

Aktives Dialogmanagement für eine natürlichsprachliche Benutzerschnittstelle

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Alexander Wachtel

aus Karlsruhe

Tag der mündlichen Prüfung: 24. Juli 2023

Erster Gutachter: Prof. Dr. rer. nat. Walter F. Tichy

Zweiter Gutachter: Prof. Dr.-Ing. Anne Koziolk

Dieses Werk ist lizenziert unter einer Creative Commons “Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International” Lizenz.



<https://creativecommons.org/licenses/by-sa/4.0/deed.de>

Inhaltsverzeichnis

Zusammenfassung	vii
Vorwort	xi
1 Einleitung	3
1.1 Problemstellung	4
1.2 Zielsetzung	5
1.3 Anforderungen	6
1.4 Abgrenzung	6
1.5 Beitrag der Arbeit	6
1.6 Struktur der Arbeit	7
2 Pilotstudie	9
2.1 Zweck	9
2.2 Fiktiver Assistent	9
2.2.1 Aufbau	10
2.2.2 Teilnehmer	11
2.2.3 Ergebnisse	12
2.3 Erzeugung neuer Funktionen	13
2.3.1 Aufbau	13
2.3.2 Teilnehmer	13
2.3.3 Ergebnisse	15
2.4 Schlussfolgerung	15
3 Verwandte Arbeiten	19
3.1 Zweck	19
3.2 Architektur eines Sprachsystems	20
3.2.1 Sprachverarbeitung	20
3.2.1.1 Stanford CoreNLP	20
3.2.1.2 Natural Language Toolkit	22
3.2.2 Sprachverstehen	23
3.2.2.1 Microsoft LUIS	23
3.2.2.2 RASA	23
3.2.3 Dialogmanagement	24
3.2.3.1 Microsoft Power Virtual Agents	24
3.2.3.2 Amazon Alexa Voice Service	26
3.2.3.3 Google DialogFlow CX	26
3.2.3.4 IBM Watson Assistant	28
3.2.4 Sprachgenerierung	28
3.2.4.1 Abrufbasiertes Antwortverhalten	29
3.2.4.2 Generatives Antwortverhalten	31
3.3 Programmieren in natürlicher Sprache	34

3.4	Benutzerprogrammierung	37
3.5	Schlussfolgerung und Forschungsfragen	38
3.5.1	Dialogverhalten	39
3.5.2	Datenstrukturanalyse	40
3.5.3	Algorithmenerfassung	40
4	Grundlagen	41
4.1	Tiefe und flache Sprachverarbeitung	41
4.2	Sprachparser	42
4.2.1	Artificial Intelligence Markup Language	42
4.2.2	Aktive Ontologie	44
4.3	Sprachverarbeitung	45
4.4	Ähnlichkeitsmaße	48
4.4.1	Lexikalische Ähnlichkeitsuntersuchung	48
4.4.1.1	N-Gramme mit Jaccard-Koeffizient	48
4.4.1.2	MongeElkan mit Jaccard-Koeffizient	48
4.4.1.3	Kosinus-Ähnlichkeit	49
4.4.1.4	Greedy String Trilling	49
4.4.2	Semantische Ähnlichkeitsuntersuchung	49
5	Natürlichsprachliche Schnittstelle	51
5.1	Übersicht	52
5.1.1	Komponenten	53
5.1.2	Softwarearchitektur	54
5.1.2.1	Präsentationsschicht	55
5.1.2.2	Steuerungsschicht	56
5.1.2.3	Geschäftslogikschicht	57
5.2	Aktives Dialogmanagement	58
5.2.1	Zweck	58
5.2.2	Geteilte Initiative	59
5.2.3	Vertikale und horizontale Dialogführung	59
5.2.4	Implementierung von <code>JustLingo.DialogManager</code>	61
5.2.5	Fazit	61
5.3	NLP-Fähigkeiten	62
5.3.1	Zweck	62
5.3.2	Sprachparser mit einer Aktivontologie	62
5.3.3	Klassifikation	64
5.3.3.1	Erkennung der Absicht	64
5.3.3.2	Training der Klassifikatoren	64
5.3.3.3	Klassifizierung einer neuen Eingabe	66
5.3.3.4	Implementierung von <code>JustLingo.Classification</code>	67
5.3.4	Dynamische Wortschatzerweiterung	68
5.3.5	Fazit	70
5.4	NLUI-Fähigkeiten	71
5.4.1	Zweck	71
5.4.2	Datenstrukturanalyse	71
5.4.2.1	Klassenmodelle aus Text	73
5.4.2.2	Klassenmodelle aus Tabellen	75
5.4.2.3	Erkennen vorhandener Datenstrukturen	80
5.4.2.4	Abfragen mehrerer Datenquellen	81
5.4.3	Erfassung der Folgenalgorithmen	83
5.4.3.1	Variablen	84

5.4.3.2	Bedingungen	85
5.4.3.3	Schleifen	86
5.4.3.4	Implementierung von <code>JustLingo.Algorithms</code>	88
5.4.4	Fazit	89
5.5	Zusammenfassung	89
6	Evaluation	91
6.1	Studie I: Vergleich der Parser	92
6.1.1	Studie I.1: Musterbasierter Parser	92
6.1.1.1	Aufbau	92
6.1.1.2	Ergebnisse	92
6.1.2	Studie I.2: Parser mit Aktivontologien	93
6.1.2.1	Aufbau	93
6.1.2.2	Ergebnisse	94
6.1.3	Fazit	94
6.2	Studie II: Klassifikation	96
6.2.1	Studie II.1: Balancierte Datensätze	96
6.2.1.1	Aufbau	96
6.2.1.2	Ergebnisse	97
6.2.2	Studie II.2: Unbalancierte Datensätze	98
6.2.2.1	Aufbau	98
6.2.2.2	Ergebnisse	98
6.2.3	Fazit	98
6.3	Studie III: Datenstrukturanalyse	99
6.3.1	Studie III.1: Klassenmodelle aus Text	99
6.3.1.1	Aufbau	99
6.3.1.2	Ergebnisse	100
6.3.2	Studie III.2: Klassenmodelle aus Tabellen	100
6.3.2.1	Aufbau	100
6.3.2.2	Ergebnisse	100
6.3.3	Fazit	101
6.4	Studie IV: Erweiterung der Funktionalität	101
6.4.1	Studie IV.1: Kontrollstrukturen	101
6.4.1.1	Aufbau	101
6.4.1.2	Ergebnisse	102
6.4.2	Studie IV.2: Einfache Folgenalgorithmen	104
6.4.2.1	Aufbau	104
6.4.2.2	Ergebnisse	105
6.4.3	Fazit	106
6.5	Akzeptanz von JustLingo NLUI	106
6.5.1	Aufbau	106
6.5.2	Ergebnisse	107
6.5.3	Fazit	107
6.6	Zusammenfassung	108
7	Diskussion	109
7.1	Erfüllung der Forschungsfragen	109
7.2	Empfehlungen für die Entwicklung von NLUI	111
7.3	Implementierung eines NLUI nach verfassten Empfehlungen	112
7.4	Bereitstellung der NLUI-Bibliotheken	113
7.4.1	Lokale Benutzerprogrammierung	113
7.4.2	Globale Bereitstellung	114

8	Zusammenfassung und Ausblick	117
8.1	Zusammenfassung	117
8.2	Ausblick und zukünftige Arbeiten	118
8.2.1	Lernende Systeme	118
8.2.2	Bedeutungsschlaue KI-Systeme	118
8.2.3	Komplexere Algorithmen	119
9	Anhang	121
9.1	Pilotstudie	121
9.2	Datenstrukturanalyse	124
9.3	Algorithmenerkennung	126
9.4	Evaluation	130
9.4.1	Aktivontologie	130
9.4.2	Klassifikation	131
9.5	Publikationsliste	137
9.6	Programmausschuss und Gutachter	138
	Literaturverzeichnis	139

Zusammenfassung

Aktuelle Sprachsysteme, wie OpenAI's ChatGPT, zeigen uns die Mächtigkeit der natürlichen Sprache in einer Mensch-Maschine-Interaktion. Durch eine textuelle, natürlichsprachliche Eingabe erteilen die Menschen einer Maschine komplexe Anweisungen und erhalten innerhalb von Sekunden textuelle Ausgaben, die einen Text enthält und von einer Formulierung eines Menschen, nicht zu unterscheiden ist. In naher Zukunft werden solche natürlichsprachlichen Benutzerschnittstellen (Engl.: Natural Language User Interface, kurz: NLUI) in der Lage sein, neben reiner Textausgabe, die Ausgabe als Datei, Bild und einem anderen Format zu erzeugen. Dadurch werden natürlichsprachliche Benutzerschnittstellen immer mehr zum festen Bestandteil unseres Alltags und ein mächtiges Hilfswerkzeug für viele Alltagsaufgaben. Dabei definieren wir eine natürliche Sprache als eine gewöhnliche, von den Menschen täglich verwendete Sprache, mit der wir mit den anderen Menschen interagieren. In der Regel brauchen die Menschen mehrere Jahre, um die Verwendung der natürlichen Sprache zu erlernen. Durch Zuhören lernt ein Kind zunächst einfache Laute und Einzelwörter und gibt diese nach einer Zeit wieder. Als nächstes erfolgt eine Wiederholung der gelernten Wörter sowie mit etwas Erfahrung eine Konkatenation der Wörter zu immer mehr komplexeren Ausdrücken. In der Schule erlernt das Kind die Bildung der Sätze nach einer vordefinierten Grammatik.

Auf der einen Seite ermöglicht die natürlichsprachliche Schnittstelle eine für den Benutzer einfache Kommunikation mit dem Softwaresystem. Die natürliche Sprache zu verwenden, hat den Vorteil, dass die Benutzer direkt mit der Interaktion beginnen können, ohne eine Systemsprache oder eine Spezifikation lernen zu müssen. Auf der anderen Seite stellt die Implementierung solcher Sprachsysteme für die Entwickler häufig aufgrund der Komplexität eine schwere Aufgabe dar. Die vorliegende Dissertation untersucht die Hürden im Umgang mit den natürlichsprachlichen Schnittstellen, sammelt aus der Sicht eines Entwicklers Erkenntnisse für eine Implementierung, formuliert die Anforderungen an derartige Systeme und leitet Auswirkungen auf den Entwicklungsprozess einer natürlichsprachlichen Benutzerschnittstelle ab. Die Dissertation präsentiert eine prototypenbasierte Implementierung einer natürlichsprachlichen Schnittstelle, genannt *JustLingo Natural Language User Interface* oder kurz *JustLingo NLUI*.

Als Ausgangspunkt der Forschungsarbeit dokumentiert eine Pilotstudie das Benutzerverhalten im Umgang mit einer natürlichsprachlichen Schnittstelle. Ein fiktiver Assistent ermittelt im ersten Experiment der Pilotstudie den Bedarf einer natürlichsprachlichen Schnittstelle. Dieses Experiment zeigt, dass unerfahrene Benutzer bei der Bearbeitung

verschiedener Aufgaben in Tabellenkalkulation einen digitalen Assistenten nach Hilfe fragen. Dabei nimmt der Informationsgehalt einer Benutzereingabe mit dem Wissen und dem Erfahrungslevel der Benutzer zu. Für die weitere Untersuchung leitet das Experiment die Anforderung eines Dialogsystems ab.

Die Erkenntnisse der Pilotstudie werden genutzt, um die verwandten Arbeiten und die bereits vorhandenen Lösungen aus dem Forschungsbereich zu untersuchen, sowie im Anschluss die offenen Forschungsfragen abzuleiten. Das Ergebnis ist, dass die aktuellen Implementierungen der natürlichsprachlichen Benutzerschnittstellen, wie OpenAI's ChatGPT, in den meisten Fällen eine brauchbare Ausgabe zu einer validen Eingabe eines Benutzers liefern. Es fehlt jedoch an den NLUIs, die mit fehlerhaften oder unvollständigen Eingabe eines Benutzers umgehen können. Der Fokus der Forschung soll dabei auf einer Verbesserung der Interaktion mit dem Benutzer liegen.

Für das Erreichen dieses Ziels ist eine NLUI mit mehreren Komponenten notwendig, die verschiedene Fähigkeiten des Gesamtsystems bereitstellen. Die Forschungsarbeit schlägt als zentrale Komponente ein mehrstufiges, zielgerichtetes und adaptives Dialogsystem vor, das mit anderen NLUI Fähigkeiten ausgestattet ist und bei Bedarf diese aufrufen kann. Durch ein aktives Dialogmanagement interagiert die natürlichsprachliche Schnittstelle mit einem Benutzer in einer geteilten Initiative. Sie kann mit den Rückbezügen auf die bereits mitgeteilten Eingaben und aus der Ungenauigkeit der natürlichen Sprache resultierenden Mehrdeutigkeiten umgehen. Des Weiteren vereint JustLingo NLUI mehrere Module, die unterschiedliche NLUI-Fähigkeiten des Gesamtsystems darstellen. Daher wird jede Benutzereingabe syntaktisch und semantisch zu den jeweiligen Fähigkeiten der natürlichsprachlichen Schnittstelle klassifiziert und an das korrekte Modul zur Verarbeitung übergeben. Außerdem verfügt es über die Fähigkeit, die Beschreibung eines Benutzers zur Funktionalitätserweiterung zu erfassen und die Einzelkomponente, wie Objekte, Methoden und Beziehungen zwischen den Objekten für die spätere Verwendung in einer geeigneten Form abzuspeichern. Die einzelnen Bestandteile einer Benutzereingabe werden dafür auf Funktionsalgorithmen untersucht. Die natürlichsprachliche Schnittstelle unterscheidet zwischen (i) Einzelbefehlen, (ii) Folgen von Einzelbefehlen sowie (iii) bedingten und wiederholenden Befehlen. Um eine Benutzeranfrage auf den Daten zu ermöglichen, erfasst JustLingo NLUI eine textuelle Beschreibung oder eine vorhandene Datenstruktur einer Tabellenkalkulation als Klassenmodell mit Attributen und Methoden. In einer tabellarischen Form stellen die Klassen die Zeilen und die zugeordneten Attribute die Spalten dar. Die Daten einer Tabelle repräsentieren jeweils eine Instanz der Klasse. Die Methoden zur Datenmanipulation können analog zu der Algorithmerkennung erfasst werden.

Im Rahmen der Forschungsarbeit wurden unabhängige Benutzerstudien durchgeführt, die den Nachweis der Funktionalität der zugehörigen Implementierung liefern. Im Vergleich zu einem musterbasierten Dialogsystem, das eine Erfolgsquote von 79,5% erreicht, schnitt ein Ansatz mit einer flexiblen Sprachanalyse um 10% besser ab. Die Evaluation der Klassifikation ergab einen F-Wert von 95%. Die Algorithmerkennung wurde in drei Komplexitätsklassen untersucht und zeigt eine Erfolgsrate von durchschnittlich 60%. Mit einer einfachen Synonymerweiterung bei unbekanntem Ausdrücken erhöht sich der Wert auf 80%. Bei der Vorstellung des finalen Prototyps wünschen 78% der Teilnehmer eine Integration der natürlichsprachlichen Schnittstelle in eine traditionelle Software.

Die Erkenntnisse der Forschungsarbeit lassen eine Definition der Best Practice für die Implementierung einer NLUI zu. Allerdings gibt es noch weiteres Potenzial für Verbesserungen. Insbesondere gilt in Zukunft die Komplexität der unterstützten Algorithmen zu erhöhen, das Dialogsystem im Umgang mit komplexen unbekanntem Daten robust zu machen sowie durch eine lernende Komponente eine bessere Anpassung an das Benutzerverhalten zu schaffen.

Vorwort

Als Erstes gilt der Dank meiner Familie, die mich über Jahre bei der Dissertation unterstützt hat. Meiner Frau Irina, die für das Ziel der Fertigstellung der Dissertation auf gemeinsame Zeit verzichtet und mich ermutigt hat, immer weiter zu machen. Unserer Tochter Elissa, die schon früh mitbekam, dass ich an der Dissertation arbeite und immer wieder eine Einleitung für die Ausarbeitung geschrieben hat. Für die Unterstützung möchte ich mich bei meinen Eltern, die mir immer hilfsbereit während der Promotion zur Seite standen und meinem Bruder für die Diskussionen sowie das Korrekturlesen der Ausarbeitung bedanken. Verwandten und Freunden danke ich für den Antrieb zum Abschluss der Dissertation.

Bei Herrn Prof. Dr. rer. nat. Walter F. Tichy möchte ich mich für die Betreuung der Promotion und die wegweisende Diskussion während der Verfassung der Dissertation bedanken. Neben der Forschungsarbeit am Lehrstuhl durfte ich interessante Lehre halten, bei der die Endergebnisse auf Wettbewerben eingereicht und als Resultat der Lehrstuhl durch einige Preise ausgezeichnet wurde.

Bei Herrn Prof. em. Dr.-Ing. Dr. h. c. Peter C. Lockemann bedanke ich mich für das Korrekturlesen der Ausarbeitung. Die Diskussion der kritischen Fragestellungen und das Abwägen der jeweiligen Aussagen haben einen wesentlichen Anteil an der Fertigstellung der Dissertation beigetragen.

Bei Frau Prof. Dr.-Ing. Anne Koziolk bedanke ich mich für das Korreferat. Die Diskussion der Forschungsarbeit und die Vorschläge zum Aufbau der Evaluation haben bei der Verfassung der Dissertation sehr geholfen.

An die PSE-Lehrveranstaltungen „Programmierung in natürlicher Sprache“ mit Martin Tillmann und „Microsoft Imagine Cup“ mit Marc Kiefer erinnere ich mich sehr gerne. Bei den Kollegen des Lehrstuhls möchte ich mich für die gemeinsame Zeit bedanken und Ihnen alles Gute für die Zukunft wünschen.

Während der Forschungsarbeit habe ich mit einigen Studenten eng zusammengearbeitet. In der Pilotstudie entstand am Anfang mit Philipp Voigt das erste Experiment sowie die erste wissenschaftliche Veröffentlichung. Danach folgte eine langfristige Zusammenarbeit mit Jonas Klamroth im PSE-Praktikum sowie einem Seminar im Forschungsgebiet. Als nächsten möchte ich Felix Eurich erwähnen, der über mein Seminar Programmieren in natürlicher Sprache auf das Forschungsgebiet aufmerksam wurde. Die tolle Zusammenarbeit wurde mit einer Abschlussarbeit und einer langfristigen Hiwibeschräftigung am Lehrstuhl

fortgesetzt. Nicht zuletzt wird Dominik Fuchß erwähnt, der ebenfalls über das Seminar zu einer Abschlussarbeit an den Lehrstuhl kam. Er strebt eine Promotion am Lehrstuhl von Prof. Dr.-Ing. Anne Koziolk an. Euch allen wünsche ich alles Gute!

Nicht zuletzt bleibt mir noch die Vision von Prof. Dr. Walter F. Tichy zu nennen, die ich über Jahre in der Forschungswelt unterstützt habe:

Programming today requires years of training.
With natural language,
programming would become available to everyone!

Karlsruhe, 24. Juli 2023

Alexander Wachtel

1. Einleitung

Seit ihrer Erfindung werden Softwaresysteme unter Verwendung spezieller, künstlicher Notationen programmiert: den Programmiersprachen [EC93, CLLS09]. Unter einem Softwaresystem wird ein Programm verstanden, das einer implementierten Logik eines Algorithmus folgt. Ein Algorithmus besteht u.a. aus Anweisungen zur Ausführung, bedingten sowie wiederholenden Aktionen und wird durch einen Entwickler in einer Programmiersprache bereitgestellt [CLLS09]. Der Vorteil eines Algorithmus ist die zuverlässige, schnelle und deterministische Verarbeitung der Daten.

Vergleichen wir die Anforderung an die Implementierung einer herkömmlichen, grafischen Benutzerschnittstelle (Graphic User Interface, GUI), stellen wir fest, dass ein Benutzer sich deterministisch durch die Menüs und die Dialoge auf den vom Entwickler vordefinierten Pfaden bewegt (Siehe Abbildung 1.1). Bei einer korrekten Ausführung des Programms ist einem Benutzer in der Regel nicht möglich, eine unvorhergesehene Aktion durch das Betätigen der GUI-Elemente zu erreichen.

Ein Softwaresystem, das mit Eingaben in natürlicher Sprache arbeitet bzw. derartige Ausgaben erzeugt, wird üblicherweise als natürlichsprachlich bezeichnet. Typischerweise wird zwischen gesprochener und geschriebener Sprache als Eingabemodalität unterschieden. Die Verarbeitung gesprochener Sprache beschäftigt sich mit der Sprachperzeption und der Sprachproduktion, also dem Verständnis und der Erzeugung von natürlicher Sprache (vgl. [CEE⁺10], [Hü06]) und liegt nicht im Fokus dieser Arbeit. Vielmehr behandeln wir im Rahmen dieser Arbeit die *geschriebene natürliche Sprache* als Ausgangspunkt.

Natürlichsprachliche Benutzerschnittstelle

Wir definieren eine natürlichsprachliche Benutzerschnittstelle (Natural Language User Interface, NLUI) als eine Mensch-Maschine-Schnittstelle, die eine Kommunikation zwischen Mensch und Computer in gewöhnlicher Sprache erlaubt.

Die Anforderung eine natürlichsprachliche Schnittstelle zu programmieren, sieht im Vergleich zu einer GUI-Implementierung viel komplexer aus. Die Mächtigkeit der natürlichen Sprache und insbesondere die Dynamik, mit der die natürlichsprachliche Schnittstelle auf den Benutzer reagieren muss, stellt eine Herausforderung für einen Entwickler dar. Um der NLUI den Umgang mit natürlicher Sprache beizubringen, bedarf es eines Backends, das für

Benutzereingaben vorbereitet ist und extrahierte Information in einem allgemeinen Modell speichert und bei Bedarf abrufen. Ein Entwickler kann das Backend unmöglich auf sämtliche Eingaben aller Benutzer vorbereiten. Es kann also kein vergleichbarer Determinismus bei den NLUIs, wie bei den GUIs, erreicht werden.

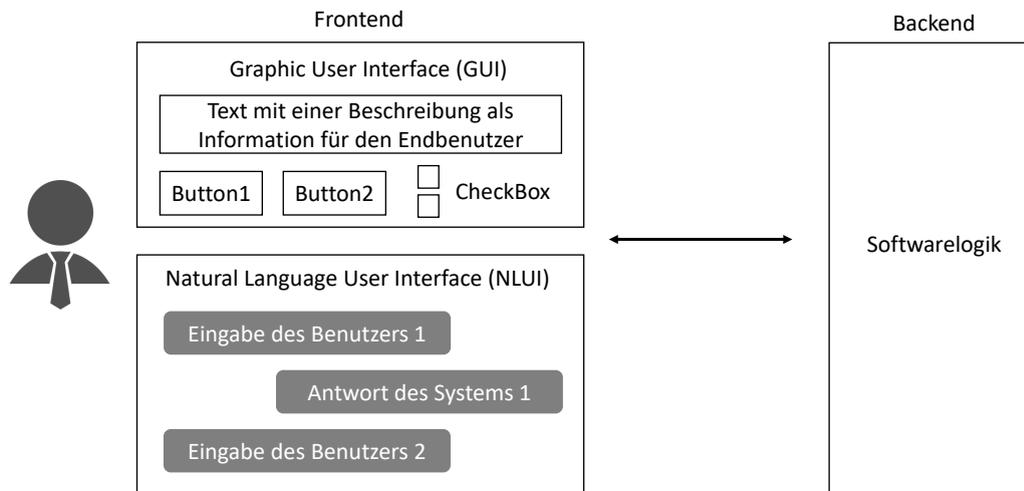


Abbildung 1.1: Schnittstellen im Vergleich

Die Benutzeranfragen an eine natürlichsprachliche Schnittstelle werden immer mehr fester Bestandteil des alltäglichen Lebens. Im Jahr 2020 wurden die sprachlichen Assistenten bereits in mehr als zwei Milliarden Geräte verwendet. Dabei teilt sich die Anzahl der Installationen laut den Herstellern auf eine Milliarde Geräte mit dem Google Assistant, 500 Mio. mit Siri, 400 Mio. mit Cortana und 200 Mio. mit Amazon Alexa auf.

1.1 Problemstellung

Seit dem Beginn der Entwicklung digitaler Rechnersysteme spielen Benutzerschnittstellen eine entscheidende Rolle bei der Akzeptanz von Softwaresystemen. Die aktuellen Veröffentlichungen, wie OpenAI's ChatGPT, unterstützen die Benutzer bei natürlichsprachlichen Eingaben mit textuellen Lösungen, die von der Formulierung kaum von die eines Menschen zu unterscheiden sind. Jedoch reagieren diese Sprachsysteme auf korrekte und vollständige Eingaben. Bei der Bereitstellung von unvollständigen oder falschen Eingaben, erfolgt eine unbrauchbare oder falsche Ausgabe.

Wie am Beispiel von ChatGPT ist es ersichtlich, dass der Erfolg der natürlichsprachlichen Schnittstelle von der Größe der Trainingsdaten abhängt. Dabei sollte eher der Fokus auf der Qualität der Daten, statt auf Quantität gelegt werden.

Wir gehen davon aus, dass Benutzer bei der Verwendung der natürlichsprachlichen Schnittstellen nicht zwangsläufig korrekt handeln und eine Unterstützung bei fehlerhaften Eingaben benötigen. Das NLUI muss durch die Entwickler in die Lage versetzt werden, robust auf unbekannte Eingaben zu reagieren, ohne eine Fehlermeldung in der Interaktion mit dem Benutzer auszugeben.

1.2 Zielsetzung

Unter der Annahme, dass eine natürlichsprachliche Schnittstelle für eine Tabellenkalkulation existiert und ein Benutzer über diese in natürlicher Sprache kommunizieren möchte, verfolgt die vorliegende Arbeit die Zielsetzung, den Benutzern eine Kommunikation mit der natürlichsprachlichen Schnittstelle zu ermöglichen und einen Lösungsansatz für die Implementierung mittels einer Schnittstelle in geschriebener natürlicher Sprache zu geben. Außerdem sind die Hürden im Umgang mit den natürlichsprachlichen Schnittstellen zu erforschen. Aufgrund der zunehmenden Verbreitung erwarten wir, dass Entwickler für Ihre Software zunehmend natürlichsprachliche Schnittstellen implementieren müssen.

Abbildung 1.2 stellt einen Ablauf bei der traditionellen Programmierung dar. Auf der einen Seite befindet sich ein Benutzer, der die Anforderungen an ein Softwaresystem nennt. Auf der anderen Seite bekommt ein Entwickler die Anforderung und implementiert einen Algorithmus, der die gewünschte Funktionalität zur Verfügung stellt.

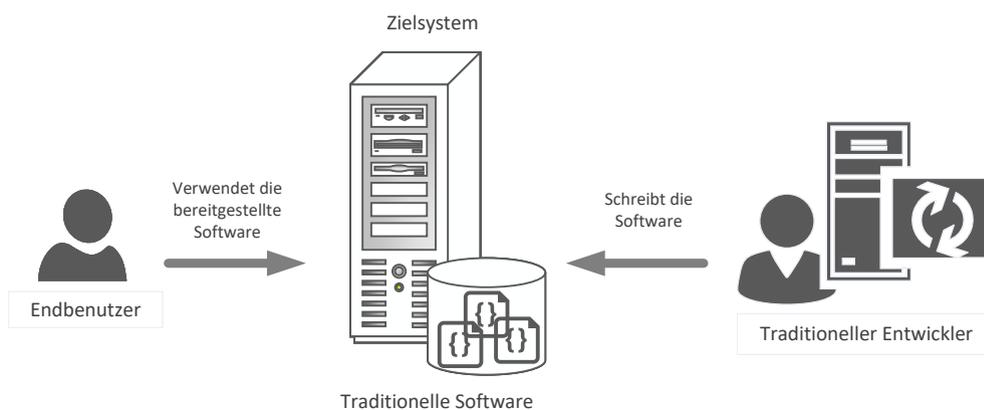


Abbildung 1.2: Traditioneller Ablauf bei der Verwendung einer Software

Abbildung 1.3 erweitert den Ablauf durch eine natürlichsprachliche Schnittstelle. In natürlicher Sprache kann ein weiterer Benutzer durch das NLUI mit dem Backend interagieren und u.a. Daten abfragen oder eine Beschreibung für eine Aktion übermitteln. Ein NLUI-Entwickler implementiert die natürlichsprachliche Schnittstelle.

