TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Conversational User Interfaces: Einführung in den Stand der Technik und Anwendungsmöglichkeiten in der Altenpflege

Projektarbeit 1

Im Rahmen des Masterstudiengangs
Applied Research in Engineering Sciences

vorgelegt von:	Markus Glas
Studiengang:	Applied Research in Engineering Sciences
Betreuung durch:	Prof. Dr. Jens Albrecht

Abstract

Der demografische Wandel ist eine wachsende Herausforderung in vielen Industrienationen. Die Gruppe älterer Menschen im Alter von 60 oder höher wächst konstant und wird voraussichtlich 2030 die Marke von einer Milliarde Menschen überschreiten. Um die steigende Nachfrage an Pflegepersonal zu befriedigen stehen, neben politischen und sozialwissenschaftlichen Anstrengungen, auch technologische Lösungen im Fokus der Wissenschaft und Öffentlichkeit. Sprachschnittstellen haben sich in den letzten Jahren als ein beliebtes Mittel etabliert, um älteren Menschen den Zugang zu neuen Technologien und Informationen zu erleichtern.

In dieser Arbeit werden Möglichkeiten zur Umsetzung einer Sprachschnittstelle durch den Einsatz aktueller Technologien gezeigt. Nach einer Einführung in den grundlegenden Aufbau von Sprachschnittstellen, werden konkrete Implementierungen betrachtet. Anhand bekannter Anforderungen zur Altenpflege werden verschiedene Systeme evaluiert und anschließend in einem Prototypen umgesetzt.

Die Ergebnisse zeigen, dass einige Systeme bereits ausgereift sind und einen produktiven Einsatz erlauben. Durch die Nutzung vorhandener Schnittstellen ist ein modularer Aufbau und somit die Kombination von Komponenten unterschiedlicher Hersteller möglich.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Conversational User Interfaces (CUI) ermöglichen eine natürlichsprachliche Kommunikation zwischen Computer und Mensch mittels natürlicher Sprache. Seit den ersten textbasierten Dialogsystemen wie ELIZA [Wei66], welches einen Psychotherapeuten mittels einfacher Textanalyse und Wortvergleichen simulierte, haben sich sowohl die verfügbaren Technologien als auch die Einsatzmöglichkeiten verändert. CUI finden heutzutage Anwendung in der Kundenbetreuung, bei Reisebuchungen oder Lieferdiensten bis hin zur Beratung bei medizinischen Fragen. Durch die Vielfalt an Anwendungsgebieten ist ebenfalls das Angebot an Softwarebibliotheken, Frameworks und Diensten zur Erstellung von CUI-Systemen gestiegen.

Neben rein textbasierten Dialogsystemen erleben sprachgesteuerte Systeme, genannt Voice User Interfaces (VUI), weite Verbreitung, vor allem durch deren Einsatz in Smartphones und Smart Speaker. Digitale Sprachassistenten wie Apple's Siri, Google's Assistant, Amazon's Alexa oder Microsoft's Cortana sind Bestandteile der Betriebssysteme des jeweiligen Herstellers und erreichen dadurch eine breite Masse an Anwendern. Gerade älteren Menschen kann durch VUI der Lernprozess komplexer Anwendungen erspart und ein einfacher Zugang zu Technologien und Informationen gewährt werden [PVG+13].

1.1 Motivation

Der demografische Wandel ist eine zunehmende Herausforderung in vielen Industrienationen. Personen der geburtenstarken Jahre Mitte des letzten Jahrhunderts erreichen zunehmend das Rentenalter. Die Bevölkerungsgruppe 60+ hatte im Jahr 2017 einen Anteil von 13 % der Weltbevölkerung und umfasste somit 962 Millionen Personen [Uni17]. Europa besitzt den größten Anteil in dieser Bevölkerungsgruppe mit 25 %.

Der demografische Wandel trifft jedoch nicht nur die Industrieländer, sondern alle Teile der Welt. Für das Jahr 2030 wird die Anzahl der Personen im Alter von 60 oder älter bereits auf 1,4 Milliarden und für 2050 auf 2,1 Milliarden prognostiziert [Uni17].

Dieser Wandel stellt die Länder vor neue Herausforderungen, um älteren Menschen durch die Bereitstellung einer ausreichenden Anzahl an Pflegepersonal und entsprechenden Einrichtungen ein selbstbestimmtes Leben zu ermöglichen. Daneben steigt die Anzahl derer, die auch im hohen Alter trotz nachlassender Selbstständigkeit im eigenen zu Hause leben möchten [VLP14]. Zunehmende physische und kognitive Einschränkung können ein selbstbestimmtes Leben im eigenen zu Hause erschweren. Des Weiteren kann eine mögliche Vereinsamung den Gesundheitszustand weiter verschlechtern.

Die Forschungsgemeinschaft begegnete dieser Problematik jüngst mit einer Vielzahl von Anwendung im Bereich des Ambient Assisted Living (AAL). Gerade Anwendungen zur Überwachung des Gesundheitszustands durch Sensoren oder Kameras haben in diesem Bereich hohe Popularität [PLJ+14]. Dem Problem der Vereinsamung begegnen jedoch nur wenige, was Anreiz zur weiteren Forschung auf diesem Gebiet gibt.

1.2 Aufbau der Arbeit

Diese Arbeit gliedert sich nach der Einleitung in vier weitere Kapitel.

Im Folgenden Kapitel 2 wird zunächst eine Einführung in den Aufbau heutiger CUI gegeben. Dabei werden die einzelnen Bestandteile näher erläutert und Möglichkeiten zur Umsetzung aufgezeigt.

Kapitel 3 definiert die Anforderungen zur Erstellung eines CUI in der Altenpflege. Anschließend werden verschiedene Softwarebibliotheken und Frameworks zur Erstellung des CUI verglichen und eine Auswahl für die spätere Implementierung getroffen.

Kapitel 4 beschreibt die Implementierung der zuvor definierten Anforderungen in Form eines Prototypen. Gemäß der Evaluation werden die Softwarekomponenten eingesetzt.

Kapitel 5 resümiert die Arbeit und gibt einen Ausblick auf zukünftige Erweiterungen des Projekts und Optimierungsmöglichkeiten.

2 Conversational User Interfaces

Conversational User Interfaces ermöglichen eine natürlichsprachliche Kommunikation zwischen Mensch und Computer. Dabei wird versucht den Konversationsverlauf durch die Nachbildung der Mensch-zu-Mensch-Konversation möglichst natürlich wirken zu lassen. Um dies zu erreichen ist es notwendig, den menschlichen Konversationsverlauf zu analysieren, um eine formale Definition der Bestandteile und Abläufe erstellen zu können. Im Nachfolgenden wird zunächst der Aufbau einer Konversationen näher betrachtet und anschließend die Bestandteile aktueller CUI-Systeme erläutert.

2.1 Text- und Sprachbasierte Interfaces

Heutige CUI-Systeme lassen sich in zwei Kategorien aufteilen [McT16]. Zum einen existieren rein textbasierte Dialogsysteme, oft auch als *Chatbots* oder *Chatterbots* bezeichnet. Zum anderen existieren *Voice User Interfaces* (VUI), welche sprachbasiert arbeiten und dem Nutzer eine natürlichere Kommunikationsform anbieten. Die im Hintergrund stattfindende Spracherkennung und Kontrolle des Dialogflusses ist in beiden Fällen ähnlich oder zum Teil komplett identisch. Teilweise werden Chatbot-Systeme durch eine Sprachschnittstelle erweitert und bieten dem Nutzer sowohl text- als auch sprachbasierte Interaktion an (vgl. Amazon Lex Service[1])

2.2 Aufbau einer Konversation

Die Grundlage natürlichsprachlicher Kommunikation bildet die *Konversation*, welche alltäglicher Gebrauchsgegenstand menschlicher Interaktion ist. Als *Konversation* wird eine informelle Form der Unterhaltung, bei der etwa Neuigkeiten oder Ansichten ausgetauscht werden, beschrieben [McT16]. Daneben können die Aufrechterhaltung, der Aus- oder Aufbau sozialer Beziehungen mögliche Ziele darstellen.

[1] https://aws.amazon.com/de/lex/

Im Gegensatz zur Konversation kann der *Dialog* als eine transaktionsorientierte Form der Interaktion beschrieben werden. Der Nutzer eines Dialogsystems kann etwa bei der Suche nach Wetterdaten die Ausgabe der entsprechenden Informationen erwarten.

Der Dialog kann durch die von Searle [SS69] publizierte *Theory of Speech Acts* weiter spezifiziert werden. Der *Sprecher* verfolgt mit seiner Äußerung eine gewisse *Absicht* (*Intention*, kurz *Intent*), woraufhin der *Zuhörer* eine bestimmte *Aktion* ausführt (vgl. 2.1). Zum Austausch dienen *Äußerungen* (*Utterances*). Der Sprecher trifft dabei ebenfalls die Annahme, dass der Zuhörer das Wissen zur Beantwortung besitzt oder sich dieses durch weitere Quellen aneignen kann.

Eine Abgrenzung zwischen *Dialog* und *Konversation* kann jedoch nicht immer erreicht werden, da eine klare Definition fehlt und Mischformen im Alltag durchaus üblich sind.

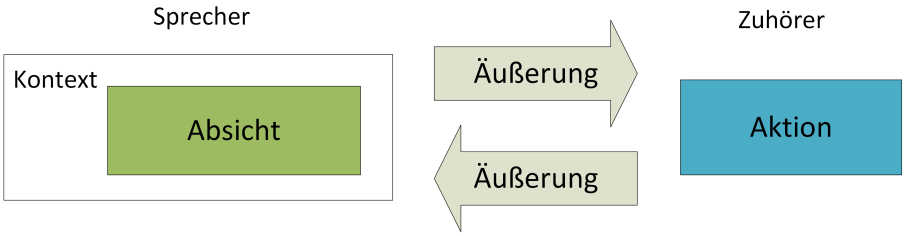

Abbildung 2.1: Aufbau einer Konversation zwischen Sprecher und Zuhörer

Neben *Konversation* und *Dialog* wird der Term *dialogorientiert* (*conversational*) verwendet, um Systeme mit menschenähnlichen Charakteristiken und dem Verständnis natürlicher Sprache zu beschreiben [McT16]. Damit wird eine Abgrenzung zu rein befehlsorientierten Systemen getroffen, welche lediglich bestimmte Wörter oder kurze Sätze erkennen können.

Die Form des Dialogs mit dem Ziel Informationen des Zuhörers zu erhalten bildet die Grundlage heutiger CUI-Systeme. Üblicherweise herrscht dabei eine klare Rollenverteilung, in welcher der Benutzer die Rolle des Sprechers und das CUI-System die des Zuhörers übernimmt. Jedoch sind Mischformen möglich und teilweise auch notwendig, um fehlende Informationen zur Durchführung einer Aktion beim Benutzer zu erfragen und eine menschenähnliche Kommunikation zu ermöglichen.

2.2.1 Absichten und Aktionen

Absichten (Intents) können durch den Benutzer auf unterschiedliche Weise geäußert werden, woraufhin eine *1-zu-n*-Beziehung zwischen Absicht und Äußerung entsteht (vgl. Abb. 2.2). Daneben können Äußerungen *Entitäten* enthalten, welche im CUI-System üblicherweise in Form von Objektklassen oder Datentypen dargestellt werden. Sie können bei der Erkennung einer Absicht, der Ausführung einer Aktion oder der Verfolgung des Kontextes helfen. Beispiele sind Orts- und Zeitangaben oder Eigennamen. Die Festlegung der Entitäten geschieht während der Implementierung und ist Teil des Entwicklungsprozesses eines CUI-Systems.

Aktionen können die Ausgabe fest hinterlegter Textnachrichten, etwa zur Begrüßung *"Hallo, ich freue mich von dir zu hören."*, bis hin zur Einbeziehung von Drittsystemen darstellen. Notwendig wird dies etwa bei der Suche eines Termins im Kalender oder der Abfrage aktueller Wetterdaten eines Wetterdienstes.

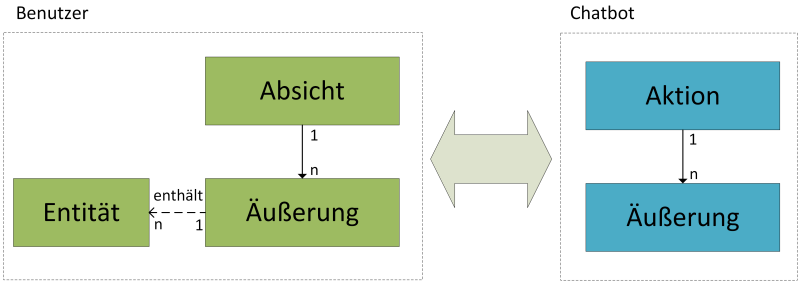

Abbildung 2.2: Aufbau einer Konversation mit Chatbots

2.2.2 Entwurf der Konversation bei Chatbots

Zum Aufbau eines textbasierten Dialogsystems (Chatbot) sollte der Funktionsumfang und dazugehörige Aktionen bei initialem Entwurf möglichst gering gehalten werden. Ein System, welches alle möglichen Äußerungen eines Benutzers erkennt und beliebige Aktionen ausführen kann ist mit heutiger Technologie schwer umsetzbar [KD18]. Ein besseres Vorgehen liegt darin, den Funktionsumfang anfangs auf ein Minimum zu beschränken und somit die Anzahl möglicher Benutzeräußerungen einzugrenzen. Ein Chatbot zur Flugbuchung sollte etwa nicht auch medizinische Fragen beantworten können. Für viele Anwendungsgebiete sind weniger als drei Funktionen ausreichend [KD18]. Sollte der Funktionsumfang wachsen, ist eine Kombination mehrere Subsystem möglich.

2.3 Komponenten eines Conversational Interface

Conversational Interfaces werden aus mehreren Komponenten zusammengesetzt (vgl. Abb. 2.3) [McT16]. Dabei bilden die Komponenten zum Verständnis natürlicher Sprache (*Natural Language Understanding*, NLU) und zur Verwaltung des Dialogflusses und -kontextes (*Dialogmanagement*) die zentralen Bestandteile. Soll das System neben textbasierten auch sprachbasierte Eingaben verarbeiten können, wird das System durch eine Komponente zu Übersetzung von Sprachdaten in Text (*Speech-to-Text*, STT) und zur Übersetzung von Text in synthetische Sprache (*Text-to-Speech*, TTS) erweitert.

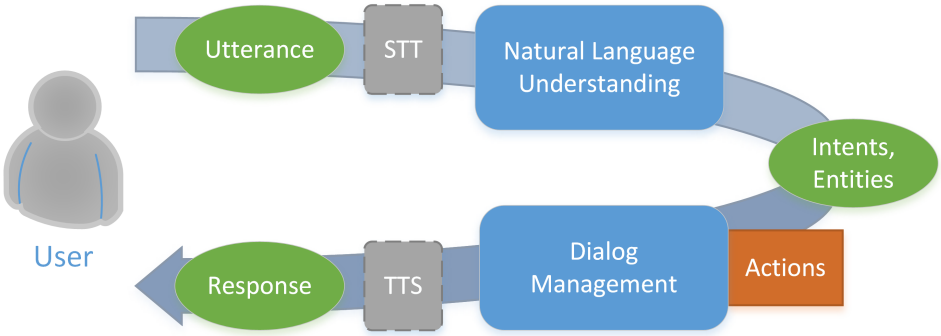

Abbildung 2.3: Ablauf eines Dialog-Systems

Der Ablauf einer Konversation gestaltet sich wie folgt: Zunächst werden die Audiodaten des Benutzers in Text gewandelt, anschließend wird die Bedeutung der Wörter durch das NLU interpretiert. Dies beinhaltet auch die Extraktion der Absichten und Entitäten. Das Dialogmanagement entscheidet über die Auswahl der passenden Antwort anhand des Intent, der Entitäten und des gespeicherten Kontextes. Gegebenenfalls werden weitere Aktionen zur Vervollständigung der Antwort ausgeführt. Die textuelle Antwort wird von der TTS-Komponente in Sprache synthetisiert und ausgegeben.

