source-Lösung Rasa.ai und der kostengebundenen, proprietären Software von IBM.

4.2 Vorstellung der Software-Lösungen

Nachdem sich im letzten Abschnitt IBM Watson und Rasa.ai im Rahmen der Vorauswahl gegen die restlichen NLP-Systeme durchgesetzt haben, werden die beiden Anwendungen nachfolgend genauer vorgestellt. Dabei stehen die prinzipielle Funktionsweise, Kostenstrukturen, Programmierschnittstellen inklusive Datenformate und die konkrete Verwendung der beiden Software-Lösungen im Mittelpunkt.

4.2.1 Das IBM Watson Framework

IBM Watson ist eine Softwarelösung des Unternehmens International Business Machines Corporation (IBM). Im Jahr 1911 in den USA gegründet, ist IBM heute einer der größten Hersteller von Software und Hardware im IT-Bereich mit Beschäftigten in unzähligen Ländern, deren Gesamtzahl auf etwas mehr als 380.000 Mitarbeiter beziffert werden kann [73]. Der Name Watson ist als Hommage an Thomas J. Watson zu betrachten, dem Gründer von IBM. [74 S. 78]

Wie bereits aus dem Kapitel *Ein Blick in die Vergangenheit - Woher kommt NLP?* bekannt, liegt der Ursprung von Watson in der Mitte der 2000er Jahre und der Idee, ein Computersystem zu entwickeln, welches in der Lage ist, bei der US-Quizshow *Jeopardy!* anzutreten. In der Quizshow werden verschiedenste Fragen aus insgesamt sechs Kategorien gestellt, wobei die Fragen in Form von Aussagen und die Antworten in Form von Fragen gestellt werden müssen [74 S. 61]. Ein Beispiel zum besseren Verständnis:

- Frage/Aussage: Er ist der Sänger der deutschen Musikband Rammstein.
- Antwort: Wer ist Till Lindemann?

Die Schwierigkeit liegt daher nicht in der bloßen Antwort, sondern zunächst zu erkennen, was überhaupt die Frage ist bzw. worauf die Aussage abzielt. Zusätzlich gilt es, möglichst schnelle Schlüsse zu ziehen, da weitere (menschliche) Kandidaten als Konkurrenz antreten.

Zur Erreichung dieses Ziels, musste Watson natürlich sprachlich formulierte Sätze verstehen lernen, Suchanfragen aus diesen generieren, abschätzen inwieweit die gefundenen Ergebnisse realistisch sind und eine Ergebniswahl treffen sowie letztendlich eine Antwort zusammensetzen oder alternativ auf die Beantwortung verzichten. Dieses System wird von den Ingenieuren IBMs DeepQA genannt. [75 S. 31]

Die Funktionsweise von DeepQA wird stets mit *„es werden mehr als 100 verschiedene Techniken in diesem Prozess eingesetzt“*, nur sehr grob umrissen [74 S. 68]. Die Architektur wird offiziell in einem abstrakten Schema dargestellt. Dieses kann in der nachfolgenden Abbildung betrachtet werden.

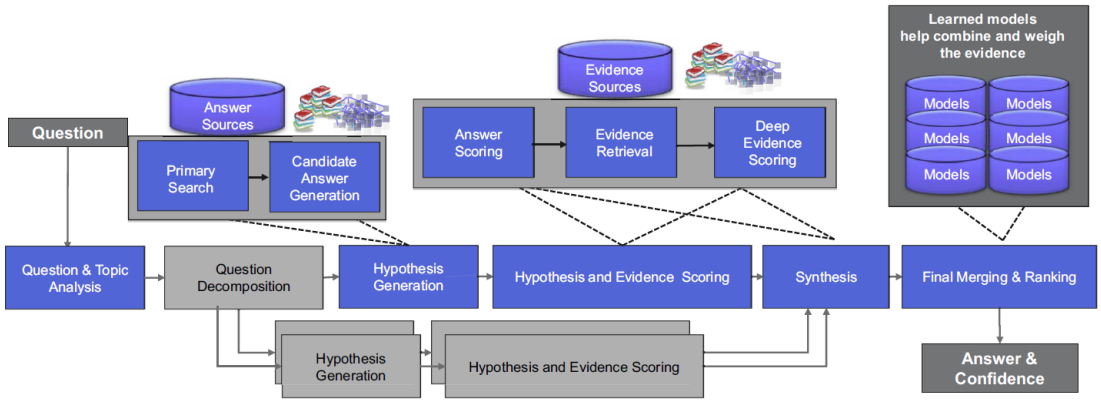


Abbildung 24: Funktionsablauf von DeepQA, dem Vorläufer von IBM Watson. [Quelle: [75 S. 32]]

Aus der Abbildung sowie zugehöriger Beschreibung lassen sich einige Erkenntnisse gewinnen, die wie folgt zusammengefasst werden können:

- *Question & Topic Analysis*: Im ersten Schritt wird der Fragetext analysiert und in einzelne Bereiche, d.h. in Worte, Phrasen und grammatische Strukturen zergliedert. [75 S. 34]
- *Question Decomposing:* Auf diesen Schritt basierend, erfolgen die NLP-Prozesse: Schlüsselwort-Extraktion[30], Informations-Extraktion[31] und Klassifikation der Frage- bzw. Antwort-Art[32]. [75 S. 34f]
- *Hypothesis Generation:* Basierend auf Schlüsselworte, werden relevante Passagen aus unstrukturierten[33] und strukturierten[34] Quellen gefiltert, um die Datenmengen massiv einzugrenzen. Die gefilterten Passagen werden ebenfalls einer Informations-Extraktion unterzogen und miteinander kombiniert, wodurch tausende von möglichen Antworten generiert werden. [75 S. 35f]
- *Hypothesis and Evidence Scoring:* Hierbei werden Beweise für jeden Antwortkandidaten gesammelt, durch die sich Wahrscheinlichkeiten für die richtige Antwort berechnen lassen. Beweise sind beispielsweise, ob Frage-Art mit Antwort-Art übereinstimmt, wie viele Schlüsselworte übereinstimmen, ob gleiche Relationen vorliegen, usw.[35] [75 S. 36f]
- *Synthesis:* Sämtliche Beweise werden für jeden Antwortkandidaten in Form eines Konfidenzwertes zusammengefasst. Dabei werden die Beweise gewichtet. Die Gewichtung erfolgt basierend auf alte Einschätzungen, nutzt entsprechend den Ansatz des Machine Learnings. [75 S. 37f]
- *Final Merging & Ranking:* Durch die Aggregation der einzelnen Beweise eines Antwortkandidaten, entsteht ein finaler Konfidenzwert für die Antwort. Die Gesamtheit aller Antworten wird anschließend anhand der Konfidenz sortiert, wodurch die wahrscheinlichste Antwort an erster Stelle steht[36]. [75 S. 38f]

[30] Schlüsselworte werden für spätere Suchabfragen benötigt.

[31] Mit der Informations-Extraktion werden die semantischen Rollen und zugehörige Relationen erkannt.

[32] Die Klassifikation bestimmt, ob die Frage durch Angabe von Person, Zeit, Ort, usw. beantwortet wird.

[33] Unstrukturierte Quellen sind u.a. Wörterbücher, Enzyklopädien, literarische Werke und Webseiten. DeepQA verwendet für die Suche vor allem Apache Lucene und Indri.

[34] Strukturierte Quellen sind u.a. Datenbanken, vordefinierte Modelle und Schemata. Welche Verwaltungssoftware DeepQA verwendet wird nicht genannt, aber die Suche erfolgt basierend auf SQL und SPARQL.

[35] Im weiteren Verlauf der DeepQA Beschreibung wird von „triggers a large number of analytics“ und „uses various strategies“ gesprochen. Diese vagen Aussagen werden nicht genauer erläutert und es erfolgt lediglich der Hinweis, dass entsprechende NLP-Ansätze verwendet werden. [75 S. 37]

[36] Es kann vorkommen, dass Antwortkandidaten übereinstimmen. In diesem Falle werden solche Kandidaten nochmals zusammengefasst. [75 S. 39]

Der eben beschriebene Ablauf und insbesondere die darin enthaltene Beweisführung wird massiv parallelisiert, so dass innerhalb von Sekunden mehrere Millionen Prozesse ausgeführt werden können. Ermöglicht wird dies durch die Verwendung des Apache Unstructured Information Management Architecture Frameworks (UIMA). [75 S. 39]

4.2.1.1 Watson Developer Cloud

Das von IBM geschaffene DeepQA-System wurde speziell für die Teilnahme an der *Jeopardy!*-Quizshow entwickelt und ist nicht für Dritte zugänglich. Die geleistete Arbeit und zugehörigen Forschungsergebnisse sollten aber, auch nach erfolgreicher Teilnahme an der TV-Sendung in 2011, weiterhin genutzt werden. Für IBM entstand mit Watson ein neues Geschäftsmodell, genannt Cognitive Computing, welches die Zielstellung hat, aus allen zur Verfügung stehenden Datenquellen Erkenntnisse zu gewinnen, Wissen aufzubauen sowie kontinuierlich zu lernen. Watson ist somit eine künstliche Intelligenz, die basierend auf strukturierten und unstrukturierten Daten wächst. [75 S. 11]

Der im DeepQA-System geschaffene Ablauf enthält Funktionen, die auch einzeln für verschiedene Anwendungsfälle verwendbar sind. Entsprechend wurden diese Funktionen zu vollständigen Diensten mit zugehöriger Schnittstelle weiterentwickelt und in IBMs Watson Developer Cloud Plattform eingefügt [75 S. 61ff]. Die Plattform bietet eine Vielzahl von Diensten an. Für die hiesige Arbeit gilt der Fokus den Services mit Bezug auf Watson. Eine Übersicht der angebotenen Dienste ist in *Abbildung 25* gegeben.

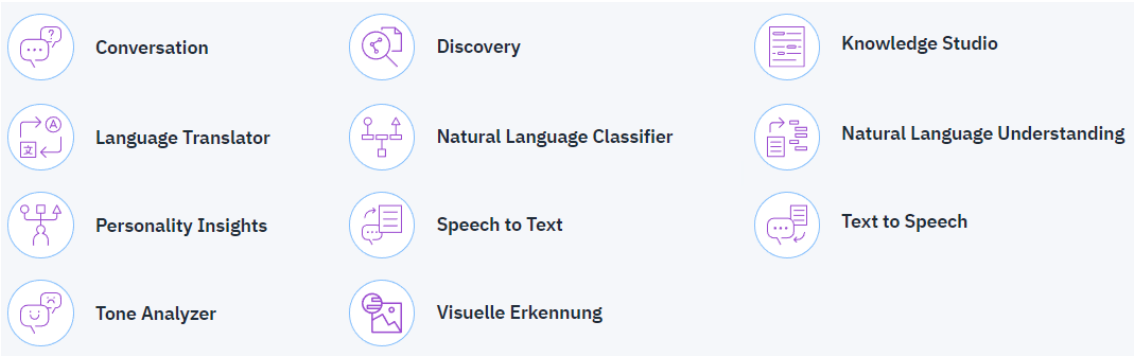


Abbildung 25: Angebotene Dienste von IBM im Kontext von Watson. [Quelle nach: [76]]

Entsprechend der Abbildung, existieren momentan neun verschiedene Dienste basierend auf Watson. Sie lassen sich wie folgt zusammenfassen:

- *Conversation:* Ermöglicht natürlich sprachliche Dialoge inklusive der Erkennung von Absichten und Entitäten aus den Benutzereingaben.

- *Language Translator:* Übersetzung von Texten zwischen Sprachen. Zusätzlich lässt sich die Sprache, in der ein Text verfasst wurde, automatisch erkennen.
- *Personality Insights:* Identifiziert Charaktermerkmale basierend auf Profildaten von Personen.
- *Tone Analyzer:* Erkennt die Stimmung eines Menschen hinsichtlich Emotion (Ärger, Ekel, Furcht, Freude, Trauer), sozialer Ausrichtung (offen, gewissenhaft, extrovertiert, introvertiert, freundlich), Sprachstil (analytisch, zuversichtlich, zögernd) und die emotionale Bandbreite. Die Erkennung basiert dabei auf Texte und nicht auf Audio-Eingaben, z.B. per Mikrofon.
- *Discovery:* Dient der Erkennung und Analyse von Mustern und Trends zur Entscheidungsunterstützung. Es können unstrukturierte und strukturierte Daten verwendet werden.
- *Natural Language Classifier:* Ordnet Text einem bestimmten vordefinierten Typ zu, mit dem Ziel einer Klassifizierung.
- *Speech to Text:* Umwandlung von verbalen Äußerungen in Text.
- *Visuelle Erkennung:* Zuordnung von Bildern hinsichtlich Orten, Entitäten, Gesichtern, Texten, unangemessenen Inhalten oder einem benutzerdefinierten Modell.
- *Natural Language Understanding:* Analyse von (längeren) Texten hinsichtlich Entitäten, Schlüsselworten, Kategorien, Stimmungen, Relationen und semantischer Rollen.
- *Text to Speech:* Umwandlung von Text in verbale Äußerungen mit Hilfe von Sprachsynthese.
- *Knowledge Studio:* Kein Dienst im klassischen Sinne, sondern ein Werkzeug zur Erstellung von benutzerdefinierten Modellen für einige der eben genannten Dienste. Sie können basierend auf das neu erstellte Modell trainiert werden. [76]

Dienste, basierend auf Watson, werden kontinuierlich von IBM weiterentwickelt, lassen sich aber nicht als Ganzes herunterladen und entsprechend lokal installieren. Somit sind jederzeit Änderungen an den Algorithmen und Schnittstellen möglich, die sich auf die Benutzung des Dienstes und somit der eigenen Anwendung auswirken können.

Sämtliche Dienste unterliegen außerdem einem Preisplan, der sich von Service zu Service unterscheidet. Folglich ist die Nutzung der Dienste mit Kosten verbunden. Zumindest im sogenannten Lite-Modus lassen sich nahezu alle Dienste innerhalb gewisser Grenzen, i.d.R. gibt es ein Maximum für Verarbeitungsanfragen, dennoch kostenfrei nutzen. Eine Ausnahme bildet hierbei der Natural Language Classifier Service, welcher immer mit Kosten verbunden ist. Die Abrechnung sämtlicher Dienste erfolgt mittels Kreditkarte.

4.2.1.2 IBM Watson Conversation

Durch die verfügbaren Watson Dienste lassen sich vielfältige und komplexe Anwendungen schaffen. Für die hiesige Evaluierung gilt der Fokus der Erkennung von Absichten und der Reaktion auf diese mittels einer natürlich sprachlichen Antwort. Unter diesen Gesichtspunkten sind nur die Dienste Conversation, Natural Language Classifier und Natural Language Understanding für diese Arbeit von Interesse.

Bei genauerer Betrachtung dieser drei Services, können Natural Language Classifier für die Klassifizierung und Natural Language Understanding für die Erkennung von Entitäten, Schlüsselworten, etc. vernachlässigt werden. Conversation enthält bereits die Klassifizierung und Entitäten-Erkennung, ergo werden die anderen beiden Services intern verwendet. Außerdem ist für die Zukunft geplant, Tone Analyzer, Speech to Text und Text to Speech in Conversation einzubinden[37] [75 S. 56]. Daher gilt der Fokus Watson Conversation.

Kosten

Von IBM im Rahmen der Watson Developer Cloud angebotene Services sind mehrheitlich mit Kosten und entsprechenden Preisplänen verbunden. Für den Conversation-Dienst gestaltet sich die Preisstruktur wie folgt:

Tabelle 6: Preispläne für die Verwendung des Dienstes IBM Watson Conversation. [Nach Quelle: [77]]

Plan	Funktionen	Preisstruktur
Lite	- 10.000 API-Aufrufe pro Monat. - Max. 5 Projekte. - Max. 100 Absichten. - Max. 25 Entitäten.	Kostenlos[38]
Standard	- Keine Begrenzung für API-Aufrufe pro Monat. - Max. 20 Projekte.	0,00188 € pro API-Aufruf

[37] Selbstverständlich können die Dienste innerhalb einer eigenen Applikation auch heute schon miteinander kombiniert werden.

[38] Die Überschreitung des Lite-Plans führt zur Ablehnung aller weiteren Anfragen für den restlichen Monat. Der Wechsel in den Standard-Plan ist aber möglich. Wichtig: Nach 30 Tagen ohne Nutzung erfolgt die Löschung aller Projekte! [77]

	- Max. 2000 Absichten. - Max. 1000 Entitäten.	
Premium	- Exklusive Instanz auf Watson Berechnungsebene. - Ende-Zu-Ende-Verschlüsselung aller Daten. - Weiteres individuell auszuhandeln.	Individuell zu verhandeln

Für die Evaluierung und den möglichen Prototyp sind die gegeben Grenzen des Lite-Plans ausreichend, weshalb dieser verwendet wird. Außerdem gilt weiterhin die Prämisse Kosten weitestgehend zu vermeiden.

Funktionsweise

Mit Watson Conversation lassen sich Dialogkonversationen innerhalb einer festgelegten Domäne zwischen Mensch und Computer erstellen. Im Gegensatz zu DeepQA handelt es sich um eine geschlossene Domäne, das heißt, die erfolgreiche Analyse ist abhängig von zuvor festgelegten Parametern, die durch Textbeispiele trainiert werden. [75 S. 64]

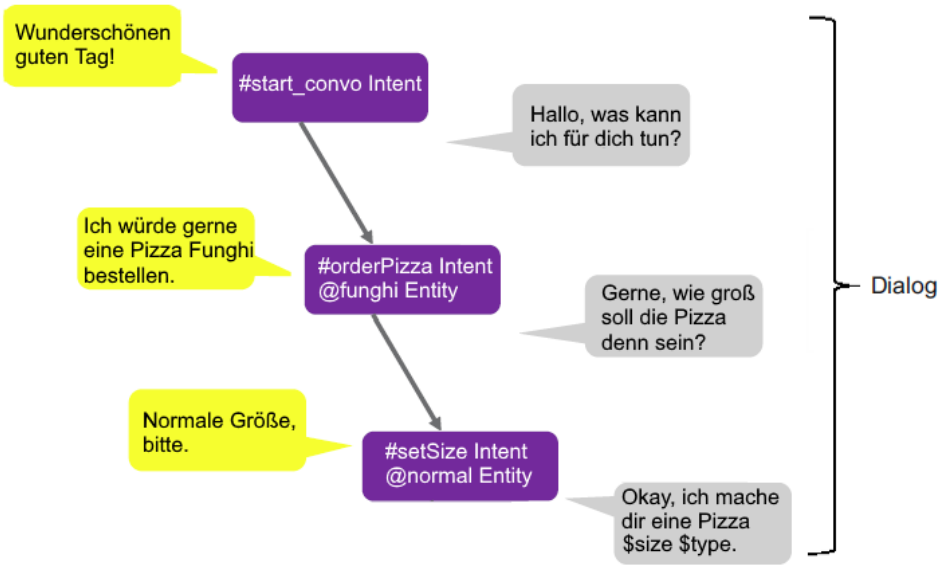


Abbildung 26: Watson-Schlüsselkonzepte in einer Konversation. [Quelle nach: [68 S. 65]]

Dabei basiert das Conversation-System im Prinzip auf vier Schlüsselkonzepte:

- *Intent*, beschreibt eine Absicht, die mit einer Aussage gemeint ist. Einem Intent ist folgendes Symbol vorangestellt: #.
- *Entity*, ist ein bestimmter Kontext-Typ innerhalb einer Absicht. Das zu verwendende Präfix ist: @.
- *Context,* enthält konfigurierbare Metainformationen. Verwendet wird hierzu das Präfixzeichen: $.
- *Dialog*, zur sprachlichen Koordination der Kommunikation. [53 S. 4ff]

Menschliche Äußerungen, in *Abbildung 26* gelb unterlegt, stellen Dialogeingaben dar, werden aber systemintern durch Intent und Entity abgebildet (lila gekennzeichnet). Bei entsprechendem Training können somit vielfältige sprachliche Variationen einer Absicht durch den Menschen formuliert und vom System erkannt werden, ohne dabei sämtliche Variationen vorher im Programmcode zu hinterlegen.

Die vom System zurückgegebenen Texte stellen die Dialogantwort dar (grau untermalt). Sie sind nicht trainierbar und werden daher im Vorfeld festgelegt. Eine Semi-Dynamik ist durch die Verwendung von Platzhaltern möglich, welche durch erkannte Entitäten oder ähnliches ersetzt werden können (durch $ in der Abbildung dargestellt). Parallel dazu ist die Angabe einer Antwort-Strategie möglich, bei der die vordefinierten Antworten zufällig oder in sequentieller Reihenfolge zurückgegeben werden.

Die Steuerung des Dialogs geschieht unter Verwendung und Verzweigung der Absichten und Entitäten. Entsprechend lässt sich ein Ablaufplan festlegen, welcher bis ins Detail durchgeplant werden kann. Ein solches Vorgehen bedeutet im Gegenzug aber wenig Handlungsspielraum innerhalb des aktuellen Gesprächskontextes und einen enormen Verwaltungsaufwand. Die Planung von Absichten, Entitäten und Gesprächsverläufen ist daher im Vorfeld der Verwendung ein enorm wichtiger Schritt.

Innerhalb einer Konversation entstehen immer auch kontextuelle Informationen, die für Dialogantworten oder dem eigenen Programmcode verwendbar sind. Eine Watson Dialogsitzung ist zunächst aber zustandslos, was bedeutet, der Gesprächsverlauf mit entsprechenden Metainformationen wird nicht gespeichert. Aus diesem Grund existiert als viertes Schlüsselkonzept das sogenannte Context-Objekt, in dem sich beliebige Metadaten des Dialogs zur Laufzeit speichern lassen ($-Symbol in der Abbildung). Folglich ist die Verwaltung des Kontextes manuell vorzunehmen.

API Übersicht

Mittels der Conversation-API lassen sich Projekte und darin enthaltene Absichten, Entitäten und Dialogmodule hinzufügen, bearbeiten und löschen. Die Verwendung der

API ist über zwei Wege möglich: Zum einen durch Verwendung einer für diverse Programmiersprachen[39] verfügbaren Client-API und zum anderen über eine spezielle von IBM entwickelten Oberfläche für Webbrowser.

Das Application Programming Interface ist in zehn Bereiche unterteilt. Diese haben folgende Namen und Funktionsbereiche:

- *Workspaces*, für das Handling einzelner Conversation-Projekte.
- *Intents*, enthält die möglichen Nutzer-Absichten in einem Workspace.
- *Entities*, repräsentiert Entitäten eines Intents in einem Workspace.
- *Values*, sind Zustände bzw. Werte, die eine Entity annehmen kann.
- *Synonyms*, sind alle bedeutungsgleichen Worte eines Entity-Values.
- *Examples*, entsprechen allen Trainingsdaten eines bestimmten Intents.
- *CounterExamples*, sind alle Trainingsdaten, die außerhalb der Domäne liegen.
- *Messages,* verarbeitet Textnachrichten in strukturierte Daten (Intents, Entities).
- *Dialog nodes*, enthält alle Dialogabläufe und wie diese verbunden sind.
- *Logs*, zeigt alle wichtigen Ereignisse (z.B. ein Konversationsverlauf) auf.

Für alle Bereiche stehen die Funktionen Auflistung, Erstellung, Bearbeitung sowie Löschung zur Verfügung. Alle Bereiche und deren Funktionen können über den sogenannten Watson API Explorer[40] in Form eines Demo-Projektes ausprobiert werden.

Die Benutzung der API ist an eine Authentifizierung gebunden, durch die eine Identifikations-Legitimation vorgenommen wird. Dazu ist die Angabe von Benutzername und Passwort nötig, welche durch das jeweilige Conversation-Projekt definiert werden. Zusätzlich kann bzw. sollte eine Versionsnummer angegeben werden, die festlegt, welche API-Version zu verwenden ist.

Alle API-Operationen sind in der Anzahl der Anfragen limitiert, das heißt, es kann nur eine gewisse Anzahl von Anfragen innerhalb einer bestimmten Zeitspanne gestellt werden. Jede Antwort einer Anfrage enthält daher drei Parameter:

- *X-RateLimit-Reset*, gibt an, wann die aktuelle Limit-Zeitspanne abläuft.
- *X-RateLimit-Remaining*, ist die Anzahl der verbleibenden Anfragen für die aktuelle Limit-Zeitspanne.
- *X-RateLimit-Limit*, zeigt die Gesamtzahl der maximal erlaubten Anfragen innerhalb einer Limit-Zeitspanne.

Die Limitierung der Anfragen ist für jede API-Operation unterschiedlich. Die drei genannten Parameter müssen aber nicht zwanghaft ständig ausgelesen werden, da bei

[39] Offizielle Client Bibliotheken stehen momentan für Python, C#, Java (und Android), Apex, JavaScript, Swift und Objective-C zur Verfügung. [130]

[40] Watson API Explorer Internetadresse: https://watson-api-explorer.mybluemix.net/apis/conversation-v1

Erreichung der Maximalgrenze die Antwort den HTTP-Fehlercode 429 enthält, welcher für genau diesen Fall verwendet wird.

Sämtliche API-Anfragen werden standardmäßig vom Watson-Dienst für Verbesserungen gespeichert *(„Logging is done only to improve the services for future users“* [78]). Laut IBM werden diese Daten nicht veröffentlicht. Es existiert zusätzlich die Möglichkeit, den Parameter *X-Watson-Learning-Opt-Out* auf den Wert *true* festzulegen. Dadurch wird IBM der Zugang zu den eigenen Daten verwehrt.

Für die Erstellung eines Watson Conversation Projektes gilt: Sämtliche Bereiche (Intents, Entities, Synonyms, etc.) müssen selbst angelegt und mit entsprechenden Beispielen bzw. Gegenbeispielen trainiert werden. Durch dieses Vorgehen entsteht im Ergebnis die eigene Anwendungsdomäne. Zeitgleich gibt es aber durchaus wiederkehrende Elemente, die übergreifend Verwendung finden. Aus diesem Grund existieren vordefinierte Entitäten, welche in allen Watson Conversation Projekten benutzt werden dürfen und lediglich manuell zu aktivieren sind. Diese System-Entitäten lauten:

- *@sys-currency*
  Dient der Erkennung von monetären Werten im Text. Dabei liegen stets der numerische Zahlenwert und die Währungseinheit vor.
- *@sys-date*
  Erkennt formulierte Datumsangaben und wandelt diese ins Jahr-Monat-Tag Format um (*„yyyy-MM-dd“).* Die Datumsangabe ist relativ zur UTC-Zeitzone, kann über die Kontext-Variable *$timezone* aber beliebig verändert werden.
- *@sys-time*
  Wie @sys-date, aber bezogen auf eine Zeitangabe. Das transformierte Format ist Stunde-Minute-Sekunde *(„HH:mm:ss“*).
- *@sys-location*
  Erkennt Orte. Diese sind zum Beispiel Kontinente, Länder, Städte, Bezirke, usw. Die @sys-location Entität befindet sich momentan in der Testphase und könnte eventuell entfernt oder stark verändert werden.
- *@sys-number*
  Zeigt gefundene numerische Werte des Textes. Dies bezieht sich auf Ganzzahlen, als auch auf Gleitkommazahlen.
- *@sys-percentage*
  Wie @sys-number, aber bezogen auf Prozentwerte.
- *@sys-person*
  Erkennt Namen von Personen. Diese Entität befindet sich momentan in der Testphase und könnte eventuell entfernt oder stark verändert werden.

Durch wohldefinierte Trainingsbeispiele und den vordefinierten Systementitäten, kann eine gute Erkennungsrate von Entitäten erreicht werden. Watson Conversation bietet als weitere Funktion Fuzzy Matching zur Erhöhung der Erkennungsrate. Mit dieser Funktionalität können auch nicht exakt übereinstimmende Entitäten-Werte zugeordnet werden. Dies geschieht durch die automatische Zerlegung der Entität in Morpheme und Stamm (vgl. Abschnitt *Wortstrukturierung (Morphology)* für Erläuterungen) sowie der automatischen Korrektur von Rechtschreibfehlern.

Datenformat

Sämtliche Watson Dienste, somit auch Watson Conversation, nutzen für den Client-Server-Datenaustausch JavaScript Object Notation (JSON). Bei Verwendung einer der Client-Bibliotheken, beispielsweise für die Programmiersprache Python, ist es nicht zwingend nötig, direkt mit dem JSON-Format zu arbeiten, da die Bibliothek die Erstellung des benötigten Formates erledigt.

Diese Aussage gilt zumindest für den Anfrage-Teil beim Datenaustausch, denn die Antwort liegt ausschließlich im JSON-Format vor und ist eigenständig in ein Objekt zu transformieren bzw. generell auszuwerten.

Das bereits vorgestellte Context-Objekt liegt ebenfalls im JSON-Format vor. Es lässt sich lesen und schreiben, so dass sich der Zustand des Kontextes dynamisch zur Laufzeit verändert. Eine Konversation ist aber prinzipiell zustandslos, bedeutet, das Context-Objekt geht nach dem Ablauf eines Anfrage-Antwort-Prozesses verloren. Zum Erhalt des Kontextes muss das Context-Objekt folglich bei jeder Anfrage mitgesendet werden und das für die gesamte Konversationsdauer.

Verwendung

In diesem Abschnitt wird aufgezeigt, wie sich die Einrichtung und Verwendung eines Watson Conversation Projektes gestaltet. Es werden die wichtigsten Schritte, von der Erstellung eines IBM Developer Cloud Accounts bis hin zum Testen eines Conversation Projektes bzw. Workspace, aufgezeigt[41]. Die Beschreibung erfolgt anhand der grafischen Benutzeroberfläche (GUI), die von IBM für diesen Zweck bereitgestellt wird. Es handelt sich bei dieser GUI um eine Webseite, die nahezu alle API-Funktionen grafisch abbildet.

[41] Die Menge an möglichen Schritten und Konfigurationen ist für eine effektive Einführung schlichtweg zu groß. Daher werden im *Anhang C* Web-Links und Buchempfehlungen für vertiefende Lektüre im Umgang mit IBM Watson Conversation bereitgestellt.

Zu Beginn gilt es, die Watson Developer Console[42] zu öffnen, dort nach dem Conversation Dienst zu suchen und diesen Service hinzuzufügen. Im Zuge dieser Aktion wird nach Zugangsdaten für ein bestehendes IBM-Konto gefragt. Alternativ kann ein neuer Account angelegt werden. Das IBM-Konto wird auch als IBM Bluemix Account bezeichnet.

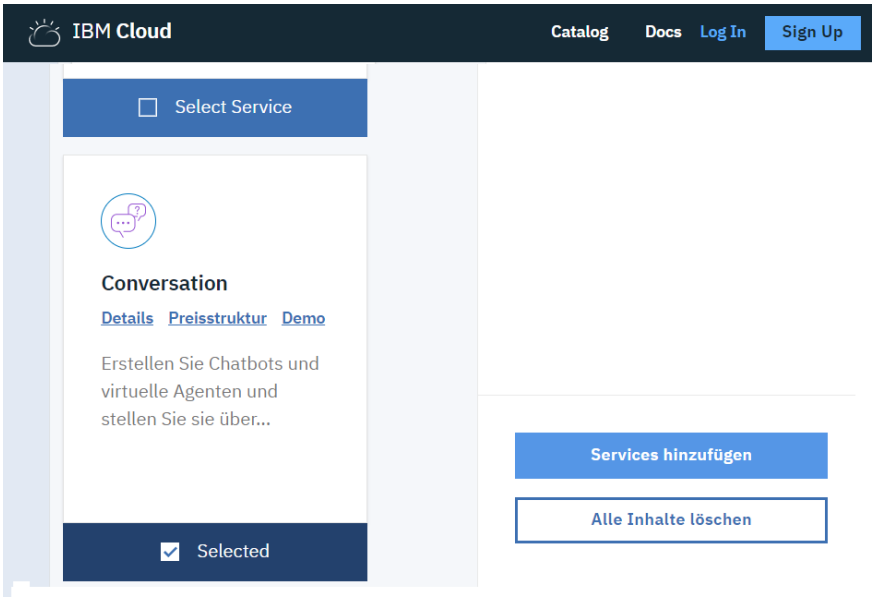


Abbildung 27: Auswahl des Conversation Services im Watson-Dienste-Katalog. [Quelle: Eigenmaterial]

Nach erfolgreicher Anmeldung bzw. Registration, ist im Anschluss ein Name für die Instanz des Watson Conversation Dienstes anzugeben und der Preisplan festzulegen. Bei Benutzung des Lite-Preisplans kann der Dienst im Anschluss sofort verwendet werden. Bei den Preisplänen Standard und Premium ist die Angabe einer Kreditkarte nötig. Für die hiesige Beschreibung wird das Projekt mit dem Lite-Preisplan angelegt.

Die nun zu sehende Webseitenoberfläche zeigt einige allgemeine Informationen über die angelegte Dienst-Instanz. Oben mittig befindet sich die Anzeige, wie viele API-Anfragen für den aktuellen Monat noch verfügbar sind (zur Erinnerung: Der Lite-Preisplan gestattet pro Monat maximal 10.000 dieser Anfragen). Linksbündig befinden sich einige Verwaltungsoptionen, über die z.B. der Preisplan angepasst werden kann oder auch Anmeldedaten (URL, Benutzername, Passwort) für die Benutzung des Service über die API einsehbar sind. Rechtsbündig ist ein Knopf mit der Aufschrift „Launch Tool“ zu sehen. Dieser ist anzuklicken, wodurch ein Wechsel zur Übersicht der eigentlichen Conversation Workspaces bzw. Projekte erfolgt.

[42] Link zur Watson Developer Console: https://console.bluemix.net/developer/watson/services

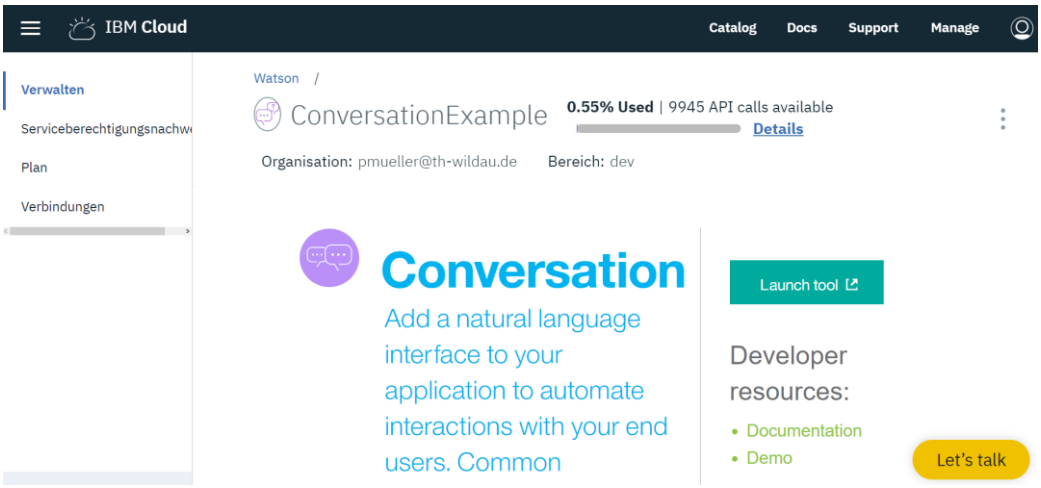


Abbildung 28: Übersicht der angelegten Watson Conversation Dienst-Instanz. [Quelle: Eigenmaterial]

Die Projekte-Übersicht wird auch als Dashboard bezeichnet. Über diese lassen sich neue Conversation-Projekte anlegen und bestehende Projekte verwalten. Im Rahmen des Lite-Preisplans können maximal fünf Workspaces zeitgleich existieren. Einer der fünf Projektplätze ist durch das Beispielprojekt *„Car Dashboard“* belegt. Dieses kann aber, wie auch selbst angelegte Projekte, gelöscht werden.