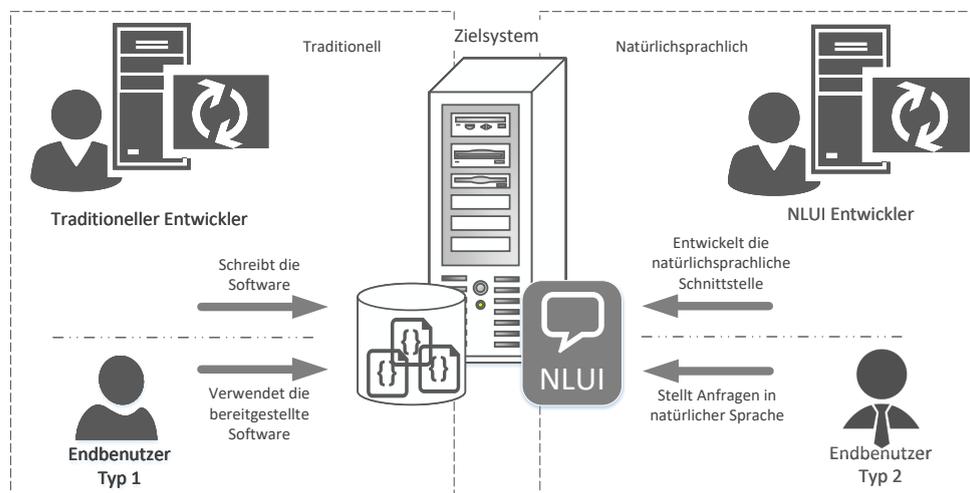


Abbildung 1.3: Ablauf mit einer natürlichsprachlichen Schnittstelle

1.3 Anforderungen

Für die Implementierung einer natürlichsprachlichen Schnittstelle wird ein Zielsystem gesucht, das für die meisten Benutzer bereits bekannt ist und einen Aufbau der Datenstrukturen aufweist, die eine natürlichsprachliche Bearbeitung ermöglichen. Wegen diesen Anforderungen haben wir uns für die Tabellenkalkulation entschieden. Die erstellten Tabellenkalkulationen sind nicht nur die traditionelle tabellarische Darstellung von Daten, sondern erlauben auch eine formularbasierte Datenmanipulation. Des Weiteren ist die Darstellung der Daten für die Benutzer leicht nachzuvollziehen, so dass vorhandene Datensätze in einer Tabelle gut zu verstehen, zu ergänzen und zu manipulieren sind. Im Allgemeinen werden Tabellen bereits seit mindestens 7000 Jahre verwendet [Hur00]. Abraham [AE04] schätzt eine jährliche Erstellung von 100 Mio. Tabellen. Im Jahr 2010 gab Microsoft die Verbreitung von Microsoft Excel mit 1,2 Mrd. Nutzer weltweit an.

Die weite Verbreitung von Microsoft Office im Jahr 2010 unterstreicht ebenfalls Statista mit der Angabe der Verbreitung bei 72% der Nutzer. Auf Platz zwei mit 22% folgt Open Office, andere Bürosoftware liegt in der Summe unter 6%.

1.4 Abgrenzung

Die Forschungsarbeit soll nicht dazu genutzt werden, um einen Ersatz von Softwareentwicklern anzustreben und stellt keine Lösung von allen natürlichsprachlichen Problemen dar. Des Weiteren sollte Grundlagenforschung an natürlichsprachlichen Schnittstellen durchgeführt werden, so dass kein Anspruch in der Implementierung einer produktreifen Software besteht.

Die natürlichsprachliche Schnittstelle soll außerdem keine Antworten auf alle möglichen Fragen liefern, nicht über das Weltwissen verfügen und nicht einem Smalltalk dienen. Es sollte vielmehr erforscht werden, wie eine natürlichsprachliche Schnittstellen mit einer kleinen Wissensbasis gute Ergebnisse liefern kann. Nicht zuletzt wird kein Maschinelles Lernen verwendet, so dass kein langes Trainieren von großen Sprachmodellen erforderlich ist.

1.5 Beitrag der Arbeit

Der Beitrag der Arbeit lässt sich in folgende Punkte aufteilen:

Implementierung einer natürlichsprachlichen Benutzerschnittstelle

Insgesamt untersucht die Dissertation die Hürden im Umgang mit den natürlichsprachlichen Schnittstellen, sammelt aus der Sicht eines Entwicklers Erkenntnisse für eine Implementierung, formuliert die Anforderungen an derartige Systeme und leitet Auswirkungen auf den Entwicklungsprozess einer natürlichsprachlichen Benutzerschnittstelle ab.

Das Programmieren der natürlichsprachlichen Schnittstellen ist eine Herausforderung für die Entwickler. Eine wichtige Voraussetzung für einen raschen Fortschritt in diesem Bereich ist ein Maßstab, der den Vergleich verschiedener Ansätze erleichtert und den Wettbewerb unter den Forschern fördert.

Empfehlungen für NLUI-Entwickler

Die gewonnenen Erkenntnisse bei der prototypenhaften Implementierung einer natürlichsprachlichen Schnittstelle werden im Rahmen dieser Forschungsarbeit als *Empfehlungen* für die zukünftigen NLUI-Entwickler formuliert.

1.6 Struktur der Arbeit

Kapitel 2 stellt den Ausgangspunkt der Forschungsanfang in einer empirischen Studie vor, die u.a. den Bedarf einer natürlichen Schnittstelle feststellt. Die Erkenntnisse und Schlussfolgerungen der Pilotstudie formulieren den Anforderungskatalog für die Implementierung einer natürlichsprachlichen Schnittstelle. Anschließend diskutiert Kapitel 3 die verwandten Arbeiten, grenzt den Anforderungskatalog aus der Pilotstudie mit bereits durchgeführter Forschung ab und präsentiert anschließend die Forschungsfragen der Dissertation. Kapitel 4 beschäftigt sich mit den Grundlagen, die zum Verständnis der Forschungsarbeit benötigt werden. Kapitel 5 diskutiert anhand der einzelnen Forschungsfragen die Komponenten einer natürlichsprachlichen Schnittstelle. Kapitel 6 präsentiert die Evaluation einzelner Komponenten und zieht ein Fazit bzgl. der gestellten Forschungsfragen. Die Erkenntnisse der Forschungsarbeit fasst Kapitel 7 zusammen und diskutiert die Vorgaben für die zukünftige Entwicklung neuer natürlichsprachlicher Schnittstellen. Im Kapitel 8 erfolgt eine Zusammenfassung der Arbeit und ein Ausblick auf weiterführende Arbeiten.

2. Pilotstudie

Eine von uns durchgeführte Pilotstudie [WWVT15, WWT15] markiert den Ausgangspunkt der Forschungsarbeit an einer natürlichsprachlichen Schnittstelle.

2.1 Zweck

Die Pilotstudie¹ besteht aus zwei aufeinanderfolgenden Experimenten:

- Die erste Experimente in Kapitel 2.2 dokumentiert das Benutzerverhalten im Umgang mit einem fiktiven natürlichsprachlichen Assistenten bei der Bearbeitung verschiedener Aufgaben in einer Tabellenkalkulation.
- Das zweite Experiment baut auf den Erkenntnisse des ersten Experiments auf und untersucht im Kapitel 2.3 die Kommunikation der Benutzer mit einer prototypischen Implementierung eines Dialogsystems und sammelt erste Erkenntnisse bei der Erweiterung von vorhandener Funktionalität mittels natürlicher Sprache.

Der Zweck der Pilotstudie ist es, (1) den Bedarf einer natürlichsprachlichen Schnittstelle für die Benutzer zu ermitteln sowie (2) erste Erkenntnisse zu den Anforderungen für eine Implementierung einer natürlichsprachliche Schnittstelle zu sammeln.

2.2 Fiktiver Assistent

Zunächst müssten wir herausfinden, wie Benutzer mit einem Assistenten in natürlicher Sprache interagieren. Für diesen Zweck, rekrutierten wir Probanden und führten eine Simulation durch. Anhand einer Tabelle sollten sich die Probanden bei der Simulation vorstellen, die Lösung zur gegebenen Aufgabenstellung einem Menschen zu erklären und die dazugehörige Formulierung in natürlicher Sprache aufzuschreiben. Eine Tabelle mit Daten wurde zusammen mit den Aufgaben von uns zur Verfügung gestellt [Voi14].

¹Die Pilotstudie umfasst insgesamt fünf Experimente. Aus Relevanzgründen werden die Experimente zur Erkennung von Tabellendaten, zur Verarbeitung einer gesprochenen Eingabe und zur Untersuchung einer Entscheidungsanalyse mit einem Langzeitgedächtnis vernachlässigt.

2.2.1 Aufbau

Die Fragestellung im Experiment besteht darin, textuelle Lösungen zu Aufgaben in alltäglichem Englisch zu formulieren. Die zwölf Aufgaben stammen aus dem Lehrbuch von Curtis D. Frye *Microsoft Excel 2013, Step by Step* [Fry13]. Die Teilnehmer sollten angeben, ob sie die Aufgabe lösen konnten oder nicht, sowie deren Lösung auf Englisch ausformulieren.

Bei den ersten sechs Aufgaben des Experiments sollen einfache arithmetischen Berechnungen formuliert werden. Dabei fokussiert die Aufgabenstellung auf die Berechnung eines Notendurchschnitts sowie einer Endnote der Schüler in natürlicher Sprache. Abbildung 2.1 zeigt die Notentabelle mit den Daten, so dass im einfachen Szenario die gesuchten arithmetischen Operation den Teilnehmern direkt ersichtlich ist.

Die Lösung besteht aus mehreren Teilen: (1) die Berechnung einer Summe sowie (2) eines Mittelwerts von mehreren Zellen, (3) das Runden einer Dezimalzahl und (4) das Übertragen einer Formel in weitere Zellen derselben Spalte.

Die Interpretation von Daten in Abbildung 2.1 ergibt:

- Zelle A1 zeigt die Hauptüberschrift
- Zellen A3 bis A15 präsentieren die vertikalen Tabellenüberschriften
- Zellen B3 bis F3 präsentieren die horizontalen Tabellenüberschriften
- Zellen B4 bis F15 enthalten die Daten der Tabelle

	A	B	C	D	E	F
1	Grades overview					
2						
3	Name	Daily grades 1	Daily grades 2	Daily grades 3	Final exam	Average
4	Marcel	35,7	35,3	20,0	40,0	
5	Andrea	75,0	95,0	65,0	95,0	
6	Simon	100,0	95,0	75,0	85,0	
7	Andrew	90,0	95,6	60,0	80,8	
8	Mike	100,0	35,0	70,0	45,0	
9	Tyler	95,5	100,0	85,1	90,0	
10	Anna	60,0	40,0	35,0	35,0	
11	Mary	80,0	85,0	80,0	90,4	
12	Flinn	65,0	20,0	70,9	40,6	
13	Rick	30,3	20,7	40,0	20,0	
14	Alex	80,0	60,0	85,0	20,0	
15	Average	73,77	61,96	62,36	58,35	

Abbildung 2.1: Ein Beispielszenario für das erste Experiment aus [WWT15]

Eine der Aufgaben bestand darin, Rechenoperationen in einer Excel-Formel anzugeben. Wie an den Beispielen zu sehen, ist die Fragestellung bewusst so formuliert, dass keine konkrete Vorgehensweise beschrieben wird:

Beispiel 1:

In der Spalte Mittelwert soll für jeden Schüler der Durchschnitt aller seiner Leistungen berechnet werden. Es soll die entsprechende Berechnung speziell für den Schüler Marcel durchgeführt und in Zelle F4 geschrieben werden.

Beispiel 2:

Der Mittelwert in Spalte F aus der vorherigen Fragestellung berücksichtigt nicht die unterschiedliche Gewichtung aus schriftlichen und mündlichen Leistungen, dies soll nun geschehen. Es soll die entsprechende Berechnung speziell für den Schüler Marcel durchgeführt und in Zelle F4 geschrieben werden.

Weitere sechs Aufgaben beziehen sich auf die Manipulation von vorhandenen Daten in einer Tabellenkalkulation. Demnach müssen die Teilnehmer natürlichsprachliche Lösungen zu folgenden Aufgaben formulieren: (1) das Einfügen von Daten in einer Zelle, (2) das Sortieren von Daten innerhalb einer Spalte sowie (3) das Erzeugen und Anpassen von Diagrammen.

Beispiel 3:

Die Erträge aus Kundenverträgen werden übersichtlich in Form einer Tabelle dargestellt. Sie möchten die Kunden nach Erträgen im Jahr 2013 in Spalte E absteigend sortieren, so dass die Umsatzstärksten zuerst dargestellt werden.

Weitere Aufgaben sind im Anhang 9.1 aufgelistet.

2.2.2 Teilnehmer

Zur Durchführung des Experiments erfolgte eine Einladung an 57 Teilnehmer verschiedenen Alters und unterschiedlicher Berufe, sich an der Pilotstudie zu beteiligen. Unter den 57 Teilnehmern waren 30 in einer schulischen Ausbildung oder Studium, 24 Berufstätige sowie 2 Nichtberufstätige vertreten. Die Studierenden teilten sich auf in 12 Studierende der Informatik, 5 der Wirtschaftswissenschaften und 11 aus anderen Studiengängen wie Mathematik, Rechtswissenschaften, und diversen Ingenieursstudiengängen. Unter den 24 Berufstätigen befanden sich 8 Ingenieure und Teilnehmer aus anderen technischen Berufen, wie Kundenberater und Programmierer. Etwa 41% der Befragten setzen Microsoft Excel als Tabellenkalkulation vorwiegend privat ein. 20 von 24 Berufstätigen bzw. 2 von 5 Studierenden setzen die Software hauptsächlich im Beruf bzw. im Studium ein.

Als Sprache für den Prototypen wird Englisch eingesetzt. Zum einen haben wir größere Mengen an Daten erhofft, zum anderen wurde die englische Sprache im Vergleich zu Deutsch als einfacher eingestuft. Die Teilnehmer gaben als Muttersprache 36 Mal Deutsch, 20 Mal Englisch und 1 Mal Türkisch an. Alle Lösungen der Teilnehmer sollten in englischer Sprache erfolgen. Die Nutzungsdauer der Teilnehmer mit Microsoft Excel als Tabellenkalkulation liegt zwischen einem und 25 Jahren.

Des Weiteren entspricht der Kenntnisstand der Teilnehmer im Umgang mit der Tabellenkalkulation Excel als Software laut der Selbsteinschätzung zu 16% (9 Personen) dem eines Anfängers, zu 72% (41 Personen) eines Fortgeschrittenen und zu 12% (7 Personen) eines Experten.

Die durchschnittliche Erfahrung liegt bei 9,5 Jahren, wobei fast 95% der Befragten die Software seit mindestens 5 Jahren verwenden. Dabei arbeiten 70% der Teilnehmer mindestens einmal pro Woche damit und lediglich 5% der Teilnehmer nutzen die Software nicht. Diese Ergebnisse zeigen, dass unter den Teilnehmern im Querschnitt eine solide Grundkenntnis und eine gewisse Vertrautheit mit der Software zu erwarten sind. Aufgrund der häufigen Benutzung durch einen Großteil der Befragten kann davon ausgegangen werden, dass diese einfache Aufgaben lösen können.

2.2.3 Ergebnisse

Abbildung 2.1 zeigt eine Notentabelle aus dem ersten Experiment. Dabei enthält die erste Spalte die Namen der Studenten, gefolgt von vier Spalten mit den jeweiligen Noten sowie einer Spalte für den Durchschnitt als Endnote.

Das Problem, zu dem die Teilnehmer eine Lösung suchen, ist:

The task is to calculate the average for Marcel and to write it to F4.

Die natürlichsprachlichen Formulierungen der Teilnehmer entsprechen dem angegebenen Erfahrungslevel. Die Anfänger verlangen in ihren Formulierungen nach Hilfe, ohne konkrete Information zur Lösung des Problems bereitzustellen:

- I need to have a cell that averages the students grades, how would I do that?
- Friend, will you help me figure out the formula for this problem?

Erfahrene Benutzer gaben mehr Informationen in der Formulierung an:

- How can I form the arithmetic average value of an arbitrary amount of values?
- How do I get this cell to show the average of Marcel's grades?
- Take every grade in the row which is named Marcel and calculate the arithmetic mean of it and write the result into cell F4.

Nicht zuletzt stellten die Experten eine komplette Beschreibung für deren Problem dar:

- Could you insert the average value of Daily grades 1, Daily grades 2, Daily grades 3, Final exam into Average?
- Please put the average of row 4 column B to E in F4.

Die Teilnehmer hatten Schwierigkeiten mit einzelnen Funktionsbereichen von Excel. Wenn die Teilnehmer die Aufgabe nicht lösen konnten, wollten sie die Möglichkeit haben, nach Hilfe zu fragen. Über alle Aufgaben hinweg zeigen die Ergebnisse, dass 17% der Experten, 80% der Erfahrenen und 100% der Anfänger eine Hilfestellung in mindestens einer Aufgabe benötigt haben. In den zwölf Aufgabestellungen haben sich im Durchschnitt die Experten bei 1,5 Aufgaben, die Fortgeschrittene bei 3,5 und die Anfänger bei sechs Aufgaben eine Hilfestellung durch den digitalen Assistenten gewünscht.

Die Ergebnisse des Experiments zeigen in [WWVT15] die Notwendigkeit eines interaktiven Dialogsystems, das insbesondere unerfahrene Nutzer sukzessive bei der Lösungsfindung unstützt. Denn die Eingaben *I need to have a cell that averages the students grades, how would I do that?* verlangt nach einem Dialog, der Schritt-für-Schritt die fehlende Information beim Benutzer abfragt und eine korrekte Lösung für die Anfrage bereitstellt:

- Die Anfänger formulierten häufig lediglich die Aufgaben in eine Frage um (*I need to have a cell that averages the students grades, how would I do that?*). Keine ihrer angebotenen Lösungsbeschreibungen reichte aus, um das Problem direkt zu lösen.
- Die Fortgeschrittene Benutzer stellten dagegen mehr Information für das System zur Verfügung, indem sie Teilschritte zur Bearbeitung der Aufgabe angaben. Meist fehlten jedoch Informationen, wie die Angabe der Operanden, um das Problem vollständig zu lösen (*I need to calculate the average of a row of values into a cell.*). Nur in 20% der Fälle waren diese ausreichend.
- Die Beschreibungen von Probanden, die sich selber als Experten einstufen, waren hingegen ausführlich formuliert, so dass das Problem direkt gelöst werden konnte (*Could you insert the average value of Daily grades 1, Daily grades 2 and Final exam into Average?*). Nur 13% ihrer Beschreibungen eigneten sich nicht zur vollständigen Lösung des Problems.

2.3 Erzeugung neuer Funktionen

Im zweiten Experiment definieren die Benutzer eigene Berechnung in natürlicher Sprache, ohne diese explizit als Formel in Excel anzugeben [Wie14].

2.3.1 Aufbau

Motiviert durch die schwierige Erweiterbarkeit der gegebenen Softwaresysteme implementierten wir eine Schnittstelle zur automatischen Erzeugung neuer Muster oder Formeln. Diese Schnittstelle ermöglicht es dem Benutzer, die neue Formel in natürlicher Sprache zu beschreiben. Wir haben einen endlichen Automaten mit vier Zuständen implementiert:

- *Allgemeine Anfrage:* Im ersten Schritt stellt der Benutzer die Anforderung bereit zum Anlegen einer neuen Formel im Dialogsystem. Der Prototyp geht in den ersten Zustand über und erwartet Benutzereingaben für eine benutzerdefinierte Formel, um das angegebene Muster in der Datenbank zu speichern.
- *Namensanfrage:* eine benutzerdefinierte Formel kann nicht verwendet werden, wenn diese keinen eindeutigen Namen hat. In diesem Schritt gleicht der Prototyp die bereits vorhandenen Datenbankeinträge auf den neuen Namen ab.
- *Formelanforderung.* Benutzer geben eine komplexe Formel in mehreren Eingaben ein und nutzen implizite sowie ausdrückliche Verweise auf die vorhergehenden Teile der Formel.
- *Eingabe wiederverwenden.* Wenn die Benutzereingabe einer Formel abgeschlossen ist, wird die Formel in der Datenbank gespeichert. Dazu muss ein Benutzer für die neue Formel einen bisher nicht verwendeten Namen angeben und kann diese Formel unter dem Namen wiederverwenden.

2.3.2 Teilnehmer

An diesem Experiment nahmen zehn Teilnehmer teil, von denen die Hälfte an den bisherigen Experimenten nicht teilgenommen hat. Sechs von ihnen betrachteten sich als Anfänger, drei als fortgeschrittene Benutzer und nur einer als Experten für Tabellenkalkulationen. Die Teilnehmer sollten mithilfe des Dialogsystems eine neue Berechnung in natürlicher Sprache formulieren. Die Formulierung erzeugt eine neue Formel, die als Muster die vorhandene Funktionalität des Gesamtsystems erweitert. In einem von drei vorgegebenen Szenarien sollte ein Benutzer anhand der Zahlen in der Übersicht das Jahr berechnen, in dem ein Unternehmen mit dem Geldverdienen beginnt:

Jahr	Einnahmen	Ausgaben	Differenz
0	5	0	
1	1	1	
2	2	5	
3	3	7	

Tabelle 2.1: Auswertung der Kommunikation im ersten Experiment

Basierend auf den Werten in der Tabelle erfolgt die Ableitung der Formel für die zukünftige Einkommenssituation:

$$f(x) = x^2 - 6x + 5$$

Der ersten Nullpunkt ist bereits in Abbildung 2.2 mit dem Wert 1 angegeben.

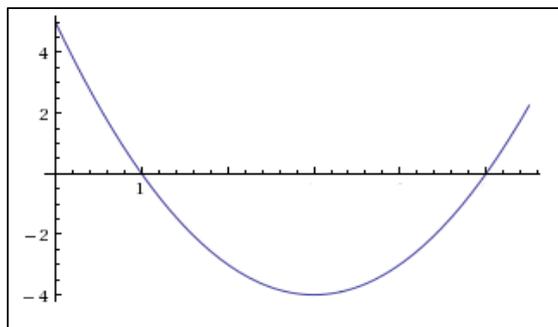


Abbildung 2.2: Funktion zur Aufgabestellung von WolframAlpha

Im nächsten Schritt berechnen die Teilnehmer den zweiten Nullpunkt mithilfe der quadratischen Gleichung, die als Mitternachtsformel bezeichnet wird:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Beispiel für einen Eingabedialog:

```

System:      How can I help you?

Benutzer:    I want to create a new formula.

System:      I created a new pattern for you.
              Would you mind telling me the name of your pattern?

Benutzer:    Rootfunc

System:      I named your pattern ROOTFUNC.

Benutzer:    Square b

System:      I found the formula SQUARE(b) and added it to your pattern.
              Your pattern called ROOTFUNC which uses the formula SQUARE(b)
              seems to be complete. Should I save it now?

Benutzer:    Subtract the product of 4 , a and c from it

System:      I added (SQUARE(b)-PRODUCT(4;a;c)) to your pattern

Benutzer:    Root it

System:      I added ROOT((SQUARE(b)-PRODUCT(4;a;c))) to your pattern

Benutzer:    Subtract b from this

System:      The pattern uses the formula (ROOT((SQUARE(b)-PRODUCT(4;a;c)))-b)
              Should I save it now?

Benutzer:    Divide it by the product of 2 and a

System:      I found the formula ((ROOT((SQUARE(b)-PRODUCT(4;a;c)))-b)/(2*a))
              and added it to your pattern. Shall I save your pattern?

Benutzer:    Yes

System:      Your pattern request was successfully saved.
              To use it you need to tell me the parameters like that:
              ROOTFUNC(b;a;c).
  
```

2.3.3 Ergebnisse

Die binomische Formel steht in Microsoft Excel nicht als eine Funktion zur Verfügung. Ziel ist es, eine neue Berechnung aus vorhandenen Formeln zu erstellen und dadurch die Funktionalität interaktiv zu erweitern. Nach einigen Interaktionen ist die Formel zu komplex und kann vom Benutzer ohne unseren Prototyp nicht einfach geschrieben werden. Nach dem Speichern der neuen Formel kann diese erfolgreich für weitere Berechnungen verwendet werden. Wie aus Tabelle 2.2 hervorgeht, reagierte das System interaktiv auf Benutzereingaben und löste ungefähr 85% der gestellten Aufgaben erfolgreich.

Insgesamt	
Anzahl der Aufgaben	120
Anzahl der Dialogiterationen	641
Erfolgreich gelöst	85,83 %
Gelöste Aufgaben	
Anfrage neuer Formel	100 %
Benennung einer Formel	100 %
Eingabe einer Formel	60 %
Speichern	83,3 %

Tabelle 2.2: Auswertung der Kommunikation im zweiten Experiment

Die Notwendigkeit des Experiments erklärt sich in der Sammlung von Erkenntnissen zu einem Ablauf einer natürlichsprachlichen Benutzerprogrammierung. Eingeschränkt auf die Domäne der mathematischen Formeln geht aus diesem Experiment hervor, dass eine weitere Anforderung an das Dialogsystem besteht. Neben der Reaktion auf die Benutzeranfragen mit einer passenden Antworten, muss das Dialogsystem für Benutzerprogrammierung *zielgerichtet eine Erweiterung der vorhandenen Funktionalität* unterstützen.

2.4 Schlussfolgerung

Zusammenfassend liefern die zwei Experimente erste Erkenntnisse für eine natürlichsprachliche Schnittstelle [WWT15]. Das erste Experiment liefert in einer Simulation die Erkenntnisse, dass Benutzer mit wenig Erfahrung nicht genug Information einem natürlichsprachlichen Assistenten zur Verfügung geben, um eine gültige Ausführung einer Aktion zu ermöglichen. Alle Teilnehmer, die sich selbst als Anfänger bezeichneten, haben bei mindestens einer Aufgabe Hilfe benötigt, im Schnitt sogar bei jeder zweiten Fragestellung. Immer noch 80% der fortgeschrittenen Teilnehmer benötigten mindestens einmal Hilfe, würden jedoch im Schnitt bei nur 3,5 Aufgaben Unterstützung begrüßen. Unter den Experten benötigte nur ein Teilnehmer Hilfe.

Aus diesem Grund haben wir im zweiten Experiment ein *musterbasiertes Dialogsystem* untersucht, das bei den Benutzern nach fehlenden Informationen fragen kann. Die Interaktion beschränkt sich jedoch aufgrund der benötigten Muster in der Anzahl der möglichen Formulierungen eines Benutzern. Aus diesem Grund wird eine *flexible Erkennungsmethode der Benutzereingabe* benötigt.

Weitere Anforderungen an eine natürlichsprachliche Schnittstelle sind [Voi14]:

- (a) *Auflösung der Rückbezüge* auf bereits mitgeteilte Information:
zum Beispiel stellt das *it* in *Please add 4 to A1. Then multiply it by 7.* einen Rückbezug auf das Ergebnis der ersten Eingabe dar
sowie
- (b) *Auflösung der Mehrdeutigkeiten* in Benutzereingaben:
zum Beispiel wird die natürlichsprachliche Eingabe *Divide A1 by 3 multiplied by A3* entweder als $(A1/(3*A3))$ oder $((A1/3)*A3)$ interpretiert.

Im letzten Experiment muss für eine natürlichsprachliche Benutzerprogrammierung *zielgerichtet* bei der Erweiterung vorhandener Funktionalität unterstützt werden. Aufbauend auf den Erkenntnissen der Pilotstudie bleibt im Folgenden zu untersuchen, ob dazu bereits vorhandene Lösungen existieren.