In den nachfolgenden Abschnitten wird die Funktionsweise der einzelnen Komponenten näher erläutert.

2.3.1 Natural Language Processing / Understanding

Natural Language Processing (NLP) beschreibt Anwendungen und Forschungsgebiete mit dem Ziel, natürliche Sprache in Computern zu verarbeiten und zu verstehen [Cho05]. Zur Umsetzung werden Techniken aus den Bereichen der Computerlinguistik, Logik und des maschinellen Lernens genutzt.

Natural Language Understanding stellt eine Untergruppe des *Natural Language Processing* dar und versucht, mit Hilfe von Wissensdatenbanken und der Analyse von Grammatik, Semantik und Pragmatik ein Verständnis des Textes zu erlangen [Bat95, Win72]. Die Analyse der Semantik stellt dabei seit jeher eine der schwierigsten Aufgaben dar. Während frühere Systeme rein auf Prädikatenlogik basierten, nutzen aktuelle Systeme Machine Learning mit Hilfe großer Textkorpora zur Bewältigung dieser Aufgabe [BFPN17, Bat95].

Am Beispiel des Open-Source-NLU-Tools *Rasa NLU* soll ein Machine-Learning-Ansatz zum Verständnis natürlicher Sprache erläutert werden (vgl. Abb. 2.4). Der Text wird zunächst in Wortbestandteile, Tokens, zerlegt und jedes Wort mittels *Part-of-Speech-Tagging* (POS-Tagging) der entsprechenden Wortart (Nomen, Verb, Adjektiv etc.) zugeordnet. Anschließend werden Vektorrepräsentationen für jedes Wort gebildet. Die Vektorrepräsentationen der einzelnen Wörter werden genutzt, um Eigennamen (Personen, Unternehmen, Orte, Produkte) zu finden, was als *Named Entity Recognition* (NER) bezeichnet wird. Im Falle von *Rasa NLU* findet dafür ein Abgleich anhand eines vortrainierten Machine-Learning-Modells statt, welches auf eine Vielzahl gängiger Eigennamen der jeweiligen Sprache trainiert wurde.

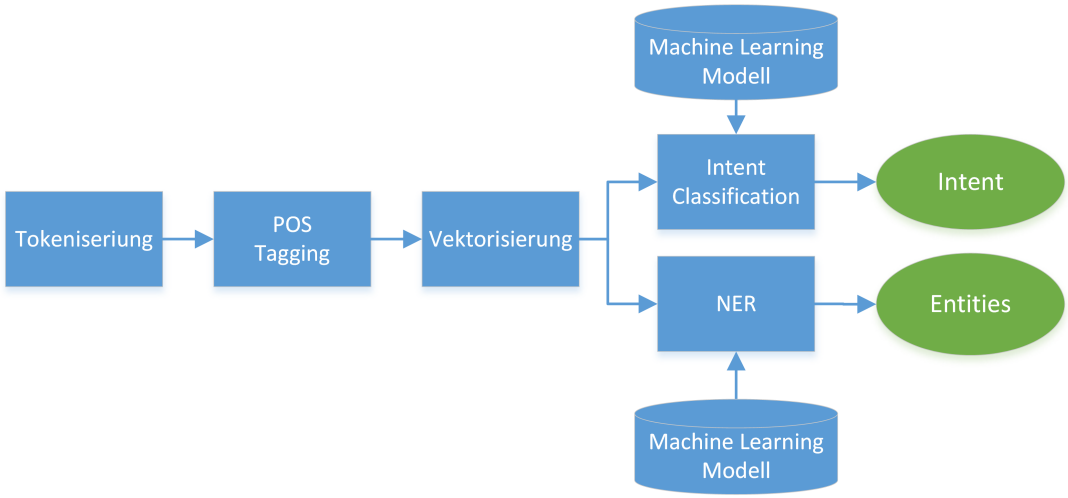

Abbildung 2.4: Bestandteile des NLU-Systems *Rasa NLU*

Ebenfalls werden Vektorrepräsentationen für jeden Satz gebildet, um die Absicht der Äußerung zu erkennen. Im Gegensatz zu NER wird hier kein vortrainiertes Modell genutzt, vielmehr wird das Modell während der Konfiguration mit Hilfe entsprechender Beispieläußerungen trainiert.

Die Auswahl des korrekten Intents wird im produktiven System als Klassifikationsproblem mit mehreren Klassen behandelt und die Ausgabe ordnet allen verfügbaren Klassen, in diesem Fall gleichzusetzen mit Intents, die entsprechenden Wahrscheinlichkeiten zu. Abbildung 2.1 zeigt eine mögliche Antwort eines NLU-Systems mit den entsprechenden Intents und deren Wahrscheinlichkeiten.

```
"query": "Wie wird das Wetter heute?",
"topScoringIntent": {
  "intent": "inform_weather",
  "score": 0.9991042
},
"intents": [
  {
    "intent": "inform_weather",
    "score": 0.9991042
  },
  {
    "intent": "welcome",
    "score": 0.00050777354
  },
  {
    "intent": "None",
    "score": 0.00312528
  }
]
```

Listing 2.1: Ausgabe eines NLU-Systems

Die *NLU-Komponente* übernimmt in Conversational Interfaces somit die Aufgabe der Erkennung eines Intents und die Extraktion der Entitäten aus den Äußerungen des Benutzers. Sie ist stark sprachabhängig und wird meist auf eine Zielsprache trainiert und optimiert. Die Erstellung der Trainingsdaten erfolgt in der Regel durch die manuelle Eingabe von Beispielsätzen durch den Entwickler.

2.3.2 Dialogmanagement

Das *Dialogmanagement* (DM) ist zuständig für die Kontrolle des Dialogflusses, das bedeutet die Generierung oder Auswahl passender Antworten, die Speicherung der Entitäten und die Entscheidungen über die Ausführung von Aktionen wie API- oder Datenbank-Abfragen [McT16]. Das DM bildet eine zentrale Komponente im Aufbau eines Conver-

sational Interface.

Traum und Larsson [TL03] definieren für das Dialogmanagement vier Kernfunktionen innerhalb eines CUI:

1. Aktualisierung des Dialogkontextes auf Basis der Kommunikation.
2. Bereitstellung eines Kontextes für die Interpretation der Anfragen.
3. Anbindung anderer Systeme (Datenbanken, Kalender, Backend-Systeme) zur Koordination des Dialogs.
4. Entscheidung über den Inhalt und Zeitpunkt der nächsten Ausgabe.

Die Entscheidung zur Auswahl einer Antwort kann auf mehreren Faktoren, wie den angegebenen Entitäten, der verwendeten Domäne und den Antworten externer Systeme basieren. Zur Handhabung der Kernfunktionen im produktiven System sind *Dialogstrategien* notwendig [McT16]. Diese definieren, wann das DM Aktionen ausführt, die Nachfrage nach fehlenden Informationen stellt oder eine Fehlerbehandlung initiiert. Des Weiteren ist es erforderlich die Dialoghistorie nachzuverfolgen, um wiederholte Nachfragen zu vermeiden. Generell existieren zur Umsetzung der Dialogstrategien unterschiedliche Ansätze, welche sich in zwei Kategorien einteilen lassen: *Regelbasiertes* und *Datengetriebenes statistisches* Dialogmanagement.

Regelbasiertes Dialogmanagement

Ein einfacher Ansatz zur Realisierung eines regelbasierten Dialogmanagement bilden *endliche zustandsbasierte* (finite-state) Systeme [McT16]. Der Pfad eines Dialogverlaufs wird in Form eines endlichen Zustandsautomaten abgebildet und fest hinterlegte Äußerungen bilden die Bedingungen für den Zustandsübergang (vgl. Abb. 2.5). Dieser Ansatz kann durch seinen expliziten Entwurf eine hohe Genauigkeit erreichen, ist jedoch in der Interaktion auf die hinterlegten Zustandsübergänge beschränkt. Des Weiteren ist die Erstellung möglicher Äußerungen im Voraus schwierig und erfordert viel Wissen über die verwendete Domäne [LJK+10].

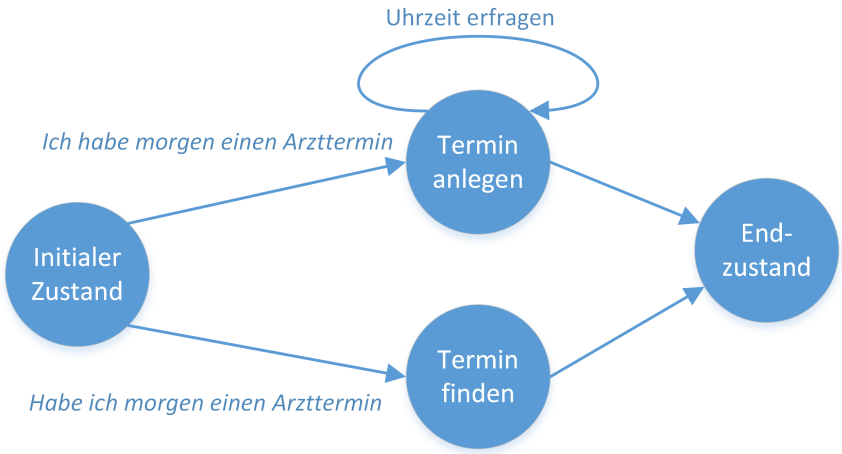

Abbildung 2.5: Regelbasiertes Dialogmanagement als endlicher Zustandsautomat

Eine Erweiterung des zustandsbasierten DM bildet das *Frame-based* Dialogmanagement [McT16]. Die Zustände besitzen keinen vorgegebenen Pfad, sondern Rahmen (Frames) mit entsprechenden Platzhaltern für jede Eigenschaft, welche das System beim Nutzer erfragen muss (vgl. Listing 2.2). Gegenüber zustandsbasierten Systemen können mehrere Informationen gleichzeitig und in jeglicher Reihenfolge angegeben werden. Die Bedingungen für einen Zustandswechsel werden in Frame-based DM Systemen, wie VoiceXML[2], in Form von If-Else-Anweisungen realisiert. Die Implementierung dieser Bedingungen kann bei komplexen Systemen sehr zeitaufwendig sein und die Erweiterbarkeit erschweren.

```
1  APPOINTMENT FRAME:
2    SUBJECT: "Zahnarzt"
3    TIME:  "Dienstag, 10:30 Uhr"
4    PLACE: "Sulzbacher Str. 1"
```

Listing 2.2: Frame-based Dialogmanagement

Statistisches Dialogmanagement

Statistisches Dialogmanagement versucht die genannten Nachteile der regelbasierten Ansätze zu beheben und gleichzeitig die Erstellung eines Systems zu beschleunigen. Der Ansatz basiert auf der Idee, aus Korpora mit Mensch-Computer-Dialogen zu lernen und automatisch Regeln für den Dialogfluss zu erstellen [You02]. Der Entwickler befasst sich dabei nicht mit einzelnen Zuständen, sondern mit der Erfüllung einer Absicht und benötigt somit nur einen abstrakten Blick auf den möglichen Dialogverlauf.

[2] https://www.w3.org/Voice/Guide/

Durch den Einsatz von Machine Learning kann dieses Verfahren in heutigen Systemen effizient umgesetzt werden.

Ein Nachteil des datengetriebenen Ansatzes ist die aufwendige Erhebung echter Konversationsdaten und die zeitintensive Annotation des Datenkorpus [LJK+10, McT16]. Um diesen Problemen entgegenzuwirken wurden alternative Techniken entwickelt, welche die Erfassung und Kennzeichnung erleichtern. Dazu zählen:

- *Wizard-of-Oz-Technik*
- *Interaktives Lernen*

Die *Wizard-of-Oz-Technik* beschreibt die Simulation des späteren Systems durch einen Menschen. Ein Proband kommuniziert dabei mit einem Menschen via Konversationsschnittstelle unter dem Vorwand, mit einer Computeranwendung zu kommunizieren. Der Konversationsverlauf wird aufgezeichnet und kann für das spätere Training des Systems und zur Erstellung der Antworten verwendet werden.

Interaktives Lernen oder *Machine Teaching* beschreibt einen Ansatz, bei dem ein Tester nach jeder Antworten des Systems oder zum Abschluss einer Konversation Rückmeldung über die Richtigkeit der Antworten gibt (vgl. Abb. 2.6). Die Rückmeldungen werden aufgezeichnet und können für die Optimierung und Erweiterung der Trainingsdaten genutzt werden.

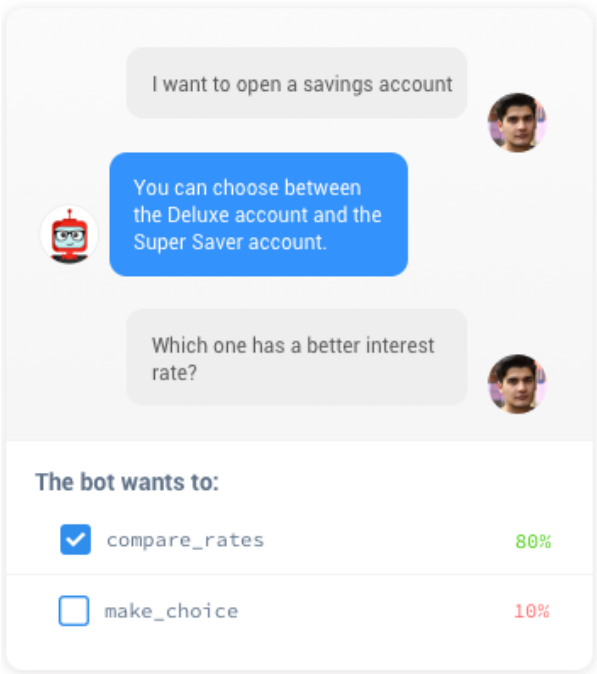

Abbildung 2.6: Beispiel eines interaktiven Lernprozesses mit einem Chatbot[3]

2.3.3 Spracheingabe und -ausgabe

Zur Erweiterung eines Conversational Interface um sprachbasierte Interaktion, sind Komponenten zur Transformation des akustischen Signals des Benutzers in Text (*Speech-to-Text*, STT) und zur Ausgabe von Text in Sprache (*Text-to-Speech*, TTS) notwendig.