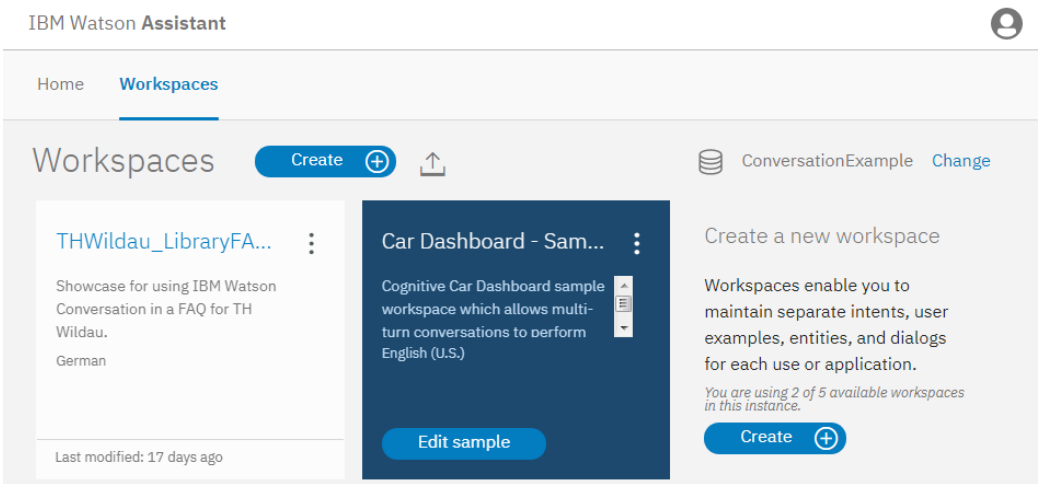


Abbildung 29: Übersicht der vorhandenen Conversation Projekte. [Quelle: Eigenmaterial]

Nach einem Klick auf den oben befindlichen *„Create“*-Knopf, kann das erste eigene Projekt erstellt werden. An dieser Stelle beginnt die Erzeugung der eigentlichen NLP-Anwendung. Zunächst werden ein Name und eine optionale Beschreibung festgelegt. Zusätzlich ist die menschliche Sprache festzulegen, die das System interpretieren soll.

Für die NLP-Anwendung sind nun Intents (Absichten), Entities (Entitäten) und der Dialogablauf zu erstellen. Die Gesamtheit dieser Elemente bildet die Domäne der Anwendung ab. Entsprechend muss die Festlegung von Werten genauestens geplant

werden und bedarf einer umfangreichen Analyse, bestenfalls basierend auf reale Anwendungsfälle und Verhaltensmuster der Benutzer.

Absichten werden durch das Klicken auf den *„Add intent“*-Knopf angelegt. Für jedes Intent ist zumindest ein Name festzulegen. Optional auch eine Beschreibung. Darauf basierend erscheint das Eingabefeld *„Add user examples“* in dem ein neues Trainingsbeispiel dieser Absicht hinzugefügt wird. Es kann sich dabei um einzelne Worte, Wortgruppen oder einen ganzen Satz handeln[43]. Letztendlich sind Varianten einzufügen, die Benutzer verwenden könnten, um ihre Absicht zu formulieren. In jedem Fall sind mehrere Beispiele zu formulieren, je mehr desto besser, aber zumindest um die 20 Stück. Die Menge an Trainingsbeispiele ist unbegrenzt, die Maximalanzahl der Absichten auf 100 limitiert.

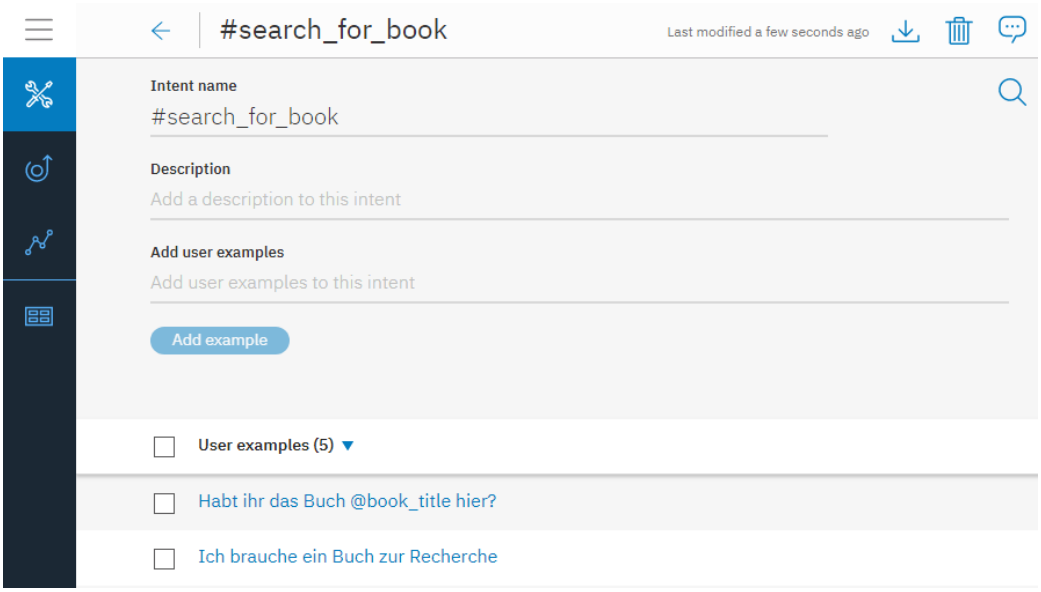


Abbildung 30: Erstellung eines neuen Intents mit Trainingsbeispielen. [Quelle: Eigenmaterial]

Neben den Absichten sind Entitäten festzulegen. Solche bestehen aus Personen, Objekte, Typen, Zustände, Länder, Dienste, etc. Letztendlich alles, worauf sich die Absichten beziehen können. Insgesamt können 25 Entitäten mit jeweils 100.000 Werten pro Entität angelegt werden. Eine Entität wird über den *„Add value“*-Knopf erstellt. Sie besitzt stets einen Namen und mindestens einen Wert *(„entity values“)*. Jedem Entitäts-Wert können über ein zugehöriges Eingabefeld, Synonyme hinzugefügt werden. Alternativ lassen sich Entitätswerte durch einen regulären Ausdruck beschreiben. In diesem Fall ist der Typ von *„Synonyms“* zu *„Patterns“* zu wechseln.

[43] Ein Intent-Beispiel darf maximal 1024 Zeichen lang sein.

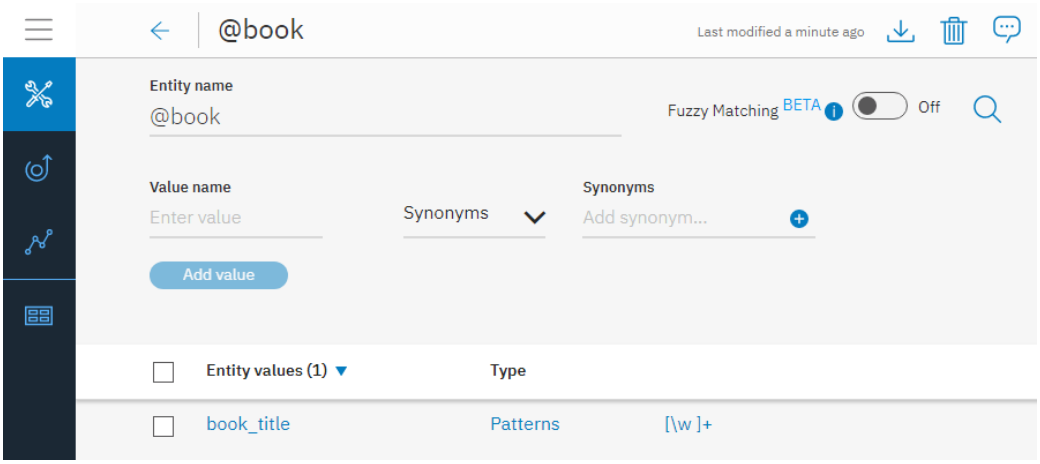


Abbildung 31: Hinzufügung einer Entity mit Values und Synonymen/Mustern. [Quelle: Eigenmaterial]

An dieser Stelle ist auch „*Fuzzy Matching*“ zur Verbesserung der Entitäten-Erkennung für die aktuell vorliegende Entität aktivierbar. Geringfügige Rechtschreibfehler werden dabei korrigiert und der Wortstamm der Entität herausgearbeitet, wie bereits im Abschnitt *API-Übersicht* erläutert.

Wie ebenfalls aus dem Abschnitt API-Übersicht bekannt, existieren mehrere System-Entitäten, die aktiviert bzw. deaktiviert werden können. Solche vordefinierten Entitäten können nicht bearbeitet werden, d.h. die Festlegung von Werten erfolgt durch IBM Entwickler. In der Regel sind diese Entitäten sehr robust, die korrekte Erkennung entsprechend gut. Es ist aber auch möglich, dass solche Entitäten in ihrer Funktion verändert oder komplett entfernt werden. Standardmäßig sind alle Systementitäten deaktiviert („*Status Off*“).

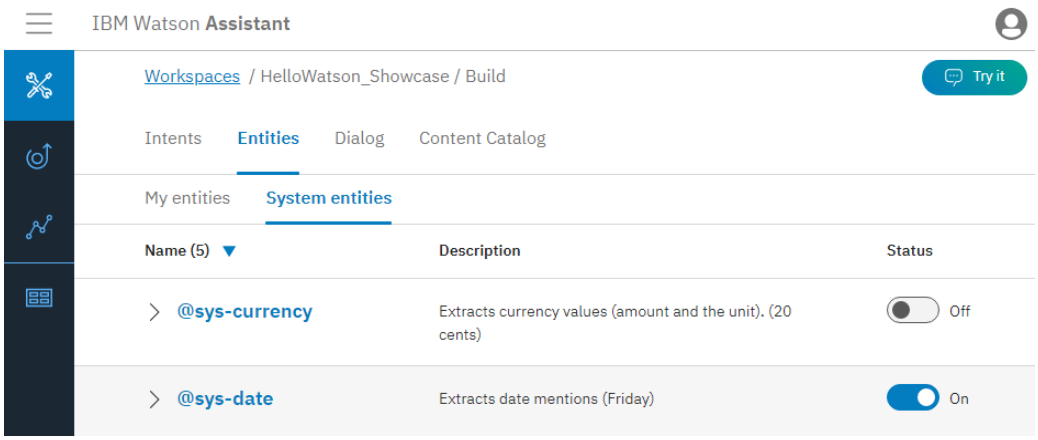


Abbildung 32: Ausschnitt der zur Verfügung stehenden Systementitäten. [Quelle: Eigenmaterial]

Basierend auf erstellte Intents und Entities, ist anschließend die Koordination der Dialogkonversation festzulegen. Dabei werden Intents mit Intents und Intents mit Entities in Beziehung gesetzt. Mit *„welcome“* und *„anything_else“* sind bereits zwei Intents vordefiniert. Die welcome-Absicht dient zur Begrüßung, noch bevor der Benutzer etwas geäußert hat, wodurch der Start einer Konversation auch durch das System ausgelöst werden kann und nicht zwanghaft auf eine vorhergehende Benutzereingabe angewiesen ist. Mit der anything_else-Absicht werden Äußerungen abgefangen, die nicht anderweitig zugeordnet werden konnten, was bedeutet, dass diese Aussage nicht zur Domäne der Anwendung gehört.

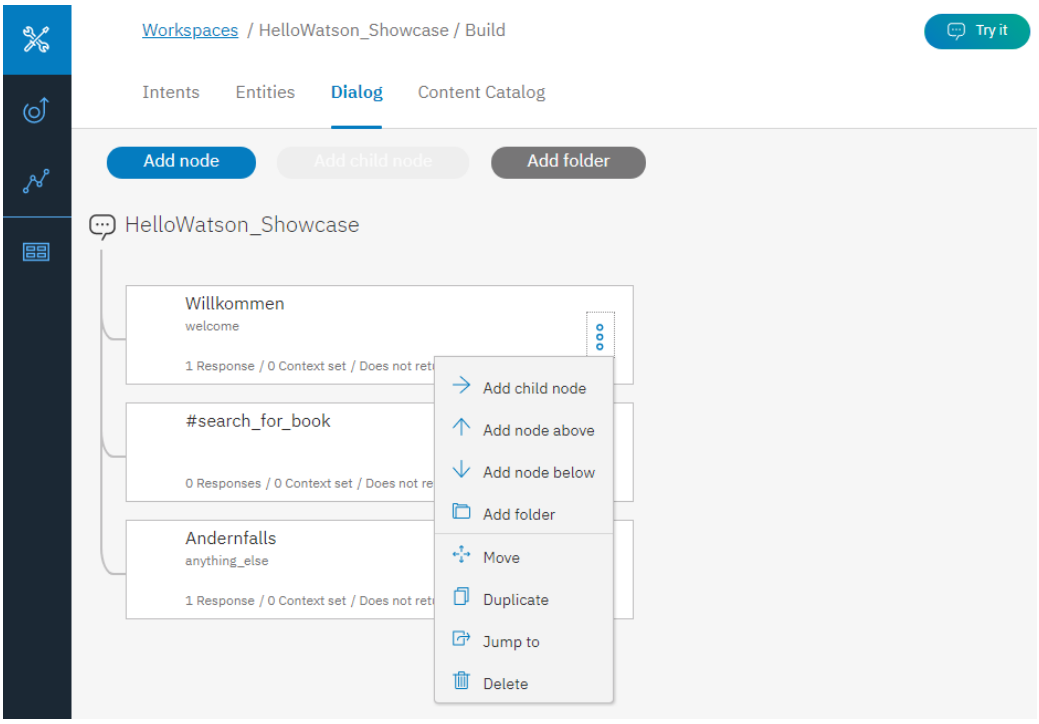


Abbildung 33: Knoten und Pfade zum Aufbau eines Dialogs in Conversation. [Quelle: Eigenmaterial]

Die Verknüpfung der Intents und Entities im Dialog entspricht Pfaden, die sich an Knotenpunkten in unterschiedliche Richtungen verteilen. Entsprechend werden Intents und Entities im Dialog als Knoten bezeichnet (bzw. in Englisch „node“). Ein neuer Dialogknoten wird per *„Add node“* hinzugefügt. Es können beliebig viele Pfade und Pfadtiefen definiert werden.

Knoten bestehen zumindest aus einer Bedingung und Dialogantwort. Die Bedingung muss erfüllt werden, damit der damit verbundene Pfad eingeschlagen werden kann bzw. die zugehörige Antwort zurückgegeben wird. Die Bedingung wird in der grafischen Oberfläche als *„If bot recognizes“* bezeichnet und lässt sich in Abhängigkeit zu Intents,

Entities und Kontext-Variablen bringen. Somit stehen Bedingungen für Daten bzw. Zustände, die durch Benutzereingaben zu erbringen sind.

Bei Verwendung der Funktion *„Slots“* lassen sich mehrere Datenzustände gleichzeitig innerhalb des aktuellen Knotens abfragen, wodurch die Anzahl von Knoten und somit die Komplexität des Pfades reduziert wird[44]. Wird ein Slot durch die aktuelle Benutzereingabe nicht bedient, so ist eine entsprechende Nachfrage im Feld *„If not present, ask“* definierbar. Slots können obligatorisch (*„Required“*) oder optional sein und lassen sich als Variable des Kontextes abspeichern (*„Save it as“*).

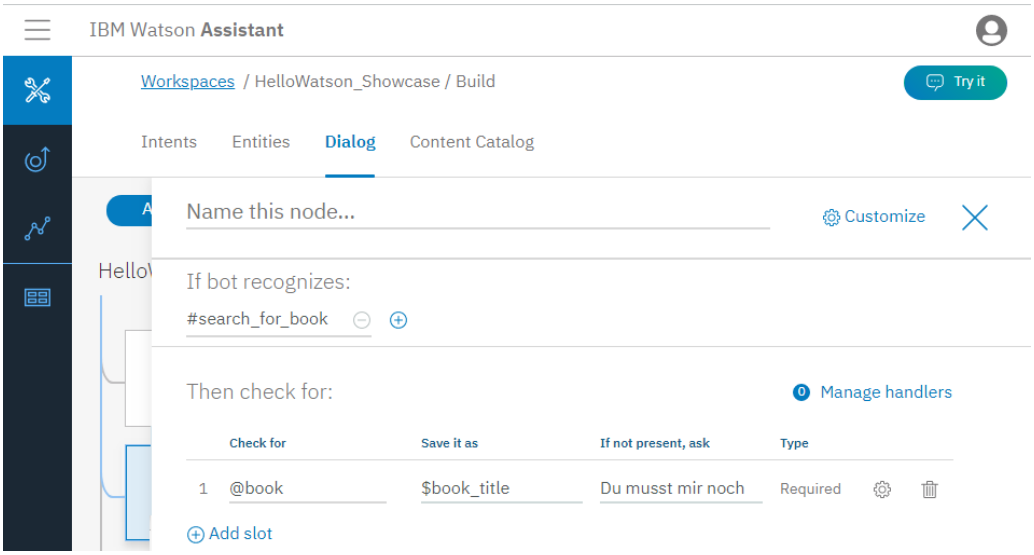


Abbildung 34: Inhalt eines Pfadknotens inklusive Slots und Variablen. [Quelle: Eigenmaterial]

Eine erfüllte Bedingung bzw. ein erfüllter Slot, ist durch eine Dialogantwort abzuschließen. Dazu wird im Bereich *„Then respond with:“* eine natürlich sprachliche Äußerung hinterlegt. Es ist möglich, mehrere Äußerungen hinzuzufügen, wodurch der Dialog variabler wirkt. In welcher Reihenfolge diese Äußerungen zurückgegeben werden, ist ebenfalls konfigurierbar: Mit den Typen Bedingung (*„If bot recognizes“*), sequentiell (*„sequential“*) und zufallsmäßig (*„random“*) stehen drei Modi zur Verfügung.

Nach der Definition eines Knotens und dessen Ablauf, ist final festzulegen, welche Aktion als nächstes ausgeführt werden soll (*„What to do next“*). In diesem Zusammenhang existieren drei Varianten: Auf eine neue Benutzereingabe warten (*„Wait for user input“*), direkt zum ersten Kindknoten des aktuellen Elternknoten

[44] Ohne die Verwendung von Slots ist es nötig für jeden Datenzustand einen Kindknoten anzulegen („Add child node“). Bis auf die Abhängigkeit zum Elternknoten, entsprechen sie einem normalen Knoten mit allen zugehörigen Funktionen. Slots sind in der Historie von Watson Conversation relativ neu und sollen das Eltern-Kind-Knotenprinzip weitestgehend ersetzen.

wechseln (*„Skip user input“*) oder zu einem bestimmten anderen Eltern- bzw. Kindknoten wechseln (*„Jump to“*).

Die Erzeugung der Dialogkonversation wird von Watson Conversation trainiert. Das Training findet automatisch statt und bezieht sich nur auf Intents, denn diese sind die einzig echten variablen Elemente einer Conversation Anwendung. Alle weiteren Funktionen sind, wenn überhaupt, pseudo-dynamisch, da sie an Bedingungen geknüpft sind oder mehrere Antwortmöglichkeiten definieren.

Dennoch sind sämtliche Schritte, ab der Definition von Intents, iterativ zu betrachten. Das bedeutet, die vorhanden Lognachrichten sind auszuwerten, um immer wieder Optimierungen am Dialogprozess vorzunehmen. Die korrekte Erkennung und eine effektive Dialogführung stehen dabei im Mittelpunkt der Verbesserungsbemühungen.

Verwendet wird die erstellte Dialoganwendung vor allem über die bereits vorgestellte API-Schnittstelle und einem zugehörigen Client-System. Zusätzlich kann der Dialog direkt im Webbrowser auf Funktionsfähigkeit getestet werden. Ein Klick auf *„Try it“* öffnet ein rechtsbündiges Fenster-Element, in dem der Dialog per Texteingaben bedient wird. Für jede Eingabe werden erkannte Intents, Entities und zugehörige Antworten angezeigt. Zusätzlich ist der aktuelle Kontextzustand einsehbar und manipulierbar.

4.2.2 Das Rasa Framework

Rasa ist eine Software-Lösung der Rasa Technologies GmbH mit Sitz in Berlin. Zurzeit arbeiten 15 Mitarbeiter für das Unternehmen. Der Ursprung der Software geht auf das Jahr 2015 zurück, in dem das damalige Kernteam einen Chatbot Prototyp entwickeln wollte und dabei erkennen musste, dass der Markt keine effektiven und kostenlosen Software-Produkte für die Erstellung eines solchen Produkts bereitstellte. [79] [80]

Auf dieser Erkenntnis basierend, entschieden die Rasa Gründer Alexander Weidauer und Alan Nichol, eine eigene Software-Lösung für diesen Zweck zu entwickeln. Als Manifest wurde festgelegt, die Software kostenfrei anzubieten und sie in eine Open-Source-Lizenz[45] einzubetten. Auch eine enge Zusammenarbeit mit freiwilligen Entwicklern gehört zu den Prämissen von Rasa. [80]

4.2.2.1 Rasa Core und Rasa NLU

Für das Framework existiert eine Unterteilung in die Module Core und NLU, welche zunächst einzelne Anwendungen darstellen, aber auch in Kombination verwendet werden können. Da unter Open-Source-Lizenz gestellt, lässt sich der gesamte Quellcode der Implementierung betrachten. Veröffentlicht ist der Quellcode auf der Plattform

[45] Lizenz-Inhalt von „Apache Licence 2.0“: https://www.apache.org/licenses/LICENSE-2.0

GitHub[46]. Dabei lässt sich feststellen, dass Rasa kontinuierlich weiterentwickelt wird und in regelmäßigen Abständen neue Versionen erscheinen[47]. Rasa ist, im Gegensatz zu IBM Watson Conversation, prinzipiell kein Online-Service und kann daher, in einer der veröffentlichten Versionen, vollständig heruntergeladen und lokal installiert werden. Auch Modifikationen am bestehenden Programmcode sind möglich.

Kosten

Prinzipiell ist Rasa kostenlos, d.h. alle Funktionalitäten sind frei verfügbar und an keine Preispläne gebunden. Ergänzend existiert Rasa Platform, ein Dienst, welcher die Arbeit im Team erleichtern soll und automatisierte Datensynchronisation sowie automatisierte Tests ermöglicht. Die Kosten für diesen Dienst sind individuell mit Rasa auszuhandeln und entsprechen einem Abonnement. [81]

Funktionsweise

Wie bei Watson Conversation, lassen sich mit Rasa Dialog-Konversationen innerhalb einer geschlossenen Domäne zwischen Mensch und Computer erstellen. Ebenfalls Watson Conversation gleichend, wird basierend auf Beispieldaten trainiert, ein Dialogablauf festgelegt und über individuell zu bestimmenden Metadaten, ein Kontext erzeugt.

Rasa NLU bzw. Rasa Natural Language Understanding übernimmt dabei die Aufgabe, natürlich sprachlich formulierten Text in Absichten und Entitäten zu strukturieren. Damit zusammenhängend, wird mittels Rasa NLU auch das Training von Beispielsätzen zur Erkennung dieser Absichten und Entitäten realisiert. [82]

Durch Verwendung von Rasa Core, werden die strukturierten Daten mit zugehörigen Aktionen verbunden. Vor allem der vordefinierte Aktionstyp Text ist dabei interessant, da durch diesen eine Antwort im Dialog gegeben wird. Mit dem Typ Custom lassen sich eigene Aktionen für die Generierung von Antworten festlegen, beispielsweise ein Datenbankaufruf zur Ermittlung relevanter Daten. Aktionen können miteinander verknüpft werden, bedeutet, das Ergebnis einer Aktion ist von nachfolgenden Aktionsprozessen verwendbar. [83] [84]

Rasa NLU und Rasa Core bieten zusätzlich die Möglichkeit, entwickelte Applikationen ohne eine umfangreiche eigene Programmierung als Server laufen zu lassen. Zu diesem Zweck steht eine vordefinierte API zur Verfügung. Durch diese API ist es außerdem möglich, in einer beliebigen Programmiersprache mit dem Rasa-Server zu

[46] Links zum Quellcode: https://github.com/RasaHQ/rasa_nlu und https://github.com/RasaHQ/rasa_core

[47] Als Indikator dieser Einschätzung wurde die Häufigkeit von Commits und neuen Release-Varianten verwendet.

kommunizieren. Standardmäßig wird Rasa aber in Python entwickelt und definiert eine zugehörige Python Client-Bibliothek für die Entwicklung eigener Anwendungen.

Zusammen bilden Rasa NLU und Rasa Core Schlüsselkonzepte analog zu Watson Conversation ab, folglich also: Intent, Entity, Context und Dialog. Die Funktionen der Schlüsselkonzepte sind ebenfalls gleich und werden daher nicht erneut beschrieben. *Abbildung 35* dient zur Verdeutlichung des Zusammenspiels zwischen den beiden Rasa Komponenten. Zur Erinnerung: gelb ist eine Benutzereingabe, lila kennzeichnet die systeminterne Datenstruktur und grau stellt die Antwort des Systems dar.

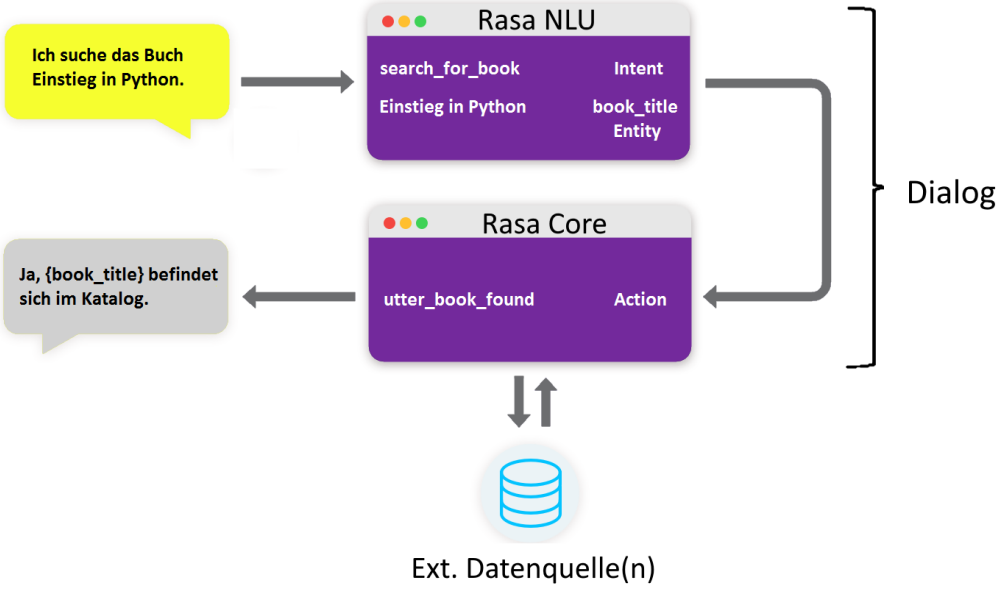


Abbildung 35: Rasa Komponenten und Schlüsselkonzepte in einer Konversation. [Quelle nach: [94]]

Ein großer Unterschied zwischen Watson Conversation und Rasa ist die Methodik zur Bestimmung einer Antwort bzw. Action. In Watson Conversation werden Antworten und Aktionen stets im Vorhinein festgelegt und durch den Ablaufplan verzweigt (Stichwort if/else). Bei Rasa werden Antworten ebenfalls im Vorfeld festgelegt, aber Aktionen sind trainierbar und basieren auf einem Wahrscheinlichkeitsmodell. Das hat zur Folge, dass Rasa bei der Auswahl einer Antwort einberechnet, welche Antwortvorlage aktuell am sinnvollsten ist. [84] [85]

Ein Beispiel: Es sind zwei Antwortvorlagen definiert, eine mit Platzhalter für eine bestimmte Entität und eine ohne Platzhalter. Später macht der Benutzer im Dialog eine Aussage, bei der die Anwendung keine Entitäten erkennt. Folglich ist es sinnvoller, also wahrscheinlicher, die Antwortvorlage ohne Platzhalter-Element zu verwenden. Bei entsprechenden Trainingsbeispielen, kann als weitere Aktion, die Nachfrage nach der benötigten Entität erfolgen. Dieser Prozess findet ohne Verwendung von

Verzweigungen und Bedingungen auf Programmierebene statt. Dadurch werden festprogrammierte Prozessflüsse vermieden und der Dialog dynamischer bzw. variabler. Auf der anderen Seite bedeutet dieses Vorgehen aber einen weiteren Trainingsschritt, der eine hinreichende Menge von Beispieldaten voraussetzt. Es werden nämlich nicht nur Eingaben, sondern auch Ausgaben trainiert, wodurch sich die Menge an Trainingsdaten mindestens verdoppelt. Letztendlich besteht weiterhin ein hoher Verwaltungsaufwand, wie bei dem Ablaufplan von Watson Conversation.

Das Wahrscheinlichkeitsmodell entsteht durch Machine Learning. Rasa verwendet dabei standardmäßig Keras[48] als abstrakte Schnittstelle und Tensorflow[49] als konkretes Machine Learning Framework. Rasa fasst dies unter dem Begriff Policy zusammen. Die vordefinierte Framework-Strategie lässt sich theoretisch durch eine andere Implementierung vollständig austauschen. [86] [87]

Die Herstellung und Verwaltung des Kontextes erfolgt bei Rasa durch sogenannte Slots, die letztendlich, in ihrer Gesamtheit, ein Kontext-Objekt darstellen. Slots werden im Vorfeld festgelegt und automatisch gesetzt, wenn Slot-Name und Entitäten-Name übereinstimmen. Stimmen diese beiden Namen nicht überein, so lassen sich Slot-Werte programmatisch in den bereits vorgestellten Actions setzen bzw. auslesen. Die Slot-Werte können zusätzlich in den Dialog-Antworten verwendet werden und ersetzen vordefinierte Platzhalter. [83] [84]

API Übersicht

Durch die Einteilung in die Module NLU und Core, existieren gleichermaßen auch zwei verschiedene APIs. Die Verwendung dieser, ist standardmäßig mittels Python vorgesehen. Alternativ können die Module als Server gestartet und mittels HTTP-Endpunkten angesteuert werden. Zum Zeitpunkt der hier vorliegenden Arbeit befinden sich die Servermodule aber noch in einer Testphase.

Rasa NLU bietet vier API-Bereiche mit folgenden Namen und Funktionen [88]:

- *Parse*, verarbeitet Textnachrichten in strukturierte Daten (Intents, Entities).
- *Train*, trainiert ein (neues) Projekt, basierend auf mitgegebenen Trainingsdaten.
- *Status*, zeigt an, welche Projekte momentan zur Verwendung verfügbar stehen.
- *Config*, enthält die Konfiguration der laufenden NLU Instanz.

Wie bereits erwähnt, übernimmt Rasa NLU die Umwandlung des Textes in Absichten und Entitäten. Mit dem Modul Rasa Core findet die eigentliche Dialogkonversation statt. Hierfür existieren die folgenden API-Bereiche [89]:

[48] Einführung in Keras: https://keras.io/

[49] Übersicht zu Tensorflow: https://www.tensorflow.org/get_started/

- *Parse*, verarbeitet Textnachrichten in Aktionen (z.B. Dialogantworten).
- *Continue*, teilt dem Server die Ausführung einer Aktion durch den Client mit.
- *Tracker*, ist das veränderliche Kontext-Objekt mit vordefinierten Slots.
- *Version*, zeigt an, welche Rasa Core Version aktiv ist.

Die Veränderung eines Projektes zur Laufzeit wird von Rasa nur rudimentär unterstützt. Bei Verwendung des Train-Endpunktes lässt sich nur die Erkennung von Intents und Entities bearbeiten. Der Dialogablauf ist zur Laufzeit nicht anpassbar. Die allgemeine Konfiguration eines Rasa-Projektes, u.a. hinsichtlich Authentifizierung und Logdaten, wird über Config-Dateien abgewickelt. Die Veränderung von Konfigurationsparametern zur Laufzeit ist nicht möglich. Die aktive Konfiguration lässt sich aber mittels Config-Endpunkt zumindest auslesen.

Neben den selbst zu definierenden Intents und Entities, existieren auch bei Rasa einige vordefinierte Entitäten. Diese werden, im Gegensatz zu Watson Conversation, nicht direkt vom System gestellt, sondern sind durch zwei Drittanbieter-Komponenten indirekt vorhanden. Komponenten sind während der Konfiguration zu aktivieren. [90]

Es handelt sich bei den Drittanbietern um spaCy[50] und duckling[51]. Die dadurch verfügbaren vordefinierten Entitäten-Extraktoren lauten wie folgt:

- *spaCy:PER*
  Enthält erkannte Namen von Subjekten bzw. Personen.
- *spaCy:LOC*
  Beschreibt erkannte Orte z.B. Kontinente, Länder, Städte, etc.
- *spaCy:ORG*
  Enthält erkannte Namen von Unternehmen und Organisationen.
- *spaCy:MISC*
  Sind (wortwörtlich) diverse Typen, z.B. Ereignisse, Produkte und Nationalitäten.
- *duckling:number*
  Beschreibt Zahlwerte, sowohl Ganzzahlen, als auch Gleitkommazahlen.
- *duckling:ordinal*
  Beschreibt Aufzählungswerte bzw. Listen.

[50] Laut spaCy Dokumentation gibt es noch eine feingranulare Variante, die den Typ MISC mit spezifischen Typ-Werten ersetzt. Rasa beschreibt diese Variante jedoch nicht und verwendet die grobgranulare Version. Die Anzahl der Typ-Werte ist auch von der verwendeten Sprache, beispielsweise Deutsch oder Englisch abhängig. An dieser Stelle wird sich auf Deutsch bezogen. [124]

[51] Die duckling Dokumentation definiert weitere Typen. Diese sind abhängig von der verwendeten Sprache, beispielsweise Deutsch oder Englisch. Hier wird sich auf Deutsch bezogen. [114] [115]

- *duckling:time*
  Erkennt formulierte Zeit- und Datumsangaben. Dieses werden ins Format Jahr – Monat – Tag – Stunde – Minute – Sekunde - Millisekunde („*yyyy-MM-ddT HH:mm:ss.SSS*“) gewandelt. Die Datumsangabe ist relativ zur UTC-Zeitzone.