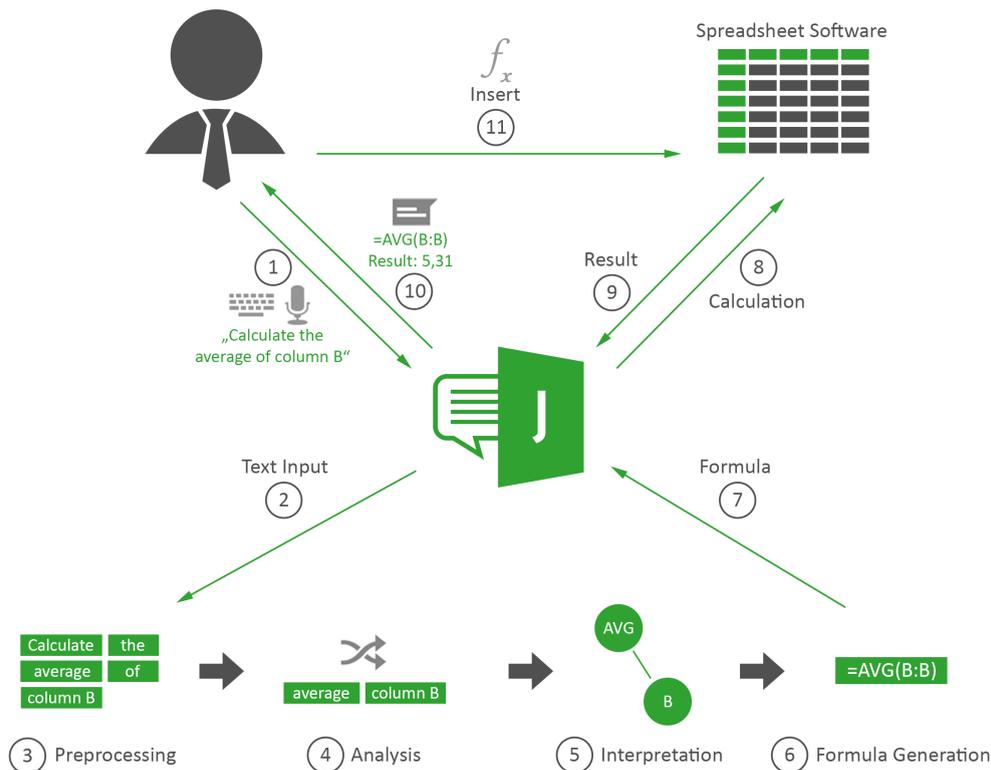


Abbildung 2.3: Verarbeitung einer Benutzereingabe nach [Voi14]

Abbildung 2.3 stellt eine allgemeine Verarbeitung einer Benutzereingabe in einer Tabellenkalkulation dar: (1) ein Benutzer gibt die Anweisung in Textform ein, (2) das System leitet diese zur Verarbeitung an eine Verarbeitungskette weiter. Um die Eingabe für die folgenden Schritte vorzubereiten, werden zunächst im Vorverarbeitungsschritt (3) überflüssige Leer- und Satzzeichen entfernt und die Eingabe in einzelne Worte aufgeteilt. (4) Anschließend werden diese Worte analysiert, Schlüsselwörter und deren Bedeutungen identifiziert, Synonyme in eine einheitliche Formulierung übersetzt sowie der Kontext entsprechend vereinheitlicht. Das Analyseergebnis wird an den Interpretationsschritt (5) weitergegeben, in dem der Zusammenhang zwischen den einzelnen Schlüsselwörtern hergestellt und diese entsprechend in einer Datenstruktur repräsentiert wird. Schließlich wird die erzeugte Datenstruktur in eine Formelsprache übersetzt (8) und im Anschluss auf den Daten in

der Tabellenkalkulation ausgeführt, so dass dem Benutzer sowohl die Formel als auch das Ergebnis angezeigt werden kann (9).

Die Benutzer müssen durch die Benutzung eines Dialogsystems die Fachdomäne des Zielsystems nicht kennen und brauchen keine Anleitung, um mit der Kommunikation zu beginnen. Sie können mittels natürlicher Sprache ihren Wunsch äußern und das Dialogsystem kann bei Unklarheiten nachfragen. Bereits durch die Unterbeantwortung und gemischten Initiative wird die Fähigkeit der natürlichsprachlichen Schnittstelle benötigt, adäquate Rückfragen zu stellen. Beispielsweise muss die natürlichsprachliche Schnittstelle in der Lage sein, eine Fehlerbehandlung durchzuführen, um zum Beispiel bei Rechtschreibfehlern bei einem Benutzer rückzufragen. Ebenso ist es durch die Mehrdeutigkeit der natürlichen Sprache oft schwer, mathematische Ausdrücke zweifelsfrei zu beschreiben. Beispielsweise enthält die Auswertung der Aussage *Calculate 5 divided by 3 times 2.* eine Mehrdeutigkeit, die aufgelöst werden muss (Siehe Abbildung 2.4):

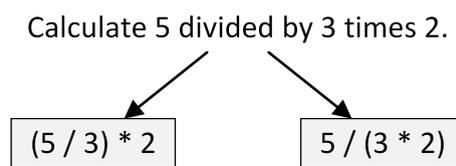


Abbildung 2.4: Auflösung der Mehrdeutigkeiten nach [Voi14]

Die Natürlichsprachlichkeit des Dialogsystems soll im Vordergrund stehen. Dies bedeutet, dass die Schnittstelle sämtliche Formulierungen eines Benutzers in natürlicher Sprache erlaubt. Des Weiteren soll statt einer Ausgabe über einen Misserfolg, ein Ansatz zur Weiterverarbeitung gefunden werden. Dadurch erfolgt eine Annäherung in der Bearbeitung des Anliegens des Benutzers.

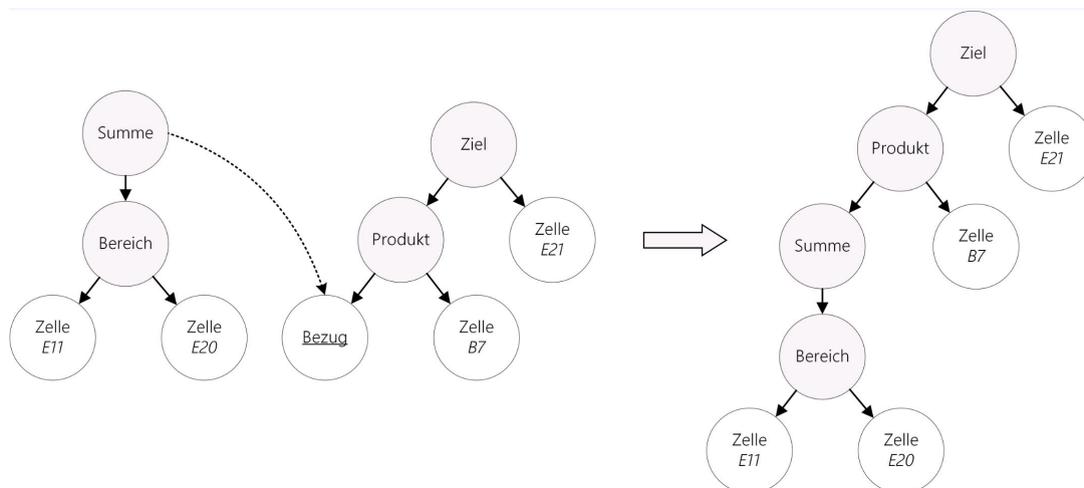


Abbildung 2.5: Auflösung der Rückbezüge in einer Benutzereingabe nach [Voi14]

Ein mächtiges sprachliches Mittel im Dialog zwischen Menschen ist es, sich auf zuvor Gesagtes zu beziehen, ohne dieselbe Information erneut zu nennen. Durch Rückbezüge (Siehe Abbildung 2.5) sollen im Rahmen dieser Arbeit zuvor erwähnte Zellen, Zeilen, Spalten, Tabellen und andere Bezeichnungen sowie zuvor berechnete Zwischenergebnisse und verwendete Zielzellen zugreifbar werden. Indem das System in der Lage ist, Schlüsselwörter wie „this result“, „that sum“, „those numbers“ oder schlicht „it“ auf zuvor Gesagtes zu beziehen und abhängig vom erwarteten Typ die entsprechende Angabe vorheriger Eingaben

weiterzuverwenden, soll ein Benutzer das Gefühl erhalten, mit JustLingo tatsächlich ein Gespräch zu führen.

Um neue Funktionalität im Dialog mitteilen zu können, müssen Algorithmen aus der natürlichen Sprache, die eine bestimmte Funktionalität ausführen, extrahiert werden. Die Benutzer sollen Algorithmen (Vgl. [CLLS09]) in der natürlichen Sprache beschreiben und eine gültige Ausgabe durch das Dialogsystem für eine gegebene Beschreibung erhalten. Ein Algorithmus besteht in der Regel aus Anweisungen, Schleifen sowie Bedingungen. Die Funktionalität soll sich an Benutzer ohne Programmierkenntnisse richten und einfache Routinen ohne Vorkenntnisse programmiert werden.

3. Verwandte Arbeiten

Dieses Kapitel diskutiert die verwandten Arbeiten und vergleicht die bereits vorhandenen Lösungen. Die im Kapitel 2 vorgestellte Pilotstudie dokumentiert das Benutzerverhalten im Umgang mit einem fiktiven Assistenten und stellt die Notwendigkeit einer interaktiven Benutzerschnittstelle fest. Insbesondere Benutzer mit wenig Erfahrung benötigen eine Schritt-für-Schritt Unterstützung. Aus diesem Grund beleuchtet Kapitel 3.2 im Allgemeinen den Aufbau von Sprachsystemen und konzentriert sich auf den Verarbeitungsstufen der Sprachverarbeitung, des Sprachverstehens sowie des Dialogmanagements. Kapitel 3.2.1 beleuchtet dabei die Sprachverarbeitung einer natürlichsprachlichen Eingabe. Das Sprachverstehen wird in Kapitel 3.2.2 präsentiert. Das Dialogmanagement als eine zentrale Komponente einer natürlichsprachlichen Schnittstelle mit deren Funktionsweise wird im Kapitel 3.2.3 besprochen. Unter anderem betrachtet das Kapitel einige Plattformen, die eine Dialogmodellierung für natürlichsprachliche Assistenten ohne eine Programmierung ermöglichen. Kapitel 3.2.4 beleuchtet die Sprachgenerierung und unterteilt die Lösungen mit einem abrufbasierten und generativen Antwortverhalten. Die Forschungsarbeiten im Bereich der Programmierung in natürlicher Sprache werden im Kapitel 3.3 beleuchtet. Kapitel 3.4 diskutiert den Stand der Forschung in der Benutzerprogrammierung.

3.1 Zweck

Insbesondere sind die bekannten Lösungsansätze für die Zielsetzung der Dissertation entscheidend. Dabei soll festgestellt werden, ob bei den vorhandenen Lösungen noch Lücken existieren, die eine Forschungsarbeit notwendig machen. Das Ziel des Kapitels ist es, nach der Diskussion für die noch nicht gelöste Probleme die jeweiligen Forschungsfragen zu formulieren. Kapitel 3.5 formuliert die Forschungsfragen der Dissertation.

3.2 Architektur eines Sprachsystems

Ein allgemeines Sprachsystem beinhaltet die folgenden Stufen [Car17]:

- Stufe 1: Automatic Speech Recognition, ASR, übernimmt die Wahrnehmung der Spracheingabe des Endnutzers und wandelt eine gesprochene Sprache zu einem geschriebenen Text um.
- Stufe 2: Natural Language Processing, NLP, unterstützt durch eine Vorverarbeitung der Sätze durch Zerteilung in Wörter und deren Rückführung auf eine Grundform sowie einem Aufbau der Abhängigkeitsbäume.
- Stufe 3: Natural Language Understanding, NLU, ist für das Sprachverstehen der geschriebenen Eingabe zuständig.
- Stufe 4: Dialog Management, DM, ist für den Aufbau einer Konversation, notwendiger Fragestellung an den Benutzer sowie eine Aktionsausführung des Dialogsystems zuständig.
- Stufe 5: Natural Language Generation, NLG, erzeugt eine natürlichsprachliche Ausgabe für den Benutzer.
- Stufe 6: Text To Speech, TTS, ist für eine Sprachausgabe zuständig.

Im weiteren Verlauf wird die Verarbeitung einer geschriebenen Benutzereingabe betrachtet. Daher sind die erste und sechste Stufe nicht relevant.

3.2.1 Sprachverarbeitung

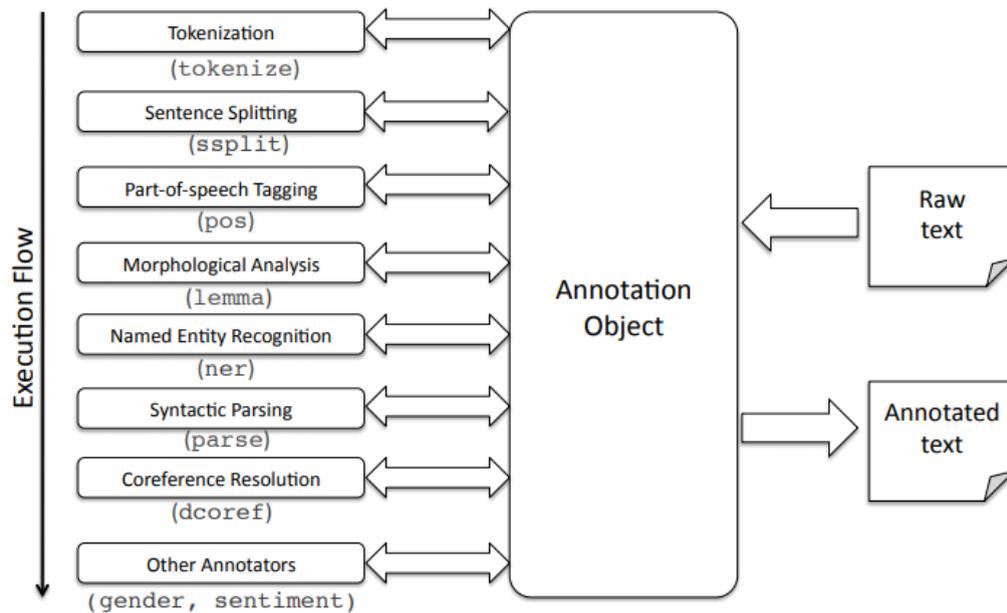
Im Folgenden wird die Sprachverarbeitung als zweite Stufe vorgestellt und vorhandene NLP-Werkzeuge beleuchtet. Insbesondere soll im Bereich der Sprachverarbeitung untersucht werden, ob es ausreichend erforscht ist und die vorhandenen Werkzeuge eine natürlichsprachliche Benutzerprogrammierung ermöglichen würden.

3.2.1.1 Stanford CoreNLP

Stanford CoreNLP Toolkit [MSB⁺14] wurde im Jahr 2010 veröffentlicht und ist eine auf Java basierte Open-Source Software, um eine Sprachverarbeitung durchzuführen. Abbildung 3.1 präsentiert die grundlegende Architektur. Der zu bearbeitende Text wird als *Annotation Object* gespeichert und auf diesem die einzelnen Tools ausgeführt. Die Ausgabe kann als Text oder als XML erfolgen.

Dabei beinhaltet das Stanford CoreNLP Toolkit eine Sammlung von Tools:

- *Tokenisierung*: die Segmentierung einer geschriebenen Eingabe in einzelne Bestandteile
- *Part-of-speech-Tagging, POS*: die Zuordnung von Wortarten zu den einzelnen Wörtern einer geschriebenen Eingabe
- *Lemmatisierung, Lemma*: die Rückführung eines Wortes auf die Grundform basierend auf einem Wörterbuch
- *Eigennamenerkennung (Named-Entity recognition, NER)*: die automatische Erkennung von Eigennamen als eine Folge von Wörtern
- *Koreferenzen*: die Auflösung der Referenzen zwischen einem Bezugswort und einer Anapher.
- *Abhängigkeitsbäume*: aufbauend auf den POS-Tagging und Koreferenzen erfolgt ein Aufbau von Abhängigkeitsräume

Abbildung 3.1: Architektur von CoreNLP nach [MSB⁺14]

Für den Aufbau der Pipeline müssen die einzelnen Tools konfiguriert werden. Quellcode 3.1 zeigt ein Beispiel, bei dem die Tools als Eigenschaften eine Pipeline initialisieren. Als nächstes wird der gewünschte Text an die Pipeline zur Bearbeitung übergeben.

Quelltextausschnitt 3.1: Bearbeitung einer Zeichenkette durch das Stanford CoreNLP

```
Properties p = new Properties();
p.setProperty (
    "annotators",
    "tokenize, ssplit, pos, lemma, ner, parse, dcoref"
);

StanfordCoreNLP pipeline = new StanfordCoreNLP(p);

String text = "Karlsruhe is the best place in Germany.";
Annotation ann = new Annotation(text);
pipeline.annotate(ann);
```

Abbildung 3.2 zeigt eine POS-Annotation durch die Stanford CoreNLP Webanwendung¹: NNP bezeichnet ein Substantiv mit einem Eigennamen (Engl.: Proper Noun), VBZ markiert ein Verb in dritter Person Präsens, DT wird als Determinator den Artikeln zugeordnet, JJS markiert Komparative, NN steht für ein generelles Substantiv (Engl.: Common Noun) und IN markiert Präpositionen und untergeordnete Konjunktionen.

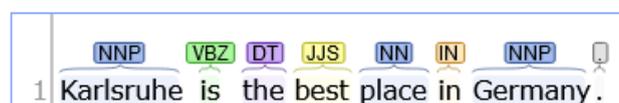


Abbildung 3.2: Durch POS annotierte Zeichenkette

¹Stanford CoreNLP Webanwendung, <https://corenlp.run/>, letzter Aufruf: 24. Juli 2023

Die erkannten Eigennamen werden in Abbildung 3.3 dargestellt. Dabei wurden Germany als Country und Karlsruhe als Organization erkannt.

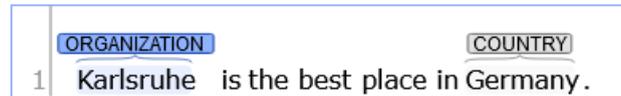


Abbildung 3.3: Eigennamenerkennung durch die NER-Analyse

Abbildung 3.4 zeigt die Abhängigkeiten durch die Pfeile zwischen den Tags an: nsubj (nominal subject) zeigt eine Abhängigkeit zu Nominalphrasen, cop (copula) zeigt eine Abhängigkeit zu einem Funktionswort, um einen Subjekt mit einem nonverbalen Prädikat zu verknüpfen, det (determiner) zeigt auf die Beziehung zu einem Artikel, amod (adjectival modifier) zeigt die Bedeutung zum Substantiv, nmod (nominal modifier) zeigt auf eine Abhängigkeit eines Substantivs oder einer Nominalphrase, case (case marking) zeigt auf eine Beziehung einer Präposition, und punct (punctuation) zeigt auf eine Satztrennung.

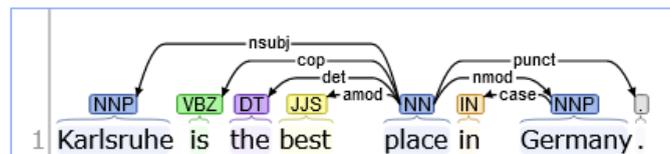


Abbildung 3.4: Abhängigkeitsbaum

3.2.1.2 Natural Language Toolkit

Natural Language Toolkit, kurz NLTK, ist eine Open Source Software, die auf Python basiert und Werkzeuge zur Verarbeitung natürlicher Sprache zur Verfügung stellt [BKL09]. Ähnlich zu CoreNLP unterstützt NLTK beim Parsen, Tokenisieren, Lemmatisieren sowie bei der Wortartbestimmung, Eigennamenerkennung, syntaktischer und semantischer Analyse. NLTK bietet Schnittstellen, um Korporas, die Sammlungen von Audioaufnahmen mit geschriebenen Texten beinhalten, und lexikalische Ressourcen, wie WordNet, zu verwenden.

Quellcode 3.2 zeigt ein Beispiel für die Verwendung von NLTK. Durch *word_tokenize* erfolgt die Satzteilung, auf die Wörter im Folgenden durch *pos_tag* die Wortarterkennung durchgeführt wird. Das Ergebnis ist eine annotierte Liste mit Wörtern und deren Bedeutung.

Quelltextausschnitt 3.2: Tokenisierung und Part-Of-Speech Tagging mit NLTK

```
import nltk

sentence= "Karlsruhe is the best place in Germany."

tokens = nltk.word_tokenize(sentence, language='english')
print(tokens)

['Karlsruhe', 'is', 'the', 'best', 'place', 'in', 'Germany', '.']

tags = nltk.pos_tag(tokens)
print(tags)

[('Karlsruhe', 'NNP'), ('is', 'VBZ'), ('the', 'DT'), ('best', 'JJS'),
 ('place', 'NN'), ('in', 'IN'), ('Germany', 'NNP'), ('.', '.')]

```

3.2.2 Sprachverstehen

Im Folgenden werden Microsoft LUIS als eine cloudbasierte- und RASA als lokale, nicht-cloud Lösung für die dritte Stufe vorgestellt.

3.2.2.1 Microsoft LUIS

Language Understanding Intelligent Service (kurz: LUIS) ist ein von Microsoft entwickeltes Framework für die Verarbeitung und Interpretation natürlicher Sprachen [Mic20a]. Es ermöglicht dem Entwickler, das Sprachverständnis (NLU) einer Benutzereingabe an Microsoft auszulagern. Dies wird durch einen domänenspezifischen Dienst erreicht, der Interpretationen bestimmter Eingaben in natürlicher Sprache abrufen. Dabei hat LUIS folgende Bestandteile:

- Absicht (Engl.: Intent):
Das Ziel von LUIS ist es, zu einer Benutzereingabe eine Absicht zuzuordnen. Das errechnete Ergebnis ist eine Wahrscheinlichkeit für die jeweilige Absicht. Die Zuordnung basiert auf den Äußerungen.
- Äußerung (Engl.: Utterance)
Die Äußerungen stellen möglichen Benutzereingaben dar, die Entwickler an LUIS pro Absicht entweder per JSON-Import oder manuell über das LUIS-Portal bereitstellen. Anhand dieser Äußerungen wird das Sprachmodell in LUIS trainiert. Es ist zur Laufzeit möglich, weitere Äußerungen bereitzustellen, um das Modell weiter zu verbessern.
- Entität (Engl.: Entity)
Entitäten sind Parameter für die jeweiligen Absichten, wie zum Beispiel die Stadt bei der Buchung eines Fluges. Absichten können mehrere Entitäten haben - entweder benutzerdefinierte oder vordefinierte Entitäten.

Der LUIS-Dienst wird in der Regel mit einem Chatbot-Projekt des Entwicklers in Visual Studio verbunden, so dass die Zuordnung zu einer Absicht mit dem Bot Framework Simulator direkt bei der Implementierung überprüft werden kann. Somit hilft LUIS bei dem Sprachverstehen und der Zuordnung einer Benutzereingabe zu einer Absicht. LUIS bietet kein Dialogmanagement an. Ein Programmierer ist bei der Dialogmodellierung auf sich gestellt [RT18].

3.2.2.2 RASA

RASA [Ras20] ist ein Open-Source-Framework für die Entwicklung von Chatbots. Es kann offline in einer lokalen Installation, als Docker Container oder in einem Kubernetes Cluster verwendet werden.

RASA NLU stellt einen Klassifikator von RASA dar. Es ermöglicht eine Definition von Absichten und Äußerungen sowie ein lokales Trainieren des Sprachmodells. RASA NLU verfolgt eine Mehrklassenauswahl, bei der die Summe der Wahrscheinlichkeit der Klassifikation genau 1 ergibt.

Abbildung 3.5 zeigt eine webbasierte Schnittstelle von RASA, die eine Verwaltung von Komponenten anbietet. Unter anderem können Absichten und Äußerungen grafisch definiert, verschiedene Trainingsmodelle verwaltet und Gesprächsverläufe angezeigt werden. *Stories* und *Antworten* definieren einfache Gesprächsabläufe. RASA erinnert an Microsoft LUIS mit dem Unterschied, dass LUIS eine Online-Plattform für die Spracherkennung anbietet.

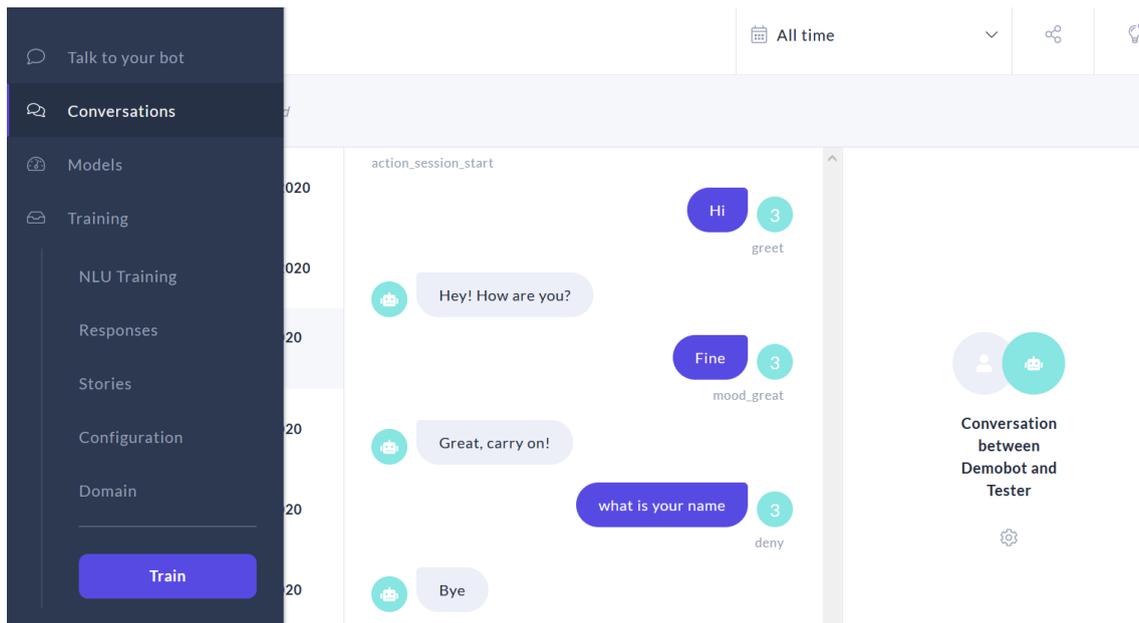


Abbildung 3.5: Gesprächsverläufe in RASA

3.2.3 Dialogmanagement

Neben den Sprachassistenten wurden bereits einige Plattformen für die Erstellung eines Dialogs entwickelt [KFM⁺18]. Diese Plattformen werden als *Conversational Computing Plattformen*, kurz CCP, bezeichnet. Die Funktionalität der zweiten Stufe der Sprachverarbeitung und der dritten Stufe einer Spracherkennung stellen CCPs zur Verfügung, so dass ein Benutzer ein Dialogverhalten durch die Bereitstellung der Eingabe und Ausgabebeispiele modellieren kann. Die bekannteste CCPs werden im Folgenden vorgestellt.

3.2.3.1 Microsoft Power Virtual Agents

Im Jahr 2019 stellte Microsoft die Power Virtual Agents (kurz: PVA) vor, welche die Spracherkennung von Microsoft LUIS verwenden und eine graphische Dialogmodellierung für einen Chatbot ermöglichen [Mic20b].

Abbildung 3.6 zeigt die Plattform. Im Unterschied zu LUIS wird eine Absicht bei PVA als Thema (Engl.: Topic) bezeichnet und zwischen System- und Benutzerthemen unterschieden. Die Systemthemen (Begrüßung, Bestätigung über Erfolg/Misserfolg, Eskalation an einen Menschen, Neustart der Konversation und Verabschiedung) sind bereits vordefiniert und können mit den gewünschten Äußerungen ergänzt werden. Dagegen müssen die Benutzerthemen für die Verwendung im Chatbot erst definiert werden. Bei der Erstellung gibt ein Benutzer spezifische Triggerausdrücke (bei Microsoft LUIS auch Äußerungen genannt), die das Thema bei einer Eingabe aktivieren - dieser Vorgang ähnelt der Erstellung von Absichten bei Microsoft LUIS. Durch die Speicherung und Aktivierung eines Themas erfolgt das Training für die Spracherkennung bei Microsoft LUIS.