Speech-to-Text

Die Transformation gesprochener Sprache in Text ist auch unter dem Namen *Automatic Speech Recognition* (ASR) bekannt. Durch signifikante Verbesserungen in den Erkennungsraten und der Verbreitung in Smartphones in den letzten Jahren bilden Spracheingaben heute eine vertraute Technologie für eine Vielzahl von Nutzern [Pea16]. Grundlagen für die Verbesserungen in den Erkennungsraten sind die Verfügbarkeit großer Sprach- und Textkorpora in Kombination mit optimierten Machine-Lerning-Algorithmen und Fortschritten in der Mikrofontechnologie [McT16].

Die Übersetzung gesprochener Sprache in Text gliedert sich in mehrere Sequenzen (vgl. Abb. 2.8) [CGB04]. Die *Endpunktbestimmung* ist zuständig für die Detektion des Anfangs und Endes einer Spracheingabe. Die *Merkmalsextraktion* transformiert die Äußerung in Merkmalsvektoren, welche eine numerische Repräsentation der Sprachcharakteristiken, wie Stimmhöhe und Sprachgeschwindigkeit bilden. Die Merkmalsvektoren werden anschließend der *Spracherkennung* übergeben, welche die passende Zeichenkette anhand eines *Erkennungsmodells* (recognition model) sucht. Das Modell beinhaltet mögliche Zeichenketten die ein Benutzer äußern kann, inklusive deren Aussprache in anwendungsspezifischer Form.

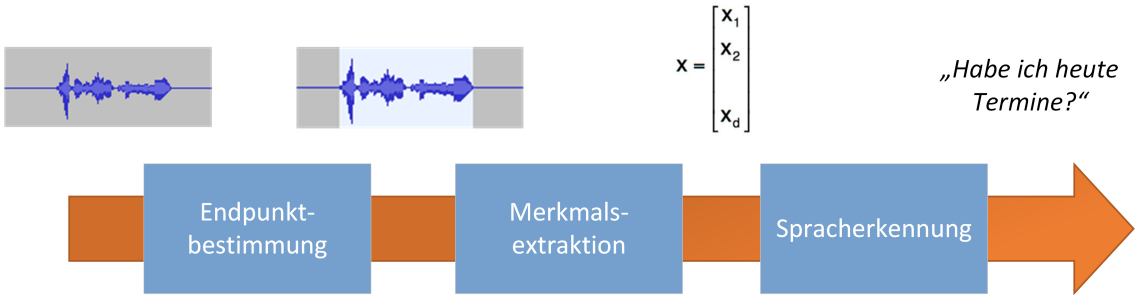

Abbildung 2.7: Ablauf eines Speech-to-Text Prozesses

Bei Umsetzung einer Sprachschnittstelle in Conversational Interfaces ist darauf zu achten, dass die gesprochene Sprache von der geschriebenen abweichen kann [McT16].

[3] https://medium.com/rasa-blog/a-new-approach-to-conversational-software-2e64a5d05f2a

Eine reine Übersetzung der Sprachdaten in Text kann in einigen Fällen nicht ausreichen und Anpassungen an der NLU-Komponente und des Dialogmanagements erfordern.

Text-to-Speech

Text-to-Speech beschreibt die Transformation von geschriebenen Text in Sprache. Früherer Ansätze verwendeten fast ausschließlich aufgenommene menschliche Sprachdaten zur Erzeugung einer Antwort [Pea16]. Die Qualität der produzierten synthetischen Sprache hat sich in den letzten Jahren deutlich verbessert und erlaubt heutzutage die Ausgabe menschenähnlicher Sprache [McT16]. Aktuelle Systeme, wie Microsoft's *Cortana*, nutzen hybride Ansätze aus menschlicher Stimme und synthetischen Elementen, welche bei Bedarf zusammengesetzt werden.

Die Transformation von Text in synthetische Sprache gliedert sich allgemein in mehrere Phasen (vgl. Abb. 2.8).

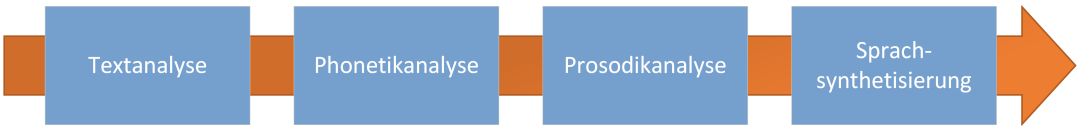

Abbildung 2.8: Ablauf eines Text-to-Speech Prozesses

Die *Textanalyse* ist notwendig, um Texte vorab zu normalisieren und Satzgrenzen oder andere Satzzeichen zu erkennen [McT16]. Dies hilft dabei, die spätere Stimmlage anzupassen und somit eine natürlichere Sprachausgabe zu erreichen. Des Weiteren ist es Aufgabe der Textanalyse Abkürzungen, Eigennamen, Datumsangaben und numerische Werte zu vereinheitlichen. Zur Umsetzung werden Klassifikatoren und Listen bekannter Wörter, Akronyme und Abkürzungen genutzt.

Die *Phonetikanalyse* sucht zu jedem normalisierten Wort die entsprechende Aussprache anhand von *Aussprachewörterbüchern*. Die verwendete Lautschrift ist dabei abhängig vom eingesetzten System. Ist ein Wort nicht Bestandteil des Wörterbuchs, wird ein *Graphem-zu-Phonem-Prozess* (G2P) mittels supervised Machine-Learning ausgeführt, um die Aussprache vorherzusagen [KTPPSI16].

Prosodie umfasst Tonhöhe, Lautstärke, Tempo und Rhythmus einer Sprache [McT16]. Damit lassen sich Unterschiede zwischen Frage, Aussage oder unterschiedlichen Emotionen zum Ausdruck bringen. Gängige Modelle zur Prosodikanalyse sind *ToBI* (Tone and Break Indices) [BHSH04] und *Tilt* [Tay00].

3 Anforderungen und Technologiewahl

Zur Umsetzung des späteren Prototypen werden Anforderung zur Entwicklung eines CUI in der Altenpflege definiert. Für das Training und zur Validierung eines geeigneten NLU-Systems wird ein domänenspezifischer Datenkorpus erstellt. Verschiedene Softwarebibliotheken und Frameworks zur Erstellung der NLU-, DM- und Sprachkomponente werden hinsichtlich ihrer Leistungsfähigkeit im vorliegenden Anwendungsfall und ihrer Kompatibilität untereinander verglichen.

3.1 Aufbau des Systems

Der Aufbau des späteren Prototypen setzt sich aus den Komponenten *Text-/Sprachschnittstelle*, *Dialogmanagement* und *Natural Language Understanding* zusammen. Die Kernkomponente bildet das Dialogmanagement, welches neben der Koordination von Nachrichten innerhalb der Subsysteme auch die Kommunikation zu externen Schnittstellen zur Aufgabe hat.

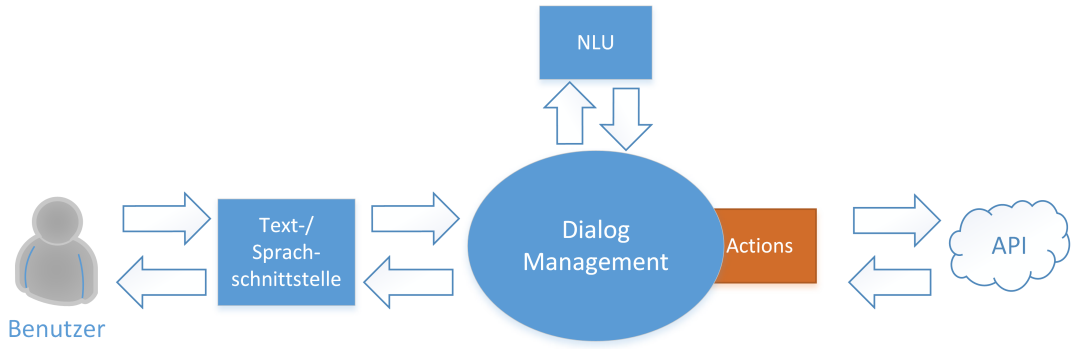

Abbildung 3.1: Schematischer Aufbau des späteren Systemes

3.2 Anforderungsdefinition

Die Anforderungen des Prototypen werden zunächst in Form von Use Cases definiert, welche auch als Grundlage für die Erstellung des Datenkorpus dienen. Die Use Cases werden auf Basis existierender CUI-Systeme zur (Alten-)Pflege definiert. Der Fokus liegt dabei auf Funktionen zur sozialen Interaktion, auch bekannt als *Sozialer Begleiter* oder *Soical Companion*.

Bisherige Anforderungsanalysen zeigten vor allem das Bedürfnis nach einem Gesprächspartner, um der Einsamkeit und Isolation entgegenzuwirken. Viele Senioren sind verwitwet, leben allein und haben nur selten Kontakt zu Familienmitgliedern, da diese bereits selbst erwachsen sind, eine eigene Familie haben oder ihre berufliche Karriere verfolgen. Des Weiteren sind viele der Freunde bereits verstorben oder leiden unter physischen Beeinträchtigungen, welche Besuche erschweren. Die Einsamkeit kann zu einer Wahrnehmung der Vernachlässigung führen [MBMU17]. Es besteht daher der Wunsch nach einer Kontaktperson mit der eine Unterhaltung möglich ist und welche ihnen zuhört. Die Akzeptanz eines virtuellen Gesprächspartners ist in vielen Teilen der Gesellschaft durchaus gegeben, auch wenn dieser keinen vollständigen Ersatz zur menschlichen Interaktion darstellen kann [MBMU17].

Soziale Eigenschaften

Um dem Benutzer das Gefühl eines menschenähnlichen Gesprächspartners zu vermitteln, soll der Bot sich mit einem Namen vorstellen und ebenfalls den Benutzer auffordern ihm seinen mitzuteilen. Dieser wird mit gegebenenfalls vorhandenen Eigenschaften, wie Alter oder Geschlecht, abgespeichert und kann zur Personalisierung der Antworten genutzt werden. Der Anwendungsfall *Personalien erfragen* resultiert aus diesen Anforderungen.

Ein *Social Companion* sollte darüber hinaus über Neuigkeiten und aktuelle Themen berichten und den Senioren erlauben, über ihre Vergangenheit zu sprechen und Ihnen zuhören [MBMU17]. Themen über psychische und physische Beschwerden, kognitive Einschränkungen oder die Isolation selbst sollten nicht Teil der Konversation sein. In der prototypischen Umsetzung wird dies durch den Anwendungsfall *Nachrichten vorlesen* umgesetzt.

Instrumentelle Aktivitäten

Virtuelle Assistenten können neben der Bereitstellung eines Konversationspartners auch Einschränkungen in den instrumentellen Aktivitäten (instrumental activities of daily living - IADL) kompensieren [OM13]. Diese umfassen alle Aktivitäten, welche für ein unabhängiges Leben notwendig sind, etwa die Zubereitung von Mahlzeiten, das Führen von Finanzangelegenheiten, die Kommunikation innerhalb der Gesellschaft, Hausarbeiten oder das Einnehmen von Medikamenten. Virtuelle Assistenten in Form von CUI oder Sprachassistenten können diese Aktivitäten nicht vollständig übernehmen, jedoch bei gewissen Teilen unterstützen, indem hilfreiche Informationen bereitgestellt werden. Die vollständige Automatisierung dieser Aktivitäten ist auch nicht wünschenswert, da sie vom eigenen Durchführen der Aufgaben abhält und somit vom Trainieren der kognitiven Fähigkeiten [OM13]. Die Anwendungsfälle *Termin finden* und *Termin erstellen* sollen durch die Hilfe bei Planung des Alltags eine erste Unterstützung der instrumentellen Aktivitäten bieten.

Erinnerungen

Viele ältere Menschen leiden unter Gedächtnisschwierigkeiten und Demenz. Die Erinnerung an Termine oder einfache Tätigkeiten fällt daher schwer, weshalb Anwendungsfälle zur Erinnerung an Termine und zum regelmäßigen Trinken erstellt werden. Diese werden, anders als die vorherigen Anwendungsfälle, selbstständig von System gesteuert und bedürfen keiner Spracheingabe des Benutzers.

Die resultierenden Use Cases für die prototypische Umsetzung sind in Tabelle 3.1 zusammengefasst.

Use Case	**Vorbedingung**	**Aktion (Erfolg)**
Personalien erfragen	keine	Antworten werden personalisiert
Nachrichten vorlesen	Verbindung zu einer News-Plattform	Nachrichten werden ausgegeben
Erinnern	Verbindung zum Kalender	Erinnerung wird ausgegeben
Termin erstellen	Verbindung zum Kalender	Termin wird im Kalender erstellt
Termin finden	Verbindung zum Kalender	Termin wird mit Details ausgegeben

Tabelle 3.1: Use Cases der prototypischen Umsetzung

3.3 Persönlichkeit und Charakter

Der Interaktion mit Sprach- und Konversationsschnittstellen sollte eine kohärente und konsistente Persönlichkeit zugrunde liegen [CGB04]. Die Form der Persönlichkeit wird dabei oft mit den Charakteren in Büchern oder Filmen verglichen. Anders ausgedrückt kann damit die Persönlichkeit betrachtet werden, welche Benutzer sich während der Interaktion mit der Applikation vorstellen.

Die Wahl eines Charakters anhand realer oder fiktiver Personen gestaltet sich jedoch schwierig, da die Implementierung Gefahr läuft, zu sehr auf den Charakter, als auf einen einfachen Dialogfluss und die Zielgruppe ausgerichtet zu werden [Pea16]. Die Gestaltung der Persönlichkeit sollte daher die Erstellung konsistenter Antworten und eine, auf die Zielgruppe angepasste Wortwahl umfassen. Dazu zählt, in welcher Form Fragen beantwortet werden, wie mit Fehlern umgegangen wird und wie umfangreich das System den Benutzer unterstützt.

Durch die Einschränkung der Altersgruppe kann der Wortschatz des Systems verkleinert und somit die Gestaltung der Antworten erleichtert werden [CGB04]. In der vorliegende Zielgruppe Senioren, welche in Deutschland der Altersgruppen 60+ zugeordnet wird, können etwa Bestandteile der Jugendsprache vernachlässigt werden.

Cohen et al. [CGB04] definieren neben der Zielgruppe zwei weitere Eigenschaften, welche bei der Wahl der Persönlichkeit beachtet werden müssen.

- Zielgruppe
- Häufigkeit der Systemnutzung
- Denkweise des Benutzers

Die Häufigkeit der Systemnutzung spielt überwiegend bei Erläuterungen der Funktionsweise eine Rolle. Einem regelmäßigen Systemnutzer müssen seltener Funktionsweise erklärt werden, als neuen Nutzern. Die Denkweise des Benutzers soll die Erwartungshaltung an das CUI decken. Erwartet der Benutzer einen seriösen Gesprächspartner zur Beratung bei Bankangelegenheiten, sind scherzhafte Äußerungen eher unangebracht. Im hier vorliegenden Fall umfasst dies die Erwartung nach einem sozialen Gesprächspartner, welcher auch ernste Themen adäquat handhaben kann (vgl. Abschnitt 3.2).

3.4 Technologiewahl

Zur Implementierung des Prototypen werden unterschiedliche Frameworks und Softwarebibliotheken eingesetzt. Diese werden für jede Komponente des CUI (vgl. Abschnitt 2.3) einzeln betrachtet und deren Eignung anhand der gegebenen Anforderungen verglichen. Die Anbindungsmöglichkeit anderer Systeme bildet ein weiteres Auswahlkriterium, um die Kompatibilität der Komponenten untereinander zu gewährleisten.