Erkannte Entitäten besitzen immer auch ein Metadaten-Feld. In diesem ist der Name der Komponente, die diese Entität erkannt hat, enthalten. Die Entitäten-Erkennung kann im Übrigen auch durch eine komplett eigene Implementierung realisiert werden. [90]

Verwendung

Im hiesigen Abschnitt wird die Einrichtung und Verwendung eines Rasa Projektes anhand einer Linux Distribution (Ubuntu 16.04 LTS „Xenial Xerus“) aufgezeigt. Im Gegensatz zu Watson Conversation, ist Rasa stark individualisierbar, entsprechend sind viele Komponenten austauschbar. Es ist nicht zielführend, alle Varianten aufzuzeigen, da der Rahmen dieser Arbeit dabei gesprengt werden würde. Stattdessen erfolgt die Beschreibung anhand der Empfehlungen und Hinweise aus der Rasa Dokumentation. Auf die Möglichkeit des Komponentenaustausches, wird zumindest in Kurzform hingewiesen und entsprechende Links für weitere Informationen bereitgestellt.

Rasa ist in Python programmiert und bedarf daher einen Python-Interpreter auf dem ausführenden Computersystem. Dabei wird mindestens die Python-Version 2.7 vorausgesetzt [91]. Vor der Verwendung von Rasa NLU und Rasa Core, müssen die zugehörigen Software-Bibliotheken installiert werden. Dieser Schritt wird nachfolgend unter der Verwendung der Python Paketverwaltung PIP realisiert.

```
$ pip install rasa_core
$ pip install rasa_nlu
```

Abbildung 36: Kommandozeilen-Code zur Installation von Rasa mittels PIP. [Quelle: Eigenmaterial]

Nach erfolgreicher Installation, kann Rasa auf dem Computersystem, in Form einer Python Anwendung, verwendet werden. Zusätzlich sollten an dieser Stelle aber eventuell später benötigte Abhängigkeiten und Sprachmodelle gleichermaßen installiert werden. Beim späteren Austausch von Komponenten, sind somit keine weiteren Installationsschritte mehr nötig. Die Rasa-Entwickler haben für die Bekanntmachung der Abhängigkeiten, den Ordner alt_requirements angelegt, der auf GitHub bereitliegt. Es empfiehlt sich, das gesamte Projekt herunterzuladen, da somit auch der Quellcode auf dem eigenen Computersystem jederzeit eingesehen werden kann. Die Befehle dazu lauten[52]:

[52] Es handelt sich hierbei um einen Vorschlag zur Installation von Rasa und seinen Abhängigkeiten. Selbstverständlich können verwendete Zielordner, Skript-Methoden, etc. gewechselt werden.

```
$ cd ~/Downloads
$ git clone https://github.com/RasaHQ/rasa_core.git
$ git clone https://github.com/RasaHQ/rasa_nlu.git
$ cd rasa_nlu
$ pip install –r alt_requirements/requirements_dev.txt
$ cd ../rasa_core
$ pip install –r dev-requirements.txt
$ python –m spacy download en
$ python –m spacy download de
$ cd ..
$ mkdir mitie_models & cd mitie_models
$ wget https://github.com/mit-nlp/MITIE/releases/download/v0.4/MITIE-models-
v0.2.tar.bz2
$ wget https://github.com/mit-nlp/MITIE/releases/download/v0.4/MITIE-models-v0.2-
German.tar.bz2
$ tar –xvzf MITIE-models-v0.2.tar.bz2
$ tar –xvzf MITIE-models-v0.2-German.tar.bz
```

Abbildung 37: Download des Rasa-Quellcodes und optionaler Abhängigkeiten. [Quelle: Eigenmaterial]

Nun ist die eigentliche Einrichtung eines Rasa Projektes möglich. Es empfiehlt sich dazu, eine integrierte Entwicklungsumgebung (IDE) für die weiteren Schritte zu verwenden. Die IDE sollte dabei die Dateitypen Markdown (.md), YAML Ain't Markup Language (.yaml, .yml) und JSON (.json) korrekt anzeigen und Python-Programmcode interpretieren können.

Die Konfiguration eines Rasa Projektes wird zunächst ohne Python-Programmcode realisiert. Mit insgesamt vier Dateien[53] lässt sich die gewünschte Anwendung abbilden:

- *config.json*
- *domain.yml*
- *nlu.json*
- *stories.md*

Mit der unter 1) definierten Datei config.json, werden allgemeine Einstellungen für das Projekt festgelegt. Sämtliche Einstellungen sind mit einem Standardwert initiiert, so dass nicht alle Parameter gesetzt werden müssen[54]. Es empfiehlt sich aber zumindest die folgenden Werte manuell festzulegen [92]:

- *project*, gibt den Namen des Projektes an.
- *path*, bestimmt, wo trainierte Modelle gespeichert werden sollen.
- *fixed_model_name*, gibt an, wie das trainierte Modell heißen soll.
- *data*, setzt den Ordner bzw. die Datei, in der die Trainingsdaten liegen.

[53] Die Dateinamen sind frei wählbar, da die Dateipfade als Parameter übergeben werden.

[54] Der folgende Link enthält alle mögliche Konfigurationsparameter: https://nlu.rasa.ai/config.html

- *language*, bestimmt die zu interpretierende menschliche Sprache.
- *pipeline*, gibt die Verarbeitungskette bei der Interpretation an.
- *num_threads*, legt fest, auf wie viel Prozesse das Training verteilt wird.
- *port*, bestimmt den Port, auf dem die Server-Anwendung läuft.
- *response_log*, gibt den Speicherort für Logs im Server-Betrieb an.
- *log_level*, legt fest, ab welchem Level die Logs zu speichern sind.

```
{
  "project": "HelloRasa_Showcase",
  "path": "models",
  "fixed_model_name": "current",
  "data": "./training_set.json",
  "language": "de",
  "pipeline": ["spacy_sklearn"],
  "num_threads": 2,
  "max_training_processes": 1,
  "port": 5000,
  "response_log": "./logs/",
  "log_level": "DEBUG"
}
```

Abbildung 38: Konfiguration eines Rasa-Projektes in der config.json-Datei. [Quelle: Eigenmaterial]

Von besonderer Bedeutung ist hierbei der Parameter pipeline. Über diesen Parameter wird bestimmt, wie und womit die eingehende Äußerung bzw. Nachricht verarbeitet wird. Wie in einem Rohleitungssystem – daher auch der Name Pipeline – fließt die Nachricht durch sämtliche Komponenten, wird jeweils analysiert und mit Metainformationen ergänzt. Folglich ist die Reihenfolge wichtig, in der die Komponenten der Pipeline hinzugefügt werden, da die Nachricht im Verlaufe der Pipeline um zusätzliche Informationen wächst, die während der Verarbeitung verwendet werden können. [92] [93]

Durch das Pipeline-Entwurfsmuster lässt sich der Gesamt-Verarbeitungsprozess hochgradig individualisieren. Entsprechend der verwendeten Komponenten und deren Reihenfolge, ist ein umfangreiches Wissen über NLP-Systeme generell und speziell über die Funktionsweise der (Drittanbieter-) Komponenten Voraussetzung für die Verwendung. Aufgrund dieser Komplexität, existieren insgesamt vier Vorlagen, die eine erfolgreiche Verarbeitung garantieren: spacy_sklearn, mitie, mitie_sklearn und keyword. Die Rasa Dokumentation rät zur Verwendung der spaycy_sklearn-Vorlage. Laut Rasa ist sie die erfolgversprechendste Kombination aus Einrichtungsaufwand, Trainingsdauer, Verarbeitungsdauer und Erkennungsrate. [93] [94]

Nach der allgemeinen Projektkonfiguration, erfolgt mit Schritt 2) die Bestimmung der Anwendungsdomäne. Es werden zu erkennende Absichten und Entitäten bekannt gegeben, Kontextvariablen definiert und auszuführende Aktionen bzw.

zurückzugebende Dialogantworten festgelegt. Wie bei IBM Watson Conversation, lauten die Namenskonversation dabei Intents für Absichten, Entities für Entitäten und Slots für Kontextvariablen. Lediglich Actions, für auszuführende Aktionen, und Templates, für Dialogantworten, weichen davon ab.

Absichten und Entitäten werden in Form von Listen definiert, wobei diese nur Namensbezeichner umfassen. Trainingsbeispiele oder auch Synonyme, werden an dieser Stelle nicht definiert. Dies geschieht erst in Schritt 3). [83]

```
entities:
- book_title

intents:
- start_conversation
- search_for_book
- end_conversation
- out_of_context
```

Abbildung 39: Definition von Entitäten und Absichten in der domain.yml-Datei. [Quelle: Eigenmaterial]

Mit Slots wird der dynamische Kontext in Form von Variablen verwaltet. Ein Slot entspricht stets einem vordefinierten Datentyp. Die möglichen Datentypen sind [83]:

- *text*, enthält Text, aber der Inhalt des Textes ist irrelevant.
- *bool*, enthält einen Wahrheitswert, also wahr oder falsch.
- *categorical*, enthält einen Wert aus einer Liste von Werten. Sinnvollerweise sollte die Werteliste aus zuvor definierten Entitäten bestehen.
- *float*, legt ein Zahlen-Intervall fest, wobei Fließkommazahlen aber auch Ganzzahlen erlaubt sind, zumindest, wenn sie im Intervallraum liegen.
- *list*, eine Menge von Werten, wobei der Inhalt irrelevant ist. Es zählt nur, ob mehrere Werte vorhanden sind oder nicht.
- *unfeaturized*, der Inhalt dieses Typs ist frei wählbar.

Das Setzen einer Slot-Variablen geschieht automatisch, wenn der Name der Slot-Variablen mit dem Namen einer Entitäten-Variablen übereinstimmt. Das Lesen einer solchen Variablen geschieht unter Verwendung eines Platzhalter-Elementes, welches den Namen des Slots verwendet und von der Anwendung automatisch gefüllt wird. Stimmen Slot-Name und Entitäten-Name nicht überein, so ist der programmatische Zugriff mittels Python-Code nötig. Der Zugriff auf Slots wird dabei durch die Erstellung einer eigenen Custom-Action ermöglicht. [83] [95]

```
slots:
  session_id:
    type: unfeaturized
  book_title:
    type: text
  is_book_found:
    type: bool
```

Abbildung 40: Slot-Definition inklusive Datentyp in der domain.yml-Datei. [Quelle: Eigenmaterial]

Actions definieren Handlungen, die von der Anwendung auszuführen sind. Die Standardhandlung im Dialog ist eine Textantwort, beispielsweise zur Begrüßung oder Verabschiedung. Aus diesem Grund gibt es die vordefinierte Handlung ActionUtter. Textantworten werden automatisch ActionUtter zugeordnet, wenn sie das Präfix *utter_* enthalten. In diesem Fall sucht die Anwendung automatisch nach der zugehörigen Textvorlage. Diese sind im Bereich der Template-Liste hinterlegt. Das Training, zu welcher Absicht welche Action auszuführen ist, erfolgt in Schritt 4). [83] [84]

```
actions:
- utter_start_conversation
- utter_book_found
- utter_book_missing
- utter_end_conversation
- utter_out_of_context
- action_search_for_book
```

Abbildung 41: Festlegung von Dialoghandlungen in der domain.yml-Datei.[Quelle: Eigenmaterial]

Die bereits im Zusammenhang mit Slots angesprochene CustomAction, definiert eine eigene Handlung. Beispielsweise eine Datenbank-Abfrage, ob ein bestimmtes Buch im Katalog der Hochschule aufgelistet ist. Anschließend kann das Ergebnis für einen zugehörigen bool-Slot mit dem entsprechenden Wahrheitswert gesetzt werden. Der Name des gesuchten Buches wurde im Vorfeld aus einem text-Slot ausgelesen und in die Suchmaske eingefügt. CustomActions basieren ausschließlich auf Python-Programmcode und erben dabei von der Rasa-Klasse Action.

```python
class ActionSearchForBook(Action):
    def name(self):
        # Der hier definierte Name ist in der domain.yml im
        # actions-Abschnitt gleichermaßen anzugeben. Andernfalls
        # wird diese Action nicht registriert.
        return "action_search_for_book"

    def run(self, dispatcher, tracker, domain):
        # Diese Methode führt die eigentliche Handlung aus.
        # Beispielsweise die Abfrage eines Buches anhand des
        # Titels in der Bibliotheks-Suchmaschine. Der Titel wird
        # aus einem entsprechenden Slot ausgelesen. Wilbert, die
        # Bib-Suchmaschine der TH Wildau, sucht das Buch
        # anschließend und gibt das Ergebnis zurück.
        book_title = tracker.get_slot("book_title")
        result = Wilbert().find(book_title)

        # Der Anwendung wird mitgeteilt, nachfolgend eine passende
        # Textantwort auszuwählen. Außerdem wird ein Slot mit dem
        # Suchergebnis für optional weitere Äußerungen gesetzt.
        if result:
            dispatcher.utter_template("utter_book_found")
        else:
            dispatcher.utter_template("utter_book_missing")

        return SlotSet("is_book_found", result)
```

Abbildung 42: Beispielhafte Implementierung einer CustomAction in Python. [Quelle: Eigenmaterial]

Damit eine Dialogantwort von Rasa ausgewählt und zurückgegeben werden kann, müssen Antworttexte definiert werden. Die Template-Liste in der domain.yml umfasst solche Texte. Diese sind immer Actions zugeordnet, wodurch für Rasa ersichtlich wird, welcher Antworttext zu welcher Handlung gehören. Es ist notwendig, sowohl bei Template-Name, als auch bei Action-Name, das Präfix *„utter_“* voranzustellen. Andernfalls erkennt das System nicht, dass es sich um eine ActionUtter, also Sprachaktion, handelt und sucht nach einer passenden CustomAction. Kann keine passende Aktion gefunden werden, verbleibt der Dialog im aktuellen Zustand, bedeutet, es passiert nichts. [83] [84]

```yaml
templates:
  utter_start_conversation:
    - "Hallo, schön dich zu sehen. Kann ich etwas für dich tun?"
    - "Hi, es ist mir eine Ehre. Was willst du wissen?"
  utter_book_found:
    - "Das Buch mit dem Titel {book_title} habe ich gefunden."
    - "Ja, {book_title} befindet sich im Katalog der Bibliothek."
  utter_book_missing:
    - "Leider konnte ich das Buch {book_title} nicht finden."
    - "Momentan ist {book_title} nicht im Katalog der Bibliothek."
    - "Du musst mir noch den Buchtitel nennen, den ich suchen soll."
  # (…)
```

Abbildung 43: Erstellung variabler Dialogantworten in der domain.yml-Datei. [Quelle: Eigenmaterial]

Pro ActionUtter lassen sich mehrere Antworten definieren, die außerdem mit Platzhaltern versehen werden können. Dadurch entsteht eine Variabilität im Dialog. Die Auswahl einer Antwort-Vorlage ist an Trainingsdaten gebunden, durch die ein Wahrscheinlichkeitsmodell entsteht, welches die Auswahl übernimmt. Werden beispielsweise drei Antwortvarianten definiert, eine mit zwei Platzhaltern, eine mit nur einem Platzhalter und eine ohne Platzhalter, dann wird, entsprechend dem aktuellen Slot-Zustand, diejenige Variante gewählt, die am ehesten mit dem Slot-Zustand übereinstimmt. Das heißt, sind zwei relevante Slots gesetzt, so wird die Antwortvariante mit zwei Platzhaltern verwendet. Bei Existenz von mehreren Antworten, die beispielsweise keine Slots oder die gleichen Slots in gleicher Reihenfolge enthalten, wird zufallsmäßig eine Antwort ausgewählt. [83] [84] [85]

Nach der Definition der Anwendungsdomäne, erfolgt mit Schritt 3) das Training zur korrekten Erkennung von Absichten inklusive Entitäten. Dazu werden in der nlu.json[55]-Datei Beispielsätze für die Absichten und Entitäten hinzugefügt. Es kann sich dabei um einzelne Worte, Wortgruppen oder einen ganzen Satz handeln. Letztendlich sind Varianten einzufügen, die Benutzer verwenden könnten, um ihre Absicht zu formulieren. Rasa empfiehlt, zu Beginn mindestens um die 20 bis 30 Beispiele hinzuzufügen. [96] [97]

Im Gegensatz zu Watson Conversation, sind bei Rasa auch Entitäten zu trainieren, da andernfalls die Entitäten-Erkennung nicht funktionsfähig ist. Wie bei den Absichten, gilt es, ca. 30 Trainingsbeispiele bereitzustellen [96]. Das Training von Absichten und Entitäten ist alternativ auch innerhalb eines Beispiels kombinierbar. In diesem Zusammenhang besteht dennoch die Notwendigkeit, zumindest einige disjunktive Beispiele zu definieren. Andernfalls könnte das Machine Learning den Schluss ziehen, nur Beispiele mit Entitäten würden zu dieser Absicht gehören und Varianten ohne Entitäten gehören nicht dazu. Zusammenfassend besteht ein Trainingsbeispiel aus einer Äußerung, dem damit assoziierten Intent und optional enthaltenen Entities. Die Intent- und Entity-Bezeichner müssen dabei mit den Definitionen aus der domain.yml übereinstimmen. [90] [97]

[55] Alternativ ist auch eine nlu.md-Datei erzeugbar. Das Markdown-Format wird jedoch von Rasa nicht vollwertig unterstützt, wodurch die Definition von regulären Ausdrücken nicht möglich wäre.

```
{
  "rasa_nlu_data": {
    "common_examples": [
      {
        "text": "Ich brauche ein Buch zur Recherche.",
        "intent": "search_for_book",
        "entities": []
      },
      {
        "text": "Habt ihr das Buch Einstieg in Python hier?",
        "intent": "search_for_book",
        "entities": [
          {
            "start": 18,
            "end": 36,
            "value": "Einstieg in Python",
            "entity": "book_title"
          }
        ]
      }
    ],
    ...
  }
}
```

Abbildung 44: Beispielhafte Definition von Trainingsdaten in der nlu.json-Datei. [Quelle: Eigenmaterial]

Für Rasa existiert kein vordefinierter Fuzzy Matching Algorithmus, wie aus Watson Conversation bekannt. Die korrekte Zuordnung von Entitäten in den Trainingsbeispielen ist daher von enormer Bedeutung. Ein jedes Entitäts-Beispiel ist daher exakt kenntlich zu machen, welches durch Angabe der Position im Textbeispiel geschieht (*start*- und *end*-Parameter). Auf manuellem Wege ist dieses Vorgehen sehr langwierig, weshalb entweder ein selbstgeschriebenes Skript zur automatischen Festlegung von start- und end-Parameter verwendet werden sollte oder der Rasa-NLU-Trainer[56] als Unterstützung benutzt werden kann.

Während der Einrichtung des Beispielprojektes erwies sich das Tool Rasa-NLU-Trainer als effektive Lösung. Das Tool ist entweder lokal zu installieren oder durch Aufruf einer Webseite[57] online verwendbar.

Für die Definition von Synonymen steht ein eigenes Datenfeld, genannt entity_synonyms, in der nlu.json Datei bereit. Dabei werden betrachtete Entitäts-Werte mit einer Liste von Synonymen verbunden. Ergänzt wird das Synonym-Datenfeld von einem weiteren Datenfeld zur Definition von regulären Ausdrücken, genannt regex_features. Die Ausdrucksmuster sind anschließend sowohl auf Entitäten als auch Absichten anwendbar. [90] [97]

[56] Link zum Rasa-NLU-Trainer Programmcode: https://github.com/RasaHQ/rasa-nlu-trainer

[57] Online-Version des Rasa-NLU-Trainer: https://rasahq.github.io/rasa-nlu-trainer/

```
"entity_synonyms": [
  {
    "value": "Immatrikulationsnummer",
    "synonyms": [
      "Immanummer",
      "Identifikationsnummer",
      "Studentenausweisnummer",
      "Registrationsnummer"
    ]
  }
],
"regex_features": [
  {
    "name": "matriculation_number",
    "pattern": "[0-9]{9}"
  }
]
```

Abbildung 45: Auflistung von Synonymen und reg. Ausdrücken in der nlu.json. [Quelle: Eigenmaterial]

Die Erkennung von Entitäten inklusive Synonyme bzw. reguläre Ausdrücke bedarf einer Pipeline, die entsprechende Komponenten zur Verarbeitung und Interpretation enthält. Bei Nutzung der Vorlagen spacy_sklearn, mitie oder mitie_sklearn ist die Vollständigkeit der Pipeline hinsichtlich dessen garantiert. [93]

Als letzte Maßnahme zur Einrichtung einer NLP-Anwendung mit Hilfe von Rasa, sind Dialogabläufe zu trainieren, auch Stories genannt. Dies geschieht in der gleichnamigen stories.md Datei, in der verschiedene Pfade möglicher Dialoge aufgebaut werden.

```
## story_1
* _start_conversation
    - utter_start_conversation

## story_2
* _search_for_book
    - utter_book_missing
* _search_for_book{"book_title": "Einführung in Python"}
    - action_search_for_book

 ## story_3
* _search_for_book{"book_title": "Heute hat die Welt Geburtstag"}
    - action_search_for_book
```

Abbildung 46: Mögliche Pfade für Dialogverläufe in der stories.md-Datei. [Quelle: Eigenmaterial]

Ein Story-Pfad besteht aus einem eindeutigen Namen und der Verknüpfung von Intents, Entities, Slots und Actions. Dabei gilt es festzulegen, welche Handlung bei welcher Intent-Entities-Kombination auszulösen ist [85]. Wird beispielsweise eine Absicht erkannt, bei der nach einem Buch gesucht werden soll, der Buchtitel aber fehlt, so wird dieser Titel vom Dialog erfragt und erst anschließend die Suchaktion ausgelöst. In einer anderen Story dagegen wird sofort eine Buchtitel-Entität erkannt, entsprechend kann ohne Verzögerung gesucht werden.

Wichtig: Intents müssen mit dem Präfix „_“ (Unterstrich) versehen werden. Andernfalls wird das Intent nicht korrekt erkannt und die Story bleibt effektlos, bedeutet, das Training wirkt sich nicht auf die Anwendung aus. Diese Besonderheit wird nicht in der Entwickler-Dokumentation bekannt gegeben und wurde eigenständig herausgefunden.

Nach Abschluss der Einrichtung eines Rasa-Projektes zur Festlegung einer Anwendungsdomäne, erfolgt anschließend das Training anhand der zuvor bestimmten Parameter und Beispieldaten. Dieses Training ist in zwei Phasen eingeteilt, wobei zunächst der Interpreter, also die Umwandlung von Äußerungen in strukturierte Daten, trainiert wird und darauf basierend die Dialogabläufe. [84]

Das Interpreter-Training ist über ein vorgefertigtes Python-Script realisierbar. Wurde das Projekt im Vorfeld konfiguriert, so reicht es aus, den Pfad zur config.json-Datei als Parameter mitzugeben. Andernfalls lassen sich sämtliche Konfigurationen (z.B. Pipeline, Path, Port) auch als Parameter übergeben. Im hiesigen Fall wird das Script mit der config-Datei direkt aus dem Projektordner ausgeführt, wodurch der Name der Datei mit vorangestelltem Parameter ausreichend ist.

```
$ python -m rasa_nlu.train -c config.json
```

Abbildung 47: Terminal-Code für das Rasa Interpreter-Training. [Quelle: Eigenmaterial]

Während des Trainings werden vereinzelt Log-Nachrichten in die ausführende Konsole geschrieben. Es ist dabei auf folgende Nachricht zu achten:

```
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with
no predicted samples
```

Abbildung 48: Warnmeldung bei einer zu geringen Menge an Trainingsdaten. [Quelle: Eigenmaterial]

Es handelt sich bei dieser Meldung um eine Warnung, die auf eine zu geringe Menge von Trainingsdaten hinweist [91]. Welche Absichten bzw. Entitäten davon betroffen sind, wird momentan nicht angezeigt[58]. Ein automatischer Abbruch des Trainings findet nicht statt, wodurch das entstehende Modell theoretisch verwendbar ist. Davon ist aber abzuraten, denn die Daten sind logischerweise korrupt. Entsprechend sind alle Trainingsbeispiele zu prüfen und ggfs. anzupassen.

[58] Diese Problematik wurde erstmals im Juni 2017 erwähnt und von den Entwicklern bestätigt. Eine offizielle Lösung steht noch aus. Der aktuelle Stand dieses Problems kann unter folgendem Link eingesehen werden: https://github.com/RasaHQ/rasa_nlu/issues/431

Nach Beendigung des ersten Skriptes, erfolgt das Training der Dialoge bzw. Stories. Auch für diesen Teil stellt Rasa ein vorgefertigtes Skript bereit. Für dieses sind zumindest drei Parameter anzugeben: -s für die Story-Datei, -d für die Domain-Datei und –o für den Speicherort des entstehenden Modells. Die Ausführung des Skriptes geschieht erneut direkt im Ordner des Projektes.

```
$ python -m rasa_core.train -s stories.md -d domain.yml -o models/dialog --epochs
300
```

Abbildung 49: Kommandozeilen-Code für das Rasa Dialog-Training. [Quelle: Eigenmaterial]

Für das Dialog-Training existieren noch einige zusätzliche Parameter, die das Training individualisieren und ggfs. optimieren. Diese Parameter werden dem Machine Learning übergeben, welches die Standardwerte mit den neuen Parametern ersetzt. Folgende Parameter stehen zur Verfügung [98]:

Tabelle 7: Optionale Parameter zur Beeinflussung des Trainings von Dialogen mit Rasa.

Parameter	Funktion	Standardwert
--history	Menge an bereits passierten Pfadschritten, die für die Wahrscheinlichkeitsberechnung der aktuell auszuführenden Action verwendet werden dürfen.	3
-- batch_size	Menge an Beispielen, die zeitgleich in einem Trainingsschritt einzubeziehen sind.	20
--augmentation	Anzahl von zufallsmäßig zu kombinierenden Stories zur Erweiterung möglicher Pfade.	50
--epochs	Anzahl der Wiederholungen des Trainings zur Generierung des Modells.	100
--validation_split	Prozentzahl der Trainingsdaten, die für eine Validierung der Korrektheit des Modells (interner Positivtest) zu verwenden sind.	0.1
--online	Ermöglicht das Lernen, während die Anwendung ausgeführt wird. In diesem Fall ist zusätzlich --nlu als Parameter anzugeben.	False
--nlu	Bei Verwendung des Parameters --online wird der Pfad zum zuvor trainierten Interpreter-Modell benötigt.	None

Im finalen Schritt können die trainierten Modelle einem Praxistest unterzogen werden. Über ein weiteres Skript werden Interpreter- und Dialog-Modell zur Verwendung eingelesen. Dabei sind erneut Paramater anzugeben, in diesem Fall die beiden Modellpfade.

```
$ python -m rasa_core.run -d models/dialog -u models/HelloRasa_Showcase/current
```

Abbildung 50: Terminal-Code für die Ausführung der trainierten Anwendung. [Quelle: Eigenmaterial]

In der Konsole können anschließend Äußerungen in Form von Text eingegeben werden. Die Antwort besteht nicht nur aus dem zugehörigen Dialogtext, sondern zeigt zusätzlich den aktuellen internen Zustand des Dialogs mit Slots und Actions an. Das Datenformat entspricht dabei JSON.

In einer vollwertigen Anwendung würde die JSON-Antwort beim Client transformiert werden, um grundlegend, vor allem den Antworttext auszugeben. Daher kann das Projekt auch in Form eines Servers verwendet werden. Die Endpunkte eines solchen Servers wurden im Unterabschnitt *API-Übersicht* bereits vorgestellt.

```
$ python -m rasa_core.server -d models/dialog -u models/HellRasa_Showcase/current
```

Abbildung 51: Terminal-Code für den serverbasierten Start der Anwendung. [Quelle: Eigenmaterial]

Für sämtliche Python-Skripte existieren auch Python-Klassen, mit denen sich der eben beschriebene Ablauf in Form von Python-Quellcode realisieren lässt. Bei entsprechender Notwendigkeit ist dabei der standardmäßige Machine Learning Algorithmus nicht nur durch Parameter anpassbar, sondern komplett austauschbar. [87] Ein Wechsel der sogenannten Policy muss wohl bedacht sein, denn in Konsequenz wird das Herzstück des Frameworks ausgetauscht. Ein solches Vorgehen ist bei Watson Conversation nicht möglich.

4.3 Anforderungsabgleich

Anhand der Erkenntnisse über Aufbau, Funktionsweise und Verwendung von Rasa sowie IBM Watson Conversation, wird mit dem aktuellen Abschnitt eine Verbindung zwischen Anforderungsanalyse und realer Leistungsfähigkeit der Software-Lösungen hergestellt. Dazu werden die formulierten Kriterien mit dem Funktionsumfang der beiden Software-Lösungen verglichen.

Aufgrund der Gesamtzahl von definierten funktionalen und nichtfunktionalen Anforderungen, ist es wenig sinnvoll, eine Vergleichstabelle mit 46 Einträgen anzuzeigen. Der Platzverbrauch wäre schlichtweg zu groß. Auf den tabellarischen Vergleich muss dennoch nicht vollends verzichtet werden: Auf der beiliegenden CD, im Verzeichnis *~/Eval/*, befindet sich die Endauswahl-Vergleichstabelle in Form eines Excel-Dokumentes.

Der Abgleich der Anforderungen soll aber nicht kommentarlos in einer Excel-Tabelle münden. Daher werden nachfolgend Auffälligkeiten des Vergleichs aufgezeigt, mit dem Fokus auf Unterschiede zwischen den beiden Software-Bibliotheken. Als weiteres Kriterium, für eine textliche Erwähnung, gelten alle Anforderungen, die als obligatorisch angesehen werden, also einer *„muss"*-Anforderung entsprechen. Vernachlässigt werden all diejenigen Anforderungen, welche bereits als Entscheidungsmerkmal für die Vorauswahl Verwendung fanden[59].

Bei Betrachtung der Anforderungen anhand der fünf Hauptbestandteile eines NLP-Systems (gemeint sind Phonologie, Morphologie, Syntax, Semantik, Konklusion – vorgestellt im Kapitelabschnitt *Die Methodik – Wie funktioniert NLP im Detail?*), fällt auf: Die Phonologie wird weder von Rasa noch von Watson Conversation unterstützt. Entsprechend sind die Anforderungen *SW0010, SW0020, SW050, SW0110, SW0120, SW0130, SW0140* und *SW370* allesamt nicht erfüllt. Diese definieren u.a. die Verfügbarkeit und Konfiguration von Klangmodellen sowie die generelle Möglichkeit zur Kommunikation mittels verbaler Sprache, folglich die Spracherkennung sowie Sprachsynthese. Schon in der Vorauswahl dieser Evaluierung wurde ersichtlich, dass die Phonologie generell selten bei NLP-Systemen vorzufinden ist. Es lässt sich sagen, dass bisher ein „entweder-oder"-Fokus vorherrscht. Entweder gilt die Konzentration der Spracherkennung und –synthese oder der Fokus liegt auf eine Dialogsteuerung unter Erzeugung von strukturierten Daten.

Im Gegensatz zu verbaler Sprache, unterstützen beide Software-Lösungen jedoch die textbasierte Eingabe und Ausgabe von Äußerungen. Damit werden die Anforderungen *SW0030* und *SW0060* abgedeckt.

Eng mit den verbalen bzw. textuellen Spracheingaben verbunden, ist die Anforderung diese Eingaben in Echtzeit abzuhandeln (*SW0040*). Dabei bedeutet Echtzeit, dass Analyseprozesse bereits während der aktiven Äußerung ablaufen[60]. Beide Systeme

[59] Die Identifikationsnummern dieser Anforderungen lauten: SW0090, SW0100, SW0440, SW0450, SW0460, SW0470, SW0480, SW0500 & SW0520.

[60] Aktive Äußerung ist der Zeitabschnitt, in der die Aussage textuell oder verbal formuliert wird, also die Aneinanderreihung einzelner Worte zu einem Satz, Befehl, Ausruf, etc.

erwarten aber ein ganzheitliches Satzkonstrukt für die Analyse, wodurch eine Latenz unvermeidbar ist.

Die Bedingung der persistenten Speicherung sämtlicher Eingaben (*SW0360, SW0380*), ist für textuelle Eingaben von beiden Systemen erfüllt. Dabei werden die Eingaben als Log-Nachrichten persistiert, die jederzeit abrufbar sind.

Analyseprozesse und Modellkonfigurationen hinsichtlich Morphologie (*SW0150, SW0160*), Syntaktik (*SW0180, SW0190*), Semantik (SW0210, SW0220) und Konklusion (*SW0230, SW0240, SW0280*) sind vorhanden. Entsprechend gelten die zugehörigen Anforderungen von beiden Software-Lösungen als erfüllt. Das Korrigieren morphologisch falscher Worte (*SW0170*) wird von Watson Conversation mittels Fuzzy Matching unterstützt. Rasa hingegen verfügt über keine ähnliche Technik. Bei syntaktisch fehlerhaften Textstücken, finden durch beide Software-Lösungen keine der geforderten Korrekturmaßnahmen statt (*SW0200*).

Durch das Konzept der Synonyme, welches von Rasa und Watson Conversation jeweils unterstützt wird, können Mehrdeutigkeiten während der Analyse weitestgehend vermieden werden (*SW0250*). Gleichermaßen existiert durch System-Entitäten, ein konzeptioneller Ansatz zur Auflösung von Repräsentationsfragen (*SW0270*), beispielsweise die einheitliche Darstellung von Datumswerten. Für Datentypen, die durch System-Entitäten nicht abgedeckt werden, können eigene Formatierungen festgelegt werden.

Keinerlei Konzept bieten die beiden Software-Lösungen zur Auflösung von Paradoxien (*SW0260*), beispielsweise beim Einsatz von Ironie. Auch die Analyse weiterer Kommunikationskanäle, z.B. Mimik und Gestik, wird nicht direkt von Rasa und Watson Conversation unterstützt (*SW0310*). Es ist jedoch möglich, über Drittanbieter-Software eine solche Analyse vorzunehmen und das Ergebnis als Kontextvariable während der Textanalyse einfließen zu lassen.

Es ist vor allem eine starke konzeptionelle Ausrichtung auf vollständige Dialoge inklusive Fluss-Steuerung auszumachen. Die Bedingungen der Dialogkommunikation (*SW0070, SW0080*) ist daher für beide NLP-Systeme als erfüllt anzusehen.