Das Konversationsverhalten des Chatbots für das jeweilige Thema kann durch einen Benutzer in der Canvasansicht eines Themas bereitgestellt werden. Abbildung 3.7 zeigt links die Navigation, daneben die Konversation mit dem Chatbot sowie rechts die Canvasansicht eines Themas.

Um einen Dialog mit dem Benutzer aufzubauen, lässt sich in einem Thema auf einen Triggerausdruck mit einer Frage oder einer Multiple-Choice Auswahl verknüpft mit einer Bedingung antworten. Die Antwort des Benutzers kann in einer Variable gespeichert und

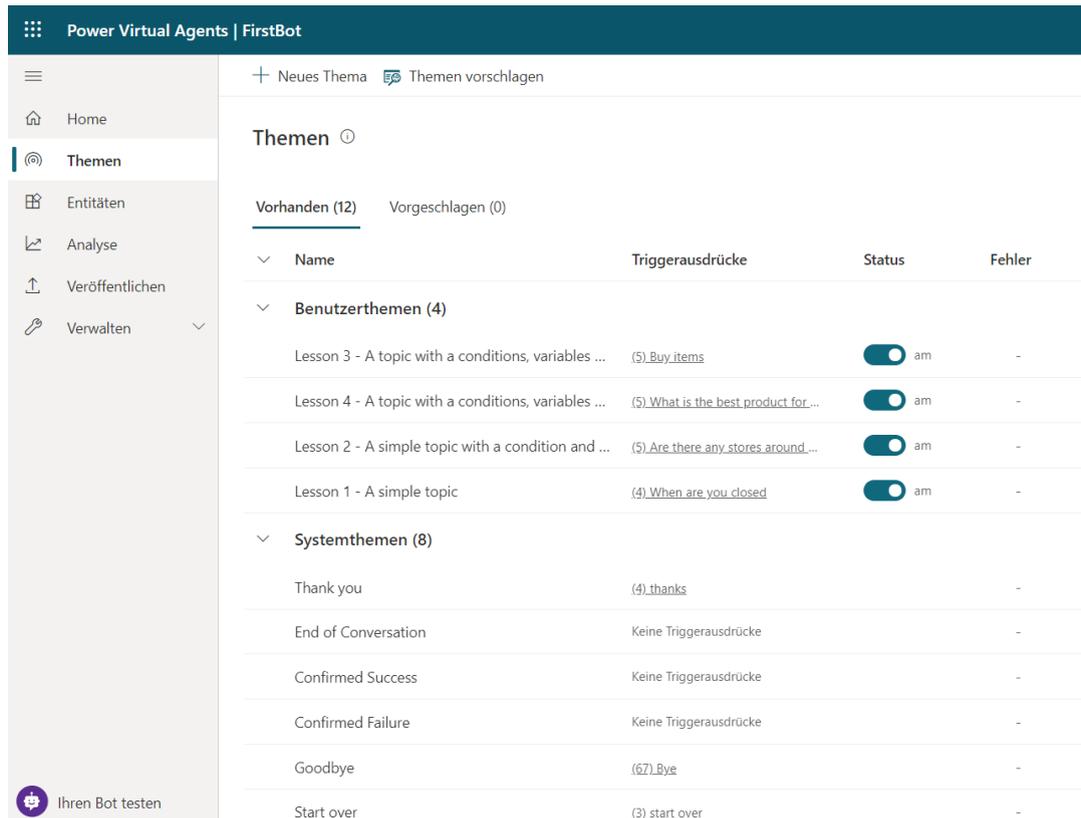


Abbildung 3.6: Auflistung der Themen bei Power Virtual Agent

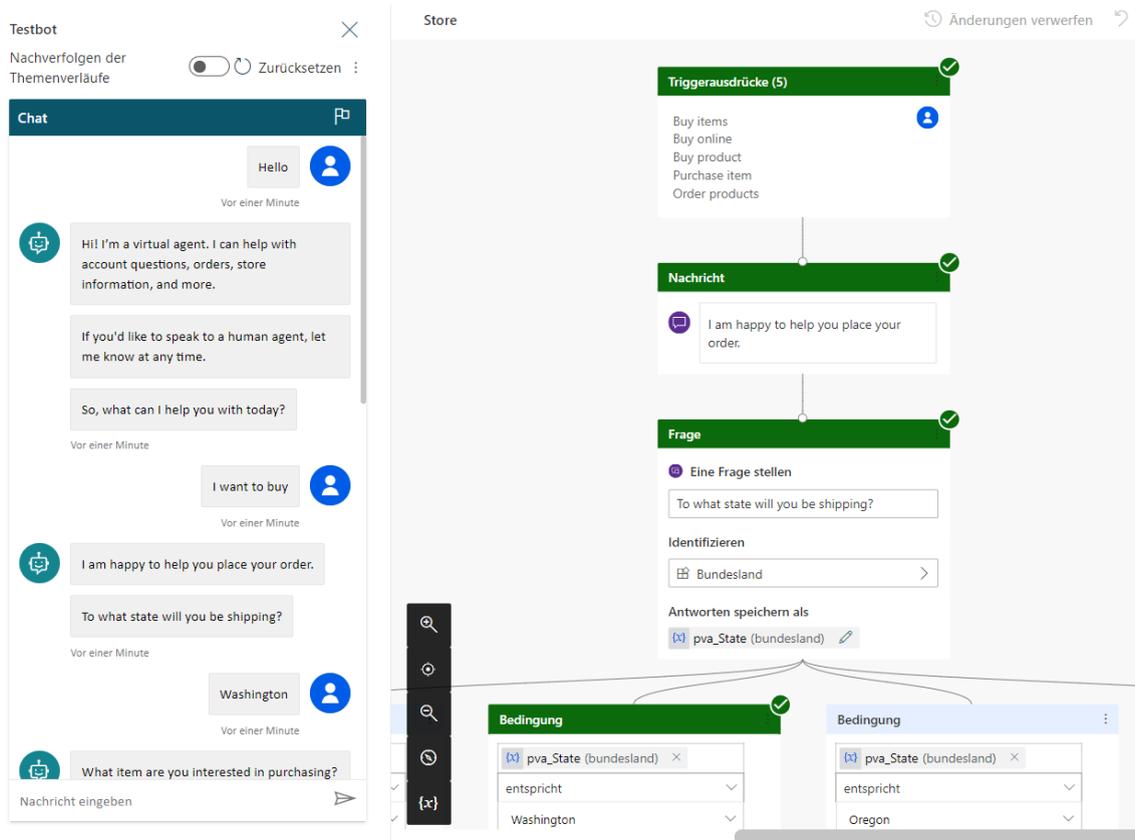


Abbildung 3.7: Power Virtual Agent mit einer graphische Erstellung einer Konversation

zum späteren Zeitpunkt im Dialog wiederverwendet werden. Bei Bedarf kann die Eingabe des Benutzers ein Ereignis auslösen und durch ein *Power Automate Flow* verarbeitet werden [Aut20]. Der erstellte Chatbot kann u.a. als Webseite oder in Microsoft Teams, Facebook, Slack bereitgestellt werden.

3.2.3.2 Amazon Alexa Voice Service

Von Amazon wird ein *Alexa Voice Service (AVS)* für die Unterhaltung mit dem Assistenten Alexa bereitgestellt [Ale20]. Mithilfe von Amazons Alexa Appkit ist es Entwicklern möglich, AVS in jedes Gerät kostenfrei zu integrieren und sich mit Alexa zu unterhalten. Dabei dient das Gerät selbst lediglich als Vermittler zum Clouddienst AVS, mit dem die Anfragen des Benutzers bei einer bestehenden Internetverbindung bearbeitet werden.

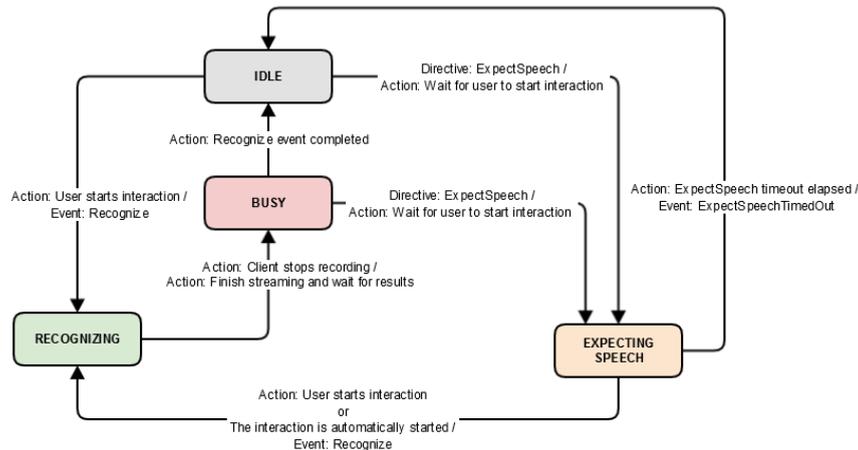


Abbildung 3.8: Zustände der AVS-Spracherkennung aus [Spe20]

Die Mikrophone eines Gerätes hören zu und warten auf ein Schlüsselwort, auch das *Wake Word* bezeichnet, um daraufhin eine Sprachaufnahme zu starten. Abbildung 3.8 zeigt das Zustandsdiagramm für die Spracherkennung [Spe20]. Dabei befindet sich der Spracherkennung im Zustand *Idle*, wenn keine Spracheingabe erfolgt. Sobald ein Benutzer interagiert, fängt die Sprachaufnahme im Zustand *Recognizing* an. Nach dem Ende der Aufnahme wechselt der Spracherkennung für die Verarbeitung in den Zustand *Busy*. Der Wechsel zum Zustand *Expecting Speech* erfolgt, wenn eine Eingabe vom Benutzer erwartet wird. Die Zustände werden durch Ereignissen unterstützt, z.B. *Recognize* für die Spracherkennung oder *ExpectSpeech* für das Fortsetzen einer Unterhaltung.

Bei AVS bilden die Fähigkeiten (Engl.: *Skills*) ein Grundgerüst, in denen die Absichten bereitgestellt werden. Wurde eine Absicht durch eine Äußerung in der Benutzereingabe klassifiziert, wird in AVS Lambda eine passende Antwort erstellt und über Alexa ausgegeben. AVS arbeitet nach dem sogenannten Slot-Filling Ansatz, in dem für eine erfolgreiche Ausführung alle Parameter vorhanden sein müssen. Bei einem fehlenden Parameter wird dieses vom AVS beim Benutzer nachgefragt. AVS beinhaltet durch die Amazon-Entwickler vordefinierte Skills, wie *Custom Interaction Model*, *Smart Home*, *Flash Briefing*, *Video*, *Music* und *List*. Eigene Skills können durch Absichten und Äußerungen definiert sowie im AVS Portal getestet werden (Siehe Abbildung 3.9).

3.2.3.3 Google DialogFlow CX

Die Implementierung von Google heißt *Dialogflow CX* [CX20] und stellt (i) eine Schnittstelle, die Sprachverständnis der Benutzereingaben in geschriebener Sprache ermöglicht,

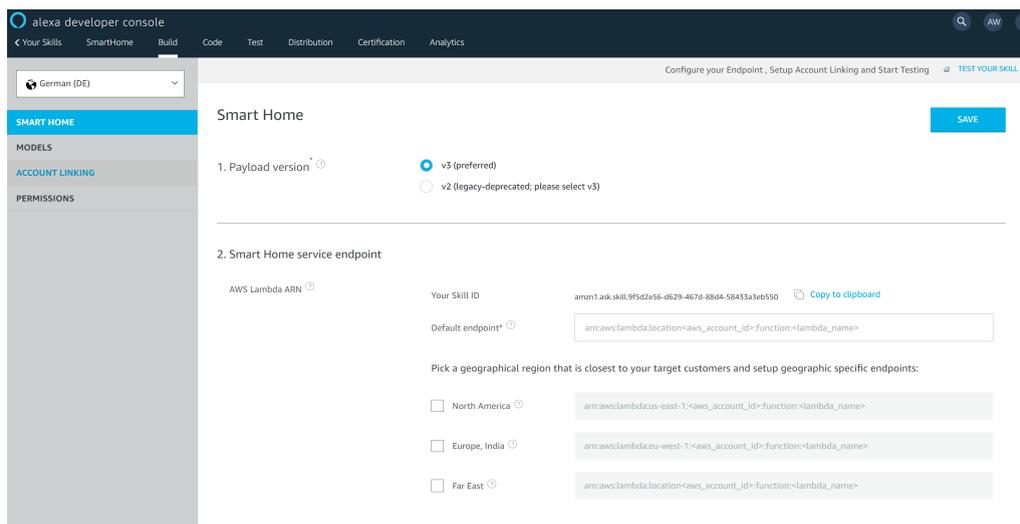


Abbildung 3.9: Alexa Developer Console

(ii) eine Modellierung von einfachen Dialogen sowie (iii) eine Integration des Agenten in externe Kommunikationskanäle zur Verfügung. DialogFlow CX ähnelt den PVAs aus dem Kapitel 3.2.3.1. Insbesondere kann bei PVA bzw. Microsoft LUIS eine Dialogmodellierung durch einen Entwickler im Bot Framework erfolgen, dagegen braucht ein Entwickler des Dialogs bei Dialogflow CX weniger Programmierkenntnisse. Die starre Dialogmodellierung erfolgt durch das Festlegen der Antworten, die wie die Äußerungen - bei Dialogflow auch Trainingsphrasen genannt - als ganze Sätze eingegeben werden. Bei der Klassifizierung einer Absicht zur Benutzereingabe werden diese Antworten unverändert ausgegeben. Das Vorgehen erinnert an unseren musterbasierten Ansatz des Parsers.

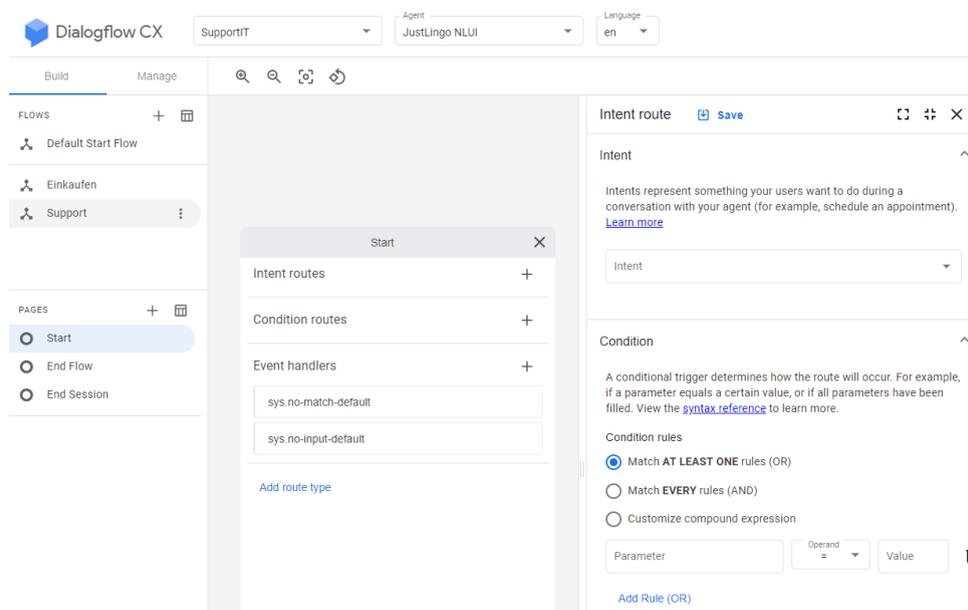


Abbildung 3.10: DialogFlow CX Plattform

Abbildung 3.10 zeigt die graphische Oberfläche von Google CX. Links werden dabei die Abläufe festgelegt. Im Arbeitsbereich in der Mitte können die Absichten festgelegt und für die Verwendung mit Phrasen und Bedingungen für die Aktivierung konfiguriert sowie die Ereignisse und deren Behandler definiert werden.

3.2.3.4 IBM Watson Assistant

Ähnlich zu den vorher vorgestellten Plattformen stellt IBM Watson Assistant [Ass20] eine Plattform für eine Bereitstellung der Chatbots dar. Nach einer Definition von Fähigkeiten mit den jeweiligen Absichten und Entitäten können beim Dialog die einzelnen Absichten in eine Hierarchie angeordnet werden. Abbildung 3.11 zeigt ein Beispiel. Bei der Auswahl einer Absicht, wie im Bild #Einkaufen, kann direkt eine Antwort des Assistenten hinterlegt und festgelegt werden, welche Aktion im Nachgang ausgeführt werden sollte. Typisch für die Plattform gibt es ganz rechts die Möglichkeit, den Assistenten im Chat auszuprobieren. Unter Analyse können die jeweiligen Konversationen und die Erfolgsraten der Verarbeitung eingesehen werden.

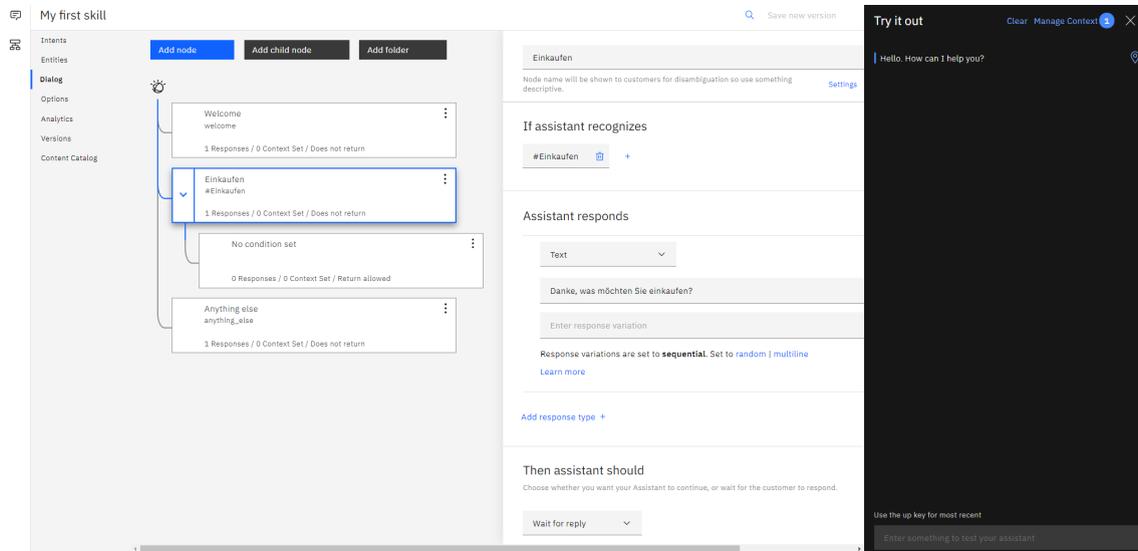


Abbildung 3.11: IBM Watson Assistant

Mit DeepQA [KPB⁺12] im Jahr 2012 hat IBM bereits dafür gesorgt, dass IBM Watson [Fer10] gegen die Jeopardy! Weltmeister gewinnen konnte. Beim Spiel Jeopardy! geht es darum, auf eine Aussage des Moderators die richtige Frage zu finden. DeepQA bildet die Basis von Watson und ist ein kognitives Computersystem aus dem Bereich des *Question Answering*. Im Allgemeinen wurde es durch IBM Research entwickelt, um aufgrund einer aufgestellten Hypothese eine Antwort auf eine Frage in natürlicher Sprache zu geben. Watson gelang es, 70% der Fragen mit einer Präzision von 85% innerhalb von drei Sekunden zu beantworten.

3.2.4 Sprachgenerierung

Es gibt bereits einige Implementierungen von Chatbots, die wir im Verlauf dieses Kapitels untersuchen. Im Allgemeinen werden diese nach der Sprachdomäne und dem Antwortverhalten unterschieden.

Definition:

Ein Chatbot ist ein Computerprogramm, das durch die Verarbeitung natürlicher Sprache automatisiert Gespräche mit einem Menschen führt.

Im Bezug auf die Eingabe haben die natürlichsprachlichen Schnittstellen entweder eine geschlossene Sprachdomäne oder eine offene Sprachdomäne. Als Ausgabe kann eine vordefinierte Antwort ausgewählt oder in Abhängigkeit zu einer Eingabe eine Ausgabe zur Laufzeit generiert werden.

Offene Sprachdomäne	Nicht möglich		GPT	GPT-2	GPT-3
			EIMo	XLNet	Turing NLG
			BERT	T5	GShard
Geschlossene Sprachdomäne	Baseball	INTELLECT	Mitsuku	Cleverbot	
	SHRDLU	Mercury	Twyla	KITT.AI	
	SmartKom	PLOW	Chatterbot	MSG.AI	
	Abrufbasierte Antwort		Generative Antwort		

Abbildung 3.12: Einordnung der natürlichsprachlichen Schnittstellen und Sprachmodelle

3.2.4.1 Abrufbasiertes Antwortverhalten

Für eine vordefinierte Aufgabe werden nur eine begrenzte Anzahl an Regeln benötigt und die Sprachdomäne ist dementsprechend geschlossen und enthält eine begrenzte Anzahl an vordefinierten Antworten. Dies sind regelbasierte Systeme, die mit wenig Daten auskommen und eine Wenn-Dann-Logik verfolgen, um eine Konversation zu führen. Dabei werden in der Regel Sprachmuster definiert, welche auf die Eingabe angewandt werden. Wird nach einem Vergleich ein Muster in der Eingabe mit einer bestimmten Wahrscheinlichkeit erkannt, kann die dazugehörige Antwort des Systems ausgegeben werden.

Frage-Antwort-Systeme

Baseball ist im Jahr 1963 als ein erstes Frage-Antwort-System (Engl.: Question-Answering-System, QA) entwickelt worden, das Fragen zu einen Datensatz über Baseballspiele beantwortet. Die Fragen mussten dabei in Englisch mithilfe von Lochkarten an das System gestellt werden. Dabei erstellt BASEBALL nach einer Benutzereingabe eine Pfadstruktur und fragt nach Slot-Filling [BBW17] die Daten ab, bis eine vollständige Abfrage zur Ausgabe eines Ergebnisses führt.

Im Jahr 2010 wurde das Frage-Antwort-System IBM Watson mit der DeepQA Architektur [Fer10] veröffentlicht. Da eine komplexe Fragestellung mehrere Hinweise auf eine korrekte Antwort enthalten kann, wird die Eingabe in mehrere Teile zur Verarbeitung zerlegt und die Lösungskandidaten berechnet [KPB⁺12]. Die Annahme für die Zerlegung ist, dass nicht alle Hinweise zur Lösung in nur einer Quelle vorkommen und durch die getrennte Untersuchung mehrere Lösungskandidaten bestimmt werden.

Von Gunther Cox entwickelt, stellt Chatterbot [Cox14] ein Python-Framework² dar, um einen Frage-Antwort-Chatbot zu erstellen. Chatterbot kann aus einem Korpus oder durch die Bereitstellung von Listen mit Beispielsätzen trainiert werden, wobei eine Liste jeweils eine Beispielkonversationen bedeutet. Zur Antwortgenerierung wählen die Such- und Klassifizierungsalgorithmen nach Ähnlichkeiten zur Eingabe oder Häufigkeit der Verwendung eines Ausdrucks einer Ausgabe aus den gelernten Trainingsdaten aus.

Natürlichsprachliche Schnittstelle zu Datenbanken

Harris stellte im Jahr 1984 mit INTELLECT [Har84] eine erste natürlichsprachliche Schnittstelle zu Datenbanken vor. Dabei handelt es sich um die Weiterentwicklung von ROBOT ebenfalls von Harris aus dem Jahr 1977. Harris ermöglicht durch seine Arbeit Abfragen gegenüber Datenbanken und anderen Schnittstellen in natürlicher Sprache. Dabei wird ein nichtdeterministischer Übergangnetzwerkparser verwendet.

Wissensbasierte Systeme

SHRDLU stellt ein erstes wissensbasiertes System im Jahr 1972 dar, das systematisch auf verschiedene Komponenten der Repräsentation sprachlichen und nicht-sprachlichen Wissens zurückgreift und von Winograd entwickelt wurde. Dabei nahm Winograd an, dass ein Leser eines Textes Wissen nutzt, um diesen zu verstehen. Dadurch führte Winograd die Verwendung von Wissensrepräsentationen in natürlichsprachlichen Systemen ein [Car17].

Siri wurde im Jahr 2010 als sprachlichen Assistenten von Apple veröffentlicht. Mit einer Client-Server-Architektur benötigt es zur Auswertung einer Benutzereingabe eine Internetverbindung und basiert auf den Aktiven Ontologien [Guz06], die im Jahr 2006 von Guzzoni erfunden wurden. Siri speichert während der Unterhaltung die Eingaben ab und pflegt die Beziehungen ein. Durch diese Personalisierung kann es zum späteren Zeitpunkt die Eingabe *Rufe Mama an* zur richtigen Person zuordnen.

Dialogbasierte Benutzerschnittstellen

Im Jahr 2001 ist SmartKom eine dialogbasierte Benutzerschnittstelle [WRB01], die modal neben der Sprache, noch die Gestik und die Mimik in der Kommunikation berücksichtigt. SmartKom unterstützt ungenaue, mehrdeutige oder unvollständige Eingaben. Es verarbeitet explizite Repräsentationen des Benutzermodells, der Domäne, der Aufgabe und des Kontextes.

Das von der MIT-Forschungsgruppe im Jahr 2002 entwickelte Mercury-System [Sen02] ist eine Telefon-Hotline für die automatisierte Buchung von Flugtickets. Es führt den Benutzer nach Slot-Filling durch einen Dialog über die Auswahl eines geeigneten Fluges nach Datum, Uhrzeit und bevorzugter Fluggesellschaft.

Außerdem beschreibt Allen [ACF07] ein an der Stanford Universität entwickeltes System namens Plow (2007). Als kollaborativer Task-Agent kann Plow bestimmte Aufgaben ausführen, z. B. das Extrahieren bestimmter Informationen aus dem Internet durch Demonstration, Erläuterung und Dialog. Für die Akzeptanz eines solchen Systems spielen das Verständnis der Benutzereingaben, die Benutzerfreundlichkeit und die Natürlichsprachlichkeit eine Rolle.

²Chatterbot, letzter Aufruf: 24. Juli 2023, <https://github.com/gunthercox/ChatterBot>

3.2.4.2 Generatives Antwortverhalten

Bei einem Chatbot mit einem generativen Antwortverhalten wird die Antwort nicht aus den vorher definierten Menge der möglichen Antwortsätzen ausgewählt, sondern in Abhängigkeit von der Eingabe zur Laufzeit generiert. Hinter solchen Frage-Antwort-Chatbots verbergen sich neuronale Netze, die nicht deterministisch auf eine Eingabe reagieren.

Bei einer offenen Sprachdomäne kann ein Benutzer über jedes mögliche Thema mit einem Chatbot sprechen. Das Sprachmodell des Chatbots muss dabei durch große Daten trainiert werden. Wenn die Trainingsdaten dagegen klein und auf die Lösung einer bestimmten Aufgabe ausgelegt sind, hat der Chatbot eine geschlossene Sprachdomäne.

Recurrent Neural Network

Ein Frage-Antwort-Chatbot kann durch das Kodierer-Dekodierer-Modell implementiert werden. Dieses Modell basiert in den meisten Fällen auf den *Rekurrenten Neuronalen Netzen* (Engl.: *Recurrent Neural Network, RNN*) [RHW86] und dient einer Abbildung von einer Wortkette auf eine andere Wortkette unterschiedlicher Länge.