3.4.1 Natural Language Understanding

Das Vorgehen zur Evaluation der NLU-Komponente erfolgt in Anlehnung an Braun et al. [BHMML17]. Es wird zunächst ein Trainings- und Validierungskorpus mit allen Intents und entsprechenden Äußerungen im JSON-Format erstellt (vgl. Listing 3.1). Die Äußerungen werden manuell eingegeben und gegebenenfalls vorhandene Entitäten annotiert. Der Trainingskorpus wird genutzt, um die NLU-Komponente anhand der Äußerungen zu trainieren. Dieser wird, falls verfügbar, per Upload-Funktion oder manuell in das jeweilige System eingegeben.

```
{
   "text": "hallo",
   "intent": "greet",
   "entities": []
},
  {
   "text": "Welche Termine habe ich heute",
   "intent": "make_appointment",
   "entities": [
     {
       "start": 24,
       "end": 29,
       "value": "morgen",
       "entity": "relativedate"
     }
   ]
  }
```

Listing 3.1: Ausschnitt des Trainingskorpus

Der Validierungskorpus folgt strukturell dem Trainingskorpus, enthält jedoch Äußerungen, welche nicht Bestandteil des Trainingskorpus sind. Die Anzahl der Utterances beträgt 20 % der des Trainingskorpus pro Intent.

Verglichen werden vier der aktuell populärsten NLU-Systeme (vgl. Tabelle 3.2). Die drei erstgenannten Systeme sind Anwendungen großer IT-Unternehmen: *LUIS* (Microsoft), *Dialogflow* (Google), *Wit.ai* (Facebook). Die Konfiguration erfolgt über eine Weboberfläche. Die Kommunikation mit dem Dialogmanagement wird über eine Webschnittstelle (REST-API) realisiert. Die Antwort (HTTP-Response) enthält die Intents und Entities im JSON-Format. Da es sich um proprietäre Systeme handelt, bleiben die internen Abläufe weitestgehend intransparent.

NLU	**Deutsches Sprachpaket**	**Kosten**	**Open Source**	**Prebuilt Entities**
LUIS[1]	ja	0,0013 €/ Anfrage (10.000 kostenlose Anfragen pro Monat)	nein	ja
Dialogflow[2]	ja	kostenlos (Standard Edition) $0,002 / Anfrage (Enterprise Edition)	nein	ja
Wit.ai[3]	ja (Beta)	kostenlos	nein	ja
Rasa NLU[4]	ja	kostenlos	ja	nein

Tabelle 3.2: NLU-Systeme im Vergleich

Ausnahme bildet *Rasa NLU*, welches eine Open-Source-Software darstellt. Die Funktionsweise liegt offen (vgl. Abschnitt 2.3.1) und kann gegebenenfalls angepasst werden. Das Training und die Nutzung erfolgt zudem direkt auf dem verwendetem System, was eine Offline-Fähigkeit ermöglicht und einen Geschwindigkeitsvorteil gegenüber den anderen Systemen bieten kann.

Als Zielsprache wird in dieser Arbeit Deutsch gewählt, d. h. es wird ausschließlich mit Äußerungen und Antworten in deutscher Sprache gearbeitet. Die Intents, Entitäten und Äußerungen werden anhand der im vorherigen Abschnitt definierten Anforderungen entworfen.

Definition der Intents und Entitäten

Die Intents werden anhand der Use Cases aus Abschnitt 3.2 definiert. Die Use Cases *Termin finden*, *Termin erstellen* und *Nachrichten vorlesen* werden als Intents übernommen und mit entsprechenden Beispieläußerungen bestückt (vgl. Tab. 3.3). Daneben wird ein Intent *inform* erstellt, welcher genutzt wird, um fehlende Informationen, wie die Uhrzeit oder den Betreff, beim Benutzer einzuholen. Der Intent *introduce* wird genutzt, um den Namen des Benutzers zu erkennen. Für den Anwendungsfall *Erinnern* wird kein Intent definiert, da dieser zeitgesteuert und ohne Spracheingabe des Benutzers abläuft. Die domänenspezifischen Intents werden nachfolgend als *Core-Intents* bezeichnet.

Intent	Beispieläußerung
find_appointment	„Habe ich heute Termine?“
make_appointment	„Erstelle für morgen einen Termin.“
read_news	„Welche Nachrichten gibt es aus der Region?“
inform	„Morgen um 9 Uhr"
introduce	„Mein Name ist Ingrid."

Tabelle 3.3: Core-Intents mit Beispieläußerungen

Zu jedem *Core-Intent* werden entsprechende Entitäten bestimmt. Dieses geben weitere Informationen zur Spezifikation der Intents an und helfen bei der Verfolgung des Kontext. Dabei wird zwischen optionalen und verpflichtenden Entitäten unterschieden.

Intent	Entitäten
find_appointment	*subject*, *date*
make_appointment	**subject**, **date**
read_news	news, news_type, date
inform	date, time
introduce	firstname, lastname, gender

Tabelle 3.4: Entitäten

Die fettgedruckten Entitäten (vgl. Tabelle 3.4 Zeile *make_appointment*) stellen Pflichtangaben zur Ausführung des Intents dar. Im Falle von *make_appointment* bedeutet das, dass ein neuer Termin nur erstellt werden kann, wenn sowohl eine Datumsangabe, als auch ein Betreff (subject) angegeben sind. Sind diese nicht vorhanden, muss das System eine entsprechende Nachfrage stellen, was in die spätere Gestaltung des Dialogmanagements einfließt. Kursivgedruckte Entitäten stellen Pflichtangaben dar, von denen mindestens eine angegeben werden muss. Zum Finden eines Termins (*find_appointment*) muss entweder der Betreff oder das Datum angegeben werden. Das Datum kann dabei in einem gängigen Datumsformat (z. B. „TT.MM.JJJJ“) oder in informeller Form, wie *„morgen“*, angegeben werden. Benutzeräußerungen welche lediglich Informationen zu einem Intent beitragen, werden durch den Intent *inform* erkannt. Listing 3.2 zeigt einen möglichen Dialogablauf mit diesem Intent.

```
1 Benutzer: Erstelle einen Arzttermin.        make_appointment
2 Bot: Wann hast du den Arzttermin?           utter_ask_date
3 Benutzer: Morgen.                           inform
4 Bot: Um wie viel Uhr?                       utter_ask_time
5 Benutzer: Um 9 Uhr.                         inform
6 Bot: Ok, ich werde dich daran erinnern.     action_save_appointment
```

Listing 3.2: Nachfrage nach fehlenden Informationen

Neben den *Core-Intents* werden nicht-domänenspezifische Intents definiert. Dazu zählen Begrüßung (*greet*), Verabschiedung (*goodbye*), Zustimmung (*agree*) und Ablehnung (*decline*), welche im Folgenden als *Basic-Intents* bezeichnet werden (vgl. Tabelle 3.5).

Intent	Beispieläußerung
greet	„Hallo"
goodbye	„Auf Wiedersehen"
agree	„Ja, das stimmt."
decline	„Nein, das ist nicht korrekt."

Tabelle 3.5: Basic-Intents mit Beispieläußerungen

Erstellung des Text- und Validierungskorpus

Zur Erstellung der Korpora wird das Werkzeug *Rasa-NLU-Trainer*[5] genutzt. Dabei handelt es sich um ein JavaScript-Tool, welches zur manuellen Erstellung der Trainingsdaten für *Rasa NLU* entwickelt wurde und die Eingabe über eine Weboberfläche erleichtert (vgl.Abb. 3.2). Zu jedem Core-Intent werden 50 und zu jedem Basic-Intent 20 Beispieläußerungen erstellt.

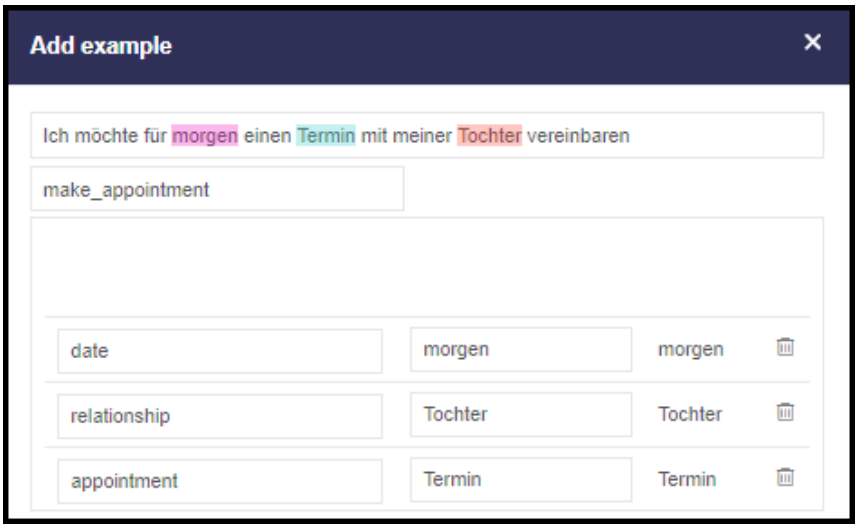

Abbildung 3.2: Rasa-NLU-Trainer zur Erstellung des Trainings- und Validierungskorpus

[5] https://github.com/RasaHQ/rasa-nlu-trainer

Die Speicherung des Korpus erfolgt bei *Rasa-NLU-Trainer* im JSON-Format (vgl. Listing 3.1) und ähnelt strukturell dem Format der anderer NLU-Systeme. *Rasa NLU* kann mit dem erstellten Korpus direkt trainiert werden. *LUIS* und *Dialogflow* bieten jeweils eine Uploadfunktion für Trainingsdaten an. Die Dateiformate sind bei beiden Plattformen JSON, folgen jedoch unterschiedlichen Strukturen.

Die Trainingsdatei von *LUIS* folgt einer ähnlichen Struktur wie *Rasa NLU* (vgl. Listing 3.3 und 3.4), weshalb ein Python-Skript zur Übersetzung der Strukturen entwickelt wird. Der übersetzte Korpus wird anschließend kontrolliert und per Upload-Funktion hochgeladen.

```
{
  "common_examples": [
   {
     "text": "guten tag",
     "intent": "greet",
     "entities": []
   },
   {
     "text": "Welche Nachrichten
         gibt es",
     "intent": "read_news",
     "entities": [
      {
        "start": 7,
        "end": 18,
        "value": "Nachrichten",
        "entity": "news"
      }
   ]
```

Listing 3.3: Trainingsdaten Rasa NLU

```
{
  "utterances": [
    {
      "text": "guten tag",
      "intent": "greet",
      "entities": []
    },
    {
      "text": "Welche Nachrichten
          gibt es",
      "intent": "read_news",
      "entities": [
      {
        "entity": "news",
        "startPos": 7,
        "endPos": 17
      }
    ]
```

Listing 3.4: Trainingsdaten LUIS

Dialogflow verfolgt eine Struktur mit verteilen Dateien, was eine Anpassung des vorhandenen Trainingskorpus erschwert. Die Trainingsdateien werden in separaten Unterverzeichnissen abgelegt und für jede Entität oder Absicht eine eigene Datei angelegt. Auf eine automatisches Skript zur Umwandlung wird bei Dialogflow, aufgrund des hohen Aufwands bei der Nachkontrolle, verzichtet und eine manuelle Eingabe durchgeführt. Ebenso werden bei *Wit.ai* die Trainingsdaten manuell über die Weboberfläche eingegeben, da zum Zeitpunkt dieser Evaluation keine Möglichkeit zum Upload existiert.

Bewertungskriterien

Es werden die drei nachfolgenden Kriterien zur Bewertung der Leistungsfähigkeit des NLU-Systems herangezogen:

- Accuracy-score (Korrektklassifizierungsrate)
- F1-score
- Entity-score: Erkennungsrate der Entitäten

Der *Accuracy-score* ist das Verhältnis von korrekt vorhergesagten Intents zur Gesamtanzahl Intents im Validierungskorpus [RM17]. Der *F1-score* bietet ein Genauigkeitsmaß, welches die Werte für Genauigkeit (precision) und Trefferquote (recall) berücksichtigt. Beide Maße ergeben sich aus den richtig-positiven (RP), falsch-positiven (FP) bzw. falsch-negativen (FN) Werten.

$$precision = \frac{RP}{RP + FP}$$

$$recall = \frac{RP}{RP + FN}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Wird ein Intent richtig erkannt, werden ebenfalls die korrekt erkannten Entitäten gespeichert und der Mittelwert gebildet, was im *Entity-score* resultiert. Zusätzlich wird die Zeit gemessen die ein Dienst benötigt, um Intent und Entities zurückzuliefern.

Ergebnisse

Eine Übersicht der Ergebnisse ist in Abbildung 3.3 dargestellt. Die schlechteste Genauigkeit erreicht *Wit.ai* mit lediglich 17 % Accuracy. Allerdings handelt es zum Zeitpunkt dieser Evaluation um eine Beta-Version für die deutsche Sprache. Somit kann zukünftig eine Steigerung erwartet werden.

Die drei anderen NLU-Systeme zeigen bei der Accuracy ähnliche Ergebnisse von über 80 %. *Dialogflow* sticht mit einer Accuracy von 96 % jedoch hervor. Die F1-scores liegen bei *LUIS* und *Rasa NLU* mit 76 % bzw. 70 % nahe zusammen. *Dialogflow* erreicht 86 % den besten Wert, *Wit.ai* mit knapp 34 % den schlechtesten.

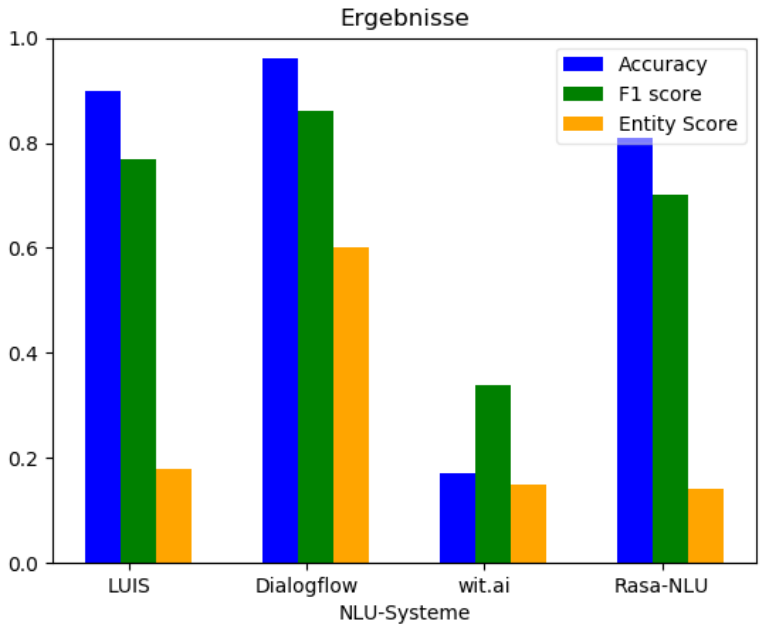

Abbildung 3.3: Ergebnisse im Vergleich

Der Entity-Score ist bei *Dialogflow* mit 62 % am größten. Die anderen Anwendungen bleiben alle unter 20 %.