Das Thema Machine Learning (*SW0400, SW0420, SW0430*) zur kontinuierlichen Verbesserung der Erkennung von Absichten und Entitäten, wird gleichermaßen von beiden Software-Lösungen abgedeckt. Die manuelle Lernfähigkeit (*SW0410*), basierend auf eigens gewählte Beispieldaten, ist dabei nicht nur möglich, sondern im ersten Schritt der Definition einer Anwendungsdomäne zwingend notwendig. Semiautomatisches Lernen (*SW0420*) ist durch Auswertung der Log-Nachrichten und der darin enthaltenen Eingaben sowie zugehörigen Absichts- und Entitätsvorhersagen möglich.

Vollautomatisches Lernen (*SW0430*), mit dem Blick auf die automatische Erweiterung der Domäne basierend auf durchgeführte Dialoge, ist weder bei Rasa noch bei Watson Conversation vorgesehen. Nur ein eigenständiger Programmieransatz würde eine solche Funktionalität momentan ermöglichen. Im Übrigen bringt das Lernen zur Anwendungslaufzeit Wartezeiten mit sich, denn sowohl bei Watson Conversation als auch Rasa, ist ein im Training befindliches Modell nicht verwendbar. Erst nach Abschluss des Trainings können neue Vorhersagen getroffen werden.

Die Anforderungen Einsicht des Quellcodes *(SW0490*) und Verwendbarkeit der Anwendung in einer Open Source Lizenz (*SW0510*) zeigen nochmals Unterschiede zwischen den beiden NLP-Systemen auf. Die proprietäre Lösung von IBM erfüllt beide Bedingungen nicht. Rasa dagegen kann vollständig heruntergeladen werden und erlaubt durch die Apache 2.0 Lizenz eine freie Verwendung.

Alles in allem, ergibt sich durch Vorauswahl und Anforderungsabgleich folgendes Bild:

- Watson Conversation erfüllt 29 der 46 Anforderungen vollständig.
- Je nach Auslegung, können die drei als teilweise bewerteten Bedingungen hinzugezählt werden[61].
- Dagegen bleiben 14 Anforderungen von Watson Conversation unerfüllt.
- Rasa verbucht 34 der 46 Anforderungen als erfüllt.
- Unerfüllt bleiben von Rasa 12 Anforderungen.

Die nicht erfüllten Anforderungen beziehen sich bei beiden Kandidaten vor allem auf den Phonologie-Bereich, wodurch bereits jeweils acht Anforderungen unerfüllt bleiben. Ebenfalls von beiden Kandidaten unerfüllt bleibt ein Lösungsansatz bezüglich Paradoxien.

Durch die Integration des Fuzzy Matchings verschafft sich Watson Conversation einen kleinen Vorteil, denn die Korrektur bzw. Akzeptanz leichter Rechtschreibfehler ermöglicht bessere Erkennungsraten und eine höhere Robustheit gegenüber fehlerbehafteten Eingaben. Bei den Eigenschaften anfallende Kosten, Offline Verwendbarkeit, Lizenzmöglichkeiten und Quellcode-Einsicht punktet die freie Software Rasa gegenüber der proprietären Software von IBM.

Insgesamt wurden die allermeisten Unterschiede bereits in der Vorauswahl kenntlich und bildeten prinzipiell die Grundlage für weitere Betrachtungen. Der Kontrast zwischen Rasa und Watson Conversation bleibt nach dem Anforderungsabgleich nahezu unverändert.

[61] Die teilweise-Bewertung betraf Anforderungen, die im Zusammenhang mit Kosten für die Anschaffung und Benutzung sowie Begrenzung von API-Anfragen standen. Sie wurden bereits in der Vorauswahl thematisiert.

4.4 Performance und Latenz

Der Anforderungsabgleich des letzten Abschnitts konnte keine schärferen Kontraste zwischen Rasa und Watson Conversation erzeugen. Die Liste der Gemeinsamkeiten und Unterschiede ist nahezu unverändert. Dieser Fakt wird bei Betrachtung der prinzipiellen Funktionsweise beider Kandidaten untermauert, denn Rasa und Watson Conversation werden durch identische Schlüsselkonzepte beschrieben.

Bisher wurden auch nur Fragen zur Kompatibilität und zum Funktionsumfang der Systeme geklärt. Wie die Software-Lösungen auf Testdaten reagieren ist aktuell unklar. Es bietet sich daher an, die Systeme einem Funktionstest zu unterziehen, der aufzeigt, welche Erkennungsraten nach vorhergehendem Training zu erwarten sind.

4.4.1 Testverfahren

Der Funktionstest folgt dem Prinzip eines Blackbox Tests: Die Bewertung der Systeme erfolgt nur von außen, das heißt, die ermittelten Ergebnisse werden mit erwarteten Ergebnissen verglichen. Welche Prozesse intern ablaufen interessiert dabei nicht.

Die erwarteten Ergebnisse sind durch eine Liste von Beispielaussagen definiert, die jeweils einer Absicht zugeordnet sind. Über die API-Schnittstelle werden die Beispielaussagen dem NLP-System übergeben. Es erfolgt die interne Analyse der jeweiligen Aussage. Das Ergebnis dieser Analyse ist die vom System vermutete Absicht der Aussage. Erwartete Absicht und vorausgesagte Absicht sind anschließend auf Übereinstimmung zu prüfen.

Für die Ergebnisse ist eine standardisierte Wertezuordnung nötig. Darauf basierend sind statistische Auswertungen und der generelle Vergleich der Messergebnisse möglich. Für Klassifikationen bietet sich das Wertungsschema einer Konfusionsmatrix an, welche zur Bewertung von Klassifikationen häufig verwendet wird und die komfortable Erstellung von Statistiken ermöglichen [99 S. 66].

Durch die Konfusionsmatrix für Klassifikationen entsteht eine Tabelle, die für eine Menge von N-Klassen eine N×N-Matrix aufbaut. Die Menge der Klassen wird mit dem Buchstaben c gekennzeichnet. Für sie gilt: $c \in \mathbb{N}\ wobei\ c \geq 2$. [100 S. 89f]

In der Matrix sind sämtliche Kombinationen aus erwarteter Klasse und interpretierter Klasse bzgl. der Menge der N definierten Klassen enthalten. Im Test wird jeweils die aktuell erwartete Klasse mit der vorausgesagten Klasse verglichen und an der entsprechenden Stelle in der Tabelle das zahlenmäßige Vorkommen gezählt. In *Abbildung 52* ist eine beispielhafte Konfusionsmatrix dargestellt.

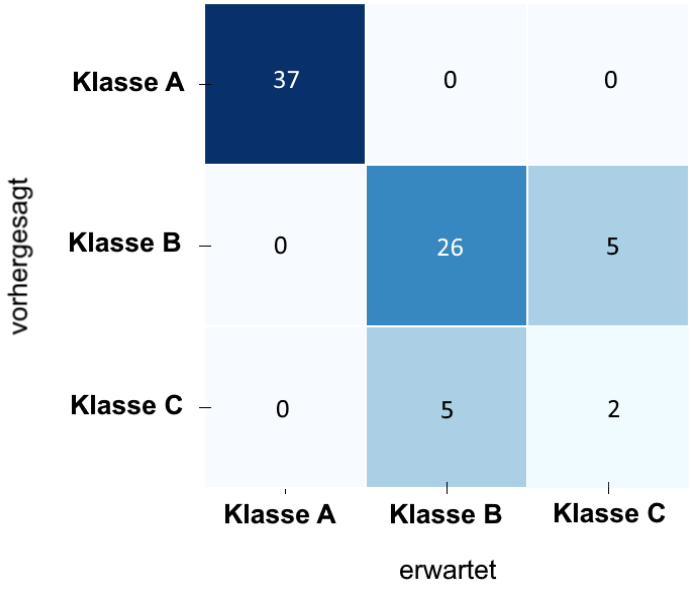


Abbildung 52: Aufbau einer 3x3 Konfusionsmatrix für Klassifikationsergebnisse. [Quelle: Eigenmaterial]

Mit einer solchen Konfusionsmatrix lassen sich verschiedene Rückschlüsse über den Test und dessen Ergebnisse ziehen. Dabei ist die Betrachtung der Diagonalen besonders wichtig, denn entlang dieser werden korrekt interpretierte Klassifikationen sichtbar. Die Zahlenwerte der Diagonale sollten gegenüber den restlichen Zeilen- und Spaltenwerten am höchsten sein. Generell zeigen Zeilen- und Spaltenwerte, die nicht auf der Diagonale liegen, falsche Interpretationen auf. Mit wenigen Blicken lässt sich daher erkennen, welche Klassen bereits gut trainiert sind und welche Klassen weiteren Trainings bedürfen bzw. welche Trainingsbeispiele zu ähnlich sind und daher momentan zu einer schwammigen Zuordnung führen. [101]

Bezogen auf das Beispiel in *Abbildung 52,* ist beispielsweise erkennbar, dass Klasse A außerordentlich robust klassifiziert wird (37 korrekte Einschätzungen) und es keine Verwechselungen mit den restlichen Klassen gibt (jeweils null Fehleinschätzungen). Dagegen scheinen die Klassen B und C noch nicht optimal trainiert. Es kommt zu einigen Verwechselungen (jeweils fünf falsche Zuordnungen), wobei Klasse B noch relativ gut abschneidet (26 korrekte Interpretationen). Bei Klasse C wird sehr schnell erkennbar, dass das bisherige Training nicht ausreichend war, denn es werden nur zwei von sieben Zuordnungen korrekt vorgenommen. Es besteht eine ganz klare Notwendigkeit das Modell hinsichtlich C, aber auch B zu verbessern.

Die Kombination aus erwarteter und vorhergesagter Klasse wird allgemein durch vier mögliche Ergebniswerte beschrieben[62] [100 S. 90]:

[62] Die englische Schreibweise der Ergebniswerte ist für die Auswertung von Konfusionsmatrizen die Standardvariante. Daher werden nachfolgend die englischen Begriffe verwendet.

Tabelle 8: Allgemeine Bezeichnung für Ergebniswerte bei Klassifikationen.

Ergebniswert Klassifikation	Bedeutung
Richtig Positiv bzw. True Positive	Beschreibt die korrekte Zuordnung eines Testbeispiels X zur Klasse X.
Richtig Negativ bzw. True Negative	Ist die korrekte Ablehnung eines Beispiels X von einer Klasse, die nicht X beschreibt, zum Beispiel Klasse Y.
Falsch Positiv bzw. False Positive	Repräsentiert die falsche Zuordnung eines Testbeispiels X zur Klasse Y.
Falsch Negativ bzw. False Negative	Ist die falsche Ablehnung eines Testbeispiels X von Klasse X.

Bei einer 2×2-Matrix, also einer auf zwei Klassen basierenden Klassifikation, sind die Bedeutungsregeln ohne Abhängigkeiten anwendbar[63]. Für eine N×N-Matrix wirkt sich das Ergebnis eines Testbeispiels aber auch implizit auf die weiteren Klassen des Klassifikationssystems aus: Der True-Positive-Fall für die Klasse X bedeutet indirekt für alle anderen Klassen einen True-Negative-Fall. Bei einem False-Positive-Fall für die Klasse X, ist zeitgleich die eigentlich erwartete Klasse Y ein False-Negative-Fall. Die restlichen Klassen werden als True-Negative eingeordnet. Ein jeder Ergebniswert hat folglich einen größeren Wirkungsgrad, also nur auf die direkt betrachteten Klassen des aktuellen Testbeispiels. [100 S. 90] [101]

4.4.2 Testparameter

Mit den vier Ergebniswerten True Positive (TP), True Negative (TN), False Positive (FP) und False Negative (FN) lassen sich verschiedenste Kennzahlen für das Klassifikationsmodell berechnen. Die Kennzahlen können dabei sowohl für eine bestimmte Klasse als auch über alle Klassen berechnet werden. [102]

Für die Bewertung von Klassifikationsmodellen auf Basis einer Konfusionsmatrix haben sich die Kennzahlen Genauigkeit, Sensitivität und F-Maß als wichtigste Berechnungen durchgesetzt. In Quelle [100 S. 89ff] werden diese Parameter verständlich beschrieben inklusive der beispielhaften Betrachtung anhand einer Klassifizierung von gesunden und erkrankten Menschen, wobei es gilt erkrankte

[63] Da nur zwei Klassen existieren, kann es keine weiteren Abhängigkeiten geben. Alle Zuordnungen bzw. Ergebniswerte wirken sich stets direkt auf beide Klassen aus. Daher sind alle Bedeutungsregeln eins-zu-eins anwendbar.

Menschen zu identifizieren. Die eigene Darstellung folgt dieser Quelle für die Beschreibung der drei Kennziffern.

4.4.2.1 Genauigkeit bzw. Precision

Die Genauigkeit – auch Precision, Wirksamkeit, Positive-Prediction-Value – beschreibt den Anteil von korrekten positiv-Zuordnungen, bezogen auf die Menge aller als positiv klassifizierten Zuordnungen. Basierend auf richtige und falsche Positiv-Zuordnungen wird die Menge der tatsächlichen Treffer aufgezeigt. [101] [102] [103]

Die Berechnungsformel lautet dementsprechend:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{TP}{TP + FP}$$

Der Ergebnis-Wertebereich liegt zwischen 0 und 1, wobei ein möglichst hoher Wert angestrebt wird.

Dem Krankheitsbeispiel folgend, ist das die Wahrscheinlichkeit, mit der eine als krank klassifizierte Person tatsächlich krankt ist. [100 S. 89ff]

4.4.2.2 Sensitivität bzw. Recall

Die Sensitivität – auch Recall, Trefferquote, Hit Rate, Richtig-Positiv-Rate – beschreibt den Anteil von korrekten Positiv-Zuordnungen, bezogen auf die Menge aller vorhandenen korrekt-Positiv-Elemente. Basierend auf richtige und fehlende Positiv-Zuordnungen wird die Vollständigkeit der Treffer aufgezeigt. [101] [102] [103]

Die entsprechende Berechnungsformel lautet:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{TP}{TP + FN}$$

Der Ergebnis-Wertebereich liegt zwischen 0 und 1, wobei ein hoher Wert für das Ergebnis angestrebt wird.

Bezogen auf das Beispiel wird damit die Frage beantwortet, mit welcher Wahrscheinlichkeit ein kranker Patient auch als krank klassifiziert wird. [100 S. 89ff]

4.4.2.3 Zusammenhang von Genauigkeit und Sensitivität

Die Auseinanderhaltung der beiden Kennzahlen kann aufgrund ihrer ähnlich klingenden Definition etwas kompliziert erscheinen. *Abbildung 53* soll als Orientierungshilfe für das Verständnis der Definitionen und zugehörigen Formeln dienen.

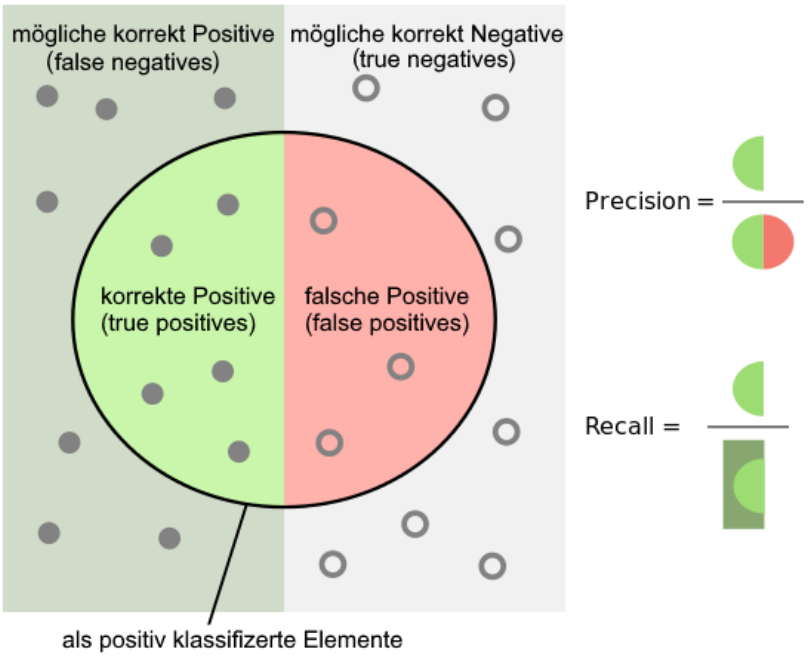


Abbildung 53: Berechnungsgrundlage für Precision und Recall im Zusammenhang mit den Ergebniswerten und zugehörigen Mengen [Nach Quelle: [104]]

Precision und Recall sind nicht nur definitionsmäßig ähnlich. Bei einer Ergebnisformulierung hat nur die gemeinsame Betrachtung eine tatsächliche Aussagekraft. Ziel ist es, maximal viele Positive zu klassifizieren ohne dass dabei falsche Positive mitklassifiziert werden [103] [104]. Entsprechende Wertungen sind:

Tabelle 9: Bewertungen eines Klassifikationsmodells basierend auf Precision und Recall.

Bewertung	Begründung bzw. Zusammensetzung
Gutes Modell	Der Anteil gefundener Positiver aus der Menge der möglichen korrekten Positiven ist außerordentlich hoch. Entsprechend sind unter den gefundenen Positiven sehr wenig falsche Positive. ➔ hohe Precision und hoher Recall
Schlechtes Modell	Es werden kaum Treffer aus der Menge der möglichen korrekten Positiven gemacht. Der Anteil von falschen Positiven ist hoch. ➔ niedrige Precision und niedriger Recall

Defensives Modell	Der Anteil gefundener Positiver aus der Menge der möglichen korrekten Positiven ist nur durchschnittlich. Dafür sind sehr wenige falsche Positive enthalten. ➔ hohe Precision und niedriger Recall
Offensives Modell	Es werden sehr viele Positive aus der Menge der möglichen korrekten Positiven gefunden. Dabei ist ein großer Teil aber falsch Positiv. ➔ niedrige Precision und hoher Recall

4.4.2.4 F1-Maß

Im Grunde bilden Precision und Recall eine Einheit. Mit dem F1-Maß bzw. auch F1 score wird daher eine Vereinigung der beiden Einzelwerte vorgenommen. Beide Ergebnisse werden zu einem gleichgewichteten Mittelwert, dem harmonischen Mittel [101] [102] [103]. Die Berechnung gestaltet sich wie folgt:

$$F1\ score = 2 \times \frac{Precision\ \times Recall}{Precision + Recall}$$

Mit dem F1-Maß werden also Precision und Recall zu einem Maßstab der allgemeinen Leistungsfähigkeit bzw. Performance abstrahiert. Der Ergebnis-Wertebereich liegt zwischen 0 und 1. Dabei ist ein hoher Wert anzustreben.

4.4.2.5 Weitere Kennzahlen

Wie bereits erwähnt, stellen Sensitivität, Genauigkeit und F1-Maß wichtige Kriterien für die Einschätzung eines Klassifikationsmodells dar. Es existieren aber noch weitaus mehr mögliche Kennzahlen, die ebenfalls Aussagen zur Güte des Klassifikators zulassen oder auch anzeigen, wie umfangreich der Test ist.

Im Rahmen des durchzuführenden Tests werden diese weiteren Kennzahlen nicht betrachtet. Das Testprogramm berechnet diese jedoch automatisiert und schreibt sie in eine Ergebnisdatei. In Quelle [100 S. 91] werden die Berechnungsgrundlagen und kurze Erklärungen bereitgestellt.

4.4.3 Datenkorpus

Im Rahmen der Evaluierung besteht die Notwendigkeit, Trainingsdaten und Testdaten zu beschaffen, welche zum einen die Domäne als solche definiert und zum anderen den

Test der Domäne auf Korrektheit gewährleistet. Im Internet existiert eine große Anzahl von öffentlichen Korpus-Datensätzen, die verwendet werden können. Für die hiesige Arbeit erfolgt die Suche nach Datensätzen, die sich in Richtung eines Bibliothekskontextes orientieren, um sowohl bei der Evaluierung, als auch für den Prototyp, realistische Beispiele bereitstellen zu können.

Das Ergebnis dieser Recherche ist ernüchternd, denn der gewünschte Kontext existiert in keiner der zugänglichen Korpus-Datenbanken. Der hier formulierte Anwendungsfall ist (noch) zu speziell und entspricht daher momentan einer Nische. Nebenher ist zu erwähnen, dass die überwältigende Mehrheit der Korpusse nur im Zusammenhang mit der englischen Sprache verwendet werden können. Als Folge dieser Erkenntnisse, wird der Datenkorpus für die Evaluierung eigenständig definiert. Die herausgearbeiteten Korpus-Datensätze sind im Übrigen im *Anhang A* aufgelistet, da sie für andere Anwendungsfälle von großer Bedeutung sein können und die Recherche-Ergebnisse für zukünftige Projekte als Grundlage dienlich sind.

Aus dem Kapitelabschnitt *3.6 Szenario* lassen sich sieben Absichten bzw. Klassen erkennen, welche die Grundlage des Datenkorpus bilden. Die Absichten lauten:

- *start_conversation*, ist der Konversationsbeginn durch einen Menschen.
- *how_to_use_library*, ob Voraussetzungen für die Benutzung der Bibliothek zu erfüllen sind, beziehungsweise wie diese generell benutzbar ist.
- *how_to_print*, inwiefern die Möglichkeit zum Drucken beststeht.
- *interrupt*, zur Unterbrechung des Systems, respektive Roboters.
- *repeat*, die Bitte nach Wiederholung der letzten Aussage.
- *end_conversation*, die Beendigung der Konversation durch den Menschen.
- *out_of_context*, repräsentiert nicht Domäne zugehörige Aussagen sowie nicht erkannte Aussagen.

Für jede der Absichten müssen beispielhafte Texte definiert werden, die ein realistisches Abbild möglicher menschlicher Äußerungen zur Erreichung der eben definierten Absichten darstellen. Zur Gewährleistung der Vergleichbarkeit von Rasa und Watson Conversation, werden beide Systeme mit identischen Beispielaussagen trainiert. Die Definition des Trainingsformates unterscheidet sich jedoch für beide NLP-Systeme voneinander[64]. Aus diesem Grund wurden zwei Ordner angelegt, die auf das jeweilige Software-System angepasst, Trainingsdateien enthalten.

[64] Rasa verlangt für jedes einzelne Testbeispiel die Angabe der damit verbundenen Absicht. Watson dagegen definiert die Absicht einmalig und erwartet eine Auflistung aller dazugehörenden Beispielaussagen. Beide Varianten verwenden standardmäßig JSON als Datenformat. Rasa bietet zusätzlich Markdown an und Watson Conversation erlaubt zusätzlich CSV.

Normalerweise existiert nur ein großer Datenkorpus, aus dem zufallsmäßig Test- und Trainingsbeispiele herausgenommen werden [105]. Aus Gründen der Nachvollziehbarkeit, sind die zum Test verwendeten Beispielaussagen aber von den im Training verwendeten getrennt. Somit können die im Rahmen dieser Arbeit durchzuführenden Tests eigenständig wiederholt und reproduziert werden.

Jede der sieben Absichten wird in drei verschiedenen Modi trainiert. Dabei basieren die Modi auf variierenden Größen der Trainingskorpusse, nämlich 5, 15 und 25 Beispiele je Absicht/Klasse. Durch dieses Vorgehen wird ersichtlich, wie die Systeme bei einer zu geringen, einer mittleren und einer empfohlenen Menge von Trainingsdaten abschneiden[65]. Auf ein noch umfangreiches Training wird verzichtet, da es schlichtweg schwierig ist auf manuellem Wege derart viele Beispiele bereitzustellen[66]. In Realität wird die Erstellung der Anwendungsdomäne mit entsprechenden Beispielen auf reale Benutzerdaten basierend vorgenommen (weitere Erläuterungen sind dem Kapitel *Fazit* zu entnehmen).

Neben den Trainingsbeispielen sind auch Testbeispiele zu definieren, die aufzeigen, wie zuverlässig die trainierten Modelle auf entsprechende Testanfragen reagieren. In diesem Zusammenhang geht es also um den Abgleich von erwarteter Absicht, definiert durch die Testdaten, und vorhergesagter Absicht, berechnet durch das trainierte Modell. Auch die Testbeispiele sind manuell festgelegt, wobei pro Absicht zehn Testanfragen vorhanden sind. Durch die bereits erläuterte indirekte Auswirkung eines jeden Testfalls, ist eine hinreichende Menge an Testdaten gegeben.

Zusammenfassend ergibt sich folgende Verzeichnisstruktur bezüglich der Trainings- und Test-Dateien für Rasa und Watson Conversation:

[65] Die Begrifflichkeit „empfohlene Menge“ ist relativ. Rasa und Watson Conversation empfehlen mindestens 20 bis 30 Beispiele zu definieren, mehr wäre aber auch denkbar bzw. schlichtweg besser.

[66] Bei der Definition weiterer Beispiele geht es vor allem um Vielfalt. Es ist nicht schwer immer weitere Beispiele zu generieren, aber die Denkweise eines jeden Menschen bewegt sich innerhalb gewisser Grenzen, welche auch durch die sprachliche Welt gekennzeichnet ist (vgl. Abschnitt *2.3.5 Konklusion (Reasoning)*). Je mehr Beispiele manuell generiert werden, desto ähnlicher werden sich diese, ergo wird das Modell zu sehr in eine Richtung beeinflusst.

```
Ordnerstruktur:                          Erklärung:

/data                                  -> Wurzel-Ordner für Modell-Dateien
  /configs                             -> Kind-Ordner für Zugangsdaten, etc.
    /...                                  bzgl. Rasa und Watson Conversation
  /test                                -> Enthält alle Test-Dateien
    TestSet.json                       -> Beispielaussagen für Modell Tests
  /training                            -> Enthält alle Trainings-Korpusse
    /rasa                              -> Korpusse im Rasa-Format
      TrainingSet_5.json               -> Korpus mit 5 Beispielen je Absicht
      TrainingSet_15.json              -> Korpus mit 15 Beispielen je Absicht
      TrainingSet_25.json              -> Korpus mit 25 Beispielen je Absicht
    /watson                            -> Korpusse im Watson-Format
      TrainingSet_5.json
      TrainingSet_15.json
      TrainingSet_25.json
```

Abbildung 54: Verzeichnisstruktur für Trainings- und Testkorpusse [Quelle: Eigenmaterial]

4.4.4 Testprogramm

Nach Festlegung des Testverfahrens und entsprechende Bewertungsparameter für die Beurteilung der beiden Software-Lösungen, wird nun eine Übersicht zum Ablauf eines Tests gegeben.

Für die sieben definierten Absichten existieren je zehn Testbeispiele, welche zugehörige Aussagen repräsentieren. Dadurch entstehen bereits 70 Überprüfungen hinsichtlich erwarteter und vorhergesagter Klasse bzw. Absicht. Entsprechend sind auch Zählungen für die Ergebniswerte True Positive, True Negative, False Positive und False Negative vorzunehmen. Zusätzlich gilt es, gleich drei Modelle zu prüfen, nämlich basierend auf 5, 15 und 25 Trainingsbeispiele je Absicht, wodurch die 70 Überprüfungen mal drei zu rechnen sind.

Da der Test einem Vergleich dient, ist dieses Vorgehen sowohl für Rasa als auch für Watson Conversation nötig, wodurch insgesamt 420 Abgleiche nötig sind. Alles in allem soll durch diese Rechnung erkennbar werden, dass ein manuelles Vergleichen zu aufwändig ist. Folglich ist ein automatisierter Testablauf notwendig.

Mit den selbstprogrammierten Python-Programmen *HelloRasa* sowie *HelloWatson* wird dieser Schlussfolgerung entsprochen. Herzstück der Implementierungen sind die Klassen *WatsonClient* bzw. *RasaClient* und *WatsonTestDataHandler* bzw. *RasaTestDataHandler*.

Im Kern sind die Client-Klassen für die Einrichtung und Verwaltung der Projekte und Modelle zuständig und übernehmen im weiteren Verlauf die Versendung der Testbeispiele und den Empfang der Antwortdaten.

Die beiden TestDataHandler-Klassen verwenden die abstrakte Klasse *AbstractTestDataHandler*, welche Methoden zur Analyse der Antwortdaten und zur Generierung von Ergebnis-Datensätzen vordefiniert. Ein einzelner Dateneintrag besteht aus den folgenden Werten:

- *sample* ist das Testbeispiel, also die analysierte Aussage dieses Dateneintrags.
- *actual* enthält, welche Absicht erwartet wurde als Ergebnis der Analyse.
- *predicted* enthält, welche Absicht durch die Analyse erkannt wurde.
- *confidence* gibt an, wie sicher das System über das analysierte Ergebnis ist.
- *duration* ist die in Millisekunden gemessene Zeitspanne von Versendung des Testbeispiels bis zum Empfang der Antwortdaten.

Die einzelnen Werte eines solchen Dateneintrags werden aus dem aktuell betrachteten Testbeispiel und der zugehörigen Antwort ausgelesen. Dies geschieht in den bereits erwähnten Methoden zur Analyse der Antwortdaten, welche in den Klassen *RasaTestDataHandler* bzw. *WatsonTestDataHandler* vorhanden sind. Alle Dateneinträge werden zur Laufzeit in eine Liste gespeichert. Diese Liste stellt somit den Ergebnis-Datensatz des Tests dar.

Nachdem alle siebzig Testbeispiele abgearbeitet worden sind, können anschließend die eigentlichen Hauptberechnungen hinsichtlich Konfusionsmatrix, Precision, Recall und F1-Maß vorgenommen werden. Nebenbei wird auch ausgerechnet, wie lange der Test insgesamt (Summe aller *duration*-Werte) und im Durchschnitt dauerte sowie für welche Testbeispiele die Analysezeit am längsten (größter *duration*-Wert) bzw. am kürzesten (kleinster *duration*-Wert) andauerte.

Aufgrund der Berechnungskomplexität von Konfusionsmatrix und zugehöriger Kennzahlen, werden die entsprechenden Parameter unter Zuhilfenahme einer Drittanbieter Software-Bibliothek kalkuliert. Ausgewählt wurde die quelloffene Software *Pandas*[67], welche speziell für Datenanalysen mittels Python geschaffen wurde. Neben komplexen Datenanalysen lassen sich Ergebnisse in verschiedensten Visualisierungsformen darstellen[68] und in diverse Formate exportieren[69]. Die Funktionen Pandas erlauben somit hinreichend viele Präsentationsmöglichkeiten der Ergebnisse.

[67] Einführung in Pandas: http://pandas.pydata.org/pandas-docs/stable/10min.html

[68] Mögliche Visualisierungen mit Pandas: http://pandas.pydata.org/pandas-docs/stable/visualization.html

[69] Standardmäßige Import/Exportformate Pandas: http://pandas.pydata.org/pandas-docs/stable/io.html

Die Berechnung der Matrixwerte übernimmt das optional zuschaltbare Modul *pandas_ml*[70]. Dieses berechnet insgesamt 26 verschiedene Kennziffern basierend auf TP, TN, FP und FN. Darin enthalten sind auch Precision, Recall und F1-Score. Die Berechnungen erfolgen stets für jede definierte Absicht respektive Klasse. Die Gesamtheit aller Kennziffern und die ursprünglichen Dateneinträge des Tests werden in eine Textdatei, genannt *results.txt*, geschrieben. Dadurch wird ein Höchstmaß an Transparenz bzgl. wie die Testergebnisse entstanden sind, gewährleistet.

Es ist jedoch relativ unübersichtlich, die gewünschten Kennziffern in der Gesamtmenge aller Kennziffern zu erkennen (vergleiche *Abbildung 55*). Daher werden TP, TN, FP, FN, Precision, Recall und F1-Score nochmals automatisch gefiltert und in ein Excel-Dokument geschrieben. Dieses enthält für jedes der getesteten Modelle eine Tabelle, welche die Ergebniswerte pro Absicht/Klasse und über alle Absichten/Klassen in deutlich übersichtlicher Art und Weise aufschlüsselt.

Classes	A	B	C	D
Population	20	20	20	20
Condition positive	0	16	3	1
Condition negative	20	4	17	19
Test outcome positive	11	7	2	0
Test outcome negative	9	13	18	20
TP: True Positive	0	6	1	0
TN: True Negative	9	3	16	19
FP: False Positive	11	1	1	0
FN: False Negative	0	10	2	1
TPR: Sensivity	NaN	0.375	0.3333333	0
TNR=SPC: Specificity	0.45	0.75	0.9411765	1
PPV: Pos Pred Value = Precision	0	0.8571429	0.5	NaN
NPV: Neg Pred Value	1	0.2307692	0.8888889	0.95
FPR: False-out	0.55	0.25	0.05882353	0
FDR: False Discovery Rate	1	0.1428571	0.5	NaN
FNR: Miss Rate	NaN	0.625	0.6666667	1
ACC: Accuracy	0.45	0.45	0.85	0.95
F1 score	0	0.5217391	0.4	0
MCC: Matthews correlation coefficient	NaN	0.1048285	0.326732	NaN
Informedness	NaN	0.125	0.2745098	0
Markedness	0	0.08791209	0.3888889	NaN
Prevalence	0	0.8	0.15	0.05
LR+: Positive likelihood ratio	NaN	1.5	5.666667	NaN
LR-: Negative likelihood ratio	NaN	0.8333333	0.7083333	1
DOR: Diagnostic odds ratio	NaN	1.8	8	NaN
FOR: False omission rate	0	0.7692308	0.1111111	0.05

Abbildung 55: Darstellung der 26 berechneten Kennziffern durch pandas_ml [Quelle: Eigenmaterial]

[70] Übersicht des Zusatzmoduls pandas_ml: http://pandas-ml.readthedocs.io/en/stable/modelframe.html

Die Betrachtung der Konfusionswerte wird von einer grafischen Konfusionsmatrix abgerundet. Diese bietet eine effektive Sichtweise auf die Ergebnisse der Klassifizierung durch die einzelnen Modelle. (vergleiche *Abbildung 52)*.

Zusammenfassend entsteht bei der Testdurchführung zur Überprüfung der Robustheit und Performance trainierter Modelle die folgende Verzeichnisstruktur der generierten Ergebnisdateien:

```
Ordnerstruktur:                              Erklärung:

/test_remote_results                        -> Wurzel-Ordner für Ergebnis-Dateien
  /TrainingSet_5_TestSet                    -> Ergebnis-Ordner für 5er-Modell
    results.txt                             -> Dateneinträge + Kennziffern
    conf_matrix_intents.png                 -> Grafisch visualisierte Konf-Matrix
    conf_matrix_intents_normalized.png      -> Wertnormalisierte graf. Konf-Matrix
  /TrainingSet_15_TestSet
    results.txt
    conf_matrix_intents.png
    conf_matrix_intents_normalized.png
  /TrainingSet_25_TestSet
    results.txt
    conf_matrix_intents.png
    conf_matrix_intents_normalized.png
  xls_intents.xlsx                           -> Excel-Dokument über alle Ergebnis-
                                                ordner zur Filterung der Kennziffern
                                                TP, TN, FP, FN, Precision, Recall
                                                und F1-Score.
```

Abbildung 56: Verzeichnisstruktur der Ergebnisdateien des Testprogramms [Quelle: Eigenmaterial]

Es lässt sich erkennen, dass bei einem Test von drei Modellen insgesamt zehn Dateien entstehen. Durch die Auflistung aller Dateneinträge mit sample, actual, predicted, confidence und duration in der results.txt-Datei lässt sich transparent nachvollziehen, wie die Ergebnisse entstanden sind.