Der Kodierer hat die Aufgabe, die Wörter in einen Vektor (auch versteckten Zustand, Engl.: hidden state) zu kodieren. Die Ausgabe des Kodierers ist ein versteckter Zustand, der als Vektor die Information der gesamten Eingabesequenz enthält und an den Dekodierer als Eingabe übergeben wird. Die Aufgabe des Dekodierers ist es, mithilfe der RNNs aus dem Vektor die jeweiligen Wörter vorherzusagen und somit zum Schluss die Ausgabesequenz zu generieren.

Abbildung 3.13 zeigt einen Ablauf bei RNN. Das Wort bzw. der Wortvektor x_t wird mit dem versteckten Zustand des vorherigen Zeitabschnitts h_{t-1} zu $[x_t + h_{t-1}]$ konkateniert. Die Hypertangensfunktion \tanh normiert die Werte des konkatenierten Vektors auf einen Bereich $[-1, 1]$ und übergibt den versteckten Zustand h_t in den nächsten RNN. Die Normierung ist wichtig, um die Werte im Vektor klein zu halten. Ohne die Normierung wird durch die über mehrere Zeitintervalle ausgeführte Konkatenation den Wertebereich des versteckten Zustands stark erhöhen.

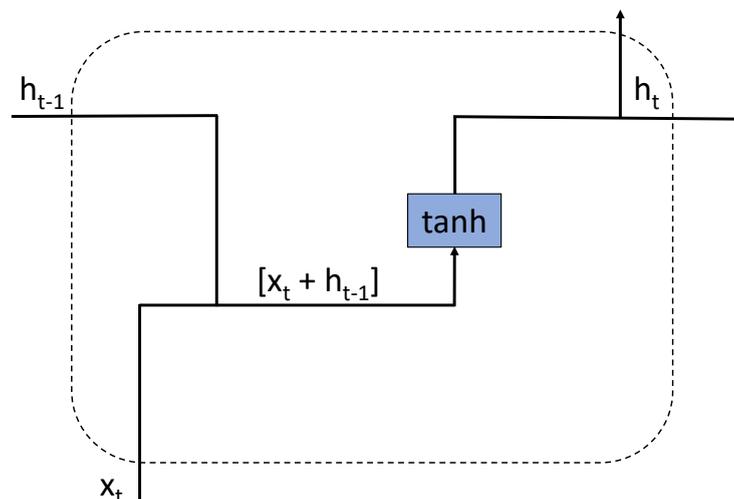


Abbildung 3.13: Aufbau von RNN

Long short-term memory

Durch Hochreiter [HS97] im Jahr 1997 eingeführt, ist das *Langes Kurzzeitgedächtnis* (*Long short-term memory*, *LSTM*) eine der bekanntesten RNN-Architekturen und hat folgenden Aufbau (Siehe Abbildung 3.14):

- Vergesstor (Engl.: Forget Gate), das mit der Sigmoidfunktion σ das konkatenierte $[x_t + h_{t-1}]$ auf den Wertebereich $[0,1]$ normalisiert und anschließend eine Punktweise Multiplikation des Ergebnisses mit dem Knotenzustand des letzten Zeitintervalls c_{t-1} durchführt. Ist das Ergebnis der Sigmoidfunktion σ näher bei 0, wird die anschließende Multiplikation den Knotenzustand vergessen lassen, ansonsten weitergegeben.
- Eingangstor (Engl.: Input Gate), welches parallel das konkatenierte $[x_t + h_{t-1}]$ mit der Sigmoidfunktion σ auf den Wertebereich $[0,1]$ und mit der Hypertangensfunktion auf den Wertebereich $[-1,1]$ normalisiert. Durch die Punktweise Multiplikation werden die Werte beibehalten, die nach der Sigmoidfunktion σ näher bei einer 1 sind, und anschließend durch eine punktweise Addition an den Knotenzustand c_{t-1} gespeichert.
- Ausgangstor (Engl.: Output Gate), das durch eine punktweise Multiplikation den versteckten Zustand h_t des Knoten berechnet. Dafür werden c_t mit der Hypertangensfunktion auf den Wertebereich $[-1,1]$ und das konkatenierte $[x_t + h_{t-1}]$ mit der Sigmoidfunktion σ auf den Wertebereich $[0,1]$ normalisiert.
- Knotenzustand c_t , der weitergegeben oder gelöscht werden kann.

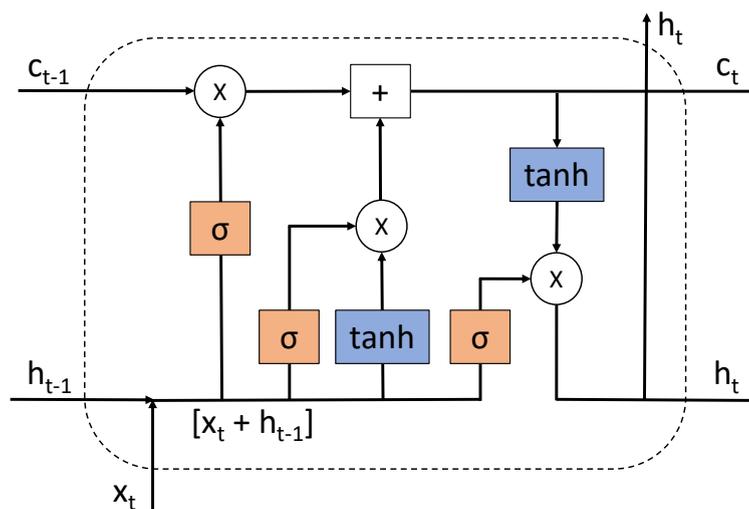


Abbildung 3.14: Aufbau von LSTM

Das häufigste Anwendungsszenario für eine LSTM-Architektur für natürlichsprachliche Schnittstellen ist die Überführung einer Zeichenkette in einer andere Zeichenkette, zum Beispiel bei einem Übersetzer. Dabei wird die Zeichenkette einer Sprache im Kodierer zu einem Vektor kodiert und dem Dekodierer für die Überführung in eine andere Sprache übergeben. In einer Kodierer-Dekodierer ist es schwierig, ein Dialogsystem zwischen den beiden Schritten einzusetzen. Im Vergleich zu RNNs, bei denen die Informationen im Kurzzeitgedächtnis gespeichert sind, kann bei LSTM ein Langzeitgedächtnis implementiert werden.

Transformer

Den letzten Stand der Technik in der Sprachgenerierung u.a. für Chatbots bildet die *Transformer*-Architektur [VSP⁺17], die durch Google im Jahr 2017 vorgestellt wurden. Dabei gibt es zwei grundlegende Unterschiede zu RNN und LSTM: (i) die Funktionsweise basiert auf einem Aufmerksamkeitsmechanismus, der für die Erzeugung einer Ausgabe aussucht, welche Bestandteile der Eingabe und den bereits erzeugten Wörter als Ausgabe für die neue Wortvorhersage relevant sind, und (ii) pro Zeitintervall werden sowohl der Kodierer als auch der Dekodierer parallel ausgeführt [VSP⁺17].

- DialoGPT (Dialogue Generative Pre-trained Transformer) [ZSG⁺20] stellt ein vortrainiertes Modell zur Erzeugung neuronaler Konversationsantworten dar, der auf dem GPT-2 Sprachmodell von Open AI basiert und auf 147 Mio. Reddit-Unterhaltungen trainiert wurde.
- Google Meena [ALS⁺20] ist ein auf Transformer basiertes Chatbot. Das neuronale Netz von Meena weist 2,6 Mrd. Parameter und wurde auf durch gefilterte Social Media Gespräche trainiert.

Seit der Einführung der Transformer-Architektur stieg die Größe der jeweiligen Sprachmodelle stark an. Tabelle 3.1 zeigt eine Auflistung mit der Angabe des Veröffentlichungsdatums und der Anzahl der Parametern. Zu den Parametern zählen die Anzahl der versteckten Schichten, die Anzahl der Knoten pro Schicht sowie deren Aktivierungsfunktion.

- ELMo [PNI⁺18] stellt einen Teil eines NLP-Toolkits von AllenNLP dar und ist ein vortrainiertes Worteinbettungssystem zur Satzdarstellung, insbesondere Wörter und Phrasen als Vektoren. Dabei verwendet es bidirektionale LSTM, um komplexe Verwendung in sprachlichen Kontext zu modellieren.
- BERT [DCLT19] ist ein bidirektionaler Transformer, der mit einer Kombination aus maskiertem Sprachmodellierung und Vorhersage des nächsten Satzes auf einem großen Korpus aus dem Toronto Book Corpus und Wikipedia trainiert wurde.
- GPT [RNSS18], GPT-2 [RWC⁺18] und GPT-3 [BMR⁺20] von OpenAI ermöglichen durch unidirektionale Transformer-Modelle, die auf sehr großen Datenmengen trainiert wurden, langfristige Abhängigkeiten in den Textdaten zu erfassen und einen lexikalisch vielfältigen Text zu generieren.
- XLNet [YDY⁺20] wurde mithilfe einer autoregressiven Methode trainiert, um bidirektionale Kontexte zu lernen, indem die erwartete Wahrscheinlichkeit über alle Permutationen der Reihenfolge der Faktorisierung der Eingabesequenz maximiert wird.
- T5 [RSR⁺20] ist ein Encoder-Decoder-Modell, das auf einer Multitask-Mischung aus unbeaufsichtigten und überwachten Aufgaben vorab trainiert wurde und für das jede Aufgabe in ein Text-zu-Text-Format konvertiert wird.
- GShard [LLX⁺20] ist zum heutigen Stand mit 600 Mrd. Parametern das größte Sprachmodell. Es stellt ein mehrsprachiges Transformer-Modell für neuronale maschinelle Übersetzung von 100 Sprachen ins Englische dar und verwendet die bedingte Berechnung und automatisches Scherben (Engl.: Sharding). In 4 Tagen wurde das Modell auf 2.048 Google Cloud GPUs v3 Cores trainiert. Eine Google Cloud GPU v3 Cores wird aktuell mit 8\$ pro Cores und pro Stunde berechnet. Zum Vergleich betrug das Training bei XLNet nur 2,5 Tagen auf 128 Google Cloud GPUs v3 Cores und bei BERT etwa 4 Tagen auf 16 Google Cloud GPUs v3 Cores.

- ChatGPT wurde im November 2022 veröffentlicht und basierte am Anfang auf GPT-3.5. Mittlerweile stehen Nachfolgeversionen wie GPT-3.5-Turbo und GPT-4.0 zur Verfügung. Bei ChatGPT handelt es sich um einen Sprachrechner für die Berechnung der Wörter bzw. des nächstwahrscheinlichen Folgeworts. Es verwendet lediglich den Decoder aus der Encoder-Decoder-Architektur und ist aufgrund großer Trainingsmengen für die Generierung menschenähnlicher Texte bekannt. Zu jeder Eingabe generiert das Large Language Model (LLM) eine textuelle Ausgabe. In der Version 4 erzeugt es neben Text, Bilder und anderen Formattypen mittels natürlichsprachlicher Beschreibung.

Veröffentlichung	Sprachmodell	Hersteller	Parameter (Mrd.)
März 2018	ELMo	Allen Institute for AI	0,0936
Juni 2018	GPT	OpenAI	0,11
Oktober 2018	BERT	Google	0,34
Juni 2019	XLNet	CMU	0,34
November 2019	GPT-2	OpenAI	1,5
Oktober 2019	T5	Google	11
Februar 2020	Turing-NLG	Microsoft	17
Mai 2020	GPT-3	OpenAI	175
Juni 2020	GShard	Google	600
November 2022	ChatGPT	OpenAI	ab 175

Tabelle 3.1: Übersicht der bekanntesten Sprachmodelle

3.3 Programmieren in natürlicher Sprache

Es sind mehr als 50 Jahre seit der ersten Idee zur Programmierung in natürlicher Sprache von Jean E. Sammet [Sam66] vergangen. Die Entwicklung befindet sich hier im Vergleich zu den anderen Forschungsgebieten noch relativ am Anfang. Dies liegt daran, dass die Verarbeitung von natürlicher Sprache komplex ist und Schwierigkeiten bereitet. Eine der Herausforderungen besteht darin, dass die Programmierung natürlicher Sprache ein domänenspezifisches System erfordert, das um eine Klärung bittet, und damit die Nachteile der natürlichen Sprache, nämlich Mehrdeutigkeit und Ungenauigkeit, überwindet.

Im Jahr 1959 entstand eine Programmiersprache, die an die natürliche Sprache angelehnt war: Common Business Oriented Language, kurz COBOL. Es war zunächst rein imperativ und wurde später durch Elemente der strukturierten, prozeduralen und objektorientierten Programmierung erweitert. Ein Vorteil von COBOL ist, dass es an die natürliche Sprache angelehnt ist und somit für Benutzer als auch Entwickler gut zu lesen und zu verstehen ist. Auf der anderen Seite sind nicht alle Mechanismen einer komplexen Programmiersprache, wie C# oder Java umsetzbar. Der folgende Codeabschnitt zeigt ein Beispiel:

```

If Nenner > 0
  Compute Zahl = Zaehler / Nenner
End-Compute
Else
  Display "Nenner sollte > 0 sein!"
  Move 0 To Zahl
End-If

```

Die *Idee einer Programmierung mit natürlicher Sprache* ist nicht neu und wurde bereits im Jahr 1966 durch Sammet [Sam66] vorgeschlagen. Sie spricht in der Vorstellung von einem Computer, der auch von einem unausgebildeten Nutzer frei programmiert werden

kann, um jede beliebige Aufgabe zu lösen. Hierfür sollte es genügen, alle Anweisungen, die abgearbeitet werden müssen, in normalen englischen Sätzen zu formulieren. Der Computer analysiert, interpretiert und arbeitet die enthaltenen Anweisungen ab. Die Grenzen dagegen sieht sie bei der Umschreibung von präzisen, mathematischen Ausdrücken, die durch natürliche Sprache und deren Umgangssprache eine syntaktische und semantische Unschärfe erhalten.

Natural Language Computer (NLC) aus dem Jahr 1979 u.a. von Ballard [BB79], [BB80], [BBS83] ermöglicht es, den Benutzern einfache *arithmetische Berechnungen in der Tabellenkalkulation* mittels natürlicher Sprache durchzuführen. Das System hat hierbei kein Wissen über die Einträge, sondern wird über eine explizite Angabe einer Zelle gesteuert: *add Y1 to Y2* oder *add to Y2 Y1*. Dabei verfolgt der Ansatz die Verarbeitung einfacher Regeln, die vorher definiert sein müssen. Bei NLC gibt es zwar die Möglichkeit arithmetische Berechnungen innerhalb der Tabellenkalkulation durchzuführen, jedoch handelt es sich hierbei um eine reihenfolgegetreue Abbildung der Sprache zu einer Formel. In einer Evaluation hatten die Teilnehmer 90% der Aufgaben ohne Programmierkenntnisse lösen können.

Mit der Erstellung *eines Programmcodes in Tabellenkalkulationen* durch einen Benutzer befasst sich im Jahr 2011 *QuickCode* [Gul11], ein Werkzeug von Gulwani. QuickCode führt die Programmierung durch Beispiele durch und automatisiert die Zeichenfolgenverarbeitung in Kalkulationstabellen anhand von Eingabe-Ausgabe-Beispielen. Der Schwerpunkt der Arbeit liegt auf der Synthese von Programmen, die aus Textoperationen bestehen.

Im Jahr 2014 stellte Gulwani ein weiteres Werkzeug *NLyze* [GM14] vor. NLyze ist ein Add-In für Microsoft Excel und erlaubt einem Benutzer, die *Tabellendaten in natürlicher Sprache* zu bearbeiten. Für die logische Interpretation der Benutzereingaben wird eine eigene domänenspezifische Sprache verwendet. Anstatt die Tabellen automatisch zu erkennen, werden kanonische Tabellen verwendet, die vom Benutzer für NLyze markiert werden müssen.

Nach der Vision von Prof. Dr. Walter F. Tichy muss das Programmieren keine Einbahnstraße sein. Der Perspektivenwechsel ist eine grundlegende Änderung in Bezug auf die Programmierung. Wenn der Computer fragen kann, was die unbekannte Eingabe bedeutet, können die Probleme der Mehrdeutigkeit, Ungenauigkeit usw. verringert, wenn nicht beseitigt werden. Tichy [TLK13] ist der Meinung, dass jeder programmieren können soll. Informelle, aber präzise natürliche Sprache würde es jedem ermöglichen, zu programmieren. Für komplexe und anspruchsvolle Anwendungen werden weiterhin Software-Ingenieure benötigt, nicht jedoch für routinemäßige Programmieraufgaben. Computer könnten Menschen nachahmen, wenn sie Anweisungen erhalten. Wenn es gelingt, Computer in gewöhnlicher Sprache zu unterrichten, wird jeder dazu befähigt sein zu programmieren, vielleicht sogar ohne es zu merken. Die Eintrittsbarriere für die Programmierung würde erheblich gesenkt werden. Es könnten zahlreiche, hochgradig personalisierte und kreative Anwendungen entstehen, die den Mangel an Programmierern verringern.

Als erste lehrstuhlinterne Forschungsarbeit auf dem Gebiet der Programmierung in natürlicher Sprache befasst sich Gelhausen [Gel10] im Jahr 2010 mit der Modellextraktion aus natürlicher Sprache. Als Zielsetzung legt Gelhausen den Schwerpunkt auf die Unterstützung des Entwicklungsprozesses in einem Softwareprojekt und überführt eine natürlichsprachliche, textuelle Beschreibung in ein UML-Modell. Diese Überführung ist einer der ersten Schritte in der traditionellen Softwareentwicklung und erfolgt in manueller, aufwendiger Arbeit durch ausgebildete Experten, die den Überblick über die Zusammenhänge in umfangreichen Spezifikationsbeschreibungen verlieren könnten. Gelhausen führt in seiner Forschungsarbeit thematische Rollen ein, die manuell an einzelne Wörter in der vorliegenden Spezifikation annotiert werden. Dadurch wird die gegebene Beschreibung mit Informa-

tion zur Semantik angereichert. Nach diesem manuellen Schritt erfolgt eine automatisierte Erstellung des UML-Modells. Auf dem Weg zur Programmierung in natürlicher Sprache schafft Gelhausen den ersten Schritt: von natürlicher Sprache zum UML-Klassenmodell. Gelhausen setzt eine manuelle Annotation der Beschreibungen durch einen Entwickler im Vorfeld voraus.

Körner [Kö14] ermöglicht in seiner Dissertation im Jahr 2014 eine automatisierte Umwandlung in UML-Klassenmodelle. Die Anforderungserhebung wird bei seiner Arbeit durch einen Menschen durchgeführt. Körner automatisiert die manuelle Annotation von Gelhausen und führt einen dreiteiligen Verarbeitungsprozess ein, der fortlaufende Überarbeitung des Modells als auch die Rückmeldung der Änderungen am UML-Modell zurück zum gegebenen Text ermöglicht. Im ersten Schritt untersucht Requirements Engineering Specification Improver (RESI) die syntaktische Struktur der Sätze und die Semantik. Dabei macht RESI den Nutzer auf sprachliche Mängel aufmerksam, die später in der Verarbeitung zu Problemen führen können. Im zweiten Schritt übernimmt AutoAnnotator die Überführung der vorverarbeiteten Beschreibung in das UML-Modell. Im letzten Schritt führt Requirements Engineering Feedback System (REFS) eine Synchronisation der Texte nach einer Anpassung am UML-Modell durch. Aus diesem Grund bleiben sowohl die Anforderungsdokumentation als auch das UML-Modell jederzeit konsistent.

Natural Language Command Interpreter, kurz NLCI, von Landhäuser [Lan16] stellt im Jahr 2016 eine Architektur für die Programmsynthese aus natürlicher Sprache dar. Landhäuser befasst sich mit der Transformation der natürlichsprachlichen Eingaben eines Nutzers in ein ausführbares Computerprogramm. Statt Domänenunabhängigkeit bei der Umwandlung für eine vielseitige Anwendung anzustreben, ermöglicht seine Forschungsarbeit einen Wechsel von Domäne zu Domäne. Für eine Programmierung in natürlicher Sprache fehlt eine Möglichkeit, Algorithmen aus einer Beschreibung eines Benutzers zu erfassen. Bei Rückfragen fehlt eine interaktive Komponente für die Kommunikation mit dem Benutzer. Zudem muss ein Entwickler erst Vorbereitungen treffen, bevor ein Benutzer das NLCI Werkzeug verwenden kann. Für die Funktionalität des NLCI muss ein Entwickler ein objektorientiertes Modell der Ziel-API und einen Quelltexterzeuger für die gewünschte Programmiersprache bereitstellen. Eine detaillierte Modellierung ist für den erfolgreichen Einsatz des NLCI entscheidend. Alle Einschübe greifen auf das API-Modell zu, um eine konsistente, funktionsfähige Lösung zu generieren. Nach der Vorverarbeitung erkennt das NLCI Sprachmuster in den natürlichsprachlichen Beschreibung, erzeugt Abhängigkeitsbäume, korrigiert die Reihenfolge der Anweisungen und identifiziert Kontrollstrukturen. Am Ende erzeugt das NLCI skriptartige Programme als Ausgabe.

ProNat von Weigelt [Wei21] zielt auf die Sprachdomäne der humanoiden Robotersysteme ab und stellt ein agentenbasiertes System dar, das die einzelnen Teilaufgaben in der Verarbeitung einer Eingabe an speziell dafür ausgelegten Agenten auslagert. Das Aufgabenspektrum der Agenten streckt sich von der Sprachverarbeitung, über die Untersuchung der Eingabe auf programmatische Strukturen bis zur Sprachgenerierung bei Zwischenfragen aus. Vor den Agenten ist eine Vorverarbeitungskomponente geschaltet, welche die gesprochene Eingabe in eine textuelle Wortsequenz umwandelt und diese anschließend in einen initialen Graphen überführt. Danach wird die Eingabe innerhalb der parallelausgeführten Agenten syntaktisch sowie semantisch untersucht und die jeweils gültigen Ergebnisse in den Graph gespeichert. Zur Unterstützung der Agenten können externe Ressourcen, wie Wissensdatenbanken, angeschlossen werden. Nach den Agenten erfolgt in der Nachbearbeitungskomponente die Umwandlung des erzeugten Graphen in den Quellcode.

In Oktober 2021 wurde Github Copilot³ von OpenAI veröffentlicht. Es handelt sich um ein Hilfswerkzeug, das Entwicklern anhand einer natürlichsprachlichen Eingabe, in der

³Github Copilot, <https://github.com>, letzter Aufruf: 24. Juli 2023

Regel eines Kommentars, passende Codesnipps vorschlägt. Entwickler kann den Vorschlag annehmen und den Code in das aktuell offene Projekt aufnehmen oder ablehnen bzw. weiteren Vorschlag anschauen. Github Copilot basiert auf dem OpenAI Codex, einem GPT AI Sprachmodell, und wurde auf öffentlichen Github Repos trainiert.

3.4 Benutzerprogrammierung

Das folgende Kapitel stellt eine Auswahl der Forschungsarbeiten in der Benutzerprogrammierung vor. Paternó [Pat13] stellt die Motivation hinter der von Liberman [LPKW06] definierten Benutzerprogrammierung vor (Siehe Kapitel 1.1) und diskutiert deren Grundbegriffe sowie den aktuellen Stand der Technik. Verschiedene Ansätze werden diskutiert und hinsichtlich ihrer Hauptmerkmale und der Technologien und Plattformen, für die sie entwickelt wurden, klassifiziert.

Im Jahr 1993 erforschte Cypher die Programmierung durch Demonstration [Cyp93]. Dabei geht der Ansatz davon aus, dass ein Benutzer die Aktion vorführt, ein Computer diese Aktion durch ein Programm wiederholen kann. Der Ansatz wird noch heute vor allem bei der Programmierung von humanoiden Robotern und der Wiederholung von Menschenaktionen verwendet.

In einer Studie aus dem Jahr 2004 identifiziert Ko [Ko04] sechs Lernbarrieren bei der Benutzerprogrammierung: Design, Auswahl, Koordination, Nutzung, Verständnis und Informationsbarrieren. 2006 stellte Myers [MKB06] einen Überblick über die Forschung im Bereich der Benutzerprogrammierung vor. Er fasste zusammen, dass viele verschiedene Systeme für die Benutzerentwicklung bereits realisiert wurden. Es gibt jedoch kein System, das eine Endnutzerprogrammierung in natürlicher Sprache ermöglicht.

Metafor [LL05] hat eine andere Ausrichtung. Basierend auf den Beschreibungen der Benutzer leitet das System die Programmstrukturen zur Unterstützung des Software-Designs ab.

Begel [Beg05] führt die Spracherkennung in der Softwareentwicklung ein. Sein Ansatz verwendet die Programmanalyse, um den Code in natürlicher Sprache zu diktieren, und ermöglicht so die Erstellung eines Programmeditors, der die sprachgestützte Programmierung unterstützt.

Im Jahr 2008 beschreibt Dorner [DSW08] Endanwendungsentwicklungsansätze aus der Literatur, die für verschiedene Gruppen von Benutzern geeignet sind. Die Umsetzung der richtigen Mischung dieser Ansätze führt zu eingebetteten Designumgebungen mit einer leichten Steigerung der Komplexität. Durch Suchumgebungen können Benutzer mit unterschiedlichen Kenntnissen Systemanpassungen selbst vornehmen.

Im Jahr 2009 stellte Williams [Wil09] dar, welches Verhalten des Systems von einem Benutzer erwartet wird. Weiter kann es häufig vorkommen, dass die Kommunikation mit dem System sehr stark vom Thema bzw. Kontext abhängig ist und eine freie Formulierung dem Benutzer schwer fällt. Insbesondere von Nachteil ist es, wenn der Benutzer sich nicht nur Gedanken darüber machen muss, was er dem System mitteilen möchte, sondern auch, wie es formuliert werden muss, damit das System es versteht.

Cunha [CFM⁺12] verwirklicht Techniken für die modellgesteuerte Tabellenkalkulation, die bidirektionale Transformationen verwenden, um Tabellenkalkulationsmodelle und synchronisierte Instanzen zu verwalten.

Eine eingeschränkte Programmierung stellt SmartSynth [LGS13] durch das Synthetisieren von Automatisierungsskripten für Smartphones aus natürlichen Sprachbeschreibungen vor.

Es erfolgt eine Ergebnisausgabe oder eine Ausgabe einer Fehlermeldung. Es gibt keine Möglichkeit zur Dialoginteraktion zwischen einem Benutzer und SmartSynth.

Sestoft [SS13] steigert die Ausdruckskraft und betont die Ausführungsgeschwindigkeit der so definierten Funktionen, indem rekursive Funktionen und Funktionen höherer Ordnung unterstützt werden und eine schnelle Ausführung durch sorgfältige Auswahl der Datendarstellung und der Compiler-Technologie ermöglicht wird.

Eines der letzten Forschungsergebnisse präsentiert Wang [WGLM17] mit einer benutzerfreundlichen Schnittstelle für natürliche Sprachen, um Benutzeraufgaben auszuführen. Das System verwendet Grammatikregeln, die natürliche Sprache in eine formale Sprache umwandeln.

Aktuelle Entwicklungen im Gebiet der Benutzerprogrammierung zeigen, dass diese durch skriptartige Programmierung, Blockprogrammierung oder durch Drag & Drop gewährleistet wird [KAB⁺11]. Jedoch erinnern die genannten Ansätze an Insellösungen, die meist nur eine bestimmte Zielgruppe ansprechen und ein Erlernen für die Verwendung dieser Schnittstellen benötigen.

3.5 Schlussfolgerung und Forschungsfragen

Im Folgenden werden die einzelnen Forschungsfelder in einer Schlussfolgerung zusammengefasst und offene Forschungsfragen ausformuliert, die im Verlauf der Ausarbeitung diskutiert werden.