Die durchschnittliche Antwortzeit ist bei *Rasa NLU* wie zu erwarten sehr kurz, da es lokal auf demselben System läuft. Im Gegensatz dazu müssen die anderen Systeme per HTTP-Request abgefragt werden. Von diesen beantwortet *Dialogflow* mit einer durchschnittlichen Dauer von 22 Millisekunden die Anfragen am schnellsten (vgl. Abb. 3.4).

Die Ergebnisse der Antwortzeiten dienen lediglich als Anhaltspunkte, da weitere Faktoren wie Internetanbindung und Auslastung des Remote-Systems nicht berücksichtigt werden. Die kurzen Antwortzeiten von *Rasa NLU* im Vergleich zu den anderen Systeme sind jedoch eindeutig.

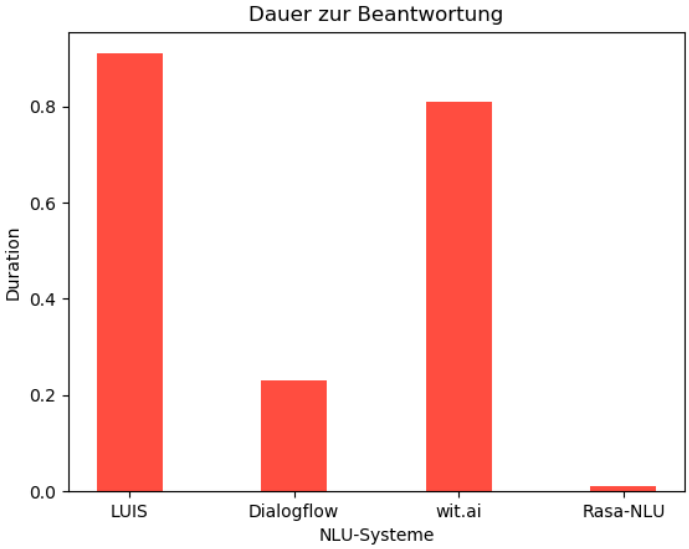

Abbildung 3.4: Durchschnittliche Dauer zur Beantwortung einer Anfrage

Die Ergebnisse zeigen, dass *LUIS*, *Dialogflow* und *Rasa NLU* für die Verwendung in Betracht kommen. Sie weisen in Accuracy- und F1-Score ähnlich gute Werte auf, wobei *Dialogflow* in allen Kategorien hervorsticht. *Rasa NLU* bietet darüber hinaus den Vorteil der Transparenz durch Open Source, der Offline-Fähigkeit und Geschwindigkeitsvorteile durch die Verwendung eines lokalen Modells.

3.4.2 Dialogmanagement

Das Dialogmanagement bildet die zentrale Komponente im späteren System. Sowohl die Kommunikation mit der NLU-Komponente, als auch mit der Sprach-Komponenten müssen gewährleistet sein. Es existieren diverse Frameworks und Programmbibliotheken zur Erstellung des Dialogmanagement, welche ähnliche Funktionsweisen nutzen. Tabelle 3.6 zeigt die quelloffenen Dialogmanagement-Frameworks *Microsoft Bot Builder SDK* und *Rasa Core*.

	Microsoft Bot Builder SDK	**Rasa Core**
Programmiersprache	Node.js, C#	Python
Messaging Plattformen	u. a. Skype, Slack, Facebook Messenger	u. a. Facebook Messenger, Slack, Telegram

Tabelle 3.6: Dialogmanagement-Frameworks

Microsoft Bot Builder SDK

Das *Microsoft Bot Builder SDK* ist ein Open-Source-Framework zur Erstellung von Chatbots in C# oder JavaScript. Es bietet Unterstützung beim Dialogmanagement, Verbinden zu gängigen Chat-Plattformen und bietet Templates zur Erstellung einfacher Chatbots an.

Das Grundkonzept[6] stützt sich auf die fünf Komponenten *Connector*, *Activity*, *Dialog*, *FormFlow* und *State*. Der *Connector* bietet eine REST-API, welche für die Kommunikation mit unterschiedlichen Ein- und Ausgabekanälen verwendet wird. Für gängige Chat-Plattformen wie Skype, Telegram oder Slack existieren vordefinierte Objekte. Für den Transport von Informationen werden dabei *Activity*-Objekte mit unterschiedlicher Typen genutzt. Der gängigste Typ ist die Textnachricht (message).

Dialog bildet eine Abstraktion zur Modellierung von Dialogen und übernimmt damit das Dialogmanagement. Im Falle von C# wird eine Implementierung des Interfaces *IDialog* durchgeführt, dessen Bestandteile u. a. Methoden zum Erhalt von Nachrichten und der Erstellung von Antworten sind. Als Einstiegspunkt dient immer die Klasse *Root-Dialog*, welche durch weitere Dialog-Klassen auf die jeweilige Domäne angepasst werden kann. Das DM arbeitet dabei regelbasiert und für die Intents werden entsprechende Methoden angelegt. Für das ebenfalls von Microsoft entwickelte NLU-System *LUIS* werden darüber hinaus eigene Annotationen angeboten (vgl. Listing 3.5).

```
1  [LuisIntent("welcome")]
2  public async Task GreetingIntent(IDialogContext context, LuisResult result)
3  {
4      await this.ShowLuisResult(context, result);
5  }
6
7  [LuisIntent("weather")]
8  public async Task CancelIntent(IDialogContext context, LuisResult result)
9  {
10     await this.ShowLuisResult(context, result);
11 }
```

Listing 3.5: Auszug des Dialogmanagement für einen Wetterbot programmiert mit Microsoft Bot Builder SDK für .NET

[6] https://docs.microsoft.com/en-us/azure/bot-service/dotnet/bot-builder-dotnet-concepts

Die Komponente *FormFlow* kann genutzt werden, um notwendige Informationen zur Erfüllung eines Intents zu sammeln. Beispielsweise kann ein *FormFlow* „ErstelleTermin“ mit den Eigenschaften „Betreff“ und „Uhrzeit“ erstellt werden, um diese als Pflichtangaben zu deklarieren.

State kann verwendet werden, um Statusinformationen zwischen den Komponenten auszutauschen, welche an einen Benutzer oder eine Konversation geknüpft sind. Damit ist es möglich auf ältere Konversationsverläufe oder Daten des Benutzer, etwa den Namen, zurückzugreifen.

Rasa Core

Rasa Core ist ein statistisches datengetriebenes Dialogmanagement welches sich ebenfalls aus mehreren Komponenten zusammensetzt (vgl. Abb. 3.5).

Die Klasse *Interpreter* bildet die Abstraktion des verwendeten NLU-Systems. Hierfür kann das hauseigene NLU-System *Rasa NLU* mittels der mitgelieferten Klasse genutzt werden. Es ist jedoch möglich jedes andere NLU-System anzubinden, was die Implementierung einer eigenen Klasse zur Transformation der Antworten in das entsprechende Zielformat erfordert.

Der *Tracker* empfängt die neuen Nachrichten und speichert den Zustand der Konversation und gegebenenfalls Intents und Entitäten.

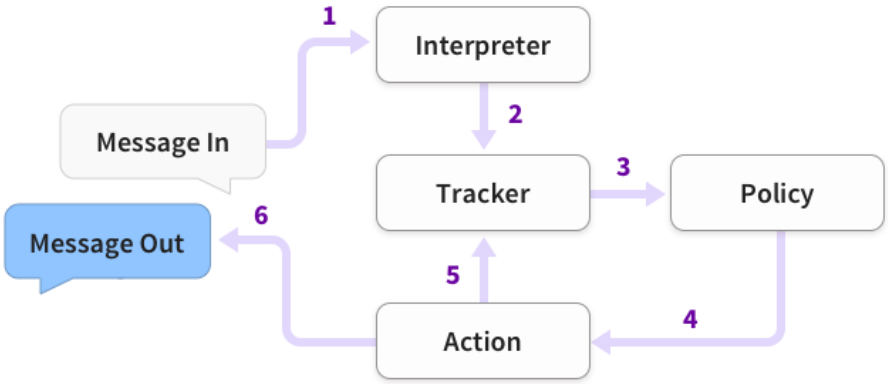

Abbildung 3.5: Rasa High-Level-Architektur[7]

[7] http://rasa.com/docs/core/architecture/

Policy erhält den Zustand vom *Tracker* und entscheidet über die auszuführenden Aktionen. Grundlage bildet ein Machine-Learning-Modell, welches domänenspezifisch trainiert wird. In der hier verwendeten Version 0.9 wird dafür ein Rekurrentes neuronales Netz mit Long-short-term-memory (LSTM) genutzt. Zur Implementierung nutzt *Rasa Core* die High-Level-API *Keras* mit der Machine-Learning-Bibliothek *Tensorflow* als Backend.

Die Trainingsdaten für das neuronale Netz werden in Form von *Stories* im Markdown-Format angelegt (vgl. Listing 3.6). Sie beschreiben mögliche Konversationsabläufe in Form von Intents (* *greet*), Entitäten (*"relativedate": "morgen", "subject": "arzttermin"*) und Aktionen (- *utter_greet*). Die Trainingsdaten sollten gängige Konversationsabläufe abdecken, um eine ausreichende Genauigkeit zu erreichen. Mit steigender Anzahl an Trainingsdaten, kann ebenfalls eine Steigerung der Genauigkeit erwartet werden. Es ist jedoch nicht notwendig alle möglichen Konversationsverläufe abzubilden.

Mittels interaktivem Lernen kann das Dialogmanagement getestet und optimiert werden. Der Konversationsverlauf wird dazu direkt im Story-Format abgespeichert.

```
* greet
  - utter_greet
* make_appointment{"relativedate": "morgen", "subject": "arzttermin"}
    - slot{"subject": "arzttermin"}
    - slot{"relativedate": "morgen"}
    - utter_ask_time
* inform{"time": "9 uhr"}
    - slot{"time": "9 uhr"}
    - action_make_appointment
```

Listing 3.6: Story-Beispiel für Rasa Core

Anbindung anderer Komponenten

Die Anbindung externer NLU-Systeme geschieht bei beiden Frameworks via HTTP-Clients. Das Dialogmanagement sendet die Äußerung des Benutzers via HTTP-GET oder HTTP-POST an das NLU-System und erhält daraufhin die Antwort mit den entsprechenden Intents, Entitäten und zugehörigen Wahrscheinlichkeiten (vgl. Listing 3.7).

```
{
  "query": "Habe ich heute Termine",
  "topScoringIntent": {
     "intent": "find_appointment",
     "score": 0.9970879
  },
  "intents": [
     {
        "intent": "find_appointment",
        "score": 0.9970879
     },
  ],
  "entities": [
     {
        "entity": "heute",
        "type": "relativedate",
        "startIndex": 9,
        "endIndex": 13,
        "score": 0.991789639
     }]
```

Listing 3.7: Antowort des NLU-Systems LUIS

Die Intents und Entitäten müssen mittels eigener Implementierung aus den Antworten extrahiert und der entsprechenden Klasse im DM übergeben werden. Für die eigenen NLU-Systeme *LUIS* bei *Microsoft Bot Builder SDK* bzw. *Rasa NLU* bei *Rasa Core* werden entsprechende Klassen angeboten, womit eine eigene Implementierung entfällt.

Eine Anbindung an gängige Chat-Plattformen bieten beide Systeme in ähnlichem Umfang an, weshalb dieses Kriterium nicht zur Auswahl in Betracht gezogen wird.

Beide Plattformen folgen ähnlichen Grundkonzepten, arbeiten jedoch bei der Entscheidung über Aktionen und Antworten unterschiedlich. Aufgrund der in Abschnitt 2.3.2 genannten Nachteile eines regelbasierten Ansatzes, wird *Rasa Core* für das Dialogmanagements gewählt.

Darüber hinaus verfügt *Rasa Core* über ein aktive Community[8] und ausführliche Dokumentationen [9] [BFPN17], welche die internen Abläufe und Funktionsweisen näher erläutern und mögliche Anpassungen erleichtern.

3.4.3 Spracheingabe - STT

Aktuell existieren diverse Systeme und Softwarebibliotheken zur Erkennung menschlicher Sprache und Übersetzung in Text (Speech to Text, STT). Die Angebote reichen von Open-Source-Projekten, freien Diensten bis hin zu kostenpflichtigen Diensten mit Abrechnungen pro Anfrage oder Monat. Tabelle 3.7 zeigt einige Dienste und Softwarebibliotheken.

Anwendungsname	**Typ**	**Deutsche Sprachunterstützung**	**Kosten**
Bing Sprach API	Proprietar	ja	€3,374 pro 1.000 Transaktionen / 5000 freie Transaktionen pro Monat
CMU Sphinx	Open Source	ja	-
Google Speech Recognition	Proprietar	ja	$ 0,006 pro 15 Sek.
Watson Speech to Text	Proprietar	nein	$ 0.0125 pro Minute

Tabelle 3.7: Übersicht zu Spracherkennungssoftware

Zu beachten ist, dass nicht alle Dienste Unterstützung für die deutsche Sprache bieten. Das Open-Source-Projekt *CMU Sphinx* bietet das Hinzufügen eigener Sprachmodelle an. Die Erstellung erfordert jedoch eine große Menge an Sprachdaten, weshalb auf existierende Modelle aus der Entwicklergemeinschaft zurückgegriffen wird.

Evaluation

Als Bewertungskriterium dient, neben der Unterstützung für die deutsche Sprache, die Genauigkeit bei der Erkennung der Sprache. Zur Evaluation der Genauigkeit werden 20 Beispielsätze aus dem Trainingskorpus (vgl. Abschnitt 3.4.1) durch einen Benutzer via Mikrofon eingegeben. Zur Verringerung der Einflüsse von Umgebungsgeräuschen und der individuellen Aussprache und Sprachgeschwindigkeit wird jeder Satz bis zu drei mal vorgelesen. Findet nach dem ersten Versuch bereits eine korrekte Übersetzung statt, wird kein weiterer Versuch unternommen und der nächste Beispielsatz vorgelesen.

[8] https://forum.rasa.com/
[9] http://www.rasa.com/docs/getting-started/overview/

Die Genauigkeit berechnet sich aus der Summe richtig übersetzter Texte geteilt durch die Summe aller Beispielsätze.

Um eine weitere Toleranz durch äußere Einflüsse zu gewährleisten wird die Überprüfung der Richtigkeit eines Satzes mit Hilfe der Levenshtein-Distanz (Editierdistanz) berechnet. Die Distanz spiegelt dabei die Anzahl der Editier-Operationen wider, welche benötigt werden, um eine Zeichenkette in eine andere zu wandeln [IMF13]. Die Editier-Operationen können je nach Implementierung variieren, sind typischerweise und auch in dieser Arbeit: Einfügen, Löschen und Ersetzen einzelner Zeichen. Zwei identische Terme resultieren somit in der Editierdistanz 0.

Damit ein Satz als richtig übersetzt gilt, darf nach dem dritten Versuch die Editierdistanz maximal zwei betragen. Verglichen werden die Anwendungen *Bing Sprach API*, *CMU Sphinx* und *Google Speech Recognition*.