4.4.5 Ergebnisse des Tests

Basierend auf die Programmierung eines Testprogramms inklusive der Definition eines Datenkorpus, ist anschließend die Durchführung verschiedener Tests bezüglich trainierter Modelle möglich.

Wie bereits in den vorhergehenden Abschnitten beschrieben, erfolgen Tests basierend auf trainierte Modelle mit 5, 15 und 25 Trainingsbeispielen pro Absicht. Insgesamt sind dabei sieben verschiedene Absichten zur Klassifizierung angelegt worden. Rasa und

Watson Conversation werden jeweils mit Hilfe der erstellten Korpus-Dateien trainiert und anschließend auf korrekte Klassifizierung überprüft.

Sämtliche Testergebnisse mit allen Einträgen, Tabellen, Statistiken und Visualisierungen können in der beiliegenden CD im Verzeichnis *~/Eval/Ergebnisse/* eingesehen werden.

4.4.5.1 Grobvergleich der Performance

Es bietet sich an, zunächst einen abstrakten Blick auf die Gesamt-Performance beider Software-Bibliotheken zu werfen. Entsprechend zeigen *Tabelle 10* und *Tabelle 11* die Verteilung von TP, TN, FP, FN und das daraus resultierende F1-Maß als Indikator für die Performance-Entwicklung über alle drei Modelle.

Tabelle 10: Rasa Performance-Werte über die verwendeten Modelle.

Testversuch	TP	TN	FP	FN	Precision	Recall	F1-Score
Set 5	27	377	43	43	0,39	0,39	0,39
Set 15	51	401	19	19	0,73	0,73	0,73
Set 25	56	406	14	14	0,80	0,80	0,80

Tabelle 11: Watson Conversation Performance über die verwendeten Modelle.

Testversuch	TP	TN	FP	FN	Precision	Recall	F1-Score
Set 5	45	395	25	25	0,64	0,64	0,64
Set 15	56	406	14	14	0,80	0,80	0,80
Set 25	61	411	9	9	0,87	0,87	0,87

Zunächst auffallend, ist die stetige Gleichheit von Precision, Recall und F1-Score innerhalb der Tests, sowohl bei Rasa als auch Watson Conversation. Es handelt sich hierbei nicht um einen Berechnungsfehler. Viel mehr zeigt dieses Verhalten, dass der Datenkorpus künstlicher Natur ist[71]. Precision und Recall entfalten jedoch erst bei unausgewogenen Datensätzen, mit einer tendenziellen Ähnlichkeit der Beispiele, eine wahrhaftige Aussagekraft (vgl. [106]). Mit anderen Worten: Die stetig gleichmäßige

[71] Zur Erinnerung: Die Beispiele wurden manuell generiert, entsprechen also nicht der Bandbreite realer Äußerungen.

Verteilung der Testdaten – je Absicht werden zehn Zuordnungen erwartet – gepaart mit den eindeutig differenzierbaren Beispielformulierungen, provoziert nur sehr wenige falsche Treffer oder unvollständige Treffer. Ungeachtet dessen sind die Werte dennoch allesamt verwertbar.

Der Blick auf das F1-Maß, als Indikator für die Performance, zeigt eine erwartungsgemäße Verbesserung der Erkennungsrate bei wachsendem Trainingskorpus. Dieser Trend lässt sich vor allem mittels eines Liniendiagramms gut erkennen. Ein solches wird mit *Abbildung 57* bereitgestellt.

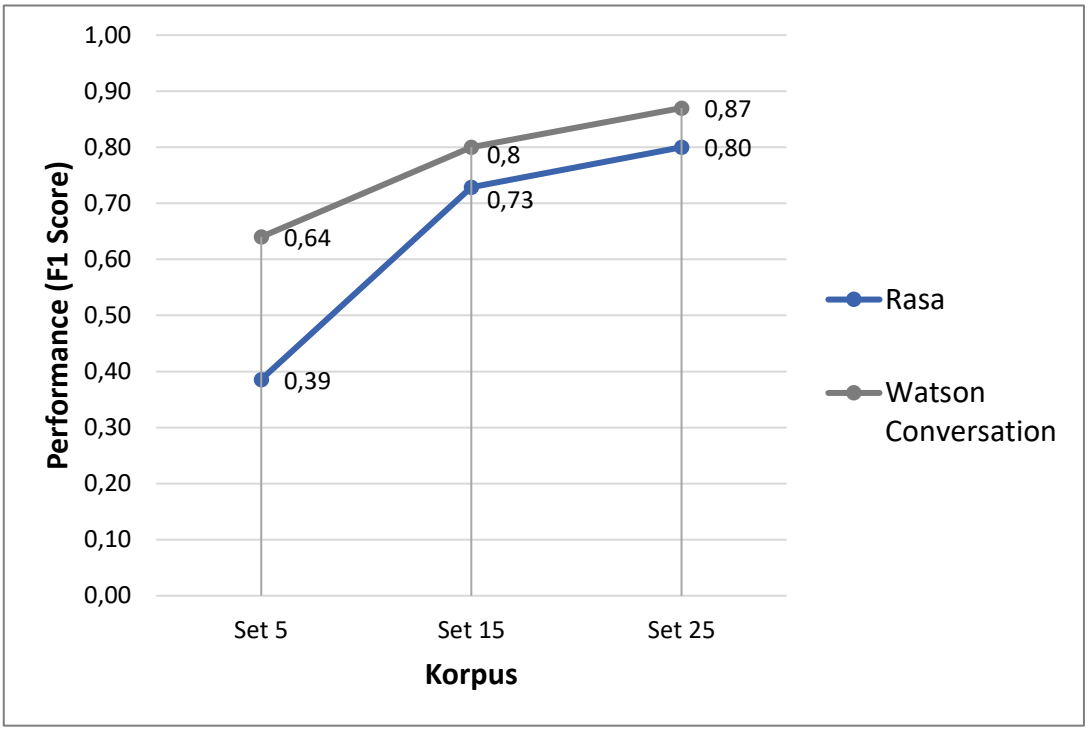


Abbildung 57: Vergleich der Performance von Rasa und Watson Conversation. [Quelle: Eigenmaterial]

Beide NLP-Systeme empfehlen als Mindestmenge 20 bis 30 Beispiele je Absicht für das Training bereitzustellen. Daher ist es wenig überraschend, dass beide Systeme bei einer absichtlich starken Unterschreitung dieser Empfehlung in schlechten Erkennungsraten münden (vgl. Set 5). Mit 0,64 erreicht Watson Conversation sogar ein respektables Ergebnis. Beim Vergleich mit dem von Rasa erreichten Wert von 0,39 wird dieser Eindruck nochmals verstärkt.

Der zweite Test, mit einer nur knappen Unterschreitung der empfohlenen Mindestmenge, lässt beide NLP-Systeme deutlich besser dastehen. Besonders Rasa überrascht mit einem enormen Performance-Sprung um 0,34 Einheiten (vorher: 0,39; jetzt: 0,73). Mit diesem Ergebnis kann Rasa den Rückstand auf Watson Conversation (vorher: 0,64; jetzt: 0,8) deutlich verkleinern. Dies liegt auch an einem deutlich kleineren Performance-Sprung von 0,16 Einheiten bei Watson Conversation.

Mit dem dritten Test, genannt Set 25, wird die Empfehlung bzgl. der Mindestmenge des Korpus erstmalig von Rasa und Watson Conversation eingehalten. Das Performance-Wachstum beträgt für beide NLP-Systeme 0,07 Einheiten, wodurch Rasa nun einen F1-Score von 0,8 vorweisen kann und Watson Conversation 0,87 erreicht. Entsprechend lässt sich sagen, dass Watson selbst bei einer leichten Unterschreitung der empfohlenen Trainingsdatenmenge bereits eine Performance erreicht, die bei Rasa erst mit Erfüllung dieser Grenze machbar wird. Zeitgleich profitiert Watson Conversation bei der Erhöhung der Korpusdaten von 15 auf 25 Beispielaussagen je Absicht aber nicht stärker als Rasa.

Insgesamt schafft es Watson Conversation mit dem gegebenen Trainingskorpus eine bessere Performance zu erreichen als Rasa. Dennoch ist für beide NLP-Systeme noch Verbesserungspotential gegeben. Dafür sind jedoch zwingend mehr Beispieldaten notwendig, welche auf realen Äußerungen basieren sollten.

4.4.5.2 Detailvergleich der Performance

Neben dem Blick auf die Gesamtperformance beider NLP-Systeme, ist auch ein detaillierter Blick auf die konkreten Kennziffern der Klassen bzw. Absichten von Interesse. Dieser Detailblick gibt Auskunft, welche Absichten bereits hinreichend zugeordnet werden und welche Absichten bessere/mehr Trainingsdaten benötigen.

Zunächst der Blick auf die Konfusionsmatrizen für Rasa (*Abbildung 58*) und Watson Conversation (*Abbildung 59*) in Abhängigkeit zur Korpusgröße von 25 Beispielen je Absicht.[72]

[72] Die Konfusionsmatrizen für Set 5 und Set 15 liegen auf der CD im Verzeichnis *~/Eval/Ergebnisse/* ebenfalls bei. Aufgrund der schlechten Gesamt-Performance werden sie an dieser Stelle nicht weiter betrachtet.

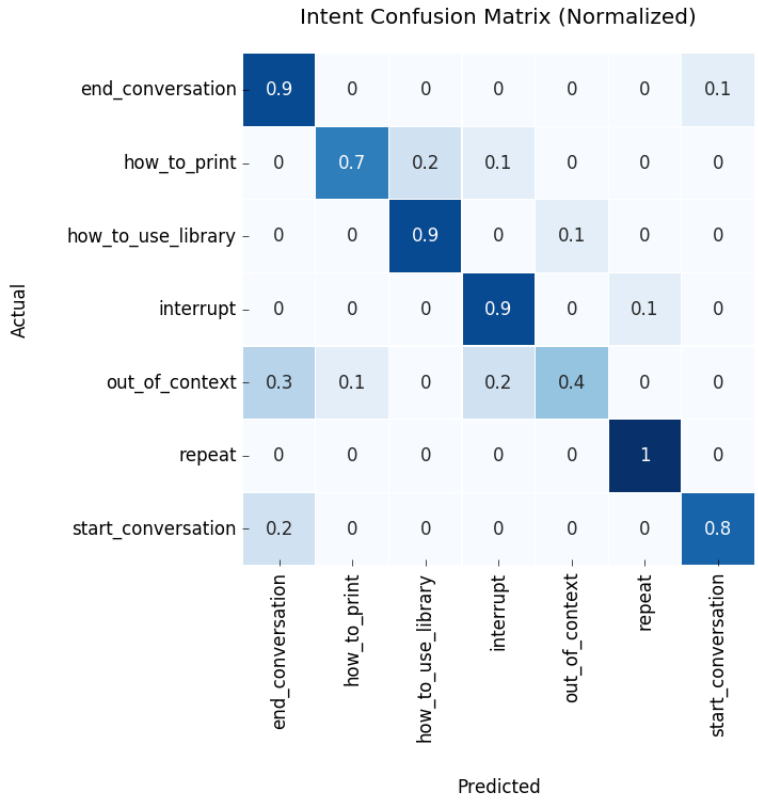


Abbildung 58: Normalisierte Konfusionsmatrix für Set 25 bei Rasa. [Quelle: Eigenmaterial]

Bereits bei einer kurzen Betrachtung offenbaren sich bzgl. Rasa häufige Abweichungen von der Diagonale. Dies bedeutet, dass es zu einer nicht zu unterschätzenden Ausprägung von False Positives und False Negatives kommt. Insbesondere out_of_context, aber auch *how_to_print* und *start_conversation* werden von Rasa mit anderen Absichten verwechselt.

Abweichungen dieser Art sind bei Betrachtung der Diagonalen von Watson Conversation kaum zu finden. Der Recall-Wert, also die Vollständigkeit der Treffer, beträgt für fünf von sieben Absichten (*end_conversation, how_to_print, how_to_use_library, interrupt, repeat*) sogar 100% - bei Rasa ist dies nur einmal der Fall (*repeat*). Dennoch sorgt *out_of_context* auch bei Watson Conversation für massive Probleme. Die korrekte Zuordnung entsprechender Absichten ist mit 20% sogar noch schlechter als bei Rasa (40%).

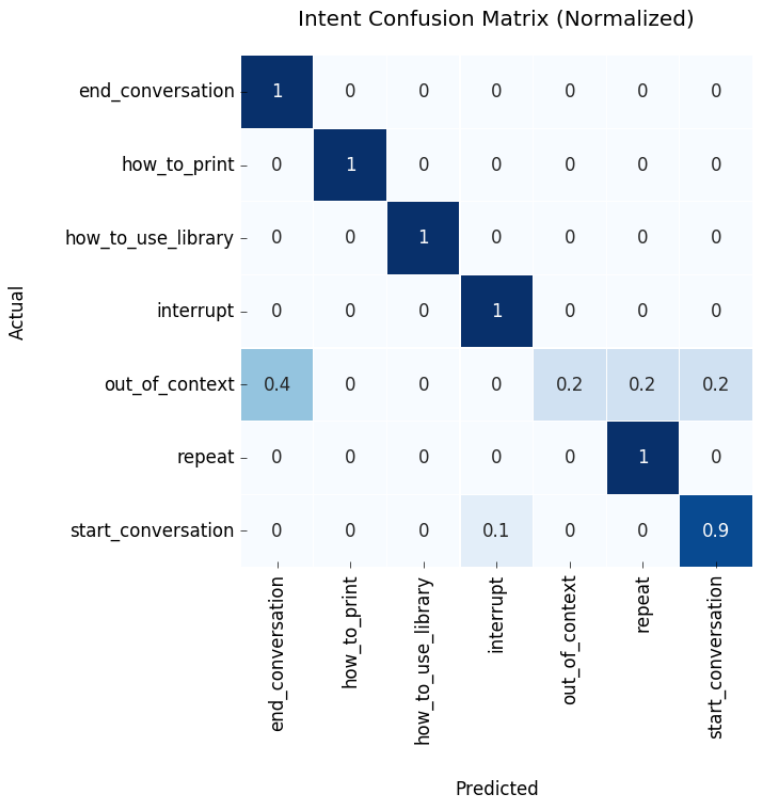


Abbildung 59: Normalisierte Konfusionsmatrix für Set 25 bei Watson Conversation. [Quelle: Eigen]

Es zeigt sich, dass beide NLP-Systeme, anhand der verwendeten Trainingsdaten, keine guten Erkennungsraten für außerhalb der Anwendungsdomäne liegende Absichten (*out_of_context*) gewährleisten können. Dieser Eindruck wird bei Betrachtung der Ergebnis-Kennziffern in zugehörigen Tabellen für Rasa (*Tabelle 12*) und Watson Conversation (*Tabelle 13*) bestätigt.

Anhand der Tabellen lässt sich, im Gegensatz zum Grobvergleich, diesmal auch eine Unterscheidung zwischen Precision, Recall und F1-Score erkennen. Die als *out_of_context* zugeordneten Beispieläußerungen sind mit 80% bei Rasa und 100% bei Watson Conversation auch tatsächliche Treffer, die Precision ist folglich hoch. Auf der anderen Seite ist die Erkennung aller möglichen korrekten *out_of_context* Zuordnungen (Recall) mit 40% bzw. 20% ziemlich niedrig. Dadurch ist das Modell hinsichtlich der Absicht *out_of_context* eher defensiv.

Diese Feststellung steht im Widerspruch zur generellen Ausrichtung der Modelle. Rasa und Watson Conversation entsprechen durchaus soliden Modellen, da Precision und Recall durchschnittlich hoch sind.

Tabelle 12: Ergebniswerte der Absichten bei Set 25 für Rasa.

Absicht	TP	TN	FP	FN	Precision	Recall	F1-Score
end_conversation	9	55	5	1	0,643	0,900	0,750
how_to_print	7	59	1	3	0,875	0,700	0,778
how_to_use_library	9	58	2	1	0,818	0,900	0,857
interrupt	9	57	3	1	0,750	0,900	0,818
out_of_context	4	59	1	6	0,800	0,400	0,533
repeat	10	59	1	0	0,909	1,000	0,952
start_conversation	8	59	1	2	0,889	0,800	0,842
Summe	56	406	14	14			
Total					0,800	0,800	0,800

Tabelle 13: Ergebniswerte der Absichten bei Set 25 für Watson Conversation.

Absicht	TP	TN	FP	FN	Precision	Recall	F1-Score
end_conversation	10	56	4	0	0,714	1,000	0,833
how_to_print	10	60	0	0	1,000	1,000	1,000
how_to_use_library	10	60	0	0	1,000	1,000	1,000
interrupt	10	59	1	0	0,909	1,000	0,952
out_of_context	2	60	0	8	1,000	0,200	0,333
repeat	10	58	2	0	0,833	1,000	0,909
start_conversation	9	58	2	1	0,818	0,900	0,857
Summe	61	411	9	9			
Total					0,871	0,871	0,871

Es handelt sich bei *out_of_context* um einen Ausreißer, der die Folge von schlechten Trainingsbeispielen ist. Es offenbart sich mit *out_of_context* ein generelles Problem: NLP-Systeme versuchen Muster in den Trainingsdaten zu finden, was bei einer extremen Variabilität der Beispiele kaum möglich ist. Als Folge dessen, ist bei einer Anwendung nicht nur auf die Absichtszuordnung durch das System zu vertrauen, sondern zusätzlich der berechnete Konfidenzwert einzubeziehen und auf den aktuellen Zustand des Dialoges zu achten.


```
{ 'sample': 'Weil es regnet wird die Erde nass', 'actual': 'out_of_context',
'predicted': 'end_conversation', 'confidence': '0.271712149255', 'duration':
'48.9799976349'}
```


Abbildung 60: Rasa Dateneintrag des Satzes „Weil es regnet wird die Erde nass“ [Quelle: Eigen]

Ein Beispiel anhand von *Abbildung 60*: Der Satz *„Weil es regnet wird die Erde nass“* wurde *end_conversation* zugeordnet, soll also, laut Rasa, eine Äußerung zur Beendigung einer Konversation sein. Es ist jedoch offensichtlich, dass diese Äußerung zu keiner der Absichten mit Bezug auf die Bibliothek passt und daher schlichtweg unsinnig für die Anwendungsdomäne ist. Die Konfidenz der *end_conversation*-Einschätzung durch Rasa beträgt 0,27 also 27%. Dies ist im Vergleich mit korrekten Einschätzungen ein sehr schlechter Wert (vgl. *results.txt* für Rasa und Watson Conversation). Entsprechend könnte die Zuordnung durch Programmierlogik abgelehnt werden. Der Beispielsatz steht außerdem für sich alleine, das heißt, ein solcher wäre die erste Aussage in einer realen Konversation. Die Logik einer jeden sinnvollen Kommunikation schließt die Absicht, ein Gespräch mit einer Verabschiedung zu beginnen, aus.

4.4.5.3 Latenz

Vergleiche hinsichtlich der Erkennungsraten von NLP-Systemen sind ohne Zweifel das wichtigste Instrument einer Evaluierung solcher Systeme. Es besteht dennoch die Möglichkeit weitere Messwerte einzubeziehen, welche eine breitere Auslegung der Bewertung bilden.

Im hiesigen Fall führt die Verwendung des NLP-Systems in einer realen Anwendung zur Notwendigkeit, die menschlichen Äußerungen möglichst in Echtzeit zu verarbeiten. Der Anforderungsabgleich hat für die entsprechende Anforderung */SW0040/* jedoch insofern für Ernüchterung gesorgt, als dass beide Systeme nicht in der Lage sind, Wort für Wort einen Analyseprozess aufzubauen, der mit jedem dieser Worte wächst. Daher ist es unerlässlich stets Äußerungen als Gesamtkonstrukt zur Analyse zu übergeben.

Die Bedingung der Echtzeitverarbeitung existiert vor allem deshalb, weil in einer menschlichen Kommunikation in der Regel keine sekundenlangen Pausen zur Verarbeitung des Gesagten entstehen. Menschen erwarten eine quasi unverzügliche Reaktion des Gesprächspartners, also eine Echtzeit-Kommunikation.

Konkrete Zahlen liefert das auf Computer-Benutzerschnittstellen spezialisierte Beratungsunternehmen Nielsen Norman Group (NN/g). Laut NN/g müssen folgende Kennziffern bzw. Intervalle beachtet werden [107]:

- Eine Reaktion innerhalb 0,1 Sekunden bzw. 100 Millisekunden fühlt sich für Menschen augenblicklich an. Die Latenz bleibt quasi unerkannt.
- Die Reaktion innerhalb einer Sekunde korreliert zur maximalen Verarbeitungszeit von Menschen. Benutzer des Systems können die Latenz zwar manchmal bemerken, akzeptieren diese aber vollkommen und haben insgesamt das Gefühl, unverzüglich mit dem System zu arbeiten.
- Bei einer Reaktionszeit zwischen einer bis maximal zehn Sekunden werden Menschen unruhig und ungeduldig. Das System wird als reaktionslahm und nicht effektiv empfunden.
- Latenz-Werte über zehn Sekunden lassen Benutzer ungehalten werden. Es folgen ein kompletter Aufmerksamkeitsverlust und der freiwillige Verzicht zur Benutzung des Systems.

Mit dem Latenztest für Rasa und Watson Conversation werden nun entsprechende Messwerte erhoben. Dabei werden für jedes Datenset Minimum, Maximum, Summe und Durchschnitt der Latenz gemessen. Um mögliche Ausreißer aufzufangen, werden die Messungen jeweils zehnmal wiederholt. Die Gesamtergebnisse der Latenztests sind in *Tabelle 14* für Rasa und *Tabelle 15* für Watson Conversation gegeben.

Tabelle 14: Ergebnisse der Latenzmessungen für Rasa.

Zeit / Korpus	**Set 5**	**Set 15**	**Set 25**
Minimum	44 ms	42 ms	43 ms
Maximum	309 ms	310 ms	54 ms
Summe	4304 ms	4250 ms	3759 ms
Durchschnitt	61 ms	61 ms	54 ms

Die Latenzmessungen offenbaren für Rasa eine stetige Einhaltung des menschlichen Akzeptanzlevels von einer Sekunde, selbst bei den Maximalwerten. Im Durchschnitt wird sogar die als augenblickliche Verarbeitung geltende Verzögerung eingehalten. Rasa erreicht somit überraschenderweise eine echtzeitgemäße Latenz.

Interessanterweise sinkt die Maximal-Latenz bei größerem Datenkorpus (Set 25). Es wäre denkbar, dass der vergrößerte Korpus dem Modell schnellere Entscheidungen ermöglicht. Dieser Gedankengang steht jedoch im Widerspruch zur Entwicklung von Set 5 zu Set 15. Die Maximalwerte sind nahezu identisch. Die Betrachtung darf nicht vernachlässigen, dass die eben genannten Sets unter der empfohlenen Mindestmenge von Trainingsbeispielen liegen. Folglich sind die Modelle nicht ausreichend trainiert. Es ist daher davon auszugehen, dass Set 5 und Set 15 nicht repräsentativ sind und ausreichend trainierte Modelle ähnliche Werte der Maximal-Latenz vorweisen.

Tabelle 15: Ergebnisse der Latenzmessungen für Watson Conversation.

Zeit / Korpus	**Set 5**	**Set 15**	**Set 25**
Minimum	406 ms	403 ms	413 ms
Maximum	2086 ms	2232 ms	2647 ms
Summe	47475 ms	46593 ms	45218 ms
Durchschnitt	678 ms	666 ms	646 ms

IBM Watson Conversation schafft ebenfalls die Einhaltung des menschlichen Akzeptanzlevels von einer Sekunde, zumindest für Minimum-Werte und im Durchschnitt. Im Maximum wird der menschliche Akzeptanzwert von einer Sekunde stetig überschritten. Dabei steigt der Maximalwert mit wachsendem Datenkorpus an (2086 ms, 2232 ms, 2647 ms). Die Annahme, ein größerer Datenkorpus führt zu längeren Entscheidungswegen, kann mit Blick auf die durchschnittliche und summierte Reaktionszeit entkräftet werden. Diese steigen nämlich nicht gleichermaßen an.

Watson Conversation schafft es zu keinem Zeitpunkt eine augenblickliche Reaktionszeit einzuhalten. Dagegen wird die von Menschen akzeptierte Zeitspanne von maximal einer Sekunde im Durchschnitt nicht verfehlt. Es werden aber durchaus kritische Reaktionszeiten im Maximum erreicht, die für eine beginnende Unruhe sorgen können.

Im Übrigen ist zu erwähnen: Die Messung der Latenzzeit umfasst den Zeitraum von Versendung der textbasierten Äußerung, bis hin zum Empfang der zugehörigen

Datenantwort. Wie lange der Verarbeitungsprozess als solches benötigt, wird daher nicht gemessen. Ein Blackbox-Test sieht eine Betrachtung innerer Systemkomponenten nicht vor[73].

Damit einhergehend stehen die Latenzmessungen in Abhängigkeit zum Netzwerk. Für Rasa sind die Latenzwerte auch deshalb deutlich niedriger, weil sich Client und Server im hochschuleigenen Campusnetzwerk befinden und somit innerhalb desselben Netzwerkes kommunizieren. Dagegen müssen Daten, welche für die Watson Conversation Instanz vorgesehen sind, das hochschuleigene Netzwerk verlassen und über das Internet kommuniziert werden. Die Möglichkeit zur lokalen Installation und die damit zusammenhängende physische wie logische Nähe von Client und Server, beschreiben den größten Vorteil von Rasa gegenüber Watson Conversation während der Betrachtung der Latenz. Durch die Handhabung von Watson Conversation als externer Dienst, kann dieser Nachteil nie vollends ausgeglichen werden.

4.5 Erkenntnisse und Auswahl

Unter Anwendung verschiedener Vergleichstechniken, gelang es mit den bisherigen Abschnitten des Kapitels Evaluierung von einer großen Anzahl Kandidaten zu einem finalen Vergleich zweier am ehesten passender NLP-Systeme zu gelangen. Der dabei bestrittene Weg von der Vorauswahl, über die Vorstellung der Software-Lösungen und dem zugehörigen Anforderungsabgleich, bis hin zu einem eins gegen eins Testvergleich hat viele Erkenntnisse hervorgebracht. Basierend auf diese Erkenntnisse, gilt es nun eine Wahl zu treffen, welche der beiden NLP-Systeme die bessere Chance auf einen erfolgreichen Prototyp für die Technische Hochschule Wildau bietet.

Mit Betrachtung der Anforderungen hatte sich ein kleiner Vorteil für Rasa ergeben. Vor allem die Punkte Open Source Lizenz, Lokale Installation und geringe Kosten sind dafür ausschlaggebend. In diesem Zusammenhang fällt auf, mit dem Anforderungsabgleich wurden mehrheitlich Unterschiede bei den nichtfunktionalen Anforderungen sichtbar. Es ist eher die unternehmerische Philosophie, die Rasa und Watson Conversation unterscheidet.

Die technische Sichtweise offenbart vor allem Gemeinsamkeiten. Beide NLP-Systeme bilden mit Intent, Entity, Context und Dialog identische Schlüsselkonzepte ab. Beide erzeugen notwendige Domäne-Modelle aus sehr ähnlichen Trainingsansätzen. Beide ermöglichen die Definition eigener Entitäten, verfügen aber auch über vorgefertigte

[73] Für Watson Conversation ist generell nur das Blackbox Testverfahren anwendbar, da Zugriff auf den Quellcode, inklusive der Einfügung von Zeitmessungen innerhalb einzelner Prozessabläufe, nicht möglich ist.

System-Entitäten. Beide erlauben eine Dialogkommunikation mit Slots, Platzhaltern und vorgefertigten Antworten, durch die eine Semi-Dynamik entsteht. Es sind im Grunde nur zwei technische Unterschiede aufgefallen: Rasa ermöglicht, neben den festen Dialogablaufplänen, zusätzlich das Training von Antworten. Diese Möglichkeit bietet Watson Conversation nicht. Dafür erlaubt Watson Conversation durch Fuzzy Matching einen robusteren Umgang mit leicht fehlerbehafteten Aussagen, was wiederum von Rasa nicht unterstützt wird.

Im Übrigen bieten beide Kandidaten gleichermaßen keinerlei technische Realisierung der Phonologie-Komponente eines NLP-Systems. Dialoge verlaufen ausschließlich textbasiert. Als Folge dessen, ist für den Prototyp eine Lösung für diese Problemstellung zu entwickeln, so dass Benutzer und Roboter in verbaler Form miteinander kommunizieren.

Die Einbeziehung der Performance-Werte zeigt einen klaren Vorteil für Watson Conversation auf, denn alle trainierten Modelle sind besser als die von Rasa. Es ist diesbezüglich anzunehmen, dass die Fuzzy Matching Methodik ein Teil dazu beiträgt. Ganz klar ersichtlich geworden ist die Notwendigkeit über einen ausreichend großen Trainingskorpus zu verfügen, ohne den keine guten Modelle erzeugbar sind. Prinzipiell bieten beide Systeme das Potential zur Erzeugung guter Klassifizierungsmodelle.

Durch die Nähe von Client und Server im Campusnetzwerk der Hochschule verbucht Rasa bei den Latenzwerten einen Vorteil. Die Installation des Servers wäre auch direkt auf dem Roboter vorstellbar, wodurch die Latenz-Werte noch weiter sinken würden. Mit Watson Conversation ist ein solches Vorgehen nicht möglich. Die Verwendung als externer Dienst führt zwangsläufig zu höheren Latenzwerten und einer insgesamt geringeren Flexibilität im Umgang mit der Software.

Über alle Faktoren betrachtet, überwiegen die Vorteile für Rasa. Aus Sicht der Hochschule bietet ein quelloffenes System deutlich mehr Konfigurationsmöglichkeiten und das Potential für weitere wissenschaftliche Arbeiten zu verschiedenen Teilaspekten des NLP-Systems. Die Vermeidung von Kosten stellt auch einen wichtigen Faktor dar, denn entsprechende Forschungsprojekte haben stets nur begrenzte finanzielle Mittel. Rasa entspricht schlichtweg mehr den Ansprüchen hinsichtlich wissenschaftlicher Forschungen.

Die Summe der Erkenntnisse lässt Rasa als Sieger dastehen. Eigentlich. Rasa macht vor allem ein Problem deutlich: Ohne ausreichende Trainingsdaten entstehen keine guten Domäne-Modelle! Die Erkennungsraten von geäußerten Absichten durch die Benutzer des Systems müssen aber zwanghaft hoch sein, da andernfalls das Interesse am Roboter als Bibliothekar unverzüglich abnimmt. Es lässt sich in diesem Zusammenhang ein

Vergleich mit menschlichen Verhaltensweisen bei unterschiedlichen Latenz-Werten aufstellen. So wie hohe Latenzen zu einem Desinteresse und dem Verzicht zur Nutzung des Systems führen, so gilt dies auch für schlechte Erkennungsraten in der Absichtsanalyse.

Personen, die das Roboter-System verwenden, sind am schnellen Informationsaustausch interessiert. Falsche Analyse-Ergebnisse und ständige Wiederholungen der ursprünglichen Äußerung werden zu einer schnellen Ablehnung des Systems führen. In der aktuellen Situation gilt es aber ein hohes Interesse zur Benutzung des Roboters bei den Studierenden zu generieren.

Mehr Trainingsdaten auf manuellem Wege zu beschaffen ist nicht zielführend, da die Abbildung möglicher Äußerungen verfälscht wird und die Auseinanderhaltung immer schwieriger wird. Nahezu alle betrachteten NLP-Systeme weisen darauf hin, im Vorfeld der Definition einer eigenen Anwendungsdomäne, auf bereits vorhandene Benutzerdaten zuzugreifen und diese als Grundlage der NLP-Anwendung zu verwenden. Dies gilt insbesondere für mögliche Formulierungen, also Trainingsbeispiele. Solche Datensätze wurden bisher weder von der TH Wildau noch von der hochschuleigenen Bibliothek angelegt.

Die hier vorliegende Arbeit stellt somit den Beginn für ein solches Vorgehen dar. Es bieten sich Mitarbeiterbefragungen und Prototyp-Feldtests als erste Maßnahmen an. Für die Feldtests sind Modelle mit guten Erkennungsraten vorzuziehen, um den Aufmerksamkeitsverlust bei den Studierenden möglich zu verhindern.

Insgesamt stellt Watson Conversation zunächst die bessere Wahl dar. Zumindest für die Startphase erster Feldtests. Mit relativ geringem Aufwand[74] lassen sich effektive Modelle entwickeln, welche den eben genannten Prozess besser unterstützen. Mit Verwendung de Lite-Preisplans stehen 10.000 API-Anfragen pro Monat kostenlos zur Verfügung, somit entstehen zunächst keine zusätzlichen Kosten[75].

Basierend auf die Auswertung der Daten des Feldtests ist zu einem späteren Zeitpunkt der Wechsel von Watson Conversation und Rasa denkbar. Eine konkrete Handlungsempfehlung wird im Kapitel *Fazit* bereitgestellt.

[74] Die Einrichtung und Verwendung ist bei Watson Conversation einfacher als bei Rasa. Vor allem die Weboberfläche zur Erstellung der Anwendungsdomäne gibt Watson Conversation einen Vorteil.

[75] Die TH Wildau verwendet den von Google entwickelten Dienst Maps. Dieser bietet ebenfalls 10.000 API-Anfragen pro Monat kostenlos an. Laut Mitarbeitern der TH Wildau reicht diese Anzahl an API-Anfragen monatlich aus.

5 Prototypische Umsetzung

In den vorhergehenden Kapiteln wurden Erkenntnisse gewonnen und Entscheidungen getroffen, die zur Wahl eines NLP-Systems zur Einbettung dessen in das Pepper-Robotersystem geführt haben. Die dabei investierte Arbeitsleistung betrifft nicht nur die Vorbereitung und Durchführung des Auswahlverfahrens, sondern ist mit großem Anteil bereits in kleinere Implementierungsversuche geflossen. Mit dem hiesigen Kapitel werden theoretische Grundlagen und konkrete Implementierungsversuche zusammengeführt. Es gilt eine umspannende Systemarchitektur zu entwerfen, die anschließend prototypisch realisiert wird und auf Funktionsfähigkeit zu testen ist. Im Ergebnis wird jedoch kein Produkt entstehen, welches sofort als vollwertiger Roboter-Bibliothekar eingesetzt werden kann. Der Prototyp gibt lediglich einen Lösungsweg an, wie das Roboter-System mit einem NLP-System verwoben werden kann.

5.1 Systemarchitektur

Durch das vorhergehende Kapitel *Evaluierung* ist ersichtlich geworden, dass Watson Conversation auf mittelfristige Sicht durchaus von Rasa oder einer völlig anderen Software-Bibliothek ersetzt werden könnte. Für das Gesamtsystem, welches Software-Bestandteile des Roboters mit der NLP-Software kombiniert, ist daher eine stark unabhängige Modularität dieser Bestandteile wichtigstes Ziel.