Eine Vielzahl an Dialogsystemen existieren bereits als (i) schlüsselwortbasierte Systeme, die nach bestimmten Schlüsselwörtern im Dialogverlauf suchen oder (ii) musterbasierte Systeme, die das sogenannte Slot-filling ausführen und die Muster für eine korrekte Ausführung zu füllen versuchen. Im Allgemeinen werden regelbasierte Systeme implementiert, die auf bestimmte Schlüsselwörter reagieren. Einen Anwendungsfall stellt eine Informationsanzeige für häufig gestellten Fragen oder die Bibliotheken von Bedienungsanleitungen. Für Endanwender bieten die natürlichsprachlichen Schnittstellen eine schnelle Variante, um Informationen anzuzeigen oder zu verarbeiten. Der Nachteil liegt darin, wie wir bereits in der Pilotstudie gezeigt haben, dass die Benutzer in der Ausdrucksweise nicht frei sind. Eine weitere Art der Dialogsysteme ist nach den vordefinierten Dialogbäumen implementiert und kann mit einer Hotline oder einem Fragebogen verglichen werden. Ein Benutzer trifft eine Auswahl in einem vordefinierten Entscheidungsbaum und wird entlang eines vordefinierten Pfades zu einer gültigen Antwort geführt. Auf der anderen Seite kann ein Benutzer eine komplexe Anfrage nur teilweise am Stück mitteilen, teilt diese unfreiwillig in kleine Abschnitte auf und erhält durch unvollständige Rückmeldung über den Zustand des Systems (Erfolgreich, Fehler, Mehrdeutigkeit, usw.). Insbesondere ist es spannend zu erforschen, wie die natürlichsprachliche Schnittstelle, ähnlich einem Menschen, bei der Verarbeitung von Information beliebig abstrahieren und spezialisieren kann. Kurzweil [Kur12] schreibt im Buch *How to create a mind?*, dass das menschliche Gehirn eine Hierarchie von Mustererkennern vorweist.

Der Ansatz eines einfachen Frage-Antwort-Systems reicht nicht aus, weil die Benutzer während der Bearbeitung unterschiedliche Gegenfragen an die natürlichsprachliche Schnittstelle stellen. Es fehlt an einer allgemeinen Schnittstelle für die Benutzerprogrammierung, bei der ein Benutzer mit den Anforderungen bzw. den Anpassungen direkt loslegen könnte. Neben den vorgestellten Ansätzen der Benutzerprogrammierung gibt es keine Lösungen, die eine natürlichsprachliche Benutzerprogrammierung versuchen. Die Sprachdomäne eines Menschen ist größer als die Sprachdomäne einer natürlichsprachlichen Schnittstelle. Daher muss sich bei der Umsetzung entweder die Sprachdomäne ähneln oder die Sprachdomäne der natürlichsprachlichen Schnittstelle dynamisch anpassbar sein.

Ein Encoder-Decoder Dialogsystem, mit RNN, LSTM sowie Transformer setzt eine Annahme voraus, dass die Eingabe eines Benutzers für die Berechnung einer Ausgabe perfekt ist. Das NLUI sollte vielmehr mit unvollständigen Eingaben umgehen können und einem Benutzer einen Schritt-für-Schritt Weg zur Lösung bereitstellen.

Aus den Erkenntnis der Pilotstudie sowie der Diskussion der verwandten Arbeiten leiten wir die Forschungsfragen in folgenden Eigenschaften ab:

- Dialogverhalten
- Datenstrukturanalyse
- Algorithmenfassung

3.5.1 Dialogverhalten

Berg [BD10], [Ber12], [Ber14] stellte in seiner Forschung an Dialogsystemen zwischen 2010 bis 2014 mehrere Kriterien zur Definition an Natürlichsprachlichkeit eines digitalen Systems fest: gemischte Initiative, implizite Bestätigung, Nachfrage, Korrektur, Rückbezüge und Umgangssprache.

Nicht-perfekte Eingaben

Die Forschung in der Dialogmodellierung ist jedoch nicht abgeschlossen. Vorhandene Lösungen bieten unterschiedliche Funktionalitäten von der NLU bis zur NLG. Diese gehen jedoch immer von einer perfekten Eingabe aus. Bei einer Kommunikation mit einem Benutzer sind Abweichungen zu erwarten. Die Forschung müsste sich eher um eine Architektur für nicht-perfekte Benutzereingaben drehen. Wenn die Eingabe nicht eingeschränkt wird, kann dadurch ein Benutzer mit der natürlichsprachlichen Schnittstelle kommunizieren und eine Beschreibung seines Anliegens liefern, ohne sich an einen Kommunikationsstil anzupassen. Durch eine zeitnahe Rückmeldung zur gewünschten Aufgabe kann auf einen Fehlerzustand zielführend reagiert und die Fehlinterpretationen durch einen Dialog korrigiert werden. Dabei habe ich vor, eine neue Technologie zu benutzen, um die Natürlichsprachlichkeit einer Benutzereingabe zu unterstützen.

Forschungsfrage I

Wie kann das NLUI eine nicht-perfekte oder nicht vollständige Eingabe eines Benutzers erkennen?

Spontaner Kontextwechsel

Die Benutzer kommunizieren mit dem NLUI in natürlicher Sprache. Dabei kann ein Benutzer zuerst eine mathematische Berechnung durchführen, und in einer neuen Eingabe einen Algorithmus mitteilen. Diese Art der spontanen Wechsel in der Kommunikation ist für das NLUI nicht so trivial zu verstehen, außerdem muss hierbei auf die Einhaltung der korrekten Zustände geachtet werden.

Forschungsfrage II

Kann das NLUI auf spontanen Kontextwechsel durch einen Benutzer adäquat reagieren?

3.5.2 Datenstrukturanalyse

Des Weiteren verlangen heutige Softwaresysteme eine Einlernzeit, auch wenn durch eine Gebrauchsanleitung mit Beispielen oder Textbeschreibungen einem Benutzer der Umgang mit dem System erläutert wird. Aus der Sicht von Benutzern muss die natürlichsprachliche Schnittstelle in der Lage sein, sich an die jeweilige Umgebung anzupassen. Die natürlichsprachliche Schnittstelle benötigt das Wissen über die Daten, um eine Benutzeranfrage zu den Daten richtig zu interpretieren. Dabei ist es wichtig, nicht nur vorhandene Daten zu untersuchen, sondern eine Technik zu entwickeln, auch zur Laufzeit mitgeteilte Daten, für natürlichsprachliche Anfragen vorzubereiten. Eine Analyse der Daten kann die natürlichsprachliche Schnittstelle auf die Anfragen der Benutzer vorbereiten.

Forschungsfrage III

Wie erfasst das NLUI die Beschreibungen und die Daten, um Benutzeranfragen zu ermöglichen?

3.5.3 Algorithmenfassung

Außerdem sollte untersucht werden, welche Komponenten zur natürlichsprachlichen Programmierung untersucht werden. Dazu müssten Algorithmen aus den Beschreibungen der Benutzer extrahiert werden. Im Allgemeinen lassen sich einfache Algorithmen durch einen Benutzer mit Hilfe von Anweisungen, Kontrollstrukturen sowie Schleifen definieren.

Wir unterscheiden daher zwischen

- (i) den Einzelbefehlen oder Anweisungen,
- (ii) Folgen von Einzelbefehlen mit Rückbezügen und Kontext,
- und (iii) bedingte und wiederholende Befehlen sowie kollektive Operationen, z.B. eine Operation auf mehreren Elementen einer Menge.

Im Rahmen der Forschungsarbeit werden wir uns auf eine Teilmenge von Algorithmen fokussieren. Des Weiteren wird ein Modell gesucht, um die gewonnene Information für die weitere Verwendung durch Benutzer permanent zu speichern.

Forschungsfrage IV

Können mittels natürlichsprachlicher Beschreibung einfache Algorithmen durch das NLUI erfasst werden?

4. Grundlagen

Dieses Kapitel gibt einen Überblick über die Grundlagen, die zum Verständnis der Forschungsarbeit benötigt werden.

Dabei erlaube ich mir die gängigen Definitionen: Sprache, Wörter, Natural Language Processing, Algorithmen, etc. wegzulassen und direkt auf externe Quellen zu verweisen:

- [SJN05] als Standardquelle für Sprachverarbeitung
- [Car17] als Standardquelle für Sprachterminologie
- [CLLS09] als Standardquelle für Algorithmen

4.1 Tiefe und flache Sprachverarbeitung

Jede Dialogkomponente baut auf einer Wissensbasis auf, die zum Sprachverständnis beisteuert und zur Dialogentscheidung führt. Dabei unterscheidet [Car17] bei der Architektur von wissensbasierten natürlichsprachlichen Systemen zwischen einer flachen und einer tiefen Verarbeitung:

- Eine tiefe Verarbeitung impliziert eine wissensreiche Analyse von Sprache durch manuell entwickelte Grammatiken.
- Eine flache Verarbeitung bleibt dagegen oberflächlich bei der Analyse der Sprache und basiert auf Statistik oder maschinellen Lernen von Texten. Aus diesem Grund fehlt der flachen Verarbeitung am Verständnis und der Möglichkeit zur Folgerung.

An dieser Definition wird die spätere Dialogführung angelehnt und von der allgemeinen Sprachverarbeitung auf die Dialogführung abstrahiert.

Wie bei der tiefen Verarbeitung taucht die vertikale Dialogführung von der Benutzereingabe über das Dialogsystem bis tief in den jeweiligen NLUI-Fähigkeit zur Bearbeitung ein. Dagegen interessiert die horizontale Dialogführung nicht der Inhalt der jeweiligen NLUI-Fähigkeit, sondern lediglich die Übersicht bzw. die korrekte Auswahl der NLUI-Fähigkeit bzgl. der vorliegenden Benutzeranfrage.

4.2 Sprachparser

Im Rahmen dieser Arbeit wird bei den Sprachparsern zwischen der Artificial Intelligence Markup Language und Aktivontologien unterschieden.

4.2.1 Artificial Intelligence Markup Language

In der Pilotstudie wurde eine XML-basierte Wissensbasis verwendet. Die Idee eine XML-basierte Annotation dafür einzusetzen, wurde durch die *Artificial Intelligence Markup Language (AIML)* [GBM13] motiviert, die eine XML-basierte Annotation für eine musterbasierte natürlichsprachliche Schnittstelle darstellt. A.L.I.C.E entwickelt 1995 von Richard Wallace sowie der Nachfolger Mitsuku entwickelt von Steve Worswick in 2005 basieren auf Artificial Intelligence Markup Language (AIML) [GBM13]. Es gewann den Loebner-Preis im Jahr 2000, 2001 sowie 2004. Mit dem Loebner-Preis werden seit 1991 Computerprogramme ausgezeichnet, die einem Turing-Test über 25 Minuten standhalten. Mitsuku gewann den Loebner-Preis im Jahr 2013 sowie 2016 bis 2019 und wird derzeit von Pandorabots weiterentwickelt. Die Wissensbasis deckt in der aktuellen Version etwa 10.000 Eingaben einer gewöhnlichen Unterhaltung ab¹.

Ein Beispiel in AIML hat die folgende an die XML-angelehnte Syntax:

```
<command> ListOfParameters </command>
```

Hinter `<command>` können folgende Ausdrücke definiert werden:

- `<aiml>` definiert den Anfang und das Ende einer AIML-Datei.
- `<category>` stellt einen Container dar, der die Inhalte der Wissensdatenbank repräsentiert und aus folgenden Elemente besteht:
 - `<pattern>` beinhaltet eine mögliche Eingabe des Benutzers
 - `<template>` zeichnet eine Ausgabe des Chatbots, wenn das `<pattern>` als Eingabe des Benutzers erkannt wird
- `<star>` stellt einen Platzhalter in der Benutzereingabe dar und kann in der Systemantwort für eine explizite Bestätigung einer Benutzereingabe verwendet werden
- `<srai>` ermöglicht verschiedene `<pattern>` anzusprechen

Im Beispiel 4.1 wird von einem Benutzer die Eingabe „Hello Bot“ erwartet und auf diese mit „Hello my friend!“ geantwortet. Dabei wurden die Tags `<aiml>`, `<category>`, `<pattern>` sowie `<template>` verwendet.

```
<aiml version="1.0.1" encoding="UTF-8"?>
  <category>
    <pattern> HELLO BOT </pattern>
    <template>
      Hello my friend!
    </template>
  </category>
</aiml>
```

Quelltextausschnitt 4.1: Beispiel einer einfachen AIML-Datei [GBM13]

Eine explizite Bestätigung einer Benutzereingabe erlaubt der `<star>`-Tag, mit dem auf die Benutzereingabe „A worker is a human“ die Systemantwort „When a worker is not a human?“ erzeugt wird. Dabei ersetzen die beiden Wörter `worker` und `human` die jeweiligen `*`-Platzhalter. `<star/>` kann als eine gleichwertige Alternative zum `*`-Symbol verwendet werden (Siehe Beispiel 4.2).

¹Pandorabots, letzter Aufruf: 24. Juli 2023, <https://github.com/pandorabots/Free-AIML>

```

<aiml version="1.0.1" encoding="UTF-8"?>
  <category>
    <pattern> A * IS a * </pattern>
    <template>
      When a <star index="1"/> IS not a <star index="2"/>?
    </template>
  </category>

  <category>
    <pattern>
      DO YOU KNOW WHO * IS?
    </pattern>
    <template>
      <srai> WHO IS <star/> </srai>
    </template>
  </category>
</aiml>

```

Quelltextausschnitt 4.2: AIML Beispiel mit Tag <star>

Eine erweiterte Modellierung ist durch den <srai>-Tag möglich: (1) Symbolreduzierung, (2) Teile-und-Hersche, (3) Synonymherleitung und (4) Erkennung der Keyword. Symbolreduzierung wird im Beispiel 4.3 gezeigt.

```

<aiml version="1.0.1" encoding="UTF-8"?>
<category>
<pattern> WHO IS ALAN TURING? </pattern>
<template>
Alan Turing was a British mathematician, cryptographer, and computer scientist.
</template>
</category>

<category>
<pattern> WHO IS ALBERT SABIN? </pattern>
<template>
Albert Sabin was the researcher who developed the vaccine.
</template>
</category>

<category>
<pattern> DO YOU KNOW WHO * IS? </pattern>
<template>
<srai> WHO IS <star/> </srai>
</template>
</category>
</aiml>

```

Quelltextausschnitt 4.3: AIML Beispiel mit einem <srai>-Tag

Abbildung 4.1 stellt das Flussdiagramm der Reduktion dar. Dabei wird bei einer Benutzereingabe „Do you know who is *“ auf eine Wildcard * abgebildet, wobei die Wildcard für einen Name steht. Durch die Ausgabe „Who is <star/>“ wird eine weitere Kategorie referenziert und bei einem Treffer mit Alan Turing oder Albert Sabin die Kategorien A oder B ausgeführt.

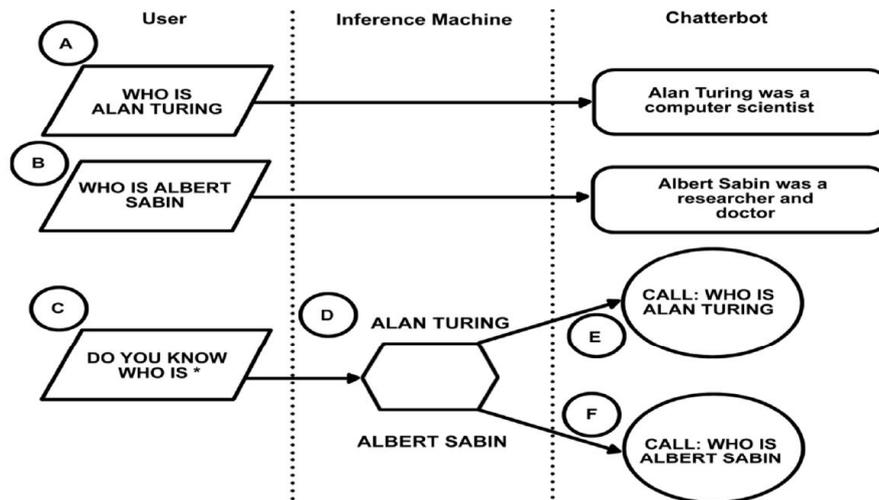


Abbildung 4.1: Flussdiagramm der Reduktion [GBM13]

4.2.2 Aktive Ontologie

Im Allgemeinen repräsentieren Ontologien [BHL⁺14] eine Datenstruktur, in der die Knoten die Konzepte und die Kanten die Beziehungen dazwischen darstellen.

Im Jahr 2006 stellte Guzzoni seine Arbeit zu den Aktiven Ontologien [Guz06] vor. Im Vergleich zu gewöhnlichen Ontologien stellt eine Aktive Ontologie eine Ausführungsumgebung dar. Konzepte sind Schlüsselemente, die Objekte repräsentieren und über Kanäle verbunden sind. Ein Konzept nimmt über einen Kanal eine Nachricht auf und entscheidet, wie es auf diese neuen Informationen mit eigenen Nachrichten reagiert. Der Nachrichtenfluss entsteht durch die Regelsätze, die einem Konzept zugeordnet sind, und einer Faktenbasis, die den aktuellen Stand der Anwendung beschreibt. Dadurch kann ein Konzept bei seiner Reaktion auf eine eingehende Nachricht den aktuellen Kontext in seine Reaktion miteinbeziehen. Eine Ontologie kann im Allgemeinen zu einer Aktiven Ontologie erweitert werden.

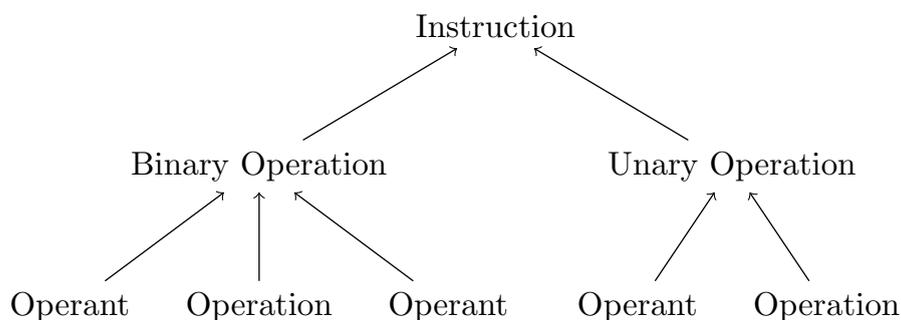


Abbildung 4.2: Beispiel einer Aktiven Ontologie nach [Guz06]

Abbildung 4.2 zeigt ein Beispiel. Damit eine Binäre Operation durchgeführt werden kann, benötigt die Aktive Ontologie die Information zur Operation sowie zu den zwei Operanden. Bei einer Eingabe *Please sum 2 and 3* werden alle Knoten (Operation = sum, Operant = 2, Operant = 3) der Binären Operation aktiviert, so dass der Treffer an den Elternknoten *Instruction* weitergegeben wird.

In der ursprünglichen Arbeit unterscheidet Guzzoni zwei Arten von Konzepten:

- Die Konzeptknoten stellen die Knoten innerhalb des Baumes dar. Sie verarbeiten die eingehenden Informationen von Kindknoten.
- Die Sensoranten repräsentieren die Blätter des Baumes. Beim Eintreten eines Ereignisses leiten sie unter bestimmten Bedingungen Informationen an Ihre Elternknoten weiter.

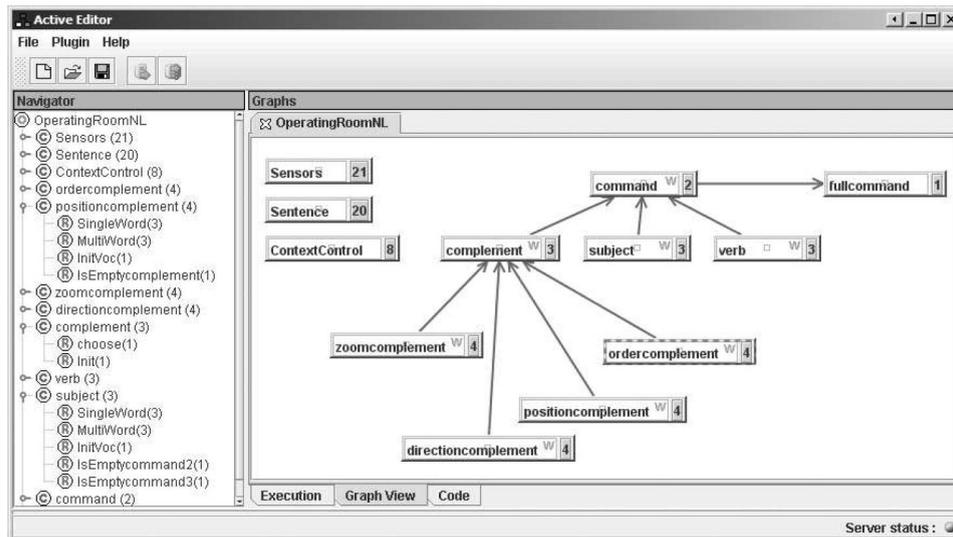


Abbildung 4.3: Beispiel einer Aktiven Ontologie nach [Guz06]

Abbildung 4.3 zeigt eine Aktive Ontologie in einem Active Editor, in dem der Aufbau nachvollzogen und editiert werden kann. Dabei stellt `command` einen Elternknoten mit den Kindern `verb`, `subject` und `complement` dar. Wobei `complement` selbst ein Elternknoten mit den Kindern `zoomcomplement`, `directioncomplement`, `positioncomplement` und `ordercomplement` ist. Damit der `command`-Knoten eine Information weiterleitet, müssen die Kinderknoten aktiviert sein. Das heißt, in einer Benutzereingabe soll die Information erkannt werden, die diesen Knoten zugeordnet werden kann.

4.3 Sprachverarbeitung

Wenn in einer Benutzereingabe unbekannte Wörter vorkommen, die nicht in der AO vorhanden sind, wird zur *Erkennung der Struktur* eine von uns implementierte Sprachverarbeitung eingesetzt. Die folgende Auflistung präsentiert die Verarbeitung der Benutzereingabe in vier Schritten nach [Wie18]:

Erste Regel: die Stichwortsuche

Die erste Regel wird durch eine einfache Schlüsselwortsuche realisiert (Siehe Abbildung 4.4). Diese Regel wird durch bestimmte Schlüsselwörter ausgelöst, das auf eine eindeutige Bezeichnung hindeutet.

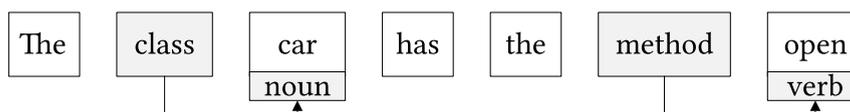


Abbildung 4.4: Beispiel: Stichwortsuche

Der vorliegende Satz *the class car has the method open* enthält sowohl ein Wort, das eine Klasse (Schlüsselwort *class*) darstellt, als auch ein Wort, das eine Methode (Schlüsselwort *method*) beschreibt. Um das übereinstimmende Wort zu ermitteln, werden alle passenden Ziele im laufenden Betrieb ermittelt und die beste Übereinstimmung ausgewählt.

Geeignet in diesem speziellen Fall bedeutet, dass der Teil der Sprache den Erwartungen der jeweiligen Regel entspricht. Zum Beispiel bedeutet *class*, dass das Ziel ein Nomen sein muss, während die Regel für *method* sowohl Verben als auch Nomen erlaubt. Es gibt weitere Unterteilungen, die sowohl für Pluraldeklarationen als auch für indirekte Phrasen verwendet werden können. Alle diese Regeltypen haben einen gemeinsamen Wahrscheinlichkeitswert. Da eine solche Äußerung immer eine explizite Angabe einer Klassifikation enthält, wird die Höchstpunktzahl angewendet. Dadurch soll verhindert werden, dass andere Regeln die Klassifizierung überschreiben, weil der Benutzer explizit angegeben hat, was er erwartet.

Zweite Regel: Suche nach einer Referenz

Die zweite Regel wird nicht nur verwendet, um die Klassifizierung bestimmter Wörter zu bestimmen, sondern ermöglicht es dem System, die Beziehung zwischen ihnen zu identifizieren (Siehe Abbildung 4.5).

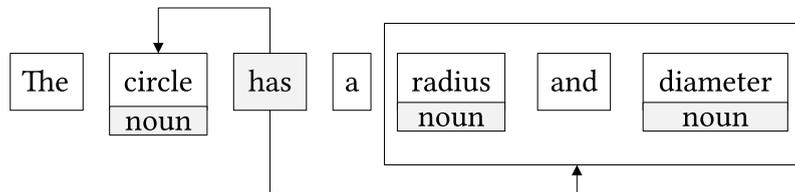


Abbildung 4.5: Beispiel: Beziehungszuordnung

In diesem Fall wird nicht nur das Objekt *circle* extrahiert, sondern auch Attribute erkannt, die den Kreis detailliert beschreiben. Das System kann zwei mögliche Fälle interpretieren:

1. Die Klassifizierung von *circle*, *radius* und *diameter* als Klassen
2. Die Klassifizierung von *circle* als Klasse sowie eine Verbindung zwischen *circle* und seinen Attributen *radius* und *diameter*.

Diese Regel wird nach folgendem Muster angewendet: Zunächst wird nach einem Schlüsselwort gesucht, das entweder eine Eigentümerbeziehung oder eine Fähigkeitsbeziehung beschreibt. Für jede Eigentumsbeziehung werden alle in Frage kommenden Attribute als solche markiert und der entsprechenden Klasse zugewiesen. In Abbildung 4.5 sind sowohl *radius* als auch *Durchmesser* als Attribute markiert und mit *circle* verbunden. Das Verfahren zum Erkennen der Fähigkeitsbeziehung wird auf ähnliche Weise durchgeführt, z. B. kann in der Eingabe *A car drives* die Methode *drive* zur Klasse *car* hinzugefügt werden.

Dritte Regel: Grammatikbasierte Übereinstimmung

Aufgrund der Mehrdeutigkeiten in der Beschreibung von Klassen und Attributen geht das System davon aus, dass der Benutzer eine Verbindung zwischen Klassen oder eine Zuordnung von Attributen beschrieben hat. Daher werden alle ermittelten Klassifizierungen mit einer niedrigeren Bewertung verglichen. Dies ermöglicht der natürlichsprachlichen Schnittstelle, die Annahme im späteren Verarbeitungsschritt neu zu bewerten.

Der Regelsatz analysiert eine Äußerung in natürlicher Sprache. Dies umfasst sowohl die Klassifizierung nach Wortarten als auch das Analysieren von Beziehungen zwischen Wörtern, die von der Grammatik abgeleitet werden. Dieser Regeltyp dient zwei verschiedenen Zwecken:

- Erstens soll es eine Rückfallregel für die Wortklassifizierung sein. Daher wird jedes Substantiv als Klasse, jedes Adjektiv als Attribut und schließlich jedes Nicht-Hilfsverb als Methode gekennzeichnet.
- Zweitens ist die grammatikalische Zusammensetzung eines Satzes hilfreich, um die Beziehungen zwischen Wörtern in einem Satz zu bestimmen (Siehe Abbildung 4.6).

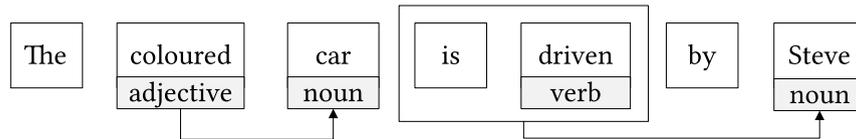


Abbildung 4.6: Beispiel: grammatikbasiertes Matching

Es enthält zwei verschiedene Beziehungen. Wenn ein Nomen ein beschreibendes Adjektiv hat, zum Beispiel *das farbige Auto*, ist das lokalisierte Adjektiv ein Attribut, das direkt mit der darstellenden Klasse verbunden ist. Außerdem bestimmt diese Grammatikregel den Akteur eines Satzes, dem das Verb zugeordnet werden kann. Aufgrund der englischen Grammatik besteht eine Phrase aus mindestens einem oder mehreren Subjekten, gefolgt von einem Verb und kann um Objekte erweitert werden. Diese Komposition kann jetzt verwendet werden, um den Akteur in jeder Phrase zu bestimmen. Dazu wird ermittelt, ob der verarbeitete Satz in aktiver oder passiver Form geschrieben ist.