Ergebnisse

Während die beiden proprietären Dienste *Google Speech Recognition* und *Bing Sprach API* auf Anhieb alle Äußerungen richtig erkannten, konnte *CMU Sphinx* auch nach drei Versuchen viele Äußerungen nicht in den korrekten Text umwandeln. Die endgültige Erkennungsrate liegt bei lediglich 42 % (vgl. Abb. 3.6). Zur Verwendung in einem produktiven System können somit aktuell nur die proprietären Systeme in Erwägung gezogen werden.

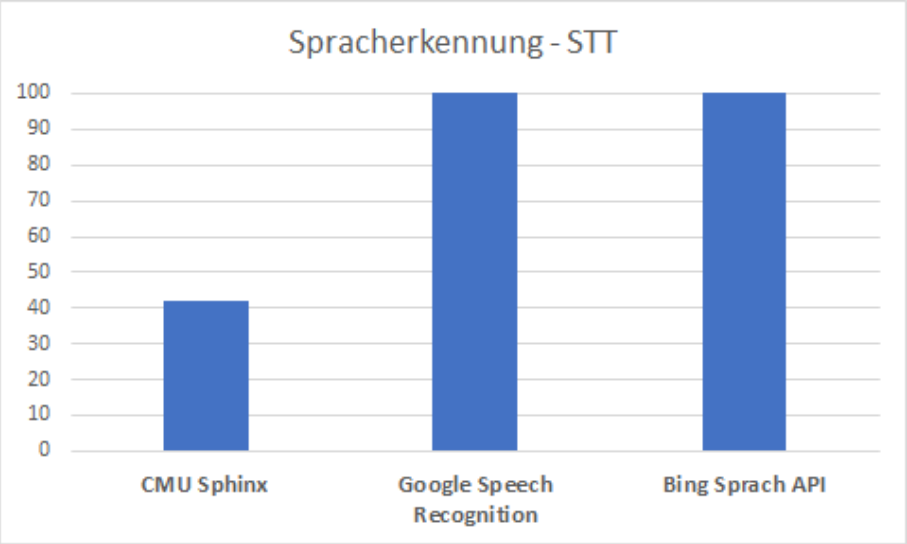

Abbildung 3.6: Ergebnisse der STT-Evaluation - Erkennungsraten in Prozent

3.4.4 Sprachausgabe - TTS

Aktuelle Sprachausgaben nutzen hybride Ansätze aus Sprachsynthese und aufgenommener menschlicher Sprache [Pea16]. TTS-Systeme werden teils in Kombination mit der Spracherkennung angeboten, weshalb einige der in Tabelle 3.8 gezeigten Anwendungen sich mit denen des vorherigen Abschnitts überschneiden.

Anwendungsname	**Typ**	**Deutsche Sprachunterstützung**	**Offline-Fähigkeit**	**Kosten**
Bing Sprach API	Proprietar	ja	nein	3,374 € pro 1000 Transaktionen / 5000 freie Transaktionen pro Monat
Google Cloud Text to Speech	Proprietar	ja	nein	$ 4,00 - $ 16,00 pro 1 Million Zeichen (je nach Stimme)
Microsoft Speech Engine (SAPI5)	Proprietar	ja	ja	In Windows integriert
Watson Text to Speech	Proprietar	nein	nein	$ 0,02 pro 1000 Zeichen / 10.000 Zeichen pro Monat frei

Tabelle 3.8: Übersicht zu Sprachausgabesoftware

Die objektive Bewertung der Sprachausgabe durch Einzelpersonen gestaltet sich schwierig und wird daher in Form einer Online-Umfrage mit entsprechenden Hörbeispielen durchgeführt.

Evaluation

In Anlehnung an [CSGF15] werden vier Bewertungskriterien mit eigener Interpretation bestimmt. Die Teilnehmer der Umfrage geben zu jedem Kriterium eine Bewertung im Schulnoten-Prinzip, von 1 sehr gut bis 6 ungenügend, ab. In Tabelle 3.9 sind die Bewertungskriterien zusammen mit Beschreibungen für die beiden Extremwerten dargestellt.

Kriterium	Note 1 (sehr gut)	Note 6 (ungenügend)
Natürlichkeit	Die Aussprache klingt sehr menschenähnlich.	Die Aussprache klingt sehr künstlich.
Verständlichkeit	Ich konnte alle Wörter im Satz verstehen.	Ich konnte keines der Wörter im Satz verstehen.
Genauigkeit	Ich konnte die Art jeden Satzes (Frage, Aussage) durch die Aussprache erkennen.	Ich konnte die Art keines Satzes durch die Aussprache erkennen.
Nachvollziehbarkeit	Ich konnte das Ziel (Was will der Sprecher) jeden Satzes verstehen.	Ich konnte das Ziel keines Satzes verstehen.

Tabelle 3.9: Bewertungskriterien zur Evaluation der Sprachausgabe

In der Online-Umfrage werden die beiden deutschen Stimmen von *Google Cloud Text to Speech* und *Microsoft Speech Engine* mit jeweils fünf Hörbeispielen evaluiert. *Bing Sprach API* bietet dieselbe Stimme wie *Microsoft Speech Engine* an und wird deshalb nicht separat evaluiert. *Watson Text to Speech* entfällt aufgrund der fehlenden Unterstützung für die deutsche Sprache.

Ergebnisse

An der Online-Umfrage haben 14 Personen im Alter zwischen 18 und 64 Jahren teilgenommen. Das Durchschnittsalter liegt 32 Jahren. Aufgrund der geringen Anzahl an Teilnehmern und des geringen Durchschnittsalters sind die Ergebnisse nicht repräsentativ für die Implementierung eines TTS in der Altenpflege. Jedoch wird das Ergebnis als Entscheidungshilfe zur Implementierung des Prototypen herangezogen.

Die Ergebnisse zeigen, dass *Google Cloud Text to Speech* mit einer Durchschnittsnote von 3,3 als natürlicher empfunden wurde gegenüber 4,2 bei *Microsoft Speech Engine* (vgl. Abb. 3.7). Beide Resultate sind gegenüber den anderen Kriterien deutlich schlechter, was zeigt, dass beide Stimmen als relativ künstlich wahrgenommen werden. In der Verständlichkeit, Genauigkeit und Nachvollziehbarkeit erzielten beide Systeme gute bis sehr gute Ergebnisse, wobei *Google Cloud Text to Speech* in jeder Kategorie geringfügig besser abschneidet.

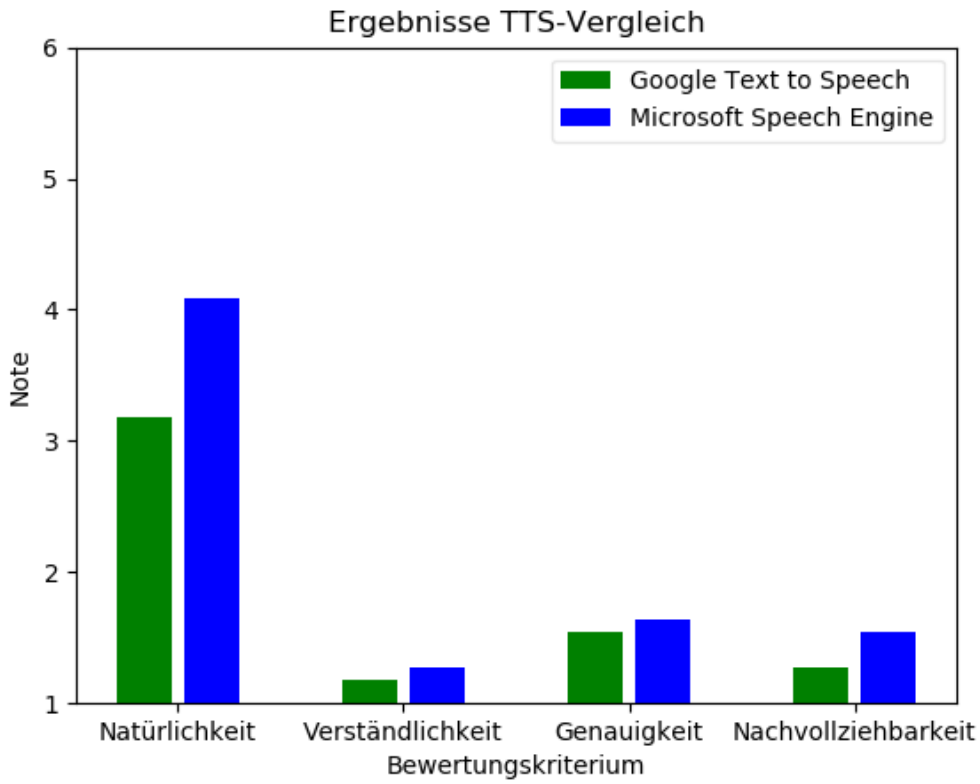

Abbildung 3.7: Ergebnisse der TTS-Evaluation im Schulnoten-Prinzip

4 Prototypische Umsetzung

In diesem Kapitel wird die Implementierung des CUI anhand eines Prototypen beschrieben. Dazu werden die vorher evaluierten Softwarebibliotheken und Anwendungen genutzt. Das System wird modular aufgebaut, um Erweiterungen in den nachfolgenden Versionen zu ermöglichen.

Der schematische Aufbau des Prototypen ist in Abbildung 4.1 dargestellt. Zentrale Komponente bildet das Dialogmanagement auf Basis von *Rasa Core*. Die Benutzereingabe erfolgt in dieser Version ausschließlich via Textnachricht. Zur Verbesserung der Benutzerfreundlichkeit wird die Chat-Plattform *Telegram*[1] als Frontend genutzt. Die Ausgabe findet auf zwei Wege, als Textnachricht in *Telegram* und als Sprachausgabe wahlweise über *Microsoft Speech Engine* oder *Google Text to Speech* statt. Um eine Offline-Fähigkeit zu gewährleisten, wird trotz der schlechteren Ergebnisse in der Evaluation, auch *Microsoft Speech Engine* angeboten. Aus demselben Grund wird neben *Dialogflow* auch *Rasa NLU* als NLU-Komponente eingesetzt. Der Prototyp überprüft beim Start die Internetverbindung und wählt gegebenenfalls die offlinefähigen Komponenten aus. Eine Spracheingabe wird in dieser Version noch nicht realisiert.

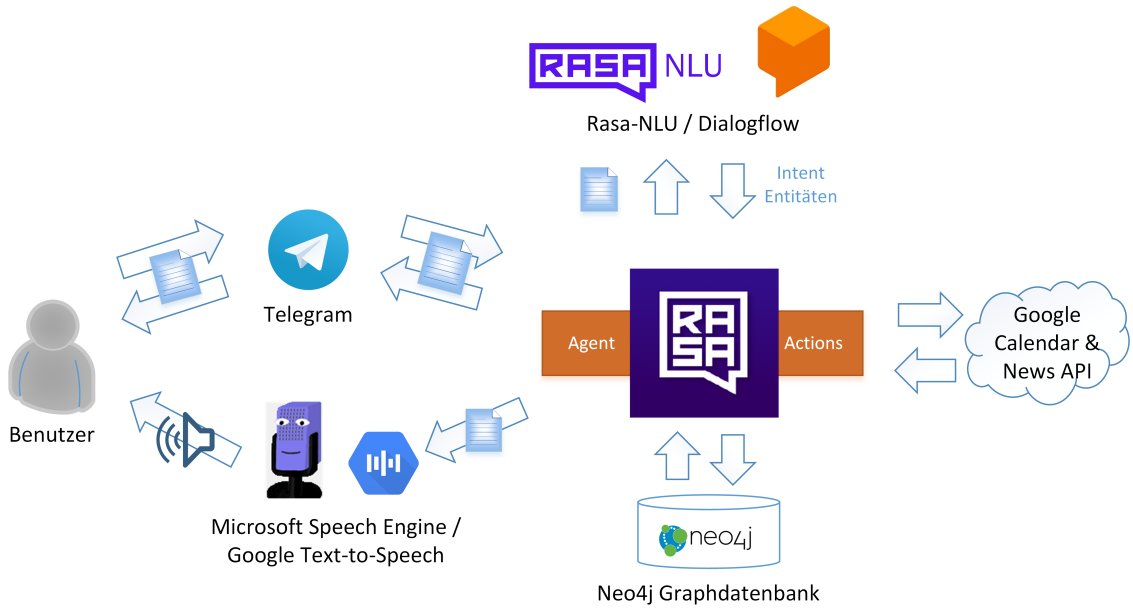

Abbildung 4.1: Schematischer Aufbau und verwendete Technologien des Prototypen

[1] https://telegram.org/

4.1 Implementierung der Komponenten

Gemäß der im vorherigen Kapitel evaluierten Technologien, werden nachfolgend die Komponenten implementiert.

4.1.1 NLU-Komponente

Die NLU-Komponente wurde bereits während der Evaluation trainiert und kann sofort verwendet werden. Aufgrund der Ergebnisse kommt Dialogflow als primäres NLU-System zum Einsatz. Jede Anfrage an Dialogflow wird mittels HTTP-GET-Aufruf an die dafür verfügbare API gesendet (vgl. 4.1). Die Äußerung des Benutzers wird URL-kodiert als Wert des Parameters *query* übergeben.

```
https://api.dialogflow.com/...?query=habe%20ich%20heute%20termine&...&lang=de
```

Listing 4.1: Aufruf von Dialogflow

Die Antwort im JSON-Format wird in einer eigene Klasse geparst und die Elemente für Intent (*intentName*), Entitäten (*parameters*) und Genauigkeit (*score*) extrahiert. Diese werden als Attribute der Interpreter-Klasse für die Verwendung in *Rasa Core* gespeichert.

```
{
   "lang": "de",
   "result": {
      "resolvedQuery": "habe ich heute termine",
      "parameters": {
         "dateperiod": "",
         "date": "",
         "relativedate": "heute",
         "appointment": "termine",
         "subject": ""
      },
      "metadata": {
         "intentId": "fe85493b-adaa-4219-a463-e78477d78c83",
         "intentName": "find_appointment"
      }
      "score": 1
```

4.1.2 Dialogmanagement

Zur Entwicklung des Dialogmanagements mit *Rasa Core* muss zunächst die Domäne in Form einer YAML-Datei (*domain.yml* - vgl. Listing 4.2) festgelegt werden.

```
slots:
   subject:
     type: text
   ...
intents:
   - goodbye
   - ...
entities:
   - subject
   - ...
actions:
   - actions_core.ActionAskLocation
```

Listing 4.2: Auszug der Datei domain.yml

Die Domäne legt fest welche Intents, Entitäten, Aktionen und Slots das System kennt und verwenden kann. Slots definieren Entitäten, welche während des Konversationsverlauf gespeichert werden und somit zu späteren Zeitpunkten des Dialogs zur Verfügung stehen.

Aktionen

Die Aktionen werden in separaten Python-Modulen *actions_basic.py* und *actions_core.py* gemäß der Intents implementiert. Für jede Aktion wird eine eigene Klasse angelegt, welche Kindklasse der Rasa internen Klasse *Action* ist. Die beiden Methoden *name* und *run* werden geerbt und mit der entsprechenden Implementierung überschrieben (vgl. Listing 4.3).

```
class ActionSearchAppointment(Action):
   def name(self):
      return 'action_search_appointment'

   def run(self, dispatcher, tracker, domain):
      ...
```

Listing 4.3: Auszug der Datei actions_core.py

Die Methode *name* gibt den Namen der Aktion in einem Format zurück, welches später innerhalb der Stories Verwendung findet.