Im Prinzip besteht der Unterbau der Anwendung aus standardmäßigen Python-Klassen, Drittanbieter-Klassen und Pepper-Klassen. Die Gesamtheit dieser Klassentypen ermöglicht es, menschliche Spracheingaben zu erkennen, diese in Text umzuwandeln, den entstandenen Text als Absichten, Entitäten und einer Dialogantwort darzustellen und die textbasierte Antwort im letzten Schritt in verbaler Form auszugeben. Diese einzelnen Aufgabenteile, bieten dabei eine ideale Grundlage zur geforderten modularen Aufteilung des Systems in eigene Programmklassen.

Auf höchster Abstraktionsebene entsteht so ein Konstrukt an Programmklassen, welches sich aus abstrakten Basisklassen, davon erbende Klassen und koordinierende Klassen zusammensetzt. In *Abbildung 61* kann die entstandene Grob-Systemarchitektur eingesehen werden. Anhand dieser erfolgen weitere Erläuterungen.

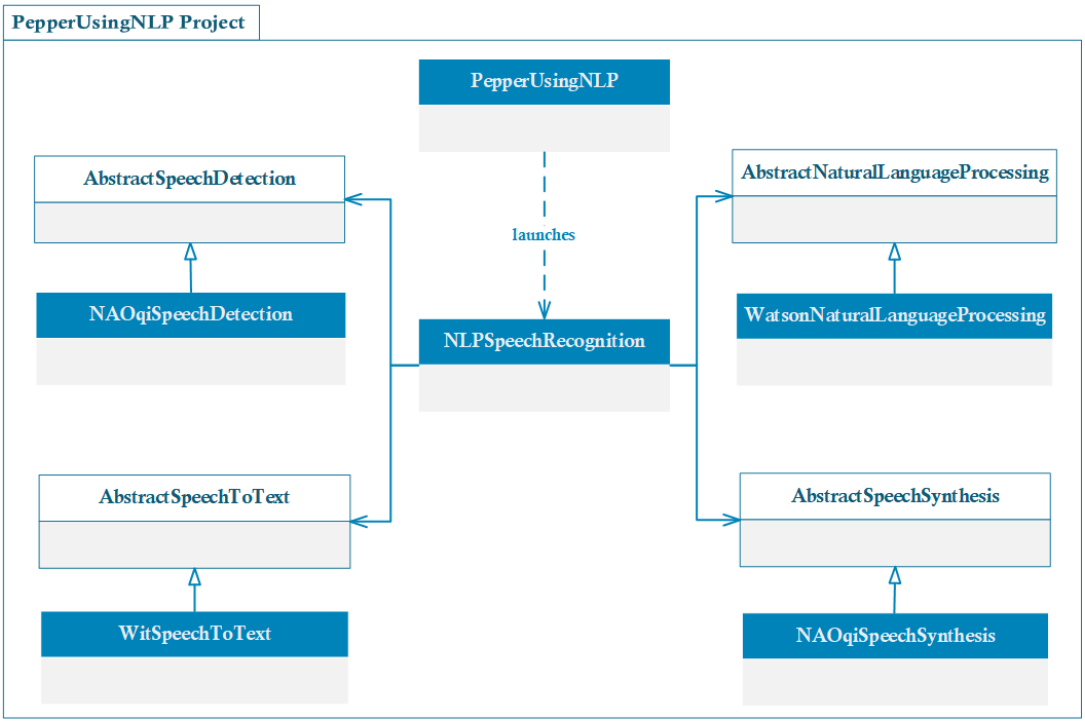


Abbildung 61: Allgemeine Systemarchitektur der prototypischen Umsetzung. [Quelle: Eigenmaterial]

Das als *PepperUsingNLP* bezeichnete Projekt besteht im Kern aus zehn Programmklassen, die den Prozess der NLP basierten Sprachverarbeitung im Pepper-Roboter realisieren. Mit den Klassen AbstractSpeechDetection, AbstractSpeechToText, AbstractNaturalLanguageProcessing und AbstractSpeechSynthesis existieren abstrakte Basisklassen, welche vor allem als Schnittstellen-Beschreibung[76] für konkrete Implementierungen der zuvor angesprochenen Teilaufgaben dienen. Die Aufgabenverteilung gestaltet sich diesbezüglich wie folgt:

- *AbstractSpeechDetection* übernimmt die Spracherkennung und Aufzeichnung von menschlichen Äußerungen über die Mikrofone des Roboters.
- *AbstractSpeechToText* wandelt die aufgezeichneten Äußerungen in eine textbasierte Repräsentationsform um.
- *AbstractNaturalLanguageProcessing* enthält die Umwandlung des Textes zu Absichten, Entitäten und Dialogantworten.

[76] Die Programmiersprache Python beschreibt standardmäßig keinen Ansatz für Interface basierte Schnittstellen. An Stelle dieser können abstrakte Klassen verwendet werden.

- *AbstractSpeechSynthesis* übernimmt die verbale Verlautbarung der erhaltenen Dialogantwort durch Sprachsynthese und Verwendung der Roboter-Lautsprecher.

Die abstrakten Basisklassen werden von konkreten Implementierungen geerbt. Dabei erfolgt die Spracherkennung (AbstractSpeechDetection) durch die bereits vorhandene Software des Pepper-Roboters (NAOqiSpeechDetection). Gleiches gilt für die Sprachausgabe (AbstractSpeechSynthesis), welche NAOqiSpeechSynthesis als faktische Implementierung verwendet. Die innerste Aufgabe des gesamten Systems, nämlich die Erkennung von Absichten und Entitäten, übernimmt die Klasse WatsonNaturalLanguageProcessing, welche von AbstractNaturalLanguageProcessing erbt. Da das NLP-Modul in diesem Zusammenhang Text als Ausgangsbasis verlangt, übernimmt AbstractSpeechToText, beziehungsweise die konkrete Implementierung WitSpeechToText, eine Umwandlung von Sprachdateien in Text. Die einzelnen Klassen werden in den nachfolgenden Abschnitten im Detail vorgestellt.

Die Koordination zwischen den einzelnen Teilaufgaben erfolgt durch die Klasse NLPSpeechRecognition. Sie folgt dem bekannten Software-Entwurfsmuster Mediator[77]. Entsprechend dieser Architektur, werden durch Kapselung direkte Abhängigkeiten der Klassen vermieden. Zu einer bestimmten Teilaufgabe gehörende Klassen können komfortabel ausgetauscht werden, ohne dass dabei andere Klassen gleichermaßen anzupassen sind. Insbesondere der später angestrebte Austausch von Watson Conversation, vorzugsweise durch Rasa, wird dadurch erleichtert.

NLPSpeechRecognition definiert in der Mediator-Rolle auch den Ablauf des gesamten Kommunikationsprozesses. Mittels des Beobachter-Entwurfsmusters[78] werden Ereignisse der Teilaufgaben NLPSpeechRecognition mitgeteilt. Basierend auf diese, erfolgt die Entscheidung, welcher Schritt als nächstes abzuarbeiten ist.

Der Einstiegspunkt der Anwendung wird durch die Klasse PepperUsingNLP definiert. In diesem Zusammenhang erfolgt die Erstellung einer NAOqi-Session, die verwendet wird, um Zugang zu den Bestandteilen des Roboters zu gewähren. Gleichermaßen wird eine NLPSpeechRecognition Instanz angelegt und gestartet.

5.2 Implementierung

Nachfolgend wird die Realisierung der Systemarchitektur im Rahmen des Prototyps aufgezeigt. Die Beschreibung erfolgt anhand der ablaufenden Prozesse, die zur Laufzeit

[77] Einführung in das Mediator-Pattern inkl. Beispiel: https://sourcemaking.com/design_patterns/mediator

[78] Observer-Pattern Einführung inkl. Beispiel: https://sourcemaking.com/design_patterns/observer

der Applikation ausgeführt werden. Entsprechend also Sprachaufzeichnung, Sprachumwandlung, Sprachanalyse und Sprachsynthese.

5.2.1 AbstractSpeechDetection und Realisierung

Die abstrakte Basisklasse AbstractSpeechDetection ist stellvertretend für die Erkennung von sprachlichen Äußerungen und die Aufzeichnung dieser. Sie dient als Wrapper bzw. Schnittstelle für eine konkrete Implementierung dieser Aufgaben. Entsprechend müssen die vordefinierten Methoden dieser Klasse von der konkreten Implementierung überschrieben werden.

Im Kapitelabschnitt *3.1.1 Robotersystem „Pepper“* wurden die technischen Möglichkeiten des Roboters aufgezeigt. Dieser verfügt über Mikrofone, welche über das Modul NAOqi Audio steuerbar sind und auch die Aufzeichnung von Audiodateien erlaubt. Somit kann die Hauptaufgabe der Schnittstelle mit Bordmitteln des Robotersystems realisiert werden und bedarf keiner Drittanbieter Software-Bibliothek.

Da die Implementierung mit eigenen Mitteln des Roboter-Frameworks realisiert wird, lautet die Klasse der Implementierung NAOqiSpeechDetection. Sie implementiert die abstrakte Klasse und deren vordefinierte Methoden. Die Zusammensetzung der abstrakten Klasse ist in *Abbildung 62* als Klassendiagramm dargestellt.

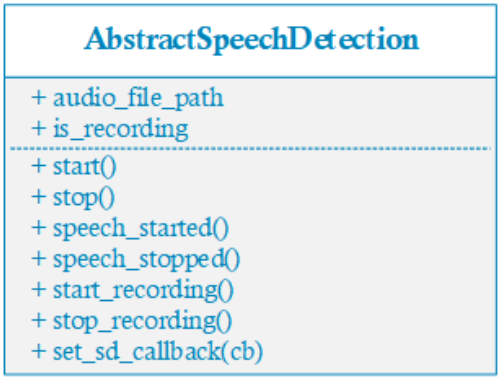


Abbildung 62: Schnittstelle zur Erkennung und Aufzeichnung von Sprache. [Quelle: Eigenmaterial]

Im Prinzip müssen Implementierungen von AbstractSpeechDetection dauerhaft auf Geräusche lauschen und entscheiden, ob es sich nur um Umgebungslärm oder menschliche Äußerungen handelt. Dabei bedürfen Umgebungsgeräusche keiner Speicherung, aber menschliche Äußerungen sind unverzüglich aufzuzeichnen. Mit Beendigung der Benutzerverlautbarung ist eine Mitteilung an die Mediatorklasse NLPSpeechRecognition zu machen. Der Speicherort der Aufzeichnung ist in diesem Zusammenhang der Mitteilung beizufügen.

Normalerweise verwendet das Pepper-System zur Erkennung von Äußerungen ALSpeechRecognition. Vorteilhaft an ALSpeechRecognition ist die sofortige Umwandlung der Worte bzw. Sätze in Text, wodurch die eigens definierte Klasse AbstractSpeechToText obsolet wäre. ALSpeechRecognition verlangt aber die Angabe aller später zu erkennender Äußerungen, d.h. es werden nur vordefinierte Worte, Begriffe und Sätze erkannt, aber niemals davon abweichende. Eine natürlich sprachliche Kommunikation wäre damit ausgeschlossen, denn es ist schlichtweg unmöglich, alle theoretisch denkbaren Äußerungen zur Beschreibung einer Absicht oder Frage im Vorfeld zu definieren.[79] Insbesondere diese Problematik gilt es durch das NLP-System zu lösen.

Als Alternative wurde die Verwendung von ALDialog zur Erkennung der Äußerungen ins Auge gefasst. Das bereits im Abschnitt *3.1.1 Robotersystem „Pepper“* vorgestellte Skript-Werkzeug erlaubt als Erkennungs-Regel sogenannte Wildcards, welche als Platzhalter für eine beliebige Äußerung verwendbar sind. Ergo ist das System auch in der Lage nicht vordefinierte Worte zu erkennen. Wie sich nach Rücksprache mit dem Support des Roboter-Herstellers herausstellte, ist dies momentan nur für die Sprache Englisch möglich und bedarf einer gesonderten Lizenz, welche etwa 1.600 Euro im Jahr kostet[80].

Die Problemstellung, welche ALSpeechRecognition und ALDialog mit sich bringen, führte zur Entwicklung eigener Lösungsansätze. Zunächst wurde über die Implementierung einer eigenen Spracherkennung nachgedacht[81]. Somit hätte die Anwendung eine verbesserte Portabilität vorweisen können und wäre auch unabhängig des NAOqi-Frameworks des Roboters einsetzbar. Durch fehlende Administrator-Rechte auf dem Roboter-System ist die Ansteuerung der Mikrofone mittels Python aber nahezu unmöglich, da mehrere Systempakete für das Linux-Betriebssystem zu installieren sind. Noch schwieriger gestaltet sich die Verwendung der eigentlichen Spracherkennung, denn auch diese benötigt Trainingsbeispiele, welche als Audiodateien vorliegen müssen. Insgesamt übersteigen die zu erbringenden Arbeitsleistungen den zeitlichen Rahmen dieser Arbeit.

---

[79] Im Übrigen werden selbst vordefinierte Worte und Begriffe eher schlecht von ALSpeechRecognition erkannt. Die standardmäßige Spracherkennung scheint nicht besonders ausgereift zu sein.

[80] Im weiteren Verlauf wurde mit verantwortlichen Stellen der TH Wildau und der Roboter-Lizenzverwaltung gesprochen. Ab Ende März 2018 werden die kostengebundenen Funktionen von ALDialog für die TH Wildau freigeschaltet. Zusätzlich wird der Hochschule, als eine der ersten Forschungseinrichtungen überhaupt, Zugang zur Sprache Deutsch innerhalb dieser Funktionen gewährt. Grund dafür ist die rege Tätigkeit der TH Wildau im Zusammenhang mit Pepper-Projekten und das Interesse an Forschungsergebnissen, u.a. auch hinsichtlich der hier vorliegenden Arbeit.

[81] Die Open Source Lösung CMUSphinx gilt in Fachkreisen als gute Grundlage für eine eigene Spracherkennung. Mehr Informationen biete der nachfolgende Link: https://cmusphinx.github.io/

Da auch dieser Ansatz verworfen werden musste, galt der Blick nochmals ALSpeechRecognition. Dabei wurde ersichtlich, dass die Klasse intern anhand sechs verschiedener Status-Zustände verwaltet wird. Mit dem Zustand *SpeechDetected* wird der Beginn einer menschlichen Äußerung gekennzeichnet. Das Ende dieser repräsentiert der Zustand *ListenOff*. Daraus lässt sich schlussfolgern, dass die Unterscheidung von einfachen Umgebungsgeräuschen und tatsächlichen Sprachverlautbarungen bereits intern vorgenommen wird. Somit bieten die eigentlich nebensächlichen Status-Zustände im logischen Schluss eine gute Grundlage für die notwendige Spracherkennung. Entsprechend werden anhand der beiden Zustände die Methoden *speech_started()* und *speech_stopped()* aufgerufen, welche durch die Schnittstelle definiert sind.

Mit den Schnittstellen-Methoden *start_recording()* und *stop_recording()* erfolgt die koordinierte Aufzeichnung der Äußerungen. In Abhängigkeit dazu sind die Mikrofone des Roboters zu steuern. Obwohl die direkte Verwendung der Mikrofone mit Python-Mitteln aufgrund der fehlenden Administrator-Rechte nicht funktioniert, ermöglicht das NAOqi-Framework durch die Klasse ALAudioDevice[82] dennoch einen entsprechenden Zugriff. Per Mikrofon erhaltene Aufzeichnungen werden standardmäßig in eine Datei geschrieben.

Der gesamte Prozess zur Erkennung von Äußerungen und die Aufzeichnung dieser, ist für andere Projektklassen vollkommen uninteressant. Die Zweckmäßigkeit liegt im Erhalt der Audio-Datei. Durch die Methode *set_sd_callback()* wird ein Zugang zu dieser Datei für unabhängige Instanz-Objekte ermöglicht. Bei Übergabe einer Rückruf-Funktion, auch Callback genannt, werden an der Audio-Datei interessierte Instanz-Objekte nach erfolgreicher Aufzeichnung über dieses Ereignis informiert. Im Gesamtkonzept der hiesigen Anwendung hat die Mediator-Klasse ein solches Interesse.

Der soeben beschriebene Prozessfluss wird mit den Methoden *start()* und *stop()* allgemein gesteuert. Ergo ist der Funktionalität nur nach Aufruf der *start()*-Methode aktiv und wird mit *stop()* angehalten.

Die Verwendung der NAOqi-Klassen setzt eine intensive Auseinandersetzung mit der API der Software voraus. So ist beispielsweise der Erhalt von Status-Zuständen mit der Registrierung an ein zugehöriges Event-System zwingend notwendig. Solche Anmelde- und Abmeldevorgänge werden in der Implementierung durch eigene, private Methoden abgewickelt. Auf eine genauere Betrachtung wird an dieser Stelle verzichtet, da sie keine besonderen Programmierleistungen erfordern.

[82] Als Alternative bietet sich die Klasse ALAudioRecorder an. Diese bietet ausschließlich Methoden zur Aktivierung bzw. Deaktivierung der Mikrofone. Dagegen ist ALAudioDevice mächtiger und erlaubt zum Beispiel die Verwaltung eines bestimmten Mikrofons oder auch das Auslesen der Audio-Buffers.

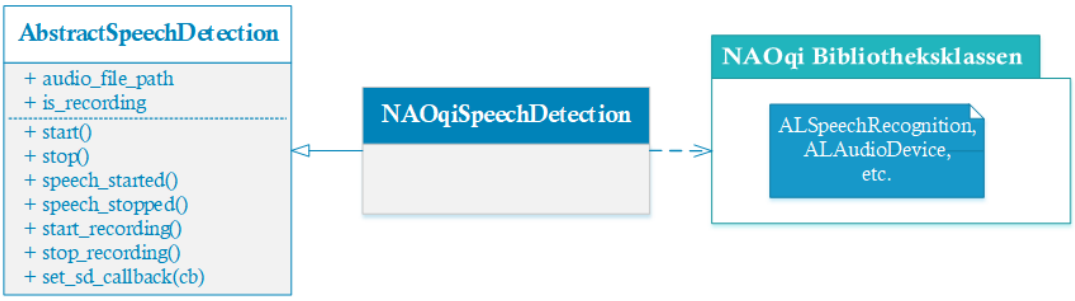


Abbildung 63: Komponenten der Spracherkennung des Protoyps. [Quelle: Eigenmaterial]

Mit *Abbildung 63* wird das eben beschriebene Zusammenspiel der Komponenten zur Spracherkennung und Sprachaufzeichnung sinnbildlich zusammengefasst.

5.2.2 AbstractSpeechToText und Realisierung

Die abstrakte Klasse AbstractSpeechToText definiert die Basis für eine Umwandlung von sprachbasierten Audio-Dateien in Text. Sie dient als Wrapper bzw. Schnittstelle für eine konkrete Implementierung dieser Aufgabe. Entsprechend müssen die vordefinierten Methoden dieser Klasse von der konkreten Implementierung überschrieben werden. Die Zusammensetzung der abstrakten Klasse ist in *Abbildung 64* als Klassendiagramm dargestellt.

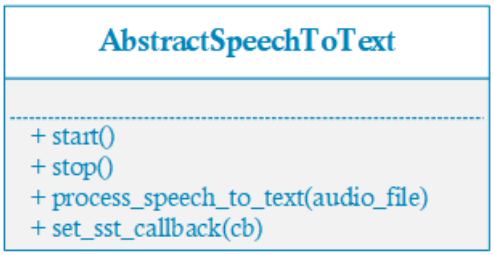


Abbildung 64: Schnittstelle zur Transformation von Audio-Dateien in Text. [Quelle: Eigenmaterial]

Der vorherige Abschnitt brachte die Erkenntnis, dass die Mittel der NAOqi-Software momentan nicht ausreichen, um verbale Äußerungen in Text umzuwandeln. Dies gilt auch für sprachliche Äußerungen im Audio-Dateiformat. Somit ist die Umwandlung von Audio-Dateien in Text durch eine Drittanbieter Software-Bibliothek zu realisieren.

Die Recherchen zu NLP-Systemen am Anfang dieser Arbeit wurden durch kleinere Implementierungsversuche gestützt. Dabei erreichten die Dienste Watson Speech To Text[83] und Wit.ai[84] sehr gute Ergebnisse. Es ist nur logisch, auf diese Implementierungsversuche aufzubauen.

[83] Übersicht zu Watson Text To Speech: https://www.ibm.com/watson/services/speech-to-text/

Watson Speech To Text scheidet für die weitere Betrachtung jedoch sofort wieder aus, denn die Sprache Deutsch wird noch nicht unterstützt. Daher fällt die Wahl auf Wit.ai, entsprechend lautet die Implementierung der abstrakten Klasse WitTextToSpeech.

Wit.ai besteht in den Grundzügen aus einem NLP-System. Es schied aufgrund des fehlenden Dialogmoduls aber bereits in der Vorbetrachtung der Evaluierungsphase aus. Die Möglichkeit zur Umwandlung von Audio in Text, macht es aber für die zu erfüllende Teilaufgabe des Prototyps wieder interessant. Zusätzlich ist Wit.ai kostenlos und unterstützt die Sprache Deutsch.

Die Installation von Wit.ai gestaltet sich bei Verwendung der Python Paketverwaltung PIP äußerst einfach:

```
$ pip install wit
```

Abbildung 65: Terminal-Befehl zur Installation von Wit.ai via PIP. [Quelle: Eigenmaterial]

Nach der Installation kann das Wit-Modul verwendet werden. Dies geschieht in der überschriebenen Methode *process_speech_to_text()* der Klasse WitSpeechToText. Im Zuge der Koordinierung durch die Mediator-Klasse, erfolgt die Übergabe des Dateipfades, der zuvor aufgezeichneten Audiodatei, in diese Methode. Dort wird die Datei im Binärformat eingelesen und *Wit.speech()* übergeben. Intern sendet das Wit-Modul die Datei an einen Server und analysiert die Audiodaten. Das Ergebnis dieser Analyse wird im JSON-Format zurückgesendet.

Bei erfolgreicher Transformation enthält das JSON-Datenfeld *„_text“* die textliche Repräsentation der Sprachaudiodaten[85]. Im Fehlerfall werden verschiedene Error-Codes mit zugehöriger Nachricht zurückgegeben. Die Meldung *„400 (Bad Request)“* deutet in diesem Zusammenhang auf eine nicht erfolgreiche Auswertung hin und das Textfeld verbleibt in einem solchen Fall leer.

Häufige Fehlerquellen sind qualitativ schlechte Audiodateien. In lauten Umgebungen oder bei einer zu schnellen Sprechweise versagt Wit.ai relativ oft. Audio-Dateien über zehn Sekunden werden teilweise sogar gänzlich abgelehnt.

Der interne Prozessablauf ist uninteressant für andere Projektklassen. Die Zweckmäßigkeit liegt im Erhalt des Textes, welcher aus der Audiodatei zu generieren ist. Durch die Methode *set_stt_callback()* wird ein Zugang zum Text-Ergebnis für unabhängige Instanz-Objekte ermöglicht. Über eine Rückruf-Funktion wird der Text an

[84] Informationen zu Wit.ai: https://wit.ai/

[85] Obwohl für das gesamte Projekt UTF-8 als Zeichenkodierung gesetzt ist, besteht die Server-Antwort aus einer anderen Kodierung. Daher wird eine manuelle Codierung zu UTF-8 vorgenommen. Ohne diese Maßnahme stürzt das System bei deutschen Umlauten sofort ab.

interessierte Objekte übergeben. Im Gesamtkonzept der hiesigen Anwendung hat die Mediator-Klasse ein solches Interesse.

Mit *Abbildung 66* wird das eben beschriebene Zusammenspiel der Komponenten zur Spracherkennung und Sprachaufzeichnung sinnbildlich zusammengefasst.

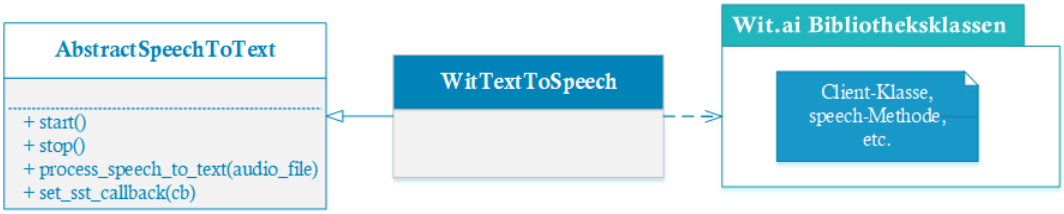


Abbildung 66: Komponenten zur Umwandlung von Sprachdateien zu Text. [Quelle: Eigenmaterial]

5.2.3 AbstractNaturalLanguageProcessing und Realisierung

AbstractNaturalLanguageProcessing übernimmt als abstrakte Basisklasse die Analyse von textbasierten Äußerungen zu Absichten und Entitäten sowie die kontextuelle Verwaltung der Dialogkommunikation. Die Klasse dient als Wrapper bzw. Schnittstelle für eine konkrete Implementierung dieser Aufgaben. Daher sind die vordefinierten Methoden dieser Klasse von der konkreten Implementierung zu überschreiben.

Implementierungen dieser Klasse folgen den theoretischen Erläuterungen aus dem Kapitelabschnitt *2.3 Die Methodik – Wie funktioniert NLP im Detail?* und den weiteren Erkenntnissen dieser Arbeit. Das heißt, an dieser Stelle des Prototyps wird eines der vorgestellten NLP-Systeme eingefügt. Da es die Bestandteile Morphologie, Syntaktik, Semantik und Konklusion enthält, kann es als das Herzstück der gesamten Implementierung betrachtet werden. Es sei die Erinnerung gestattet, dass die Klassen AbstracSpeechDetection, AbstractSpeechToText und AbstractTextToSpeech als Ergänzung dienen, da die betrachteten NLP-Systeme der Evaluierung nahezu alle auf die Phonologie-Komponente verzichten. Diesbezüglich bilden auch Rasa und Watson Conversation keine Ausnahme.

Watson Conversation stellte sich als Sieger der Evaluierung heraus, folglich lautet die implementierende Klasse WatsonNaturalLanguageProcessing. Zu überschreiben sind die vordefinierten Methoden der Basisklasse, welche in *Abbildung 67* dargestellt sind.

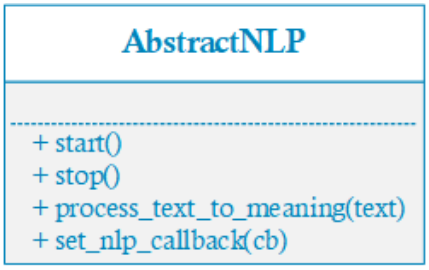


Abbildung 67: Schnittstelle zur Analyse von Text mittels NLP-Bestandteilen. [Quelle: Eigenmaterial]

Die Verwendung von WatsonNaturalLanguageProcessing gestaltet sich bewusst einfach. Die textbasierte Aussage ist *process_text_to_meaning()* in Form eines Methodenparameters zu übergeben. Anschließend erfolgt eine automatische Client-Server-Kommunikation anhand der API-Schnittstelle der Python-Bibliothek von Watson Conversation. Die Server-Instanz analysiert nun die textbasierte Aussage hinsichtlich definierter Absichten und Entitäten, um darauf basierend, eine Dialogantwort auszuwählen. Anschließend sendet der Server eine Antwort an den Client zurück. Dieser übergibt das Antwort-Ergebnis schlussendlich an den Mediator.

Die Versendung und der Empfang benötigter Daten erfolgt über eine selbst entwickelte Client-Klasse, welche aus der entstandenen Anwendung zur Evaluierung übernommen wurde (vgl. *4.4.4 Testprogramm*). Erst in dieser Client-Klasse erfolgen konkrete Zugriffe auf die Watson-Bibliotheksklassen, welche API-Zugriffe auf die Watson Conversation Instanz des Servers erlauben. Durch dieses Vorgehen werden Abhängigkeiten zur Watson-Bibliothek in eine Klasse gebündelt und zeitgleich findet eine Wiederverwendung von Teilen des Testprogramms statt.

Im Zuge einiger kleinerer Tests, stellte sich die Zustandslosigkeit des Dialogmoduls als ein Problem heraus. Erkannte Aussagen und Entitäten sowie gesetzte Kontextvariablen einer Anfrage gingen mit der nächsten Anfrage verloren. Relative Sätze wie *„Bitte wiederhole das*“ konnten daher nicht von Watson Conversation aufgelöst werden. Aus diesem Grund erfolgte die Einführung der Klasse WatsonDialogContent. Sie enthält sämtliche angehängte Daten der aktuellen Antwort und daher u.a. das Datenfeld *context*. Innerhalb einer Dialogkonversation ist die *context*-Variable der aktuellen Analyse-Anfrage in einem gleichnamigen Variablenfeld beizufügen. Mit Beendigung einer Dialogkonversation ist das *context*-Feld mit einem Null-Eintrag zu überschreiben. Andernfalls verbleiben die Kontextinformationen der vorherigen Konversation in der neuen Dialogkommunikation bestehen. Dies kann zu verwirrenden Zuständen führen und stellt auch datenschutzrechtlich ein Problem dar.

Der eben beschriebene Prozessablauf dient dem Zweck, aus textbasierten Äußerungen Absichten und Entitäten auszulesen, um darauf basierend kontextuelle

Dialogkonversationen abzuhalten. Dabei liegt das Augenmerk auf den Erhalt einer Dialogantwort, welche vom Roboter ausgesprochen wird. Durch die Methode *set_nlp_callback()* wird der Zugang zur Dialogantwort für unabhängige Instanz-Objekte der Anwendung ermöglicht. Bei Übergabe einer Rückruf-Funktion, auch Callback genannt, wird diese Funktion nach Erhalt einer Antwort durch Watson Conversation zurückgerufen und WatsonDialogContent als Parameter mitgegeben. Im Gesamtkonzept der hiesigen Anwendung verwendet die Mediator-Klasse dieses Parameter-Objekt.

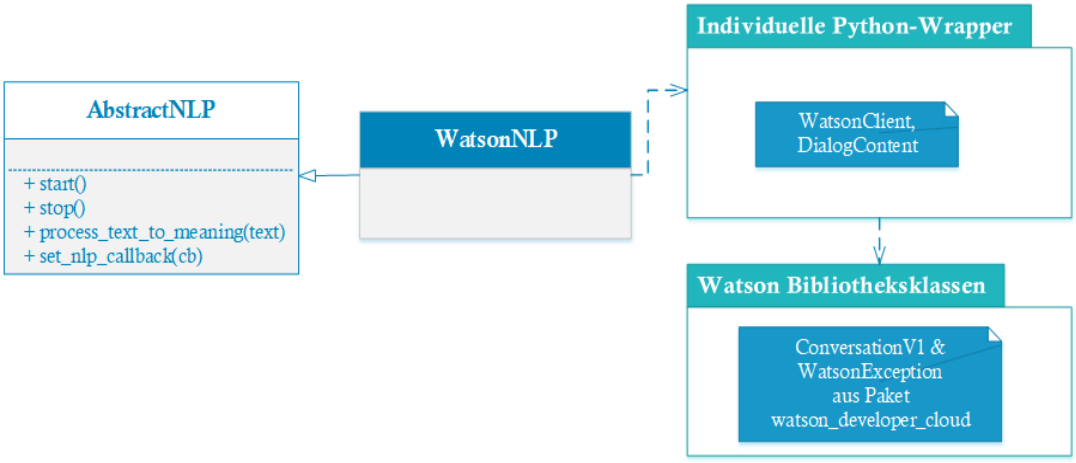


Abbildung 68: Umwandlungs-Komponenten für Text zu Bedeutung im Prototyp. [Quelle: Eigenmaterial]

In *Abbildung 68* wird das eben beschriebene Zusammenspiel der Komponenten zur Verdeutlichung sinnbildlich dargestellt.

5.2.4 AbstractSpeechSynthesis und Realisierung

Mit der abstrakten Basisklasse AbstractSpeechSynthesis wird der letzte Part zur verbalen Dialogkommunikation mittels eines NLP-Systems und dem Pepper-System definiert. Die Klasse repräsentiert die sprachbasierte Ansage bzw. Aussprache der zuvor erhaltenen Dialogantwort. AbstractSpeechSynthesis dient als Wrapper bzw. Schnittstelle für eine konkrete Implementierung dieser Aufgabe. Entsprechend müssen die vordefinierten Methoden dieser Klasse von der konkreten Implementierung überschrieben werden. Die Zusammensetzung der abstrakten Klasse ist in *Abbildung 69* als Klassendiagramm dargestellt.

Die Aufgabe von AbstractSpeechSynthesis ist das Gegenstück zu AbstractSpeechDetection. Währenddessen zuvor Sprache zu Text transformiert wurde, erfolgt nun die Umwandlung von Text zu Sprache. Folglich stehen diesbezügliche Implementierungen wieder in Relation zur Phonologie-Komponente vollwertiger NLP-Systeme. Ein anderer Begriff für diesen Vorgang ist Sprachsynthese.

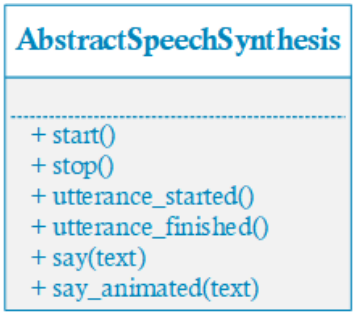


Abbildung 69: Schnittstelle zur (animierten) Sprachsynthese von Text. [Quelle: Eigenmaterial]

Wie auch bei AbstractSpeechDetection, ist es möglich, die Bordmittel des Pepper-Roboters zur Implementierung dieser Aufgabe zu verwenden. Im Kapitelabschnitt *3.1.1 Robotersystem „Pepper“* wurde die Verfügbarkeit zweier Lautsprecher kenntlich gemacht, welche über das Modul NAO Audio steuerbar sind. In diesem Zusammenhang verfügt der Roboter auch über eine vollwertige Sprachsynthese, das heißt, sämtliche Texteingaben werden vom Roboter ausgesprochen. Auf Drittanbieter Software kann daher verzichtet werden.

Da die Implementierung mit eigenen Mitteln des Roboter-Frameworks realisiert wird, lautet die implementierende Klasse NAOqiSpeechSynthesis. Die prinzipielle Funktionsweise besteht darin, übergebenen Dialogtext zur entsprechenden NAOqi-Klasse weiterzuleiten. Die einfachste Realisierungsform ist eine schlichte Ansage des übergebenen Textes. Eine natürlich sprachliche Kommunikation besteht indirekt jedoch auch aus Körpersprache (an dieser Stelle sei auf *2.1 Eine Definition - Was ist NLP?* verwiesen). Das Roboter-System enthält dafür bereits eine vorgefertigte Klasse, genannt ALAnimatedSpeech.