Die Analyse der Sätze erfolgt anhand der vorgestellten Muster. Dabei funktioniert die Erkennung auch, wenn die Wörter in der Eingabe nicht in der Aktivontologie vorhanden sind. Abbildung 4.6 zeigt ein Beispiel. *Steve* gilt als ein Akteur, der das *Auto fährt*. Dabei befinden sich die Wörter nicht in der Aktivontologie. Mit diesem grundlegenden Regelsatz können Annahmen über die Beziehungen zwischen den in den einzelnen Eingabesätzen enthaltenen Wörtern getroffen werden.

Vierte Regel: Erkennung einer Beziehung

Die vierte Regel wird verwendet, um die Beziehungen zwischen zwei Entitäten zu bestimmen, die als Klassen in und zwischen Sätzen betrachtet werden. Wir unterscheiden drei verschiedene Arten von Beziehungen:

- *is-a*-Beziehung beinhalten die von UML bekannte Vererbung und Instanzbeziehung. Der Name erklärt sich daraus, dass diese durch den einfachen Satz *A is a B* in natürlicher Sprache ausgedrückt werden können.
- in den zugrundeliegenden sprachlichen Beschreibungen konzentrieren wir nur Aggregationen. Die Kompositionen werden vernachlässigt.
- Die dritte Art von Beziehung sind Assoziationen. Wenn nicht entschieden werden kann, ob es sich um eine Vererbung oder eine Instanzbeziehung handelt, findet eine Interaktion mit dem Benutzer statt.

4.4 Ähnlichkeitsmaße

Für die Klassifikation einer neuen Eingabe zu den vorhandenen NLU-Fähigkeiten wird auf eine Kombination der ausgewählten Klassifizierer von DKPro Similarity [BZG13] zurückgegriffen. Wir verwenden das open-source Framework DKPro Similarity im standalone Modus. Dies bedeutet, dass die Ähnlichkeitsmaße durch die Bereitstellung zweier Texte als String berechnet werden:

```
String[] tokens1 = "This is a short example text ".split(" ");
String[] tokens2 = "A short example text could look like that ".split(" ");

double score = measure.getSimilarity(tokens1, tokens2);

System.out.println("Similarity: " + score);
```

Das Token-Array beinhaltet die einzelnen Wörter der Texteingabe, zu denen die Ähnlichkeit berechnet wird. Dabei betrachten wir mehrere Ähnlichkeitsmaße als Bündel und untersuchen neben der lexikalischen Ähnlichkeit auch die semantische Ähnlichkeit. Das Bündel berechnet den Durchschnittswert aller Ähnlichkeitsmaße. Dadurch wird eine falsche Berechnung durch die Berechnung des Durchschnittswert aller Ähnlichkeitsmaße geglättet. Alle Algorithmen sind auf Github² veröffentlicht.

4.4.1 Lexikalische Ähnlichkeitsuntersuchung

Ein lexikalisches Verfahren basiert auf der Untersuchung von einzelnen Wörtern und berücksichtigt den Zusammenhang der Wörter im gesamten Text nicht.

Die Grundlage für den Vergleich zweier Wörter ist der Jaccard-Koeffizient, der die Anzahl des Schnittes der Wortmenge beider Sätze geteilt durch die Menge der Vereinigung berechnet. Diese Schnittmenge besteht dabei aus allen Wörtern, die in beiden Sätzen vorkommen. Bei zwei identischen Sätzen ergibt die Berechnung unabhängig von der Reihenfolge der Wörter genau 1.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

4.4.1.1 N-Gramme mit Jaccard-Koeffizient

Wir verwenden den Ansatz zur Ähnlichkeitsuntersuchung von einem bereits vorliegenden Trainingssatz zu einer neuen Eingabe der Endbenutzer. Dabei werden die Sätze in N aufeinanderfolgende Fragmente - in unserem Fall sind es Wörter - zerlegt. In unserem Fall werden Bigramme verwendet und durch den Jaccard-Koeffizient die Ähnlichkeit zwischen den Wörtern aus den beiden Sätzen berechnet.

4.4.1.2 MongeElkan mit Jaccard-Koeffizient

MongeElkan [ME01] ist ein hybrides Ähnlichkeitsmaß für den Vergleich zweier Sätze, bei dem die Reihenfolge der Wörter keine Rolle spielt. Somit berechnet es, wie gut die Wörter eines Satzes im Durchschnitt zu den Wörtern eines anderen Satzes passen.

²Github DKPro Similarity, letzter Aufruf: 24. Juli 2023, <https://github.com/dkpro/dkpro-similarity>

4.4.1.3 Kosinus-Ähnlichkeit

Die Kosinus-Ähnlichkeit definiert ein Maß für die Ähnlichkeit von zwei Vektoren [Sin01]. [BZG13] verwendet das Maß, um die Ähnlichkeit von zwei Sätzen zu berechnen. Laut der Definition hat der Vektor eines Satzes an den Stellen eine 1, wenn es das Wort enthält, ansonsten eine 0. Die Vektoren sind auf die Länge 1 normiert. Dies bedeutet, dass das Ähnlichkeitsmaß nur vom Winkel zwischen den Vektoren und nicht von deren Größe abhängig ist. Zeigen die Vektoren in die gleiche Richtung - stimmen also die Sätze überein - ist die Kosinus-Ähnlichkeit gleich eins. Bei unabhängigen Eingaben, ergeben sich orthogonalen Vektoren, deren Kosinus-Ähnlichkeit bei der Berechnung 0 ergibt. Die Gewichtung kann bei Vergleich der Sätze nicht negativ werden, so dass die Kosinus-Ähnlichkeit stets zwischen 0 und 1 liegt.

$$\text{CosinusSimilarity} = \cos(\gamma) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i^2} \sqrt{\sum_{i=1}^n b_i^2}}$$

4.4.1.4 Greedy String Trilling

Greedy String Trilling aus [BZG13] betrachtet den längsten zusammenhängenden Teil, den beide Textkandidaten gemeinsam haben. Das Verfahren wurde aus Jplag von Prechelt [PMP00] übernommen und erstellt ein Maß für die Ähnlichkeit zweier Texte über die Anzahl der selben Wörter.

Die Berechnung erfolgt laut [PMP00] in zwei Phasen:

- Phase 1:
Die beiden Zeichenketten werden auf die längste Übereinstimmung untersucht. Hierbei wird jeder Token aus dem ersten Text mit jedem Token aus dem zweiten verglichen. Ist eine Übereinstimmung gefunden, wird das folgende Token angeschaut. Somit findet die Methode alle Zeichenketten, die sich übereinstimmen.
- Phase 2:
Um Wiederholungen bei den Übereinstimmungen in der Phase 1 zu vermeiden, werden die gefunden Zeichenketten markiert. Das bedeutet, dass bei einem weiteren Treffer im Text, nur die erste Übereinstimmung gilt und weitere nicht mehr betrachtet wird.

Weitere Erläuterung zum Greedy String Trilling ist in [PMP00] zu finden.

4.4.2 Semantische Ähnlichkeitsuntersuchung

Zur Analyse der semantischen Ähnlichkeit verwendet [BZG13] ein konzeptvektorbasiertes Maß in Kombination mit der Konzeptdarstellung von Wikipedia und Wordnet [ZMG08].

Das Wissen über die semantischen Zusammenhänge kann aus Beschreibungen extrahiert werden. Diese Beschreibungen enthalten viele Konzepte, die von verschiedenen Menschen formuliert wurden. Anhand der Häufigkeit eines Wortes in der Beschreibung wird ein Index erstellt und für einen Vergleichswert auf der Menge der Wörter gewichtet. Aus der vorliegenden Beschreibung kann ein Vektor berechnet werden, der die gewichteten Konzepte der Beschreibung enthält [GM07].

Beispiel:

```
sentenceToVector = new HashMap<>();  
..  
public double getSimilarity(String sentence1, String sentence2)  
{  
    List<String> wordsArray1 = Arrays.asList(sentence1.split(" "));  
    vector1 = reader.getVector(wordsArray1);  
    sentenceToVector.put(sentence1, vec1);  
  
    List<String> wordsArray2 = Arrays.asList(sentence2.split(" "));  
    vector2 = reader.getVector(wordsArray2);  
    sentenceToVector.put(sentence2, vec2);  
  
    return innerProduct.apply(vec1, vec2) / normalization.apply(vec1, vec2);  
}
```

5. Natürlichsprachliche Schnittstelle

Dieses Kapitel stellt passend zur Zielsetzung der Dissertation im Kapitel 1.2 sowie den verfassten Forschungsfragen im Kapitel 3.5 die Lösungsansätze für die natürlichsprachliche Schnittstelle vor.

Die Forschungsarbeit berichtet von einer Implementierung der natürlichsprachlichen Schnittstelle, die den Namen *JustLingo NLUI* trägt.

Der Name JustLingo ist vom Englischen *Just* und *Language* abgeleitet. Somit dreht sich bei der Forschung alles *nur* um die *Sprache*. Die Abkürzung NLUI steht für den Englischen Begriff einer natürlichsprachlichen Benutzerschnittstelle, also **N**atural **L**anguage **U**ser **I**nterface.

Um eine interaktive Kommunikation mit einer natürlichsprachlichen Benutzerschnittstelle zu ermöglichen, wird als zentrale Komponente ein Dialogsystem benötigt. Das Dialogsystem steuert die einzelnen Komponenten von *JustLingo NLUI* an, die durch eigene Funktionalität einen Beitrag zur Lösung einer Benutzeranfrage bereitstellen. Dabei kommt es auf die Anfrage an, ob eine Komponente diese alleine oder im Zusammenspiel mit einer anderen Komponente bearbeitet.

Die jeweiligen Komponenten von *JustLingo NLUI* werden im Folgenden vorgestellt. Kapitel 5.1 zeigt eine allgemeine Übersicht von *JustLingo NLUI*. Die zentrale Komponente zum Aktiven Dialogmanagement führt das Kapitel 5.2 ein, das bei *JustLingo NLUI* die NLP-Fähigkeiten und den NLUI-Fähigkeiten steuert. Zu den NLP-Fähigkeiten im Kapitel 5.3 gehören im Rahmen unserer Forschung ein AO-basierter Sprachparser in Kapitel 5.3.2, Klassifikation in Kapitel 5.3.3 und dynamischen Wortschatzerweiterung in Kapitel 5.3.4. Durch die Klassifikation werden der unbekannteren Benutzereingaben zu den vorhandenen NLUI-Fähigkeiten zugeordnet. Nicht zuletzt erläutert dynamischen Wortschatzerweiterung die Methode im Umgang mit Wörtern, die nicht in der Aktivontologie vorhanden sind oder als mehrdeutig erkannt werden. Die NLUI-Fähigkeiten werden im Kapitel 5.4 präsentiert. Um Anfragen der Benutzer in natürlicher Sprache über das NLUI auf Daten zu ermöglichen, erfolgt im Kapitel 5.4.2 die Vorstellung der Datenstrukturanalyse. Zum Schluss beschreibt Kapitel 5.4.3 eine Methode der Erfassung von Folgenalgorithmen.

5.1 Übersicht

Ein Benutzer kann mittels geschriebener, natürlicher Sprache eine Anfrage an das NLUI stellen und erhält eine passende Rückmeldung zurück. Eine zentrale Rolle spielt bei JustLingo NLUI ein Dialogmanager, der die internen Fähigkeiten der natürlichsprachlichen Schnittstelle zu einer Benutzereingabe ansteuert. Wir unterscheiden zwischen den *NLP-Fähigkeiten*, die dazu dienen, eine Absicht zu der jeweiligen Eingabe des Benutzers herauszufinden, und den Fähigkeiten der natürlichsprachlichen Schnittstelle, welche die Gesamtfunktionalität von *JustLingo NLUI* definieren, und später als *NLUI-Fähigkeiten* näher vorgestellt werden.

Unter anderem ist *JustLingo NLUI* dafür zuständig,

- (i) einen Dialog mit dem Benutzer aufzubauen,
- (ii) unbekannte oder mehrdeutige Eingaben eines Benutzers zu klären,
- (iii) eine Benutzeranweisung durch ein gezieltes Hinterfragen zu spezifizieren,
- (iv) Fehler und Ungenauigkeiten in der Eingabe herauszufinden und diese einem Benutzer mitzuteilen.

Insbesondere strebt *JustLingo NLUI* bei Unklarheiten, fehlenden Informationen und Mehrdeutigkeiten statt der Ausgabe einer Fehlermeldung eine Interaktion mit dem Benutzer an. Die Akzeptanz einer natürlichsprachlichen Schnittstelle steigt, wenn es nicht nur Fehlermeldungen ausgibt, sondern einen Benutzer anleitet, es zu vermeiden.

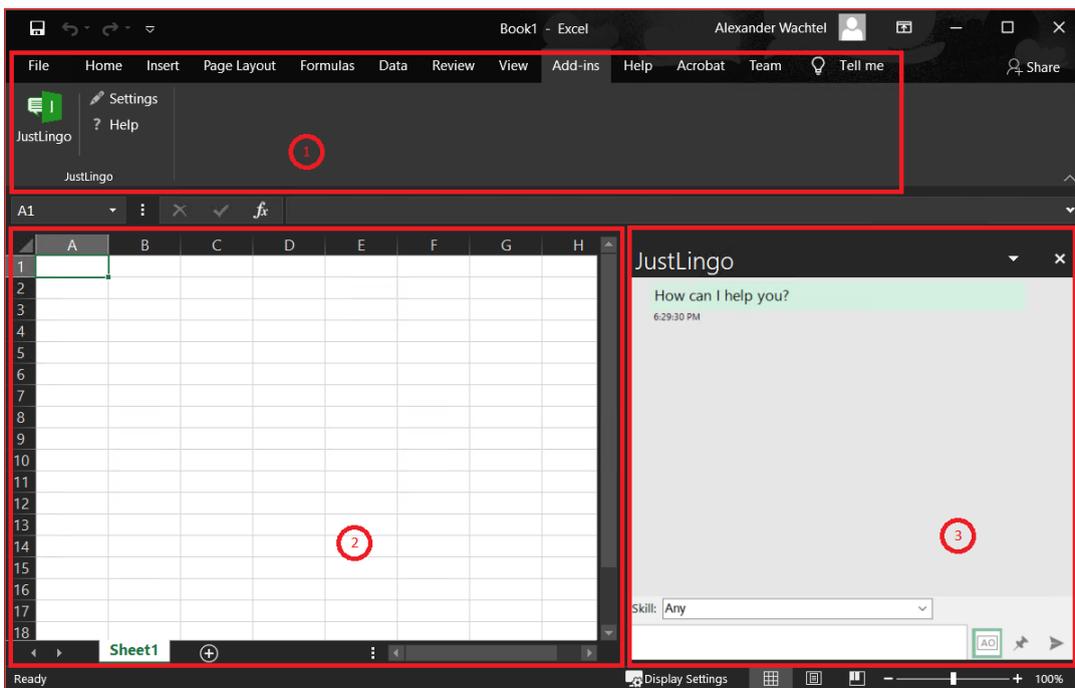


Abbildung 5.1: Die Integration von JustLingo NLUI in Microsoft Excel

JustLingo NLUI integriert sich als ein Add-In in das Microsoft Excel (Siehe Abbildung 5.1) und ist für eine lokale Installation von Microsoft Office mit der englischen Sprachedition verfügbar. In der Befehlsleiste (1) können wir unter Add-ins JustLingo NLUI sehen. Beim Betätigen des Buttons *JustLingo* kann das Chatfenster (3) ein- und ausgeblendet werden. In (2) befinden sich die Daten einer gewöhnlichen Microsoft Excel Arbeitsmappe. Durch das Chatfenster (3) können Anweisungen an *JustLingo NLUI* abgegeben werden.

5.1.1 Komponenten

JustLingo NLUI wurde mit Visual Studio in der Programmiersprache C# entwickelt und umfasst knapp 35.000 ausführbare Quellcodezeilen, die auf 21 Projekte verteilt sind. Dabei zeichnet sich die Software durch einen Wartbarkeitsindex von 78,55 und einer Klassenkopplung von 2,6 aus¹. Die folgende Auflistung zeigt die jeweiligen Komponenten:

1. *JustLingo* beinhaltet als Hauptprojekt einen *JustLingo.DialogManager* und das Excel Add-In.
2. *JustLingo.AO* stellt den Spracherkenner mit der Aktivontologie dar. Zur manuellen Erzeugung und Pflege von Aktivontologien für die jeweiligen Dienste in JustLingo NLUI steht der von uns entwickelte *JustLingo.AOEditor* zur Verfügung.
3. Damit *JustLingo.DialogManager* eine Benutzereingabe zur jeweiligen NLUI-Fähigkeit korrekt zuordnet, berechnet *JustLingo.Classification* den Ähnlichkeitsgrad zwischen den vorhandenen Beispieldaten der jeweiligen NLUI-Fähigkeiten und der Benutzereingabe.
4. *JustLingo.Facade* kapselt die NLUI-Fähigkeiten. Über Schnittstellen werden die NLUI-Fähigkeiten dem *JustLingo.DialogManager* mitgeteilt.
5. *JustLingo.Algorithms* dient zur Interpretation von Folgenalgorithmen mit Variablen, Schleifen und Bedingungen aus natürlicher Sprache.
6. *JustLingo.DataStructure* wird zur Erkennung von Klassen mit Attributen und Methoden aus den Beschreibungen der Benutzer verwendet. *JustLingo.DataRelations* untersucht die Eingabe auf mögliche Beziehungen zwischen den Sätzen.

JustLingo.Facade ist eine Fassade nach dem Entwurfsmuster aus der Vorlesung Software-technik und hat als Ziel die Schnittstelle zu den Subsystemen zu vereinfachen.

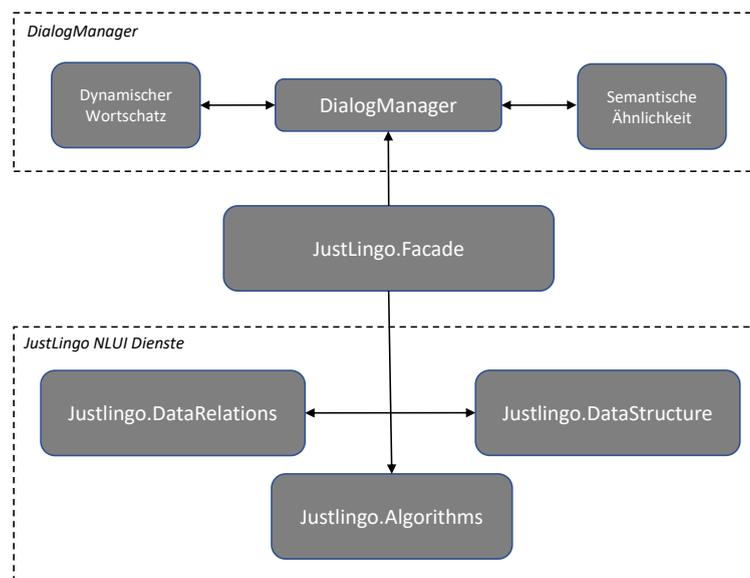


Abbildung 5.2: Vereinfachte Darstellung von *JustLingo.Fassade*

¹Quelle: Code Metriken, Microsoft Dokumentation, letzter Aufruf: 24. Juli 2023
<https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values>
 Wartbarkeitsindex: Wertebereich von 0 bis 100; sehr schlechte Wartbarkeit bei einem Wert unter 10, mittlere Wartbarkeit bei unter 20 und eine gute bis sehr gute Wartbarkeit von 20 bis 100.
 Klassenkopplung: Wertebereich von 0 bis ∞ ; je niedriger der Wert, um so loser die Kopplung.

Komplexe Fähigkeiten von *JustLingo NLUI* können somit leicht durch weitere NLP-Fähigkeiten sowie NLUI-Fähigkeiten erweitert werden. Der *JustLingo.Dialogmanager* benutzt somit die *JustLingo.Fassade*, die als Schnittstelle die verfügbaren NLUI-Dienste über eine einheitliche Schnittstelle zur Verfügung stellt. Dabei spielt die *JustLingo.Fassade* für die NLUI-Fähigkeiten eine wichtige Rolle, weil bei der Entwicklung einer weiteren NLUI-Fähigkeit die Fassade dem Entwickler die Vorgaben des Gesamtsystems präsentiert. Durch den Aufbau ist eine leichte Erweiterbarkeit der natürlichsprachlichen Schnittstelle mit zukünftigen Fähigkeiten gegeben. Durch die *JustLingo.Fassade* sind die NLUI-Fähigkeiten dem *JustLingo.DialogManager* nicht direkt zugänglich. Alle NLUI-Fähigkeiten werden durch eine zentrale Schnittstelle angesprochen. Dies dient der Robustheit des Gesamtsystems bei *JustLingo NLUI* und senkt die Fehlerquellen bei der Erweiterung der natürlichsprachlichen Schnittstelle.

5.1.2 Softwarearchitektur

Als Softwarearchitektur wurde bei *JustLingo NLUI* das Entwurfsmuster Model-View-Presenter (MVP) gewählt. Abbildung 5.3 zeigt die Aufteilung der MVP-Architektur in Präsentations-, Steuerungs- sowie Geschäftslogikschicht. Die Darstellung weiterer Schichten, wie Datenzugriff- bzw. Datenhaltungsschicht, wird zwecks der Übersichtlichkeit vernachlässigt. Die Ansicht stellt die Daten dar und ist für die Interaktion mit dem Benutzer zuständig und keine steuernde Logik. Das Modell enthält die Geschäftslogik und kennt weder die Ansicht noch den Präsentierer. Die zuletzt genannte Komponente übernimmt die Steuerung zwischen dem Modell und der Ansicht. Durch den Aufbau wird ein Vorteil des Entwurfsmusters MVP deutlich: die Geschäftslogik lässt sich unabhängig von der Ansicht erweitern. Dies werden wir im Verlauf des Kapitels noch verdeutlichen.

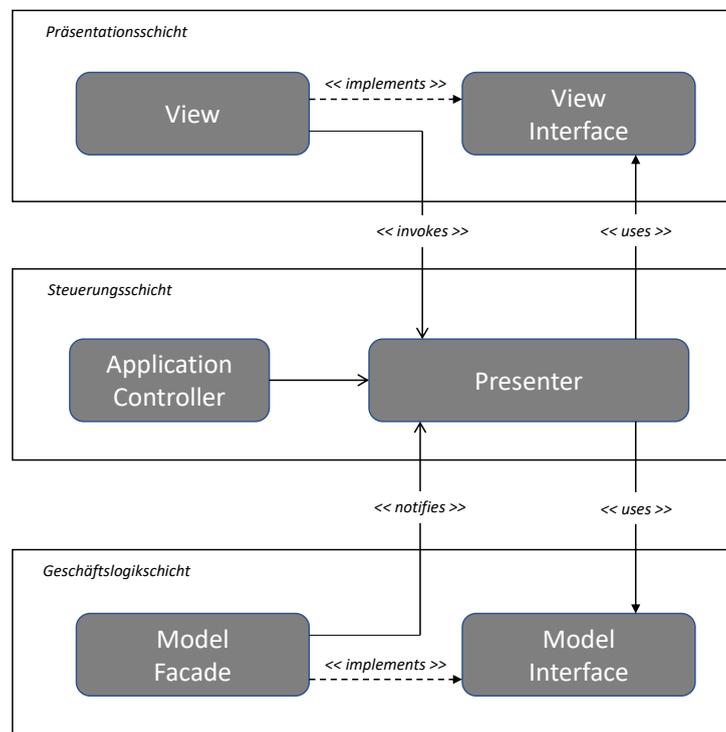


Abbildung 5.3: Schichtenstruktur der MVP-Architektur von *JustLingo NLUI*

5.1.2.1 Präsentationsschicht

Die Präsentationsschicht bildet die Ansicht der MVP-Architektur ab und stellt Daten für einen Benutzer dar. Des Weiteren dient es als Schnittstelle für die Eingaben des Benutzers sowie die Ausgabe der Software. Wie in Abbildung 5.4 dargestellt wird, enthält die Präsentationsschicht keine Programmlogik.

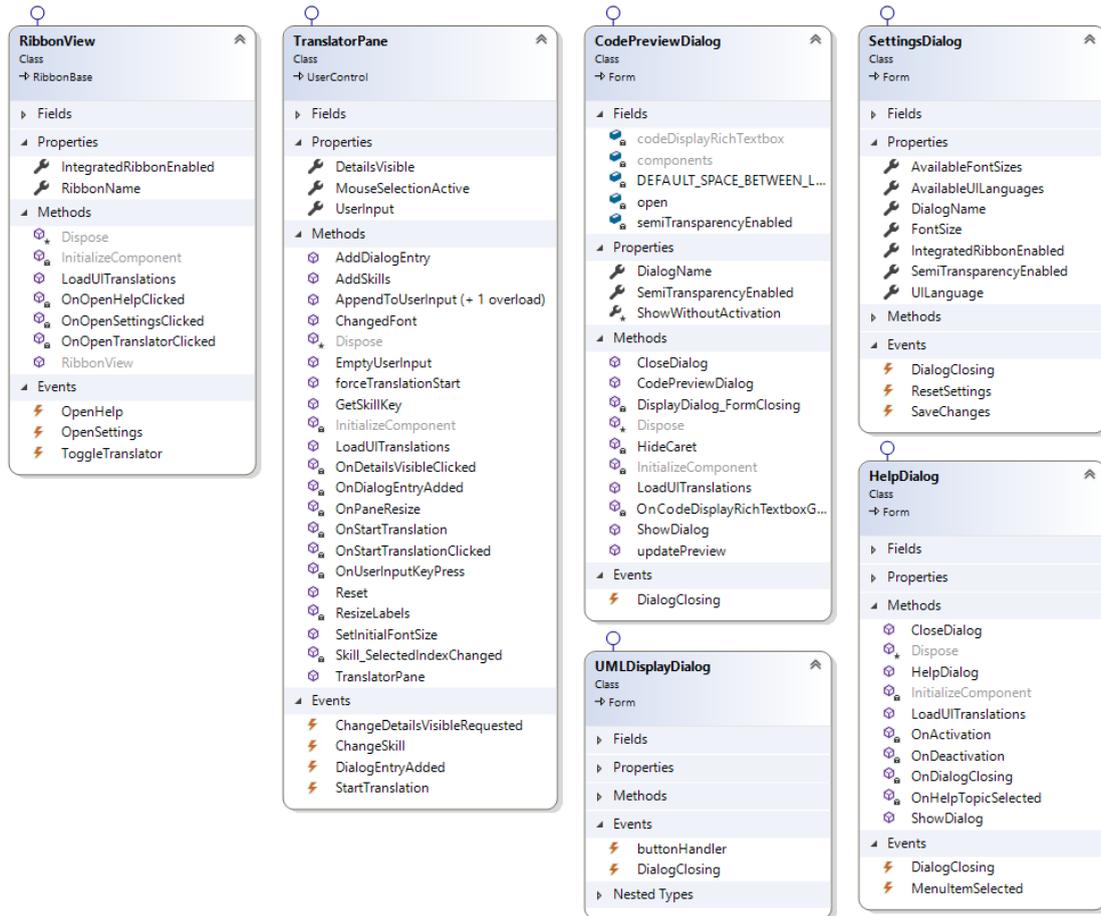


Abbildung 5.4: Auflistung der einzelnen Dialogfenster in der Ansicht

Aus Abbildung 5.1 kann folgendes entnommen werden: die Befehlsleiste (1) ist als Ribbon-View implementiert und ermöglicht uns die Einstellungen (HelpDialog und SettingsDialog) zum Add-in *JustLingo NLUI* anzuschauen. Beim Betätigen des Buttons *JustLingo* kann das Chatfenster (3) ein und ausgeblendet werden. Das Chatfenster (3) ist als TranslatorPane implementiert und kann die Anweisungen entgegennehmen und eine Ausgabe anzeigen. Des Weiteren sind die Hilfsdialoge CodePreview und UMLDisplayDialog implementiert, die in den Vordergrund treten, sobald die jeweilige NLUI-Fähigkeit tätig wird.