Die Methode *run* erwartet, neben der Objektinstanz *self*, die Parameter *dispatcher*, *tracker* und *domain*. All diese sind Objekte der Rasa-Core-Bibliothek und besitzen folgende Eigenschaften:

- Tracker: Erlaubt den Zugriff auf Slots und die letzte Nachricht des Benutzers.
- Dispatcher: Wird genutzt, um Nachrichten an den Benutzer zurückzuschicken.
- Domain: Die Domäne des Bots gemäß der Definition in der *domain.yml*.

Stories

Um generelle Konversationsverläufe in den Stories abzudecken, werden vorab Flussdiagramme zu jedem Intent erstellt (vgl. Beispiel in Anhang A). Innerhalb des Diagramms weden Fallunterscheidungen erstellt, um zwischen vorhandenen bzw. fehlenden Entitäten zu unterscheiden. Für jede Fallunterscheidung wird anschließend eine eigene Story angelegt. Der Intent zur Erstellung eines Termins aus Anhang A besitzt somit initial acht Stories (vgl. Tabelle 4.1). *Story 1* ist in Listing 4.4 dargestellt.

Story-Nr.	1	2	3	4	5	6	7	8
Subject	-	x	-	-	x	x	-	x
Time	-	-	x	-	x	-	x	x
Date	-	-	-	x	-	x	x	x

Tabelle 4.1: Kombinationen der Entitäten anhand des Konversations-Flussdiagramms

```
## Story 1
* greet
   - utter_greet
* make_appointment{}
   - utter_ask_subject
* inform{"subject": "arzt"}
   - slot{"subject": "arzt"}
   - utter_ask_date
* inform{"relativedate": "morgen"}
   - slot{"relativedate": "morgen"}
   - utter_ask_time
* inform{"time": "9 Uhr"}
   - slot{"time": "9:00"}
   - action_make_appointment
* goodbye
   - utter_goodbye
```

Listing 4.4: Story zur Erstellung eines Termins

Zu beachten ist, dass lediglich erfolgreiche Konversationsabläufe (Happy Path) abgebildet werden. Abweichende Abläufe hervorgerufen durch falsch interpretierte Intents, fehlerhafte Benutzereingaben oder einen Kontextwechsel des Benutzers sind in diesem Stadium des Dialogentwurfs nicht abgedeckt.

Fehlerbehandlung

Als Fehler werden hier Vorgänge bezeichnet, welche zu ungeplanten Schritten im Konversationsverlauf führen oder die Entscheidungsfindung des Systems behindern. Nicht alle Fehler können während einer Konversation behandelt werden und erfordern gegebenenfalls eine Anpassung des Systems (vgl. Tabelle 4.2).

Fehler	**Behebbar während der Konversation**	**Maßnahme**	**Konsequenz**
Intent wird nicht erkannt	ja	Fallback-Antwort	Benutzer muss die Frage erneut eingeben.
Intent wird falsch erkannt	nein	Optimierung des NLU-Systems	Benutzer erhält falsche Antwort. / Falsche Aktion wird ausgeführt.
System wählt falsche Aktion aus	nein	Optimierung der Stories	Benutzer erhält falsche Antwort. / Falsche Aktion wird ausgeführt.

Tabelle 4.2: Mögliche Fehler während des Konversationsverlauf

Für die Erkennung eines Intents wird ein Schwellwert von 50 % festgelegt, d. h. die Wahrscheinlichkeit mindestens eines Intents in der Antwort des NLU-Systems muss 50 % überschreiten. Ist dies nicht der Fall, wird die Rückweisungsantwort *„Ich habe Sie leider nicht verstanden“* ausgegeben.

Wird ein falscher Intent erkannt, kann keine Behandlung während der Konversation stattfinden und der Benutzer erhält eine falsche Antwort. Dies kann nur durch eine Anpassung oder Erweiterung der Beispieläußerungen im NLU-System erfolgen. Ebenfalls kann keine direkte Fehlerbehandlung durchgeführt werden, wenn das DM eine falsche Aktion ausführt. Dies Bedarf einer Erweiterung bzw. Optimierung der Stories.

Interaktives Lernen

Nach Fertigstellung der initialen Stories anhand der Flussdiagramme wird das neuronale Netz von *Rasa Core* trainiert und das Modell gespeichert. Anschließend wird der Dialogfluss mittels interaktivem Lernen getestet und optimiert. *Rasa Core* bietet dazu eine eigene Methode (vgl. Abb. 4.2) an und speichert die interaktiv geführte Konversation im entsprechenden Markdown-Format. Die so entstanden Stories können anschließend zum erneuten Training des neuronalen Netzes verwendet werden.

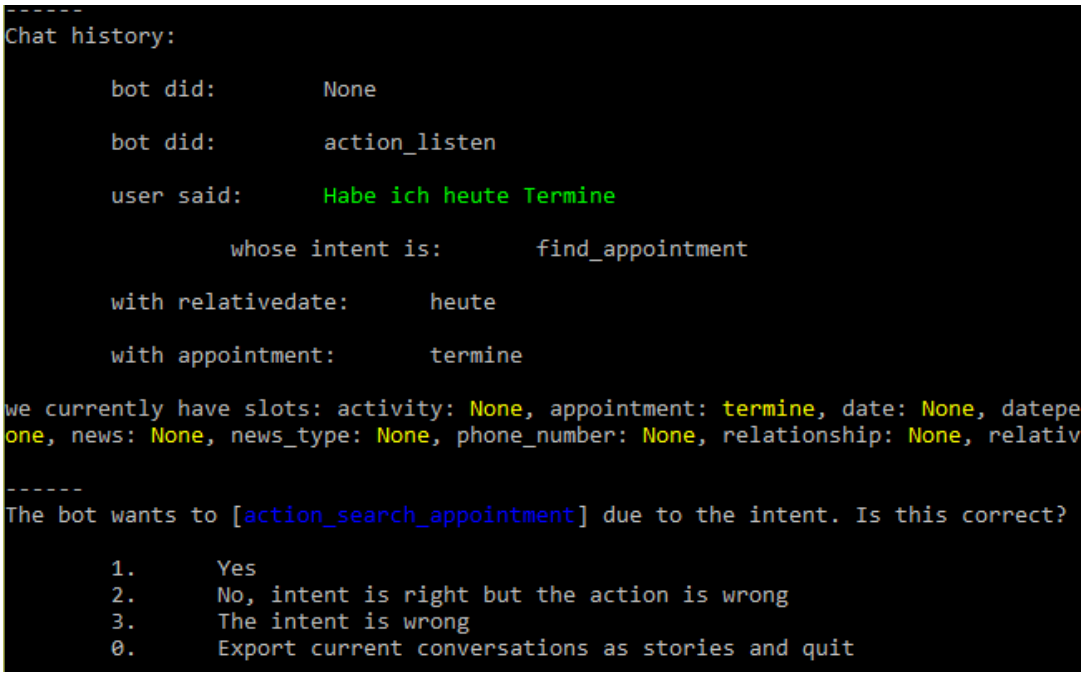

Abbildung 4.2: Interaktiven Lernen mit Rasa Core

Anbindung des Kalenders

Zur prototypischen Umsetzung der Kalenderfunktionen wird eine Anbindung an einen Google-Kalendar via *Google Calendar API*[2] implementiert. Zur Erkennung von Zeitangaben unterschiedlicher Formate werden drei Entitäten definiert. *Relativedate* beschreibt Angaben, welche im Verhältnis zum aktuellen Datum stehen, etwa „morgen“ oder „übermorgen“. *Dateperiod* beschreibt Zeiträume, wie „am Wochenende“ oder „die nächsten Tage“. Die Entität *Date* nimmt Daten in gängigen Datumsformaten, wie „01.01.“ oder „01.01.2018“, auf.

Zur Vereinheitlichung der unterschiedlichen Angaben wird das Format mittels regulärem Ausdruck erkannt und in ein einheitliches ISO-8601-Format der Form „YYYY-MM-DDThh:mm:ss“ transformiert. Die Uhrzeit wird in diesem Fall auf 00.00 Uhr gesetzt. Das „T“ dient als Trennzeichen zwischen Datum und Uhrzeit.

Wird neben dem Datum auch die Uhrzeit angegeben, wird diese ebenfalls per regulärem Ausdruck erkannt und Stunden und Minuten extrahiert. Sekunden werden generell auf Null gesetzt. Gibt der Benutzer nur die Uhrzeit an, wird davon ausgegangen, dass der Termin am heutigen Tag stattfindet. Tabelle 4.3 zeigt einige Beispiele der Transformation.

[2] https://developers.google.com/calendar/

	Benutzereingabe	ISO 8601 Format
Aktuelles Datum 14.09.2018	*14 Uhr*	2018-09-14T14:00:00
	morgen	2018-09-15T00:00:00
	am 20.10.	2018-10-20T00:00:00
	am 23.11.18	2018-11-23T00:00:00
	übermorgen um 9.00 Uhr	2018-09-16T09:00:00

Tabelle 4.3: Beispiele zur Transformation der Datums- und Zeitangaben nach ISO 8601

Wird weder Datum noch Uhrzeit, jedoch ein Betreff angegeben, werden Termine anhand des Betreffs gesucht. Um geringfügige Tippfehler und unterschiedliche Schreibweisen zu erlauben, wird der Betreff mittels Fuzzy-String-Matching (auch Approximate-String-Matching) gesucht. Die Zeichenkette wird mittels, der aus Kapitel 3.4.3 bekannten Levenshtein-Distanz, gesucht. Anders als in Kapitel 3.4.3 wird nicht nur die Levenshtein-Distanz, sondern das Verhältnis p in Bezug auf die Länge der Zeichenkette genutzt. Dieses berechnet sich aus der Länge des längeren Terms m und der Levenshtein-Distanz l: $p = (1 - l/m) * 100$. Das Verhältnis muss mindestens 85 % betragen, um in einer erfolgreichen Suche zu resultieren.

Anbindung der News-API

Zur Suche der Nachrichten wird die *Google News API* genutzt. Als prototypische Umsetzung werden die fünf neuesten Schlagzeilen mit entsprechenden Linkverweisen zum vollständigen Artikel ausgegeben. Wird kein Thema, wie Politik oder Sport angegeben, werden die fünf neuesten Schlagzeilen gesucht. Bei Angabe eines Themas, etwa „Welche Sportnachrichten gibt es“, werden die fünf neuesten Nachrichten zu dieser Thematik gesucht und ausgegeben.

4.1.3 Chat-Plattform

Der Prototyp wird zur benutzerfreundlichen Verwendung mit der Chat-Plattform *Telegram* verknüpft. In Telegram wird dafür ein eigenes Chatbot-Profil angelegt. Der Chatbot erhält beim Start des CUI-Systems eine Webhook-URL, an welche alle Anfragen weitergeleitet werden (vgl. Listing 4.5). Ebenso wird im CUI-System eine API-Schlüssel hinterlegt, welcher beim Senden der Antworten vom Prototypen an Telegram genutzt wird.

```
1 https://api.telegram.org/bot'+ telegram_api_key +'/setWebhook?url='+ webhook
```

Listing 4.5: URL zum Setzen des Webhooks in Telegram

Der Benutzer kann den Chatbot anschließend äquivalent zu einem Chatkanal mit einer realen Person verwenden. Bei Start des Kanals wird eine Begrüßungsnachricht und einer grundlegenden Beschreibung der Funktionen ausgegeben (vgl. Abb. 4.3). Das Chatbot-Profil erhält den Benutzernamen *Car(e)ina* in Anlehnung an die Funktionsweisen zur Pflege.

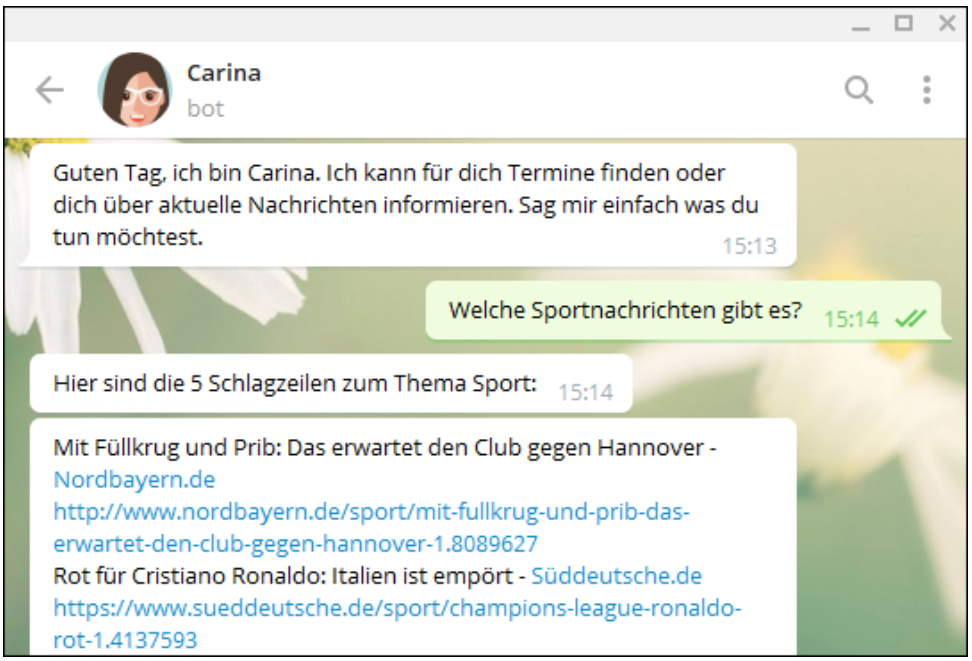

Abbildung 4.3: Integration des Systems in die Chat-Plattform Telegram

4.1.4 Sprachausgabe

Neben der textuellen Ausgabe über die Chat-Plattform gibt das System die Antworten in Form von synthetischer Sprache aus. Über eine Konfigurationsdatei kann zwischen *Google Text to Speech* und *Microsoft Speech Engine* gewählt werden. Ist keine Internetverbindung vorhanden, wird automatisch *Microsoft Speech Engine* verwendet, da diese durch die Integration in Windows eine Offline-Fähigkeit bietet. Die Sprachausgaben sind weitestgehend identisch zu den textuellen Ausgaben. Lediglich bei Ausgabe der Nachrichten wird zwischen Sprach- und Textausgabe unterschieden, um die URL zum vollständigen Artikel (vgl. Abb. 4.3) nicht sprachlich auszugeben, was den Benutzer irritieren könnte.

4.1.5 Anbindung Neo4j-Datenbank

Die Anbindung der Neo4j-Graph-Datenbank stellt eine erste Umsetzung zur Abbildung sozialer Beziehungen anhand der Konversation dar. In der aktuellen Version des Systems wird der Name des Benutzers erkannt und als Knoten in der Datenbank gespeichert (vgl. Abb. 4.5). Einige Antworten werden daraufhin personalisiert und sollen eine persönliche Kommunikation erlauben. Ist der Name bereits in der Datenbank hinterlegt, wird beim Benutzer nachgefragt, ob es sich um seine Instanz handelt (vgl. 4.4).

Abbildung 4.4: Personalisierung der Antworten

In zukünftigen Version sollen darüber hinaus die sozialen Beziehungen des Benutzers erkannt und abgebildet werden.