ALAnimatedSpeech erwartet, neben dem auszusprechenden Text, einen weiteren Parameter, der den Roboter während des Sprechvorgangs um Bewegungen ergänzt. Dieser Parameter lautet *bodyLanguageMode* und erlaubt drei Werte: *disabled, random* und *contextual*. Für den hiesigen Prototyp wird *random* verwendet, wodurch der Roboter zufällige Bewegungen ausführt, was bedeutet, die Bewegungen stehen nicht im Einklang mit dem Text. Sollen Bewegungsanimationen in Abhängigkeit zu bestimmten Worten ausgeführt werden, so ist der Modus *contextual* zu verwenden. Es können dabei sowohl vorgefertigte Animationen, als auch eigens programmierte verwendet werden[86]. Im Modus *disabled* erfolgen keine Bewegungen, der Roboter sagt lediglich den Text an.

[86] Auf eine genauere Betrachtung der Verwendung von Bewegungsanimationen wird an dieser Stelle verzichtet. Die Abläufe und Programmierschritte würden den Rahmen dieser Arbeit sprengen. Ein Einstieg zur konkreten Verwendung kann folgender Link bieten: http://doc.aldebaran.com/2-5/naoqi/audio/alanimatedspeech.html#alanimatedspeech

Die Einbindung von ALAnimatedSpeech bedingt die Verwendung von ALMotion und ALRobotPosture. Der Roboter befindet sich zur Schonung der Servomotoren in einer Schonhaltung und muss diese Haltung verlassen, sobald Bewegungen auszuführen sind. Dabei zeigt ALMotion über die Methode *robotIsWakeUp()* an, ob der Roboter bereit ist oder zunächst die Schonhaltung verlassen muss. Die Methode *wakeUp()* weckt den Roboter bei Bedarf entsprechend auf. Nach jeder durchgeführten Bewegungsanimation verweilt der Roboter im letzten Zustand der Bewegung (vgl. *Abbildung 70*). Es kam während der Testphasen immer wieder zu „komischen" Haltungen, die nicht natürlich wirkten. Nach jeder Animation wird daher über ALRobotPosture durch den Aufruf der Methode *goToPosture(„StandInit, 0.5)*[87] der Ausgangszustand wiederhergestellt.

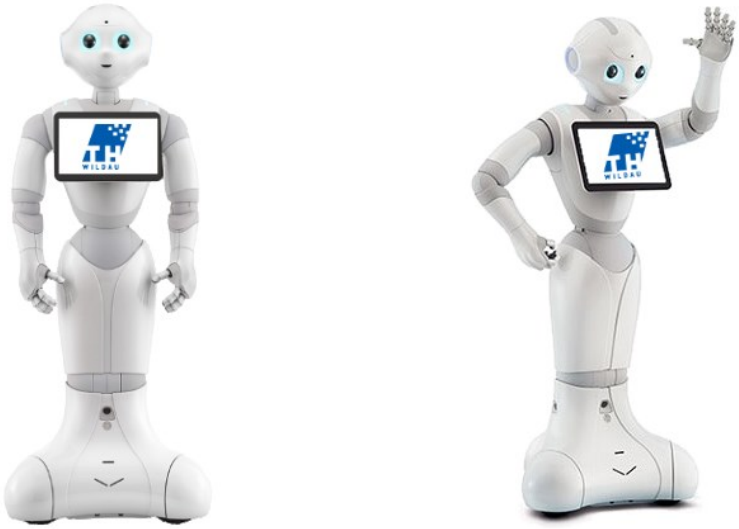


Abbildung 70: Neutrale StandInit-Haltung (links) und „komische" Endhaltung nach Ausführung einer Bewegungsanimation (rechts) des Pepper-Roboters. [Quelle: Eigenmaterial]

Mit den Schnittstellen-Methoden *say()* und *say_animated()* erfolgt die Übergabe des Dialogtextes zu den NAOqi-Klassen. Für *say()* sind Bewegungsanimationen deaktiviert, währenddessen *say_animated()* zufällige Animationen auslöst.

Die Verwendung der Klassen des NAOqi-Frameworks setzt eine intensive Auseinandersetzung mit der API der Software voraus. So ist beispielsweise im Zuge der Textansagen die Spracherkennung zu deaktivieren, denn das Roboter-System ist nicht in der Lage zwischen seiner eigenen Stimme und menschlicher Stimmen zu unterscheiden. Bei ausbleibender Deaktivierung der Spracherkennung während der Sprachsynthese, beginnt der Roboter, die eigenen Äußerungen zu analysieren, wodurch eine Endlosschleife entsteht, die im Endeffekt zur Funktionsunfähigkeit führt. Solche

[87] StandInit ist der initiale Haltungszustand des Roboters, welcher mit einem normalen, aufrechten Stand eines Menschen vergleichbar ist. Der Wert 0,5 gibt die relative Bewegungsgeschwindigkeit für den Wechsel zur initialen Haltung an.

Aktivierungs- und Deaktivierungsvorgänge werden in der Implementierung durch eigene, private Methoden abgewickelt. Auf eine genauere Betrachtung wird an dieser Stelle verzichtet.

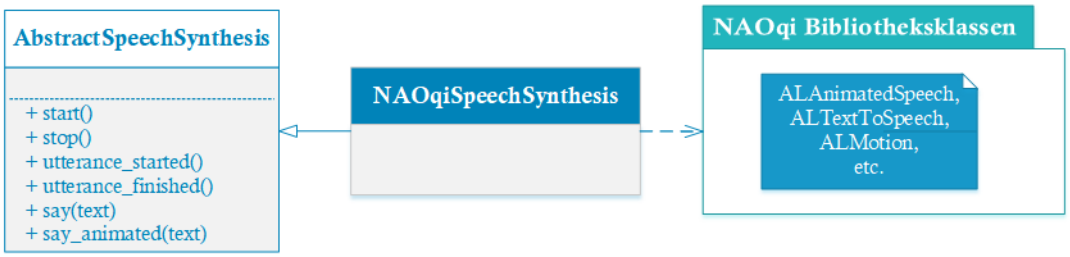


Abbildung 71: Komponenten der Sprachsynthese des Prototyps. [Quelle: Eigenmaterial]

Mit Abbildung 71 wird das eben beschriebene Zusammenspiel der Komponenten zur Sprachsynthese im Pepper Roboter sinnbildlich dargestellt.

5.3 Aktualisierte Anwendungsdomäne

Im Zuge der Programmierung des Prototyps, erfolgte eine Anpassung und Erweiterung des ursprünglichen Watson Conversation Projektes aus der Evaluierung. Dieses wird von WatsonNaturalLanguageProcessing im Prototyp verwendet.

Basierend auf die Befragung von Bibliotheksmitarbeitern während Feldtests, konnte eine Liste häufig gestellter Fragen von Besuchern im Zusammenhang mit der Bibliothek und deren angebotene Dienste angefertigt werden. Die Liste befindet sich im *Anhang B*.

Die Analyse des Fragenkatalogs ergibt vier typische Absichten der Bibliotheksbesucher und eine Vielzahl damit verbundener Entitätsziele. Dabei beziehen sich die Absichten auf die generelle Funktionsweise von Diensten, wo sich etwas Bestimmtes befindet, welche Maßnahmen zur Buchung von Etwas nötig sind und Kosten, die für solche Dinge eventuell anfallen.

Indirekte Ziele dieser Absichten sind Entitäten wie Arbeitsräume, Endgeräte, Medien, Schließfächer, der hochschuleigene VPN-Zugang, usw. oder auch Dinge, wie USB-Sticks und die Toiletten. Dabei werden die Entitäten durchaus noch sprachlich spezifischer von den Bibliotheksbesuchern verwendet. So werden beispielsweise Medien in Bücher, Ebooks, Magazine, Zeitschriften und Abschlussarbeiten unterteilt.

Die Abbildung aller Absicht-Entität-Kombinationen ist aufgrund der Ähnlichkeit einiger Anfragen für eine manuelle Erstellung von Trainingsdaten nur sehr schwierig möglich. Daher werden die vier hauptsächlich vorkommenden Absichten programmatisch eins-zu-eins übernommen. Die Unterscheidung, welche Dialogreaktion

konkret nötig ist, wird mit Hilfe der Slot-Funktionalität gelöst. Entsprechend lauten die vier Hauptabsichten des Watson Conversation Projektes:

- *how_to* beschreibt die Absicht eine Hilfestellung zur Verwendung einer Bibliotheksfunktionalität zu benötigen bzw. in Erfahrung zu bringen, wie etwas Bestimmtes funktioniert.
- *searching_for* ist stellvertretend für die Suche nach etwas Bestimmten in der Bibliothek.
- *booking_of* repräsentiert den Wunsch, etwas zur Bibliothek gehörendes im Rahmen einer Anmietung zu reservieren.
- *costs_of* ist die Absicht zu erfahren, ob die Benutzung bzw. der Zugang zu etwas Bestimmten in der Bibliothek mit Kosten verbunden ist.

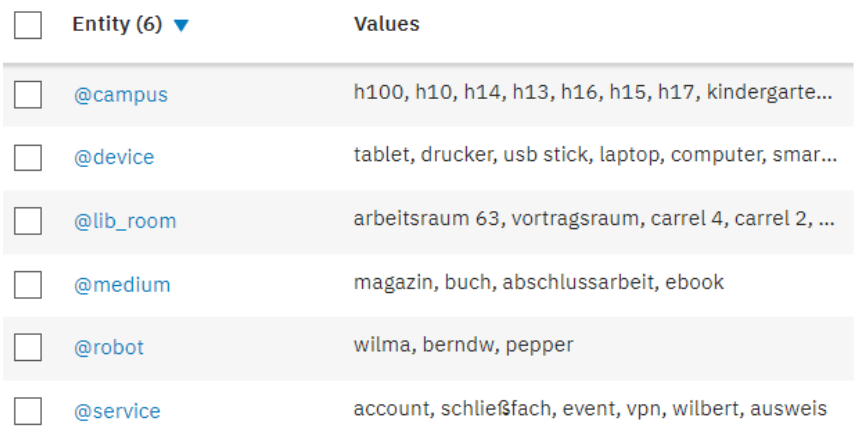
Es existieren noch weitere Intents: *start_conversation, end_conversation, repeat, yes* und *no*. Dabei entsprechen *start_conversation, end_conversation* und *repeat* den gleichen Funktionalitäten, wie aus dem Testprogramm bekannt. Durch die Einführung von *yes* und *no* kann das System Zustimmung und Ablehnung erkennen. Die Unterbrechung des Roboters, bekannt aus dem Testprogramm als Absicht *interrupt*, ist in der momentanen Implementierung nicht enthalten. Während der Roboter spricht, muss die Spracherkennung deaktiviert werden, da sich der Roboter andernfalls selbst hört und in einer Endlosschleife Absichtserkennung vornimmt. Mit sprachlichen Mittel ist eine Unterbrechung des Roboters daher nicht möglich.

Im Zusammenhang mit Entitäten existieren sechs Grundtypen. Diese unterteilen sich aber noch weiter und sind zusätzlich mit Synonymen ausgestattet. Die sechs Basistypen lauten:

- *campus*
  Beschreibt Objekte des Hochschul-Campus, also vor allem Gebäude.
- *device*
  Besteht aus verschiedenen Gerätetypen, beispielsweise Drucker, Computer und USB-Stick.
- *lib_room*
  Eine Auswahl von zur Bibliothek gehörender Räumlichkeiten und Orte, zum Beispiel die Carrel eins bis vier, der Empfang, der Glaskasten und die Toiletten.
- *medium*
  Beschreibt die in der Bibliothek vorhandenen Medien wie Bücher, E-Books, Zeitschriften und Abschlussarbeiten.
- *robot*
  Repräsentiert den Roboter selbst als Entität und wird über Werte wie Wilma, BerndW, Pepper und Personalpronomen definiert. Dem Roboter wird damit eine

rudimentäre Selbstwahrnehmung ermöglicht. Beispielsweise wenn Benutzer die Frage stellen „*Wie funktionierst du?*“.

- *service*
  Repräsentiert Dienste und Dinge wie den Bibliotheksausweis, Schließfächer, die Suchmaschine Wilbert oder auch Events wie Konzerte, etc.



Entity (6) ▼	Values
@campus	h100, h10, h14, h13, h16, h15, h17, kindergarte...
@device	tablet, drucker, usb stick, laptop, computer, smar...
@lib_room	arbeitsraum 63, vortragsraum, carrel 4, carrel 2, ...
@medium	magazin, buch, abschlussarbeit, ebook
@robot	wilma, berndw, pepper
@service	account, schließfach, event, vpn, wilbert, ausweis

Abbildung 72: Gliederung der Entitäten und Entitätswerte des Prototyps. [Quelle: Eigenmaterial]

Mit *Abbildung 72* kann die Struktur von Entitäten und Entitätswerte bzgl. der prototypischen Implementierung ausschnittsweise betrachtet werden. Zu jedem Entitätswert existieren zusätzlich Synonyme, beispielsweise *Thesis, Masterarbeit, Bachelorarbeit,* etc. für den Entitätswert *Abschlussarbeit* der Basisentität *@medium*.

Im Dialogmodul erfolgt die kombinierte Verwendung von Absichten und Entitäten, wie bereits im Abschnitt *4.2.1.2 IBM Watson Conversation* erläutert. Alle Absichten beschreiben dabei einen eigenen Elternknoten. Dieser ist aber zu allgemein, denn beispielsweise *how_to*, also die Absicht zu erfahren wie etwas funktioniert, definiert keine Zielentität. Daher verwenden die Elternknoten *how_to, searching_for, booking_of* und *costs_of* das Slot-Prinzip. Entsprechend erfolgt die Setzung einer Kontext-Variablen, sobald eine der definierten Entitäten in einer Äußerung erkannt wird. Bezüglich des Beispiels lautet diese Variable *$howto_target* und kann die Entitäten *@campus, @device,* usw. enthalten.

Dieses Vorgehen ermöglicht Erklärungen und Beschreibungen in Form von Dialogantworten einzufügen und auf Wunsch mit mehreren Antwortvarianten zu versehen. Bei Definition von mehreren Antwortvarianten wird momentan das *random*-Prinzip verwendet, wodurch zufallsmäßig eine dieser Varianten zurückgegeben wird. Im Falle, dass keine Entität erkannt wurde, verlangt das System die Angabe einer solchen.

5.4 Systemablauf und Verwendung

Der aktuelle Prototyp ist für jede testweise Verwendung explizit zu starten und entspricht daher keinem vollwertigen ALModul des NAOqi-Frameworks. Ein solches Modul wäre mit Start des Roboters automatisch aktiviert und verfügbar.

Als Einstiegspunkt der Anwendung dient momentan die Klasse PeperUsingNLP. Im Zuge der Initialisierung, wird der Prototyp dynamisch mit die NAOqi-Umgebung verwoben. Dies geschieht in dem eine Session eröffnet wird, welche den Zugriff auf den Roboter und die NAOqi-Klassen zur Steuerung des Roboters erlaubt. Außerdem wird eine Instanz der Mediator-Klasse NLPSpeechRecognition angelegt, welche den gesamten Ablauf des prototypischen Systems koordiniert. Mit Erzeugung der Instanz werden auch die Teilmodule initialisiert.

Tabelle 16: Typen von möglichen Verarbeitungsfehlern des Prototyps.

Fehlercode	Numerischer Wert	Bedeutung
DEFAULT_PROCESSING _ERROR	0	Allgemeiner, nicht weiter spezifizierter Fehler in der Verarbeitungskette.
AUDIO_INPUT _ERROR	1	Indiziert einen Fehler bei der Aufzeichnung von Spracheingaben.
AUDIO_TO_TEXT _ERROR	2	Kennzeichnet Fehler bei der Umwandlung der Audiodatei zu Text.
UTTERANCE_UNDESTANDING _ERROR	3	In der Analyse des Textes zur Erkennung von Absichten und Entitäten ist etwas schief gegangen.
NO_UTTERANCE_DEFINED _ERROR	4	Für eine Absichts-Entität-Kombination ist keine Dialogantwort hinterlegt.

Die Mediatorfunktion erreicht NLPSpeechRecognition durch die Registrierung von Callbacks zu allen Teilmodulen. Die abzuarbeitende Prozesskette folgt dabei den Erläuterungen aus den letzten Abschnitten. Im Falle, dass ein Teilmodul die Verarbeitung aufgrund eines Fehlers nicht gewährleisten kann, wird dem Callback *None* zurückgegeben. Das Robotersystem verfügt in diesem Zusammenhang über Fehlercodes, welche in *Tabelle 16* aufgelistet sind.

Zur Verwendung der Applikation ist der Projektordner *NLP_Projects*, beigefügt auf der CD-ROM dieser Arbeit, auf das Robotersystem zu kopieren. Anschließend stehen sämtliche notwendigen Konfigurations- und Programmdateien zur Verfügung. Die Anwendung wird aus dem Verzeichnis *~/NLP_Projects/PepperUsingNLP* mit folgendem Terminal-Befehl gestartet:

```
$ python PepperUsingNLP.py
```

Abbildung 73: Kommandozeilenbefehl zur Ausführung der Prototyp Applikation. [Quelle: Eigenmaterial]

Die entwickelte Anwendung kann prinzipiell als stabil angesehen werden, das heißt, die Häufung von Abstürzen wurde nicht registriert und sprachliche Äußerungen werden kontextuell verarbeitet. Dennoch sind Fehler in der Verarbeitungskette nicht auszuschließen und es ist aus Entwicklersicht notwendig, Einblick in die Prozesse zu erhalten. Insbesondere die Umwandlung der Audiodateien zu Text spielt hierbei eine wichtige Rolle, da Wit.ai diese Transformation vornimmt und falsch verstandene Worte Einfluss auf die restliche Verarbeitungskette haben.

Um die Prozesse kenntlich zu machen und die Fehlersuche zu erleichtern, wird ein intensives Logging der gesamten Applikation vorgenommen. Ein beispielhafter Ausschnitt solcher Log-Nachrichten des Prototyps ist in *Abbildung 74* gegeben.

Eine Log-Nachricht besteht, der Abbildung folgend, aus einem Level[88], dem UNIX-Zeitwert[89], der Identifikationsnummer des ausführenden CPU-Prozesses, dem Namen der betreffenden Klasse und einer Nachricht der Objektinstanz dieser Klasse.

Bei Betrachtung der Zeitwerte offenbart sich die größte Problematik des aktuellen Prototyps: Die von Menschen als akzeptabel empfundene Latenz von maximal einer Sekunde wird deutlich überschritten. Entsprechend der Log-Nachrichten dauert der Verarbeitungsprozess für die Äußerung *„Wo befinden sich hier die Masterarbeiten?“* ungefähr sieben Sekunden. Die Hauptverzögerung entsteht während der Umwandlung der Audio-Datei in Text durch die Wit.ai Software, welche ca. sechs Sekunden benötigt.

[88] Die Log-Level folgen den Werten der NAOqi. Diese sind in folgendem Link einsehbar: http://doc.aldebaran.com/2-5/dev/libqi/api/python/logging.html

[89] Im hiesigen Fall liegt der UNIX-Zeitstempel als numerischer Wert und nicht als Datumswert vor. Mehr Informationen bietet: https://de.wikipedia.org/wiki/Unixzeit

Damit empfinden Menschen das System als reaktionslahm. Die Freischaltung der Wildcard-Funktion für die hauseigene Spracherkennung des Roboters könnte diesen Nachteil in Zukunft ausgleichen[90].

```
[I] 1519998615.507242 3619 NAOqiSpeechDetection: status: value: ListenOn
[I] 1519998645.104338 3620 NAOqiSpeechDetection: status: value: SpeechDetected
[I] 1519998645.107638 3620 NAOqiSpeechDetection: speech_started
[I] 1519998645.116833 3620 NAOqiSpeechDetection: start_recording: success
[I] 1519998647.739464 3619 NAOqiSpeechDetection: status: value: ListenOff
[I] 1519998647.742379 3619 NAOqiSpeechDetection: speech_stopped
[I] 1519998647.748496 3619 NAOqiSpeechDetection: stop_recording: success!
[I] 1519998647.916733 3622 NAOqiSpeechDetection: status: value: EndOfProcess
[I] 1519998647.932642 3622 NLPSpeechRecognition: on_speech_detection_result:
/home/nao/recordings/speech.wav
[I] 1519998647.936254 3622 WitSpeechToText: process_speech_to_text:
/home/nao/recordings/speech.wav
[I] 1519998652.807527 3622 NLPSpeechRecognition: on_speech_to_text_result: Wo
befinden sich hier die Masterarbeiten
[I] 1519998654.022613 3622 NLPSpeechRecognition: {
 'input_text': 'Wo befinden sich hier die Masterarbeiten',
'detected_intents': '[{u'confidence': 0.9794283866882325, u'intent':
u'searching_for'}]',
'detected_entities': '[{u'confidence': 1, u'value': u'abschlussarbeit',
u'entity': u'medium }]',
'output_text': 'Du suchst also im Zusammenhang mit: abschlussarbeit .',
'session_context': '...',
'session_input': '...',
'session_output': '...'
}
[I] 1519998654.027836 3622 NAOqiSpeechSynthesis: say_animated: Du suchst also im
Zusammenhang mit: abschlussarbeit.
```

Abbildung 74: Ausschnitt von generierten Log-Nachrichten durch den Prototyp. [Quelle: Eigenmaterial]

5.5 Erkenntnisse und Zusammenfassung

Die Konzeption und praktische Umsetzung eines Prototyps zur natürlich sprachlichen Steuerung des Robotersystems Pepper mit Hilfe von Natural Language Processing, ist als letzte große Maßnahme dieser wissenschaftlichen Arbeit gelungen.

Die fehlende Phonologie-Komponente zur Spracherkennung und Sprachsynthese gegenüber menschlichen Kommunikationspartnern, konnte durch die Verwendung von Bordmitteln des Pepper-Roboters ausgeglichen werden. Die NLP-Komponente, als Herzstück der Anwendung, wurde erfolgreich mit dem Robotersystem verwoben. Beide

[90] Die Funktion „Wildcard“ wurde im Abschnitt *3.1.1 Robotersystem „Pepper“* dargestellt. Die damit verbundene Problematik ist in *5.2.1 AbstractSpeechDetection und Realisierung* beschrieben.

Maßnahmen ermöglichten in Kombination die Schaffung eines neuen Moduls zur Sprachverarbeitung auf dem Robotersystem des Typs Pepper.

Das in diesem Zusammenhang angewandte Mediator-Entwurfsmuster, ermöglicht eine zentralisierte Koordination der einzelnen Teilmodule und verhindert durch Kapselung direkte Abhängigkeiten der Klassen zueinander. Da zusätzlich abstrakte Klassen einheitliche Schnittstellen bilden, ist ein zukünftiger Austausch von Teilbestandteilen der Implementierung komfortabel möglich.

Diese Vorgehensweise scheint schon jetzt der richtige Weg, denn die viel zu lange Verarbeitungszeit für die Umwandlung von Audiodaten in Text, ist ein Nachteil, der unbedingt ausgeglichen werden sollte. Eine Verwendung des Moduls im Rahmen weiterer Feldtests ist zwar durchaus denkbar, aber eine vollwertige Einbindung als Ergänzung zu menschlichen Bibliothekaren noch nicht. Zunächst müssen die Latenzwerte dringend reduziert werden. Ein eventueller Wechsel zur Wildcard-Platzhalterfunktion, könnte die Lösung dieser Problemstellung sein. Nachteilig ist die späte Freischaltung dieser Funktion kurz vor Ende der Bearbeitungszeit dieser Abschlussarbeit. Die Zeit ist zu knapp, um noch diesbezügliche Erkenntnisse bereitzustellen. Bei Möglichkeit werden diese im Rahmen der mündlichen Verteidigung nachgeliefert.

Nach Anpassung und Erweiterung der Anwendungsdomäne, kann nun zumindest die gewünschte Funktionalität zur Beantwortung von häufig gestellter Fragen praktisch getestet werden. Komplexere Vorgänge, beispielsweise die Buchung eines Raumes durch die Sprachsteuerung, sind bei Zuhilfenahme der Watson Conversation Konfigurationswebseite ebenfalls in greifbare Nähe gerückt. Feldtests und die Auswertung von Log-Nachrichten erlauben diesbezüglich die Optimierung von Dialogflüssen und abzudeckende Bereiche der Anwendungsdomäne.

6 Fazit

Man muss seine Ideen verwirklichen,

sonst wuchert Unkraut darüber.

Jean Paul[91]

In diesem Kapitel wird ein abschließender Überblick über die Ergebnisse dieser Masterarbeit gegeben. Zusätzlich erfolgt, durch die Nennung von Verbesserungspotentialen und neuen Aufgabenfeldern, ein Blick in die mögliche Zukunft der Implementierung.

6.1 Ausblick

Das bis dato geschaffene System stellt noch lange nicht die maximale Ausreizung von NLP basierten Kommunikationen in der Bibliothek der TH Wildau dar. Ganz im Gegenteil. Es wurden im Grunde nur alle Voraussetzungen geschaffen, um nun vollwertige Anwendungsprojekte und Forschungsarbeiten durchführen zu können. Mit diesem allerletzten Kapitel werden einige Anregungen gegeben, welche Handlungen in Zukunft denkbar sind.

Erweiterung des Korpus anhand eines Chatbots

Die Vergrößerung des Korpus-Datensatzes hat die höchste Priorität für eine zukünftige Weiterentwicklung des NLP-Systems. Nur so ist eine Optimierung der Performance zur zielgenauen Absichts- und Entitäten-Erkennung möglich.

Eine Verwendung des Systems in Form von Feldtests in der Bibliothek ist zwar möglich, jedoch stehen der TH Wildau momentan nur zwei Roboter zur Verfügung, wobei einer davon bereits als Entwicklungsplattform anderweitig verwendet wird. Dies stellt eine Flaschenhals-Problematik dar, denn es können schlichtweg nicht genügend Konversationen in relativ kurzer Zeit durchgeführt werden. Eine möglichst starke Benutzung des Systems ist aber notwendig, da nur so genügend beispielhafte

[91] Jean Paul (1763 – 1825), bürgerlich Johann Paul Friedrich Richter, war Autor und Dichter verschiedener Werke wie beispielsweise *Leben des Quintus Fixlein*. In ärmlichen Verhältnissen geboren, wurde er noch zu Lebzeiten eine bekannte Figur im deutschsprachigen Raum und lernte später auch Goethe und Schiller kennen. [120]

Äußerungen und Absichten für die Vergrößerung des Korpus-Datensatzes zusammenkommen.

Als Alternative könnte das System in Form eines textbasierten Chatbots eingesetzt werden. Dieser Anwendungsfall entspricht auch der am häufigsten vorkommenden Verwendungsform von NLP-Systemen. Dabei verwenden Unternehmen das NLP-System als Ergänzung zu Mitarbeitern des Supports. In dieser Aufgabe können häufig gestellte Fragen der Kunden automatisiert beantwortet werden.

Bei einer Eingliederung eines solchen Chatbots in die Webpräsenz der TH Wildau, könnten Studierende und Mitarbeiter der Hochschule ihre Suchanfragen über das Bot-System vornehmen. Der Durchfluss an Fragen wäre somit maximal und die Flaschenhals-Problematik umgangen. Basierend auf die Auswertung sämtlicher Anfragen kann dann die Domäne optimiert werden und das eigentliche Roboter-System den Dienst in der Bibliothek aufnehmen. Auch ein Parallelbetrieb beider Varianten ist in diesem Zusammenhang denkbar.

Die zu investierende Arbeitsleistung könnte innerhalb einer Bachelorarbeit oder sogar als Praktikum umgesetzt werden. Der Aufwand ist als niedrig bis mittel einzuschätzen.

Generierung automatischer Antworten

Die aktuelle Implementierung verwendet ausschließlich vordefinierte Antwort-Texte. Diese werden zwar durch Platzhalter-Elemente und der Slot-Funktionalität in gewisser Weise quasi-dynamisch, aber im Endeffekt muss die Aktualität der hinterlegten Informationen dennoch ständig geprüft werden.

NLP-Systeme können in diesem Zusammenhang um die Funktionalität Question Answering (QA) ergänzt werden. Dazu werden relevante Quellen herausgesucht und auf ihre Inhalte analysiert. Im hiesigen Fall könnte vor allem die gesamte Web-Präsenz der TH Wildau eine solche Quelle sein. Zusätzlich wäre es wünschenswert die Web-Dienste der Bibliothek einzubinden. Die Menge aller Informationsquellen bedeutet umfangreiche Analyseprozesse. Hilfreiche Tools können unter den Begriffen Web Crawler, Web Scraping, Data Extraction, Text Analysis gefunden werden. Zusammengefasst werden solche Tools unter dem Begriff Data Mining.

QA-Systeme, als Teil von NLP-Systemen, sind nicht weniger kompliziert. Die notwendige Arbeit ist außerordentlich hoch, so dass notwendige Grundlagen vermutlich auch nur im Rahmen einer Masterarbeit herausgearbeitet werden können.

Einbau von Eye-Catcher Funktionalitäten

Ein humanoider Roboter bringt nicht nur durch das menschenähnliche Aussehen eine hohe Anziehungskraft auf eben diese Menschen mit sich. Grund für das Interesse sind vor allem Darstellung wie *2001: Odyssee im Weltraum*, *Ex Machina* und *Terminator* oder auch Serien wie *Westworld* und *Black Mirror*. In der Film- und Fernsehwelt werden humanoide Systeme mit einer überwältigenden Genauigkeit und überlegenden Intelligenz dargestellt.

In der Realität stoßen solche Systeme (noch) schnell an ihre Grenzen. Mit Überschreitung der Anwendungsdomäne verlieren sie jegliche „Intelligenz“. Dieser Fakt führt bei interessierten Menschen schnell zur Ernüchterung gegenüber den Möglichkeiten des Systems. Bezüglich des Robotersystems heißt das, selbst Anfragen wie „Was ist 2 plus 2?“ oder „Erzähle mir einen Witz“ können momentan nicht sinnvoll beantwortet werden.

Um den Effekt der Immersion im Umgang mit dem Roboter zu erhöhen wäre es daher sinnvoll einige solcher Funktionen einzubauen. In diesem Zusammenhang sei auf den nächsten Abschnitt verwiesen.

Zusammenführung mit anderen Arbeiten

In der Einleitung dieser Arbeit wurde vom langen Weg des Gesamtprojektes gesprochen. Damit einhergehend, bestehen bereits einige Umsetzung und prototypische Anwendung für den Roboter.

Beispielsweise wurde bereits eine Witzefunktion von Studenten der Telematik programmiert. Diese müsste nur noch über NAOqi Event Handling mit der Sprachverarbeitung des NLP-System verbunden werden.

In einem weiteren Projekt befasste sich eine Studentin der Telematik mit der Möglichkeit, bestimmte Worte an Gesten zu koppeln, die an entsprechender Stelle im Sprechvorgang vom Roboter ausgeführt werden. In Verbindung könnte der Roboter somit deutlich lebendiger wirken, vor allem wenn die Bewegungsanimationen nicht mehr zufallsbasiert, sondern kontextuell sind.

In diesem Zusammenhang müssten für alle Projekte programmatische Schnittstellen definiert werden. Besser noch wäre die Verwendung eines eventbasierten Entwurfsmusters in Abhängigkeit zum NAOqi-Framework. Der entsprechende Arbeitsaufwand ist als mittel einzuschätzen.

6.2 Zusammenfassung

Gemeinsam mit der hochschuleigenen Bibliothek, versucht die TH Wildau seit einigen Jahren den Schulterschluss von Medien der analogen Welt mit Medien der digitalen Welt zu erreichen. Dieses, als Hybridbibliothek bezeichnete Konzept, besteht aus sechs Grundsäulen: Indoor-Ortung, Humanoide Roboter, Digitales Wissen, Open Acess, Web-Dienste und Showroom. Sie sollen, in Form eines kooperierenden Gesamtkonzeptes, die Digitalisierung der Hochschulbibliothek umsetzen.

Die Einbeziehung eines Roboters stellt eine besondere Herausforderung dar, denn zum einen sind Roboter, fernab industrieller Einsatzzwecke, noch relativ neu und zum anderen sollen sie in der Bibliothek als Schnittstelle zwischen dem digitalen Angebot und den „analogen“ Menschen dienen. Daher ist es erklärtes Ziel, den Roboter als Assistenzsystem einzusetzen, das heißt, der Roboter nimmt die Rolle als Experte für Anliegen rund um die Bibliothek ein und versucht bei der Beantwortung von Fragen und Problem zu helfen. Der Roboter wird also zum Bibliothekar.

Zu diesem Zweck, wurde die Wichtigkeit der sprachlichen Kommunikation zwischen Mensch und Roboter am Anfang dieser Arbeit herausgearbeitet. Da auf dem Gebieten künstliche Intelligenz und Sprachverarbeitung in den letzten Jahren große Fortschritte erzielt werden konnten, existieren eine Vielzahl von softwaretechnischen Lösungen zum Aufbau einer sprachlich intelligenten Kommunikation bzw. eines Dialogs zwischen Mensch und Maschine.

Für die Arbeit ergab sich daher als allumfassendes Ziel, eine prototypische Implementierung zur verbalen Kommunikation zwischen Mensch und Pepper-Roboter umzusetzen. Daraus ergaben sich vier große Kapitel für die hier vorliegende Arbeit: Theorie, Anforderungsanalyse, Evaluierung und Umsetzung.

Im Theorie-Teil wurden Grundlagen der Mensch-Maschine-Kommunikation betrachtet. Dies betraf die Erarbeitung einer Definition von Natural Language Processing als Methode dieser Kommunikationsform. In diesem Zusammenhang wurde auch die prinzipielle Funktionsweise mit allen Bestandteilen dargelegt und auf aktuelle Herausforderungen im Umgang mit NLP eingegangen. Dies sollte später noch einen enormen Einfluss auf die Systemarchitektur des Prototyps haben.

Der Theorie-Teil ermöglichte außerdem einen Blick in die Vergangenheit von NLP. Als besonders spannend zeigte sich dabei die Wahrnehmung in der Fachpresse. Zunächst als „Wunder“ bezeichnet, führte der ALPAC-Bericht aufgrund der langsamen Entwicklung zur Beendigung sämtlicher staatlicher Förderungen. Durch private Firmen wurde aber dennoch weitergeforscht, wobei dabei vor allem IBM eine wichtige Rolle

spielte. Mit dem Supercomputer Watson kann IBM diese Rolle auch heute immer noch unterstreichen.

Mit dem nachfolgenden Kapitel Anforderungsanalyse erfolgte die Übersetzung der theoretischen Erkenntnisse in Anforderungen, die eine NLP-Software beschreiben. Somit konnte eine Liste von erwarteten und gewünschten Funktionen erstellt werden, welche in der anschließenden Evaluierung als einheitliche Bewertungskriterien für einen Vergleich solcher NLP-Systeme diente. In diesem Zusammenhang wurden nicht nur klassische funktionale und nichtfunktionale Anforderungen definiert, sondern auch technische Voraussetzungen und Rahmenbedingungen beschrieben. Diese basieren auf Vorgaben der TH Wildau und Einschränkungen, die das Roboter-System setzt. Insgesamt wurden zehn solcher Einschränkungen festgelegt und grundlegend 46 Anforderungen definiert.