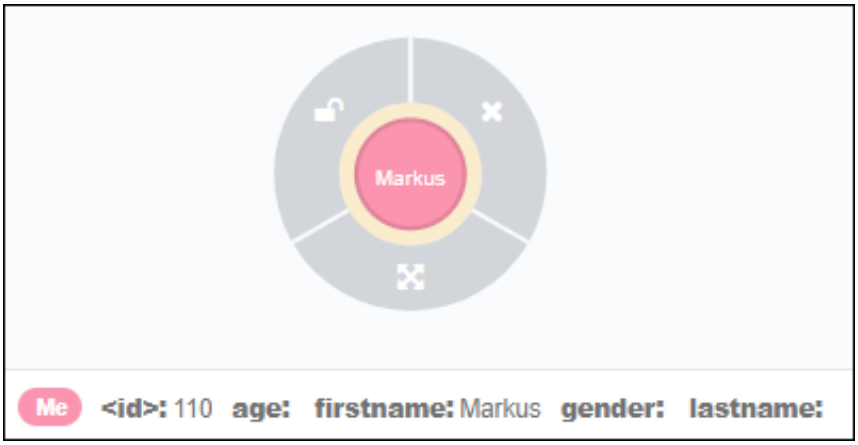

Abbildung 4.5: Knoten des Benutzers in der Neo4j-Graph-Datenbank

5 Schlussbetrachtung

5.1 Zusammenfassung

Diese Arbeit gibt eine Einführung in aktuelle Techniken zur Umsetzung von Conversationl Interfaces. Neben den Grundlagen zum Aufbau konversationsorientierter Anwendungen, wurden aktuelle Technologien verglichen und hinsichtlich ihrer Verwendung in der Altenpflege evaluiert. Die Implementierung eines Prototypen zeigt eine erste Möglichkeit zur Umsetzung domänenspezifischer Anforderungen mit Hilfe aktueller Softwarebibliotheken und Frameworks. Der modulare Aufbau erlaubt eine Kombination unterschiedlicher Komponenten und erleichtert die zukünftige Erweiterung.

Um die Funktionsweise heutiger CUI-Systeme nachvollziehen zu können ist es notwendig, den grundlegenden Aufbau eines menschlichen Konversationsverlaufs zu verstehen. Dessen formale Definition gibt Aufschluss über interne Abläufe und Bestandteile aktueller CUI-Anwendungen. Die Komponenten gliedern sich dabei in Sprachein- und Sprachausgabe, Natural Language Understanding und Dialogmanagement.

Die Evaluation verschiedener Softwarebibliotheken und Frameworks zur Implementierung der Komponenten zeigte, dass vor allem die Systeme namhafter IT-Unternehmen eine große Präsenz besitzen und bereits über ausreichend Reife und Robustheit zur Umsetzung produktiver Anwendungen verfügen. Doch auch Open-Source-Projekte sind auf dem Gebiet von CUI präsent und in einigen Bereichen, wie dem Natural Language Understanding, bereits konkurrenzfähig zu proprietären Systemen.

Durch die Nutzung existierender HTTP-Schnittstellen ist ein modularer Aufbau und die Kombination von Softwarebibliotheken und Diensten unterschiedlicher Hersteller möglich. Eine vorherige Definition des Dialogflusses ist auch in aktuellen probabilistischen Systemen noch notwendig. Jedoch ermöglicht der Einsatz von Machine Learning eine abstraktere Definition und eliminiert die Notwendigkeit zur expliziten Definition aller möglichen Konversationsverläufe.

5.2 Ausblick

Die prototypische Umsetzung eines CUI stellt eine erste Implementierung digitaler Sprachassistenten in der Altenpflege unter der Verwendung aktueller Technologien dar. Dabei werden die Möglichkeiten durch Nutzung aktueller Softwarebibliotheken und notwendige Schritte zur Planung eines Dialogablaufs und zur Fehlerbehandlung aufgezeigt.

Der Funktionsumfang deckt erste Aufgaben zur Unterstützung älterer Menschen in ihrem Alltag ab, lässt jedoch Raum für Erweiterungen. Die Anbindung der Chat-Plattform Telegram ermöglicht eine benutzerfreundliche Eingabe, bildet für älteren Menschen jedoch eine neue oder ungewohnte Benutzerschnittstelle und Bedarf einer Einführung. Eine natürlichsprachliche Eingabe sollte daher das Ziel zukünftiger Erweiterungen sein. Ebenso ist die Erweiterung durch Avatare oder humanoide Roboter denkbar, um eine menschenähnliche Kommunikation zu ermöglichen.

A Anhang

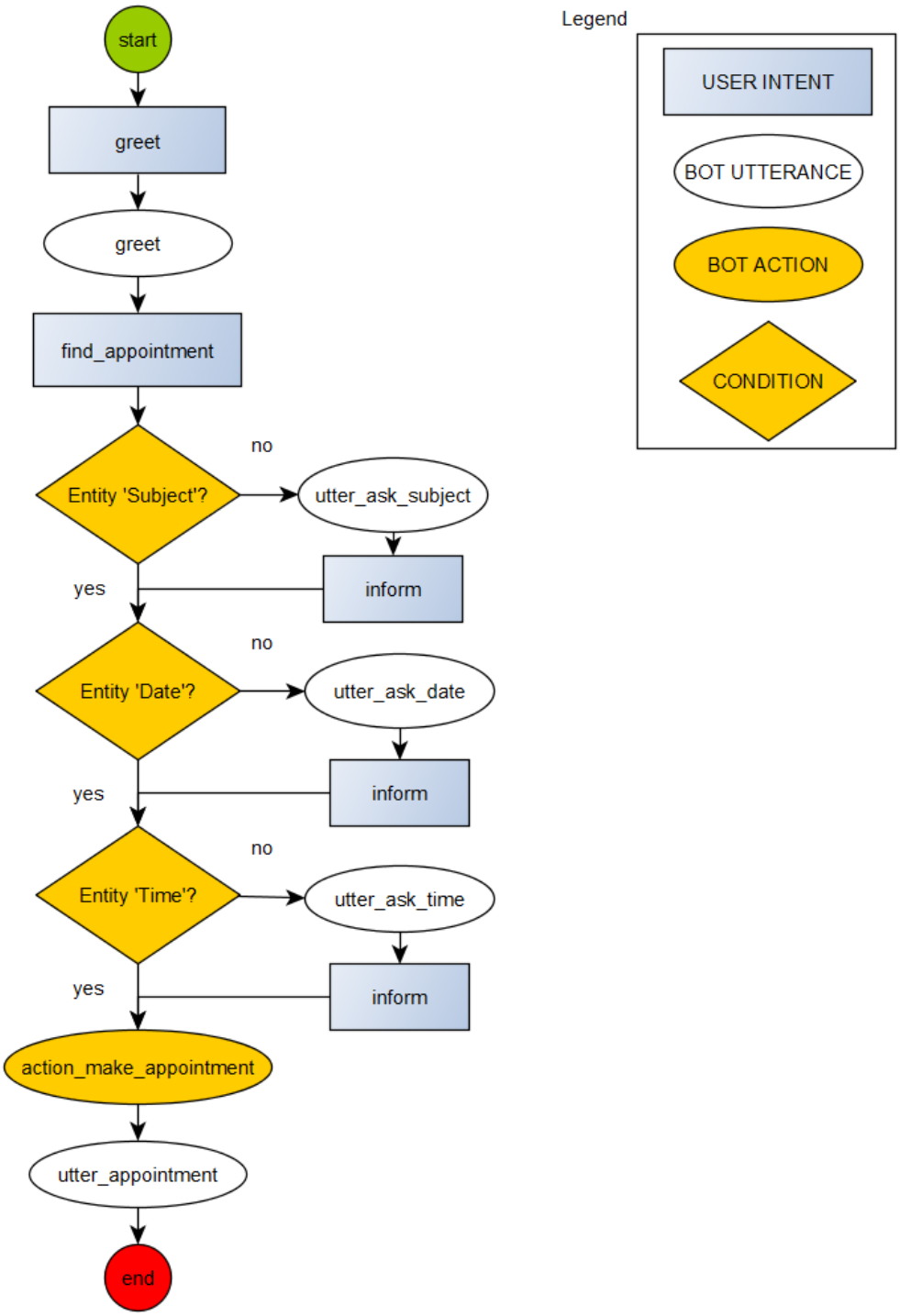

Abbildung A.1: Flussdiagramm eines Konversationsablaufs zur Erstellung eines Termins

Literaturverzeichnis

[Bat95] BATES, M: Models of natural language understanding. In: *Proceedings of the National Academy of Sciences of the United States of America* 92 (1995), Oktober, Nr. 22, 9977–9982. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC40721/. – ISSN 0027–8424

[BFPN17] BOCKLISCH, Tom ; FAULKNER, Joey ; PAWLOWSKI, Nick ; NICHOL, Alan: Rasa: Open Source Language Understanding and Dialogue Management. In: *31st Conference on Neural Information Processing System (NIPS 2017)* (2017), S. 9

[BHMML17] BRAUN, Daniel ; HERNANDEZ-MENDEZ, Adrian ; MATTHES, Florian ; LANGEN, Manfred: Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. In: *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. Saarbrücken, Germany : Association for Computational Linguistics, August 2017, 174–185

[BHSH04] BECKMAN, Mary ; HIRSCHBERG, Julia ; SHATTUCK-HUFNAGEL, Stefanie: The Original ToBi System and the Evolution of the ToBi Framework. In: *Prosodic Typology: The Phonology of Intonation and Phrasing*. 2004, S. 9–54

[CGB04] COHEN, Michael H. ; GIANGOLA, James P. ; BALOGH, Jennifer: *Voice User Interface Design*. Addison-Wesley Professional, 2004 http://proquest.tech.safaribooksonline.de/book/telephony/0321185765. – ISBN 978–0–321–18576–1

[Cho05] CHOWDHURY, Gobinda G.: Natural language processing. In: *Annual Review of Information Science and Technology* 37 (2005), Nr. 1, 51–89. http://dx.doi.org/10.1002/aris.1440370103. – DOI 10.1002/aris.1440370103. – ISSN 1550–8382

[CSGF15] CARDOSO, Walcir ; SMITH, George ; GARCIA FUENTES, Cesar: Evaluating text-to-speech synthesizers. In: *Critical CALL – Proceedings of*

the 2015 EUROCALL Conference, Padova, Italy, Research-publishing.net, Dezember 2015. – ISBN 978–1–908416–29–2, 108–113

[IMF13] INGERSOLL, Grant S. ; MORTON, Thomas S. ; FARRIS, Andrew L.: *Taming text: how to find, organize, and manipulate it*. Shelter Island : Manning, 2013. – ISBN 978–1–933988–38–2. – OCLC: ocn772977853

[KD18] KHAN, Rashid ; DAS, Anik: *Build Better Chatbots*. Springer, 2018

[KTPPSI16] KYAW THU, Ye ; PA PA, Win ; SAGISAKA, Yoshinori ; IWAHASHI, Naoto: Comparison of Grapheme-to-Phoneme Conversion Methods on a Myanmar Pronunciation Dictionary. In: *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*. Osaka, Japan : The COLING 2016 Organizing Committee, Dezember 2016, 11–22

[LJK+10] LEE, Cheong-Jae ; JUNG, Sang-Keun ; KIM, Kyung-Duk ; LEE, Dong-Hyeon ; LEE, Gary Geun-Bae: Recent Approaches to Dialog Management for Spoken Dialog Systems. In: *Journal of Computing Science and Engineering* 4 (2010), Nr. 1, S. 1–22. http://dx.doi.org/10.5626/JCSE.2010.4.1.001. – DOI 10.5626/JCSE.2010.4.1.001. – ISSN 1976–4677

[MBMU17] MIEHLE, Juliana ; BAGCI, Ilker ; MINKER, Wolfgang ; ULTES, Stefan: A Social Companion and Conversation Partner for Elderly. (2017), S. 6

[McT16] MCTEAR, Michael: *The conversational interface*. [Cham] : Springer International Publishing, 2016. – ISBN 978–3–319–32967–3

[OM13] OPPENAUER-MEERSKRAUT, Claudia: Would a virtual butler do a good job for old people? psychological considerations about benefits and risks of technological assistance. In: *Your Virtual Butler*. Springer, 2013, S. 11–15

[Pea16] PEARL, Cathy: *Designing Voice User Interfaces*. O'Reilly Media, Inc., 2016. – ISBN 978–1–4919–5541–3

[PLJ+14] PEETOOM, Kirsten ; LEXIS, Monique ; JOORE, Manuela ; D DIRKSEN, Carmen ; WITTE, Luc: Literature review on monitoring technologies and their outcomes in independently living elderly people. In: *Disability and rehabilitation. Assistive technology* 10 (2014), S. 1–24

[PVG+13] PORTET, François ; VACHER, Michel ; GOLANSKI, Caroline ; ROUX, Camille ; MEILLON, Brigitte: Design and evaluation of a smart home voice interface for the elderly: acceptability and objection aspects. In: *Personal and Ubiquitous Computing* 17 (2013), Januar, Nr. 1, 127–144. http://dx.doi.org/10.1007/s00779-011-0470-5. – DOI 10.1007/s00779–011–0470–5. – ISSN 1617–4917

[RM17] RASCHKA, Sebastian ; MIRJALILI, Vahid: *Python Machine Learning - Second Edition*. 2. Packt Publishing, 2017. – ISBN 978–1–78712–593–3

[SS69] SEARLE, John R. ; SEARLE, John R.: *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969. – ISBN 978–0–521–09626–3

[Tay00] TAYLOR, P.: Analysis and synthesis of intonation using the Tilt model. In: *The Journal of the Acoustical Society of America* 107 (2000), März, Nr. 3, S. 1697–1714. – ISSN 0001–4966

[TL03] TRAUM, David R. ; LARSSON, Staffan: The Information State Approach to Dialogue Management. In: *Current and New Directions in Discourse and Dialogue*. Springer, Dordrecht, 2003 (Text, Speech and Language Technology). – ISBN 978–1–4020–1615–8 978–94–010–0019–2, S. 325–353

[Uni17] UNITED NATIONS, DEPARTMENT OF ECONOMIC AND SOCIAL AFFAIRS: World Population Prospects: The 2017 Revision, Key Findings and Advance Tables. In: *Working Paper No. ESA/P/WP/248* (2017)

[VLP14] VACHER, Michel ; LECOUTEUX, Benjamin ; PORTET, Francois: Multichannel Automatic Recognition of Voice Command in a Multi-Room Smart Home: an Experiment Involving Seniors and Users with Visual Impairment. (2014), S. 5

[Wei66] WEIZENBAUM, Joseph: ELIZA—a Computer Program for the Study of Natural Language Communication Between Man and Machine. In: *Commun. ACM* 9 (1966), Januar, Nr. 1, 36–45. http://dx.doi.org/10.1145/365153.365168. – DOI 10.1145/365153.365168. – ISSN 0001–0782

[Win72] WINOGRAD, Terry: Understanding natural language. In: *Cognitive Psychology* 3 (1972), Januar, Nr. 1, 1–191. http://dx.doi.org/10.1016/0010-0285(72)90002-3. – DOI 10.1016/0010–0285(72)90002–3. – ISSN 0010–0285

[You02] YOUNG, Steve: Talking to Machines (Statistically Speaking). (2002), S. 8