Speziell das Roboter-System wurde im Anforderungsteil nochmals separat detailliert vorgestellt. Sämtliche Hardware- und Software-Bestandteile wurden dazu beleuchtet, um Möglichkeiten und Grenzen für die Mensch-Roboter-Kommunikation des Prototyps aufzuzeigen. Dabei wurde ersichtlich, dass der Pepper-Roboter im Prinzip alle Bestandteile vorweisen kann, die eine Zusammenführung mit einem NLP-System voraussetzt.

Nachdem einheitliche Bewertungskriterien für einen Vergleich der recherchierten Software-Bibliotheken definiert wurden, galt es anschließend, den tatsächlichen Nutzen bzw. die reale Leistungsfähigkeit der NLP-Systeme zu analysieren und miteinander zu vergleichen. Die Evaluierung stellte in gewisser Weise das Herzstück dieser Arbeit dar. Basierend auf den bisherigen Erkenntnissen, sowohl bezüglicher theoretischer Grundlagen, als auch durch konkrete Anforderungen, sollte nun die am besten passendste NLP-Lösung gefunden werden, mit der sich ein realer Prototyp verwirklichen lässt.

Im Zuge der Recherche zur Auffindung von NLP-Software, entstand eine Liste mit über zwanzig Einträgen, die potentiell zur genaueren Betrachtung in Frage kamen. Die vollumfängliche Evaluierung aller Listeneinträge war hierbei nicht möglich, da die Arbeit zum einen nicht den entsprechenden zeitlichen Spielraum bot und zum anderen ein qualitativ hochwertiges Ergebnis angestrebt wurde, welches auch bereits Fragen zur Implementierung, in Form eines Prototyps, klären sollte. Diese Gründe machten es nötig, den zahlenmäßigen Umfang der betrachteten Softwarebibliotheken einzugrenzen. Folglich wurde eine Vorauswahl durchgeführt. Als Kriterien dieser Vorauswahl wurden die Rahmenbedingungen und technische Voraussetzungen verwendet.

Es wurde ein sehr heterogenes Bild, hinsichtlich der erfüllten Rahmenbedingungen und technischen Voraussetzungen, erkennbar. Der Großteil aller Systeme bewegte sich bei sechs bis sieben von zehn möglichen Punkten in der Gesamtbewertung. Lediglich die Systeme IBM Watson und Rasa.ai konnten sich leicht absetzen. Somit fand die Evaluierung zwischen Watson Conversation und Rasa statt. Es existierte damit ein interessanter Kontrast zwischen der kostenfreien, Open-Source-Lösung Rasa und der kostengebundenen, proprietären Software von IBM.

Was folgte, war eine detaillierte Vorstellung der beiden NLP-Systeme hinsichtlich prinzipieller Funktionsweise, Kostenstrukturen, Programmierschnittstellen und die konkrete Verwendung der beiden Software-Lösungen in einem Software-Projekt. Anhand der damit einhergehenden Erkenntnisse, wurde der Anforderungsabgleich vorgenommen. Interessanterweise konnten beide Systeme keine Anforderungen bzgl. der Phonologie erfüllen[92]. Gleiches gilt für die Problemstellung der Paradoxien.

Der Anforderungsabgleich sollte durch konkrete Funktionstests unterstützt werden, um zu sehen, wie die Software-Lösungen auf Testdaten reagieren. Dazu wurde nach dem Prinzip des Blackbox-Tests vorgegangen. In diesem Zusammenhang wurde auch ein eigenes Testprogramm in Python entwickelt und zugehörige Trainings- und Testdaten in Form von Korpus-Datensätze angelegt.

Insgesamt schafft es Watson Conversation mit dem gegebenen Trainingskorpus eine bessere Performance zu erreichen als Rasa. Dagegen hatte Rasa bei den Latenzwerten einen Vorteil. Diese sind deutlich geringer als bei Watson Conversation und ermöglichen ein Echtzeitverhalten der Anwendung.

Obwohl Rasa, über alle Vergleichskriterien betrachtet, eine bessere Gesamtbewertung erhalten hatte, wurde eine andere Entscheidung getroffen, mit welchem NLP-System der Prototyp zu entwickeln ist. Der Grund für diese Entscheidung war, dass ohne ausreichende Trainingsdaten keine guten Domäne-Modelle entstehen. Die Erkennungsraten von geäußerten Absichten müssen aber zwanghaft hoch sein, da andernfalls das Interesse am Roboter als Bibliothekar unverzüglich abnimmt. Personen, die das Roboter-System verwenden, sind am schnellen Informationsaustausch interessiert. Falsche Analyse-Ergebnisse und ständige Wiederholungen der ursprünglichen Äußerung führen zu einer schnellen Ablehnung des Systems.

In der aktuellen Situation gilt es aber ein hohes Interesse zur Benutzung des Roboters bei den Studierenden zu generieren. Nur so kann der Ausbau des Korpus-Datensatzes garantiert werden, der so wichtig ist für eine zukünftige Verwendung des Gesamtsystems als Roboter-Bibliothek.

[92] Diese Aussage lässt sich auf nahezu alle betrachteten NLP-Systeme erweitern.

Im abschließenden Kapitel Prototypische Umsetzung galt es, eine umspannende Systemarchitektur zu entwerfen und diese zu realisieren. Die Architektur sollte dabei eine starke Modularität vorweisen, um den Austausch einzelner Komponenten zu erleichtern, entsprechend also: Sprachaufzeichnung, Sprachumwandlung, Sprachanalyse und Sprachsynthese.

Die Konzeption und praktische Umsetzung eines Prototyps zur natürlich sprachlichen Steuerung des Robotersystems Pepper mit Hilfe von Natural Language Processing, ist als letzte große Maßnahme dieser wissenschaftlichen Arbeit gelungen. Die fehlende Phonologie-Komponente zur Spracherkennung und Sprachsynthese gegenüber menschlichen Kommunikationspartnern, konnte durch die Verwendung von Bordmitteln des Pepper-Roboters ausgeglichen werden. Die NLP-Komponente, als Herzstück der Anwendung, wurde erfolgreich mit dem Robotersystem verwoben. Beide Maßnahmen ermöglichten in Kombination die Schaffung eines neuen Moduls zur Sprachverarbeitung auf dem Robotersystem des Typs Pepper.

Glossar

Absicht Ist die Botschaft einer übermittelten Nachricht, also welches Ziel eine Frage bzw. Äußerung hat.

Assistenzsystem Computerbasierte Systeme, die Menschen bei der Erledigung einer Aufgabe unterstützten.

Entität Sind Objekte, Subjekte, Zeiten, Namen, usw. Im Grunde alles, was objektgebundenes Ziel einer Absicht sein kann.

Framework Eine Zusammenstellung von Software-Werkzeugen, die zur einheitlichen Programmierung einer Komponente verwendet werden können.

Humanoider Roboter In der Gestalt dem Menschen nachempfundener Roboter. Wichtigstes Merkmal ist dabei das Gesicht, welches einen Widererkennungswert für Menschen bietet. Häufig sollen solchen sich solche Roboter wie Menschen bewegen und mit diesen interagieren.

Indoor Ortung Standortermittlung in geschlossenen Räumen. Bekannte Systeme wie GPS können hier nicht verwendet werden, weshalb komplett neue Lösungen zu verwenden sind.

Konklusion Fremdwort für den Begriff Schlussfolgerung. Entsprechend ist die Bedeutung, aus einer Aussage eine logische Konsequenz zu ziehen.

NAOqi Ist ein Framework zur Programmierung und Steuerung des Pepper Roboters der Firma SoftBank Robotics Corp.

Pepper Roboter Ein humanoides Robotersystem der Firma SoftBank Robotics Corp. Er wird sowohl zu Forschungszwecken als auch im Privatgebrauch eingesetzt.

Unidos Wildau Applikation für mobile Endgeräte (App) mit Informationen über die Technischen Hochschule Wildau.

Abkürzungsverzeichnis

AF	Anforderung
ALPAC	Automatic Language Processing Advisory Committee
API	Application Programming Interface
ASR	Automatic Speech Recognition
ATN	Augmented Transition Network
FAQ	Frequently Asked Questions
FN	False Negative
FP	False Positive
GUI	Grafische Benutzeroberfläche
HMM	Hidden Markov Model
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifikationsnummer
IDE	Integrierte Entwicklungsumgebung
JSON	JavaScript Object Notation
MT	Machine Translation
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
QA	Question Answering
REST	Representational State Transfer
SDK	Software Development Kit

TH Wildau	Technische Hochschule Wildau
TN	True Negative
TP	True Positive
UI	Benutzeroberfläche
UIMA	Unstructured Information Management Applications
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtuelles Privates Netzwerk

Abbildungsverzeichnis

Tabellenverzeichnis

Quellenverzeichnis

Letztes Abrufdatum: Siehe *Hinweise zum Lesen dieser Arbeit.*

1. **Technische Hochschule Wildau.** *Bestand und Aufstellung.* https://www.th-wildau.de/hochschule/organisation/stabsstellen-und-zentrale-einrichtungen/hochschulbibliothek/benutzung-und-auskunft/bestand-und-aufstellung/.

2. **Technische Hochschule Wildau**. *Räume und Ausstattung.* https://www.th-wildau.de/hochschule/organisation/stabsstellen-und-zentrale-einrichtungen/hochschulbibliothek/benutzung-und-auskunft/raeume-und-ausstattung/.

3. **Technische Hochschule Wildau**. *Bewerbung zur Bibliothek des Jahres 2018 - Small but smart.* https://icampus.th-wildau.de/bewerbung-bdj-2018/index.html.

4. **Deutscher Bibliotheksverband.** *Bibliothek des Jahres 2012.* http://www.bibliotheksverband.de/dbv/auszeichnungen/bibliothek-des-jahres/preistraeger/2012.html.

5. **Davros, Yanni .** *Prolific Pen Comics.* https://prolificpencomics.tumblr.com/.

6. **Lobin, H.** *Computerlinguistik und Texttechnologie.* 1. Auflage. Wilhelm Fink GmbH & Co. Verlags-KG, 2010. ISBN: 978-3-7705-4863-7.

7. **Brownlee, J.** *What Is Natural Language Processing?* https://machinelearningmastery.com/natural-language-processing/, 2017.

8. **Meinel, C. und Sack, H.** *Digitale Kommunikation - Vernetzen Multimedia Sicherheit.* 1. Auflage. Springer X.media.press Verlag, 2009.ISBN: 978-3-540-92922-2.

9. **Chipman, S.** *The Oxford Handbook of Cognitive Science.* 1. Auflage. Oxford University Press Verlag, 2016. ISBN: 978-0199842193.

10. **Zysk, W.** *Körpersprache - eine neue Sicht.* Dissertation Universität Duisburg, 2002.

11. **Lehmann, C.** *Grammatik - Zweifache Gliederung.* https://www.christianlehmann.eu/ling/lg_system/grammar/morph_syn/index.html.

12. **Grimm, R. und Delfmann, P.** *Digitale Kommunikation - Sprache, Protokolle und Datenformate in offenen Netzen.* 1. Auflage. De Gruyter Oldenbourg Verlag, 2017. ISBN: 978-3110495355.

13. **Gebhardt, H.** *Kommunikationskanäle und deren Einsatz in Softwareunternehmen.* http://www.se.uni-hannover.de/priv/lehre_2008winter_seminar/Henrik_Gebhardt-Kommunikationskanaele-Folien.pdf, 2008.

14. **Watzlawick, P.** *Menschliche Kommunikation. Formen, Störungen, Paradoxien.* 1. Auflage. Bern, 1969. ISBN: 3-456-82825-X.

15. **Eichler, W.** *Sprachdidaktik Deutsch. Ein kommunikationswissenschaftliches und linguistisches Konzept.* 2. Auflage. Wilhelm Fink Verlag, 1979. ISBN: 3770507916.

16. **Geipel, M.** *Kommunikation und Sprache - Kommunikationsstörungen.* https://www.br.de/alphalernen/faecher/deutsch/5-kommunikation-stoerungen100.html.

17. **Carstensen, U. K., et al.** *Computerlinguistik und Sprachtechnologie - Eine Einführung.* 1. Auflage. Springer Verlag, 2010. ISBN: 978-3-8274-2023-7.

18. **Hancox, P. J.** *A brief history of Natural Language Processing.* http://www.cs.bham.ac.uk/~pjh/sem1a5/pt1/pt1_history.html.

19. **Johnson, R. G.** *Andrew D Booth - Britain's Other "Fourth Man".* Dept of Computer Science and Information Systems Birkbeck University of London : Springer Verlag, 2010. Bd. 325. ISBN: 978-3-642-15199-6.

20. **Rees, M.** *WARREN WEAVER 1894 - 1978.* National Academy Of Sciences, 1987.

21. **Hutchins, J.** *From first conception to first demonstration: the nascent years of machine translation, 1947 - 1954.* Kluwer Academic Publishers, 1997. S. 195–252.

22. **Schöning, U.** *Theoretische Informatik- kurz gefasst.* 5. Auflage. Spektrum Akademischer Verlag, 2008. ISBN: 978-3-8274-1824-1.

23. **Russell, S. und Norvig, P.** *Artificial Intelligence - A Modern Approach.* 3. Auflage. Pearson Verlag, 2009. ISBN: 978-0136042594.

24. **Jurafsky, D. und Martin, J. H.** *Speech and Language Processing - An Introduction to Natural Language Processing, Computional Lingustics and Speech Recognition.* 2. Auflage. Prentice Hall, 2008. ISBN: 978-0131873216.

25. **Stanford HCI Group.** *SHRDLU.* http://hci.stanford.edu/winograd/shrdlu/.

26. **Wikipedia**. *SHRDLU.* https://de.wikipedia.org/wiki/SHRDLU.

27. **Wikipedia.** *History of natural language processing.* https://en.wikipedia.org/wiki/History_of_natural_language_processing.

28. **Wikipedia**. *PARRY.* https://en.wikipedia.org/wiki/PARRY.

29. **Reitmaier, T.** *Aktives Lernen für Klassifikationsprobleme unter der Nutzung von Strukturinformationen.* Kassel University Press, 2015. ISBN: 978-3-86219-999-0.

30. **Cisco.** *The Zettabyte Era - Trends and Analysis.* https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.

31. **Fürnkranz, J., Gamberger, D. und Lavrač, N.** *Foundations of Rule Learning.* 1. Auflage. Springer Verlag, 2012. ISBN: 978-3-540-75196-0.

32. **Thompson, C.** *What Is I.B.M's Watson?* http://www.nytimes.com/2010/06/20/magazine/20Computer-t.html : The New York Times Magazin, 2010.

33. **IBM.** *The DeepQA Research Team - The Jeopardy! Quiz Show.* http://researcher.watson.ibm.com/researcher/view_group_subpage.php?id=2158.

34. **IBM.** *Products and services.* https://www.ibm.com/watson/products-services/.

35. **Olavsrud, T.** *10 IBM Watson-Powered Apps That Are Changing Our World.* https://www.cio.com/article/2843710/big-data/10-ibm-watson-powered-apps-that-are-changing-our-world.html, 2014.

36. **Forrest, C.** *IBM Watson: What are companies using it for?* http://www.zdnet.com/article/ibm-watson-what-are-companies-using-it-for/, 2015.

37. **tutorialspoint.** *AI - Natural Language Processing.* https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm.

38. **magizbox.** *Introduction to Natural Language Processing.* http://magizbox.com/training/natural_language_processing/site/introduction.html.

39. **Gerdsen, P. und Kröger, P.** *Digitale Signalverarbeitung in der Nachrichtenübertragung.* 2. Auflage. Springer Verlag, 1997.ISBN: 978-3-540-61194-3.

40. **Khurana, D., et al.** *Natural Language Processing: State of The Art, Current Trends and Challenges.* https://arxiv.org/ftp/arxiv/papers/1708/1708.05148.pdf.

41. **Clark, A., Fox, C. und Lappin, S.** *The Handbook of Computational Linguistics and Natural Language Processing.* Blackwell Publishing, 2010. ISBN: 978-1-4051-5581-6.

42. **TU-München, SR Wiki.** *Welcome to recognize-speech.com!* http://recognize-speech.com/.

43. **Meibauer, J., et al.** *Einführung in die germanistische Linguistik.* : J.B. Metzler Verlag, 2007. ISBN: 978-3476021410.

44. **Gabriel, C. und Meisenburg, T.** *Romanische Sprachwissenschaft.* 1. Auflage. UTB GmbH, 2007. ISBN: 978-3-7705-4325-0.

45. **magizbox.** *Natural Language Processing Tasks.* http://magizbox.com/training/natural_language_processing/site/tasks.html.

46. **Lämmel, U. und Cleve, J.** *Künstliche Intelligenz.* 4. Auflage. Carl Hanser Verlag, 2012. ISBN: 978-3-446-42758-7.

47. **Lehmann, C.** *Logik - Prädikatenlogik.* https://www.christianlehmann.eu/ling/logic/praedikatenlogik.html.

48. **Lehmann, C.** *Pragmatik - Aktualisierungen.* https://www.christianlehmann.eu/ling/pragmatics/index.html.

49. **Lehmann, C.** *Pragmatik.* https://www.christianlehmann.eu/ling/pragmatics/index.html.

50. **Lehmann, C.** *Pragmatik - Sprechsituation.* https://www.christianlehmann.eu/ling/pragmatics/index.html.

51. **Lehmann, C.** *Pragmatik - Deixis.* https://www.christianlehmann.eu/ling/pragmatics/index.html.

52. **Gill, Navdeep Singh.** *Overview of Artificial Intelligence and Natural Language Processing.* https://www.upwork.com/hiring/for-clients/artificial-intelligence-and-natural-language-processing-in-big-data/.

53. **IBM.** *Building Cognitive Applications with IBM Watson Services: Volume 2 Conversation.* 1. Auflage. International Business Machines Corporation, 2017. ISBN: 0738442569.

54. **Lehmann, C.** *Grammatik - Varia et curiosa - Lücken in der deutschen Grammatik.* https://www.christianlehmann.eu/ling/lg_system/grammar/index.html.

55. **Bender, E. M. und Good, J.** *A Grand Challenge for Linguistics - Scaling Up and Integrating Models.* https://ubir.buffalo.edu/xmlui/handle/10477/37382?show=full.

56. **Bird, S., Klein, E. und et al.** *Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit.* 1. Auflage. O'Reilly Media, 2009. ISBN: 978-0596516499.

57. **Nuance.** *Understanding Natural Language - Learning to speak customer-ese.* http://www.procat.com.tr/dokuman/Nuance/call-steering-(sesle-yonlendirme)-whitepaper.pdf, 2009.

58. **Bock, T.** *Der humanoide Kompagnon-Roboter Pepper als Interaktionsschnittstelle zwischen unterschiedlichen SmartBuilding- bzw. SmartHome-Produkten.* Masterarbeit TH Wildau, 2018.

59. **magizbox.** *Natural Language Processing Applications.* http://magizbox.com/training/natural_language_processing/site/applications.html.

60. **Guizzo, Erico.** *Aldebaran Robotics Sells Majority Stake for $100 Million [UPDATED].* https://spectrum.ieee.org/automaton/robotics/humanoids/aldebaran-robotics-sells-majority-stake, 2012.

61. **Technische Hochschule Wildau.** *HUMANOIDE ROBOTER IM EINSATZ.* https://icampus.th-wildau.de/bewerbung-bdj-2017/robotic.html#assistenzsysteme.

62. **SoftBank Robotics.** *Who is Pepper?* https://www.ald.softbankrobotics.com/en/robots/pepper.

63. **SoftBank Robotics.** *SOLUTIONS - FOR BUSINESS.* https://www.ald.softbankrobotics.com/en/solutions/business.

64. **SoftBank Robotics.** *Pepper - Documentation - Technical overview.* http://doc.aldebaran.com/2-4/family/pepper_technical/index_pep.html.

65. **SoftBank Robotics.** *Pepper - Documentation - ALAudioDevice.* http://doc.aldebaran.com/2-5/naoqi/audio/alaudiodevice.html#alaudiodevice.

66. **SoftBank Robotics.** *NAOqi APIs.* http://doc.aldebaran.com/2-5/naoqi/index.html.

67. **SoftBank Robotics.** *Pepper - Documentation - ALAudioPlayer.* http://doc.aldebaran.com/2-5/naoqi/audio/alaudioplayer.html#alaudioplayer.

68. **SoftBank Robotics.** *Pepper - Documentation - ALAudioRecorder.* http://doc.aldebaran.com/2-5/naoqi/audio/alaudiorecorder.html#alaudiorecorder.

69. **SoftBank Robotics.** *Pepper - Documentation - ALSpeechRecognition.* http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition-api.html#alspeechrecognition-api.

70. **SoftBank Robotics.** *Pepper - Documentation - ALTextToSpeech.* http://doc.aldebaran.com/2-5/naoqi/audio/altexttospeech-api.html#altexttospeech-api.

71. **Balzert, Helmut.** *Lehrbuch der Softwaretechnik - Basiskonzepte und Requirements Engineering.* 3. Auflage. Spektrum Akademischer Verlag Heidelberg, 2009.

72. **Hoffert, Johan.** *Watson - Pepper Robot.* https://www.ibm.com/blogs/nordic-msp/pepper-robot/.

73. **Wikipedia.** *IBM.* https://de.wikipedia.org/wiki/IBM.

74. **Ferrucci, David, et al.** *Building Watson: An Overview of the DeepQA Project.*2010.

75. **Gliozzo, Alfio, et al.** *Building Cognitive Applications with IBM Watson Services: Volume 1 Getting Started.* 1. Auflage. International Business Machines Corporation, 2017. ISBN: 073844264X.

76. **IBM.** *Katalog - IBM Cloud.* https://console.bluemix.net/catalog/?taxonomyNavigation=watson&search=watson&category=watson.

77. **IBM.** *IBM Cloud - Conversation.* https://console.bluemix.net/catalog/services/conversation.

78. **IBM.** *Watson Conversation - API Reference.* https://www.ibm.com/watson/developercloud/conversation/api/v1/.

79. **Rasa.** *About.* https://rasa.com/about/.

80. **Rasa.** *Why we started Rasa.* https://rasa.com/mission/.

81. **Rasa.** *Rasa Platform: Features & support to kick-start your deployment.* https://rasa.com/products/rasa-platform/.

82. **Rasa.** *Rasa Core - Motivation.* https://core.rasa.ai/motivation.html.

83. **Rasa.** *Rasa Core - Domain, Slots, and Actions.*

84. **Rasa.** *Rasa Core - Building a Simple Bot.* https://core.rasa.ai/tutorial_basics.html.

85. **Rasa.** *Rasa Core - Stories - The Training Data.* https://core.rasa.ai/stories.html.

86. **Rasa.** *Rasa Core - Supervised Learning Tutorial.* https://core.rasa.ai/tutorial_supervised.html.

87. **Rasa.** *Rasa Core - Custom Policies.* https://core.rasa.ai/policies.html.

88. **Rasa.** *Rasa NLU - Using rasa NLU as a HTTP server.* , .

89. **Rasa.** *Rasa Core - HTTP server.* https://core.rasa.ai/http.html.

90. **Rasa.** *Rasa NLU - Entity Extraction.* https://nlu.rasa.ai/entities.html.

91. **Rasa.** *Rasa NLU - Frequently Asked Questions.* https://nlu.rasa.ai/faq.html.

92. **Rasa.** *Rasa NLU - Configuration.* https://nlu.rasa.ai/config.html.

93. **Rasa.** *Rasa NLU - Processing Pipeline.* https://nlu.rasa.ai/pipeline.html.

94. **Rasa.** *Rasa NLU - Installation.* http://nlu.rasa.ai/installation.html.

95. **Rasa.** *Rasa Core - Common Patterns.* https://core.rasa.ai/patterns.html.

96. **Rasa.** *Frequently Asked Questions.* http://nlu.rasa.ai/faq.html.

97. **Rasa.** *Rasa NLU - Training Data Format.* https://nlu.rasa.ai/dataformat.html.

98. **Rasa.** *rasa_core - train.py.* https://github.com/RasaHQ/rasa_core/blob/master/rasa_core/train.py.

99. **Feindt, Michael und Kerzel, Ulrich.** *Prognosen bewerten - Statistische Grundlagen und praktische Tipps.* Springer Verlag, 2015. ISBN: 978-3-662-44682-9.

100. **Runkler, Thomas A.** *Data Mining - Modelle und Algorithmen intelligenter Datenanalyse.* 2. Auflage. Springer Fachmedien Wiesbaden, 2015. ISBN: 978-3-8348-1694-8.

101. **Wikipedia.** *Confusion matrix.* https://en.wikipedia.org/wiki/Confusion_matrix.

102. **Lohninger, Hans.** *Gütemaße für Klassifikatoren.* http://www.statistics4u.com/fundstat_germ/ee_classifier_performance_metrics.html.

103. **Pinkal, Manfred.** *Einführung in Computerlinguistik - Statistische Modellierung I.* http://www.coli.uni-saarland.de/courses/I2CL-10/material/Folien/CL10folien9_StatistischeModelle.pdf, 2010.

104. **Wikipedia.** *Precision and recall.* https://en.wikipedia.org/wiki/Precision_and_recall.

105. **Brownlee, Jason.** *What is the Difference Between Test and Validation Datasets?* https://machinelearningmastery.com/difference-test-validation-datasets/.

106. **Saito, Takaya und Rehmsmeier, Marc.** *The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets.* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800/, 2015.

107. **Nielsen, Jakob.** *Response Times: The 3 Important Limits.* https://www.nngroup.com/articles/response-times-3-important-limits/, 1993.

108. **Lilli, W. und Frey, D.** *Die Hypothesentheorie der sozialen Wahrnehmung - Theorien der Sozialpsychologie.* 2. Auflage. 2001. S. 49 - 78. Bd. I.

109. **Költzsch, T.** *Tay - Microsofts Chat-Bot wird zum Rassisten.* https://www.golem.de/news/tay-microsofts-chat-bot-wird-zum-rassisten-1603-119976.html.

110. **Afflictor.** *Old Print Article: Electronic Brain Translates Language, Brooklyn Daily Eagle (1954).* http://afflictor.com/2016/08/29/old-print-article-electronic-brain-translates-language-brooklyn-daily-eagle-1954/.

111. **Baron, Justine.** *SAP Acquires Recast.AI to Accelerate Natural Language Processing Capabilities.* https://recast.ai/blog/sap-acquires-recast-ai-accelerate-natural-language-processing-capabilities/.

112. **Abts, D.** *Grundkurs Wirtschaftsinformatik - Eine kompakte und praxisorientierte Einführung.* 8. Auflage. Springer Vieweg Verlag, 2013. ISBN: 978-3834816696.

113. **Chaubard, F., et al.** *Natural Language Processing with Deep Learning - Part I.* https://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf, 2017.

114. **Duckling.** *Duckling.* https://github.com/facebook/duckling.

115. **Duckling.** *Introduction.* https://duckling.wit.ai/.

116. **Technische Hochschule Wildau.** *Studienführer 2017/18.* https://www.th-wildau.de/files/2_Dokumente/Broschueren-Flyer/studienfuehrer_2017_18_web_2_.pdf.

117. **Wikiquote.** *Diskussion: Victor Hugo.* https://de.wikiquote.org/wiki/Diskussion:Victor_Hugo.

118. **Wikipedia.** *Victor Hugo.* https://de.wikipedia.org/wiki/Victor_Hugo.

119. **Wikipedia.** *Thesaurus.* https://de.wikipedia.org/wiki/Thesaurus.

120. **Wikipedia.** *Jean Paul.* https://de.wikipedia.org/wiki/Jean_Paul.

121. **Wikipedia.** *Deixis.* https://de.wikipedia.org/wiki/Deixis.

122. **Traut-Mattausch, E. und Frey, D.** *Handbuch der Sozialpsychologie und Kommunikationspsychologie - Kommunikationsmodelle.* [Hrsg.] W. Bierhoff und D. Frey. 1. Auflage. hogrefe Verlag, 2006. Bd. 3. ISBN: 978-3801718442.

123. **Teufel Blog Redaktion.** *Die Abtastrate - Tastend nach dem besten Sound.* https://blog.teufel.de/abtastrate/.

124. **spaCy.** *Annotation Specifications.* https://spacy.io/api/annotation#named-entities.

125. **SoftBank Robotics.** *Pepper - Documentation - Supported languages.* http://doc.aldebaran.com/2-5/family/pepper_technical/languages_pep.html ?highlight=supported%20languages.

126. **SoftBank Robotics.** *NAOqi SDKs.* http://doc.aldebaran.com/2-5/dev/programming_index.html.

127. **SoftBank Robotics.** *Humanoid Robot 'Pepper' to Support Android.* https://www.ald.softbankrobotics.com/en/press/press-releases/humanoid-robot-pepper-to-support-android.

128. **SoftBank Robotics.** *ABOUT US - GALLERY - PEPPER.* https://www.ald.softbankrobotics.com/en/press/gallery/pepper.

129. **Rammstein.** *History.* https://www.rammstein.de/de/history/.

130. **IBM.** *IBM Watson APIs.* https://github.com/watson-developer-cloud?page=1.

Anhang

Anhang A: Web-Links zu Korpus-Datensätzen

An dieser Stelle werden gefundene Korpus-Datensätze aufgelistet:

- https://github.com/niderhoff/nlp-datasets
  Enthält nach Quellursprung sortierte Datensätze.

- https://github.com/awesomedata/awesome-public-datasets#complementary-collections
  Enthält über 30 verschiedene Domain-Datensätze.

- https://www.reddit.com/r/datasets/
  Auflistung verschiedenster Datensätze durch die Programmier-Community.

- https://www.kaggle.com/datasets
- Suchmaschine für Korpus-Datensätze.

- http://linguatools.org/tools/corpora/wikipedia-monolingual-corpora/
- Besteht aus Textabschnitte von Wikipedia für 23 Sprachen.

- https://github.com/rkadlec/ubuntu-ranking-dataset-creator
- Klassifizierte Fragen und Äußerungen Rund um das Thema Computer, Betriebssysteme und Ubuntu.

- https://github.com/karthikncode/nlp-datasets
- Allgemeine Auflistung einiger Datensätze und wissenschaftliche Dokumente zu diesen Korpus-Daten.

Einige der hier vorkommenden Links enthalten weitere Verzweigungen zu noch mehr Datensätzen. Der Großteil dieser Auflistung bezieht sich auf die Sprache Englisch.

Anhang B: Häufig gestellte Fragen bzgl. der Bibliothek

Im Zuge des Feldtests nannten die Mitarbeiterinnen und Mitarbeiter des Empfangs der Hochschulbibliothek Wildau die folgenden Äußerungen als „häufig gestellte Fragen“:

- *Können Sie mir mit dem Drucker helfen?*
- *Kann man hier drucken?*
- *Wo sind die Toiletten?*
- *Mein Computer erkennt den USB-Stick nicht. Können Sie mir helfen?*
- *Mein Dokument kommt nicht beim Drucker an.*
- *Kann man hier farbig drucken?*
- *Wie viel kosten Kopien?*
- *Wieso kann ich dieses Buch nicht ausleihen?*
- *Haben Sie Bücher zu …?*
- *Ich suche Bücher zu …*
- *Wo befinden sich die Abschluss-/Bachelor-/Masterarbeiten?*
- *Wie greife ich auf mein Bibliothekskonto zu?*
- *Was sind die Zugangsdaten zu meinem Bibliothekskonto?*
- *Wie funktionieren die elektronischen Schließfächer?*
- *Wie funktionieren die Schlösser mit dem PIN?*
- *Wie funktioniert das mit den Schließfächern?*
- *Wie schließe ich meine Sachen ein?*
- *Kann ich E-Books auch von zu Hause runterladen?*
- *Kann ich E-Books auch zu Hause nutzen?*
- *Wie lauten die Login-Daten für das VPN?*
- *Ist ein Arbeitsraum frei?*
- *Wo kann ich meine Karte mit Geld aufladen?*

Anhang C: Web-Links zu Watson Conversation Tutorials

Für IBM Watson Conversation gibt es sehr viele Anlaufstellen und Tutorials. Mit der nachfolgenden Liste wird eine kleine Übersicht gegeben:

- https://console.bluemix.net/docs/services/conversation/getting-started.html
  Offizielle A bis Z Anleitung zur Erstellung eines Watson Conversation Projektes mit Hilfe der Weboberfläche.

- https://www.ibm.com/watson/developercloud/conversation/api/v1/curl.html?curl
  Referenzdokumentation der API-Schnittstelle von Watson Conversation.

- https://github.com/IBM-Cloud?utf8=✓&q=watson&type=&language
  Offizielle Programmierbeispiele und Anleitungen von IBM zu Watson.

- https://github.com/IBM-Cloud/watson-conversation-variables
  Ausführliche Erklärung wie Variablen und Slots programmatisch verwendet werden können.

- https://www.ibm.com/blogs/bluemix/2017/11/enhance-chatbot-conversation-context-variables-system-entities/
  Anleitung eines IBM Mitarbeiter zur Verwendung von Kontext-Variablen und System-Entitäten.

- https://developer.ibm.com/in/2017/04/11/building-complex-dialogs-using-watson-conversation/
  Anleitung eines IBM Mitarbeiters zur Erstellung einer komplexen Dialogkonversation.

- https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248387.html
  Generelle Einführung in Watson mit Übersicht zu allen Diensten.

- https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248394.html
  Einführung in Watson Conversation unter allen Gesichtspunkten.

Anhang D: Inhalt der beigelegten CD-ROM

Der Arbeit ist eine CD-ROM mit folgenden Inhalten beigelegt:

- **„Anf"-Ordner:** Innerhalb dieses Ordners befinden sich sämtliche User Stories und Anforderungen als Excel-Tabellen. Sie sind der Ursprung für diesbezügliche Erklärungen im Text.
- **„Eval"-Ordner:** Dieses Verzeichnis enthält die Vorauswahl der zwanzig NLP-Systeme, sämtliche Performance- und Latenztests für Rasa und Watson Conversation (ergibt alleine 34 Dateien) sowie die Endauswahl zwischen diesen beiden Systemen.
- **„Impl"-Ordner:** Enthält den Quelltext des Testprogramms (HelloRasa, HelloWatson), des Prototyps (PepperUsingNLP) und die zugehörigen Konfigurations-, Test-, und Trainingsdateien. Mit HelloRasaServer liegt eine rudimentäre Version des Prototyps unter Verwendung von Rasa vor.
- **„Install"-Ordner:** Besteht aus Markdown-Dateien, welche die Installation von Rasa, Watson Conversation und Wit für das Ubuntu-Betriebssystem beschreiben.
- **Masterarbeit.pdf:** Ist die hier vorliegende Abschlussarbeit im PDF-Format.