Zum Austausch werden die Schnittstellen des Pakets *ViewInterface* implementiert, die aus Einfachheit nicht grafisch dargestellt werden. Die Komponente *View* beinhaltet die Darstellung des Add-Ins innerhalb des Ribbons sowie die Dialogfenster. Der Ribbon und die Dialoge implementieren je ein Interface aus der Komponente *ViewInterface*, welches den Zugriff auf die Darstellungselemente steuert.

5.1.2.2 Steuerungsschicht

Die Steuerungsschicht bildet den Presenter der MVP-Architektur ab und beinhaltet die Komponenten *Presenter* und *ApplicationController* (Siehe Abbildung 5.5). Die Komponente *Presenter* dient als Bindeglied der Steuerung zwischen Präsentations- und Geschäftslogikschicht. *ApplicationController* regelt beim Programmstart das Initialisieren der *Presenter*. Für den Ribbon und jedes Dialogfenster wird ein Presenter erzeugt.

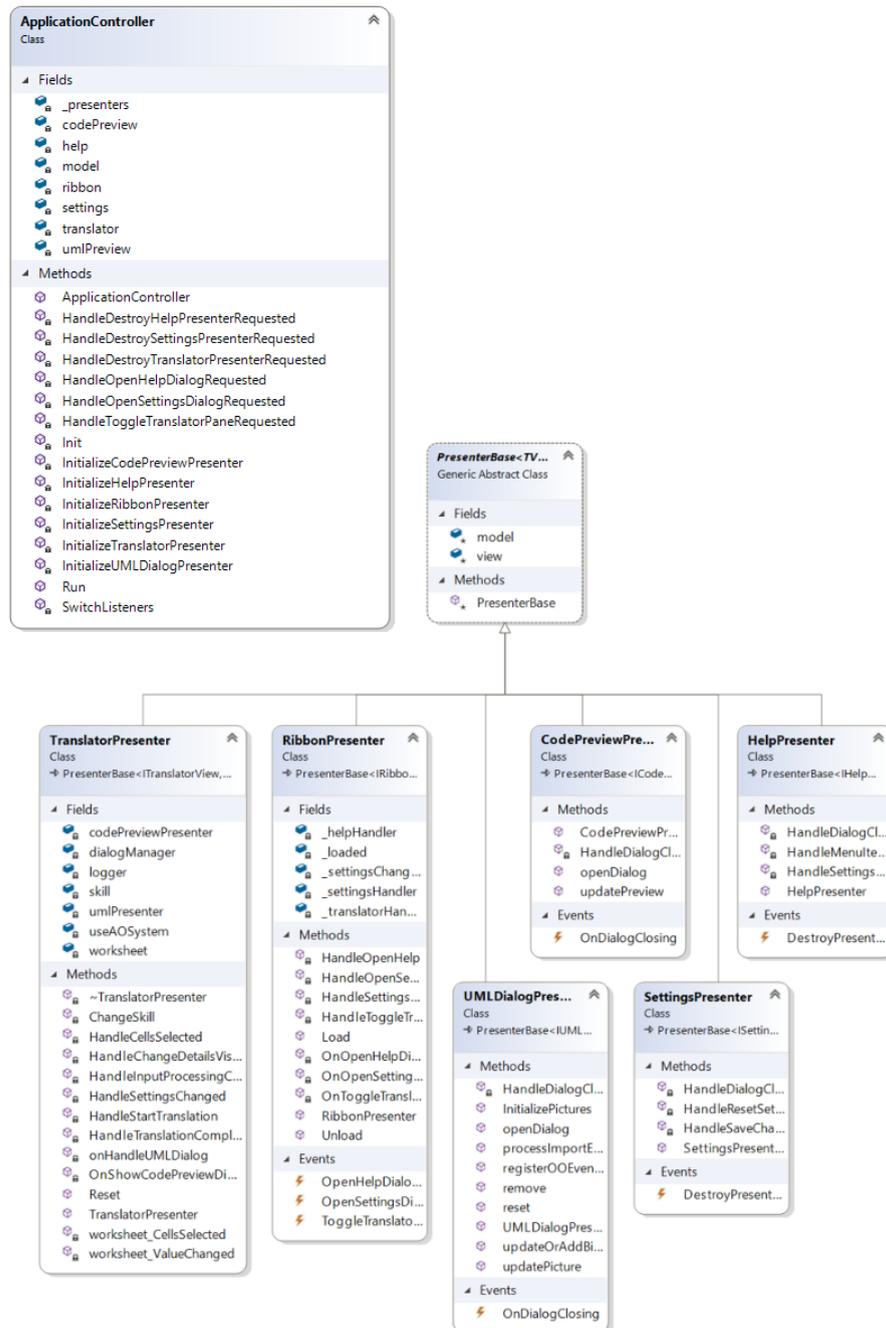


Abbildung 5.5: ApplicationController und Presenter im Überblick

5.1.2.3 Geschäftslogikschicht

Die Geschäftslogikschicht stellt einen Teil des Modells der MVP-Architektur dar und dient dem Zugriff der jeweiligen *Presenter* auf die Geschäftslogik (Siehe Abbildung 5.6). Dafür stehen in der Komponente *ModelInterface* unterschiedliche Schnittstellen zur Verfügung, die jeweils den Zugriff auf bestimmte Teile der Geschäftslogik und der Daten erlauben. Die Schnittstellen werden von der gemeinsamen Fassade *ModelFacade* implementiert. Dies stellt sicher, dass alle *Presenter* stets auf den selben aktuellen Daten arbeiten. Die *ModelFacade* sendet Ereignisse an die *Presenter*, um Änderungen der Daten zu signalisieren.

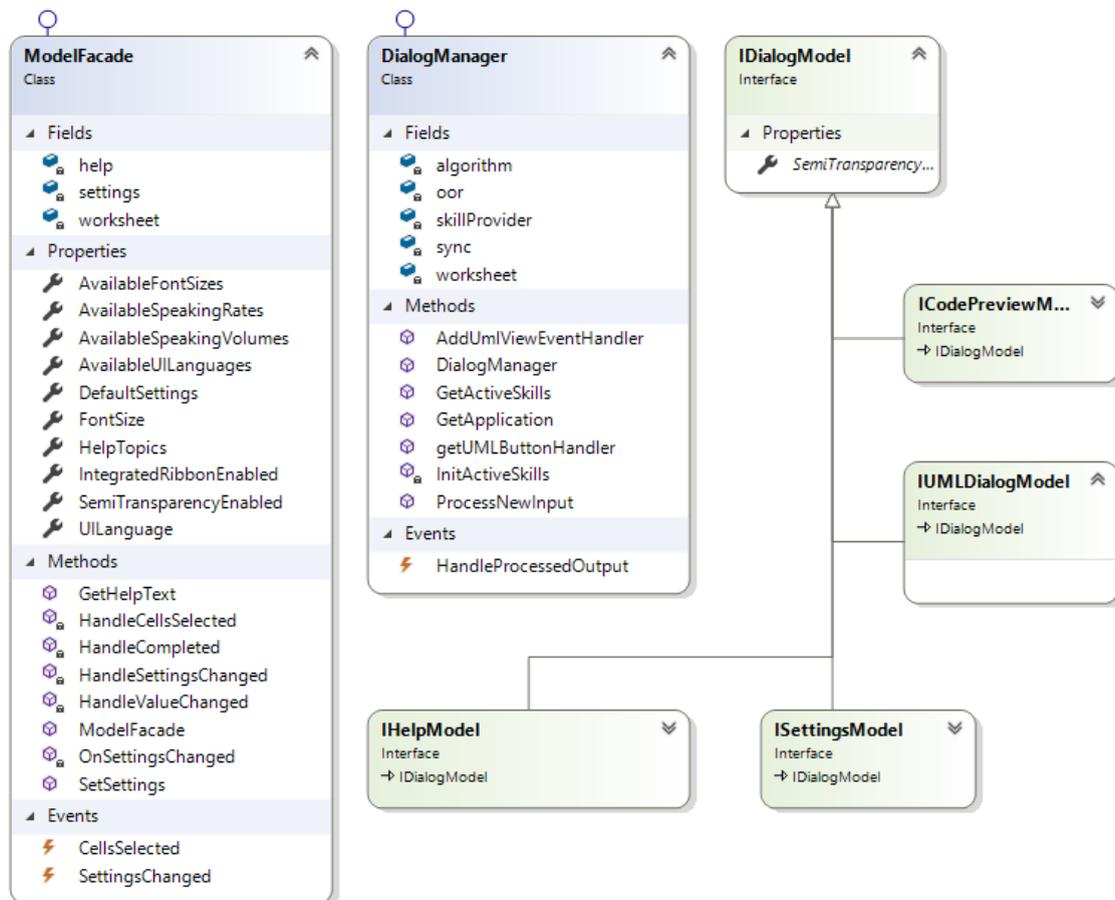


Abbildung 5.6: Die Komponenten der Geschäftslogikschicht im Überblick

Eine zentrale Komponente in *JustLingo NLUI* stellt *JustLingo.DialogManager* dar. Die vom *TranslatorPresenter* gesteuerte Komponente regelt den Aufruf der bereitgestellten NLP-Fähigkeiten sowie implementierten NLUI-Fähigkeiten. *JustLingo.DialogManager* wird in diesem Kapitel als nächstes erläutert.

5.2 Aktives Dialogmanagement

Dieses Kapitel stellt mit dem *JustLingo.Dialogmanager* eine zentrale Komponente *JustLingo NLUI* vor, die den Fokus auf eine natürlichsprachliche Interaktion zwischen dem Benutzer und den Komponenten legt.

5.2.1 Zweck

Durch die natürliche Sprache ist es möglich, in einer nicht-ingeschränkten und nicht-technischen Sprache mit dem System zu kommunizieren. Dabei entsteht ein Dialog, bei dem beide Teilnehmer zur Interaktion beitragen.

Ein *JustLingo NLUI* hilft einem unerfahrenen Benutzer, fehlende Informationen in den jeweiligen Eingaben zu identifizieren und Schritt-für-Schritt in einem Dialog zu vervollständigen.

Abbildung 5.7 stellt den Ablauf nach einer Benutzereingabe grafisch dar:

- Die Benutzereingabe wird zuerst durch den Dialogmanager bearbeitet. Dieser ist für die Dialogführung zuständig und bedient sich mehrerer Hilfsdienste, wenn die Absicht hinter einer Anfrage nicht trivial herausgefunden werden kann und die Eingabe Einzelwörter aufweist, die nicht in der Aktivontologie enthalten sind.
- Wenn *JustLingo NLUI* die Absicht des Benutzer identifiziert hat, folgt eine Bearbeitung der Anfrage durch die NLUI-Fähigkeiten. Zur Bearbeitung stehen diesen die aktuellen Eingabehistorie sowie das Excel mit den Daten im Arbeitsblatt und die spezifische NLU-Aktivontologien der jeweiligen NLUI-Fähigkeit zur Verfügung.
- Als letztes erfolgt in *JustLingo NLUI* eine Antwortgenerierung, die mittels einer NLG-Aktivontologie eine Antwort generiert.

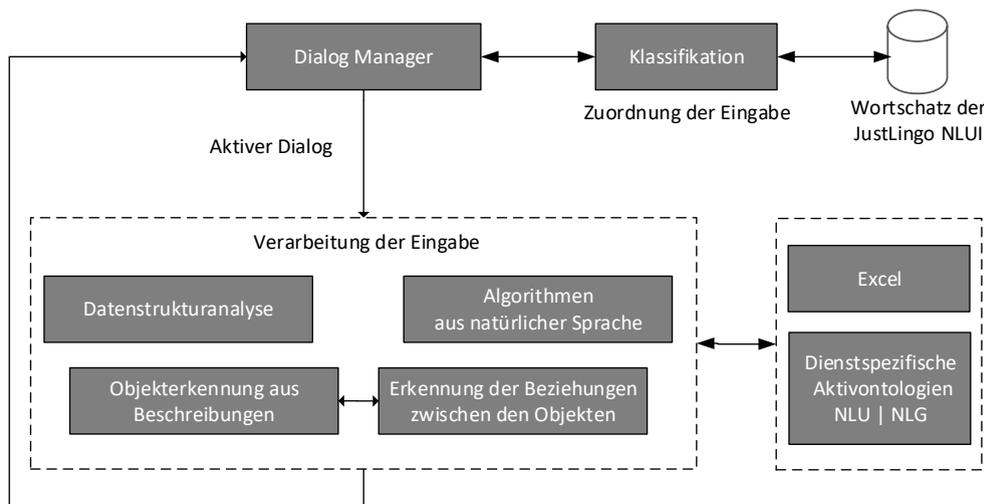


Abbildung 5.7: Ablauf einer Benutzereingabe bei *JustLingo NLUI*

In einer natürlichsprachlichen Konversation können Mehrdeutigkeit in der Zuordnung einer Eingabe zur NLUI-Fähigkeiten auftreten. Dies wird u.a. durch das aktive Dialogmanagement verhindert, denn bei einer Rückfrage erwartet diese aktive NLUI-Fähigkeit eine Weiterverarbeitung der Eingabe. Bei der Antwortgenerierung wird die aktuelle Eingabehistorie gespeichert und im Fehlerfall eine Aufgabe für einen Benutzer aus den verstandenen Teilen der Eingabe formuliert.

5.2.2 Geteilte Initiative

Eine der Kerneigenschaften des Dialogsystems bildet die geteilte Initiative.

Bei einer geteilten Initiative sind Benutzer als auch *JustLingo NLUI* in der Kommunikation gleichberechtigte Partner und steuern beide zum Dialog bei, ohne aufgefordert zu werden. Eine geteilte Initiative wird durch eine Aktivontologie begünstigt, weil dadurch *JustLingo NLUI* fehlende Informationen in der Eingabe erkennt und aktiv hinterfragen kann.

Eine geteilten Initiative erweitert eine gemischte Gesprächsinitiative (Vrgl. [Ber12]) durch einen gezielten Beitrag zum Dialog. Durch die geteilte Initiative hat *JustLingo NLUI* die Fähigkeit, einen zielgerichteten Dialog mit einem Benutzer aufzubauen. Dabei strebt *JustLingo NLUI* bei geteilter Initiative einen perfekten Dialog an: 50% Benutzer zu 50% *JustLingo NLUI*, wobei die gleichverteilte Kommunikation in beide Richtungen erwünscht ist. Zum Vergleich ist die Aufteilung bei ChatGPT als reines Frage-Antwort-Werkzeug 100% Benutzer zu 0% ChatGPT. Benutzer fragt, ChatGPT antwortet. Es werden keine Rückfragen von ChatGPT gestellt, es sei denn ein Benutzer hat es dem System als Aufgabe gegeben. Dabei bilden die Frage wieder die Antworten des ChatGPT auf das *Prompt Engineering* des Benutzers.

Die Aktivontologie hilft eine geteilte Initiative zu erreichen, da sie aufgrund des Aufbaus eine Benutzereingabe auf die fehlenden Informationen prüfen kann und dies an den *JustLingo.Dialogmanager* zurückgibt. Der Aufbau einer Aktivontologie wird in Kapitel 5.3.2 vorgestellt. Der Benutzer bekommt anschließend eine Frage zur fehlenden Information, so dass angepasste Rückfragen zur Eingabe ausgegeben werden. *JustLingo NLUI* ermöglicht es, einem Benutzer frei mit dem System zu interagieren und stellt Hilfestellungen sowie Anleitung im Rahmen eines Dialogs zur Verfügung. Ein Benutzer findet sich dadurch schneller zurecht und erledigt die geforderte Aufgabe zielgerichtet. Dabei agiert die Dialogführung bei *JustLingo NLUI* in einer geteilten Initiative. Eine Eingabe des Benutzers erweitert das gemeinsame Wissen und wird durch *JustLingo.DialogManager* bearbeitet. Ist die Anfrage mehrdeutig interpretiert, stellt das *JustLingo.DialogManager* in der Rückmeldung an den Benutzer mehrere Möglichkeiten der Lösung vor. Diese wird durch den Benutzer in der nächsten Eingabe an das System spezifiziert. Durch die Bearbeitung kann sich eine Rückfrage ergeben, die wiederum an den Benutzer übermittelt wird. Die Anfragen und Antworten erweitern den gemeinsamen Kontext. Dabei trägt die geteilte Initiative zu einem natürlichen Gesprächsverlauf bei, da ein fließender und spontaner Übergang zwischen Anfragen und Rückmeldungen auf die Anfragen für beide Gesprächspartner ermöglicht wird.

5.2.3 Vertikale und horizontale Dialogführung

Eine weitere Eigenschaft der interaktiven Schnittstelle ist eine zielgerichtete Dialogführung. Dabei gibt es im Dialogmanager zwei Konzepte: horizontale und vertikale Dialogführung. Im Verlauf der Forschungsarbeit hat sich gezeigt, dass beide Konzepte in der Kombination einen natürlichen Dialogfluss erzeugen.

Bei den meisten Dialogsystemen wird eine vertikale Dialogführung implementiert. Dabei kann ein Benutzer zu einer spezifischen Aufgabe Dialoge aktivieren, bis diese Aufgabe erfüllt ist, und der Dialog zu Ende ist. *JustLingo NLUI* verwendet eine vertikale Dialogführung innerhalb der vorhandenen NLUI-Fähigkeiten (Siehe Abbildung 5.8). Dabei wird zielorientiert eine bestimmte Aufgabe verfolgt. Das heißt, dass die Dialoge aufeinander aufbauen und nacheinander abgearbeitet werden. Dies geschieht zum Beispiel innerhalb eines NLUI-Fähigkeit bei der Mitteilung noch fehlender Information im Dialog.

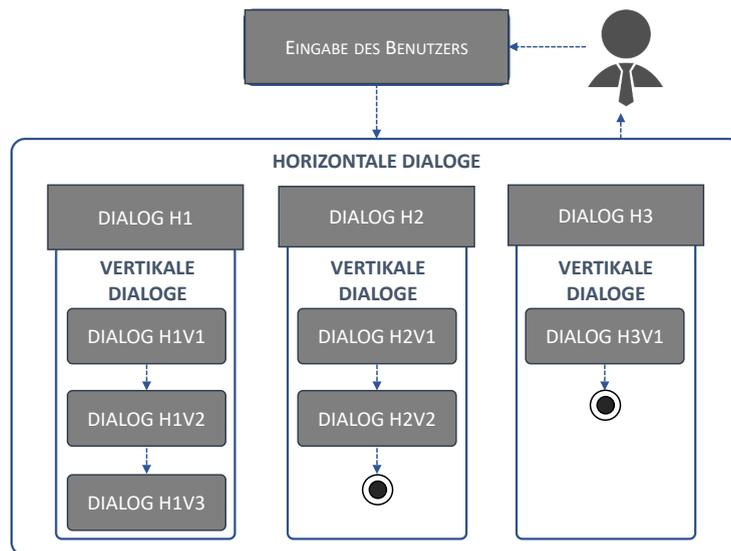


Abbildung 5.8: Natürlicher Dialogfluss mit vertikalen und horizontalen Dialogen

Beispiel einer vertikalen Dialogführung:

```

JustLingo NLUI: How can I help you?

Benutzer:      Put 7 percent of cell C15 plus 500 in cell E15

JustLingo NLUI: Your input is ambiguous.
                Following interpretations are possible:
                a) product(7/100, sum(C15, 500))
                b) sum(product(7/100, C15), 500)

                Please choose which one you were trying to tell me.

Benutzer:      b)

JustLingo NLUI: Ok, I did that for you.
  
```

Erfolgt ein Kontextwechsel durch eine Benutzereingabe, welche die Bearbeitung durch eine andere NLUI-Fähigkeit verlangt, wird die vertikale Ausführung des Dialogsystems angehalten und horizontal der gewünschten NLUI-Fähigkeit gesucht. Bei einer horizontalen Dialogführung sind alle Dialoge parallel aktiv und können gleichzeitig angesteuert werden.

Dieses Konzept ermöglicht dem JustLingo NLUI einen reibungslosen Kontextwechsel, den die Benutzer öfters in den Dialogen ausführen. In unseren Nutzerstudien haben wir festgestellt, dass die Benutzer durch die Eingaben öfters einen Kontextwechsel erzwingen, weil sie an der vorherigen Aufgabe nicht mehr arbeiten wollen und schon zur nächsten übergehen. Die natürlichsprachliche Schnittstelle ist durch die horizontale Dialogführung auf einen Kontextwechsel eines Benutzers vorbereitet und passt sich dem Verhalten des Benutzers an. Der Kontextwechsel wird durch die NLP-Fähigkeit *Klassifikation* ermöglicht, die in Kapitel 5.3.3 vorgestellt wird.

Beispiel einer horizontalen Dialogführung:

```

NLUI-Fähigkeit Mathematische Berechnung:
Substract 10 from all cells in column C that are larger than 25.

Wechsel zur NLUI-Fähigkeit Algorithmenerfassung:
Please create a variable.
  
```

5.2.4 Implementierung von JustLingo.DialogManager

Der Dialogmanager spielt in der Implementierung der natürlichsprachlichen Schnittstelle JustLingo NLUI eine zentrale Rolle (Siehe Abbildung 5.9). Zum einen lädt er beim Start des Programms durch *InitActiveSkills* die Information zu den verfügbaren Diensten und ist für die Dialogführung zuständig. Zum anderen nimmt er durch *ProcessNewInput* die Benutzereingabe entgegen und entscheidet, an welchen Dienst diese zur Bearbeitung weiterzuleiten ist.

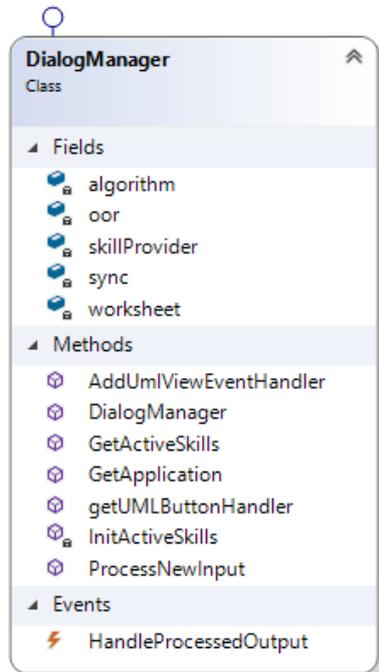


Abbildung 5.9: DialogManager

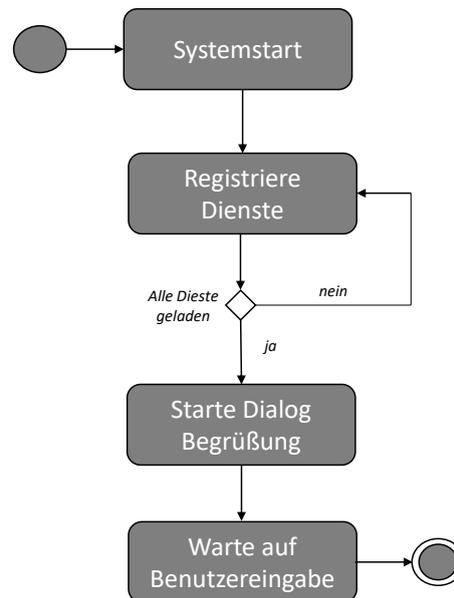


Abbildung 5.10: Dialogmanager Ablauf

Abbildung 5.10 zeigt den initialen Ablauf beim Start der Anwendung. Dabei müssen nicht nur die im Kapitel 5.1.2 vorgestellten grafischen Komponenten, sondern vielmehr die zur Verfügung stehenden Dienste initialisiert werden. Durch eine dynamische Initialisierung aller verfügbaren Dienste zum Programmstart kann das Gesamtsystem der natürlichsprachlichen Schnittstelle JustLingo NLUI leicht durch neuimplementierte Dienste erweitert werden. Der Dialogmanager überprüft bei einer neuen Eingabe des Benutzers an welchen Dienst die Anfrage zur Verarbeitung zu schicken ist. Die Umsetzung der Klassifikation wird detailliert im Kapitel 5.3.3 erläutert.

5.2.5 Fazit

Dieses Kapitel stellt das Aktive Dialogmanagement als den Kern von *JustLingo NLUI* dar. Die Komponente kennt die registrierten NLUI-Fähigkeiten des Gesamtsystems und leitet die Benutzeranfragen zielorientiert an diese weiter. Um die vorgestellte Eigenschaft des Dialogsystems anzubieten, bedient sich das NLUI mehrerer Hilfsdienste, die im Folgenden als NLP-Fähigkeiten vorgestellt werden.

5.3 NLP-Fähigkeiten

Während der Forschungsarbeit an der natürlichsprachlichen Benutzerschnittstelle *JustLingo NLUI* wurden mehrere NLP-Fähigkeiten implementiert, die als Hilfswerkzeuge dem NLUI bei der allgemeinen Bearbeitung einer Benutzereingabe dienen.

5.3.1 Zweck

Während der Forschungsarbeit an der natürlichsprachlichen Benutzerschnittstelle *JustLingo NLUI* haben wir beobachtet, dass das NLUI sehr gut mit bestimmten Beschreibungen in natürlicher Sprache zurechtkommt, wenn eine 1-zu-1-Beziehung zwischen der Eingabe des Benutzers und der Reaktion der natürlichsprachlichen Schnittstelle besteht. In diesem Fall ist eine Weiterleitung der Benutzereingabe an die richtige NLUI-Fähigkeit zur weiteren Bearbeitung trivial.

Die Komplexität ändert sich, wenn die Zuordnung, (i) aufgrund der Vielfalt einer natürlichsprachlichen Eingabe, die nicht immer eindeutig zu einer Aktion bzw. zur NLUI-Fähigkeit zugeordnet werden kann, oder (ii) mehrere NLUI-Fähigkeiten zur Bearbeitung einer Benutzereingabe zur Verfügung stehen.

Während der Forschung an *JustLingo NLUI* wurden mehrere NLUI-Fähigkeiten implementiert, die u.a. mathematische Berechnungen, Datenanalyse, Algorithmusinterpretation, Entitätserkennung, etc. durchführen. Bei einer Benutzereingabe reicht *JustLingo.Dialogsystem* eine Eingabe an eine implementierte Schnittstelle weiter. Aufgrund der Mehrdeutigkeit von natürlicher Sprache können mehrere NLUI-Fähigkeiten in Frage kommen.

Die Benutzereingabe $input_1$ für die Systemaktion $action_1$ ist eindeutig, wenn das *JustLingo NLUI* anhand der Syntaxuntersuchung in Kapitel 5.3.2 die Zuordnung korrekt und eindeutig die richtige Aktion bestimmt.

Ist dies nicht der Fall, unterscheidet ein Klassifikator (Kapitel 5.3.3) zwischen den NLUI-Fähigkeiten. Bei unbekanntem Benutzereingaben wird eine dynamische Wortschatzerweiterung (Kapitel 5.3.4) der Aktivontologie von *JustLingo NLUI* durchgeführt.

5.3.2 Sprachparser mit einer Aktivontologie

Die Aktivontologie nimmt bei *JustLingo NLUI* eine zentrale Rolle bei der Interpretation einer Benutzereingabe ein. Es ist verantwortlich für (i) die Zerteilung einer Eingabe und (ii) die Umwandlung in ein geeignetes Format für die Weiterverarbeitung, (iii) die Erschließung der Semantik einer Eingabe im Anschluss an den Analysevorgang und (iv) die Durchführung geeigneter Funktionen.

Eine Ontologie ist eine formale Repräsentation von Wissen. Eine Aktivontologie ist im Rahmen dieser Forschungsarbeit viel mehr. Es wird als Parser und zur Informationsextraktion verwendet.

Eine *Aktivontologie* ist eine Ausführungsumgebung, die als Ontologie mit zusätzlichen Informationen aufgebaut ist. Dabei entsprechen die Knoten den Konzepten und Kanten den Kommunikationskanälen.

Abbildung 5.11 zeigt eine Aktivontologie (kurz: AO) als Parser. Die Wörter in der Benutzereingabe werden in Knoten der AO gesucht. Die Elternknoten registrieren sich als Zuhörer der Kinderknoten. Sofern ein Kinderknoten einen Treffer landet, löst dieser eine Meldung nach oben zum Elternknoten aus. Ein Elternknoten kann diese Meldung entgegennehmen und weiterleiten oder ignorieren.

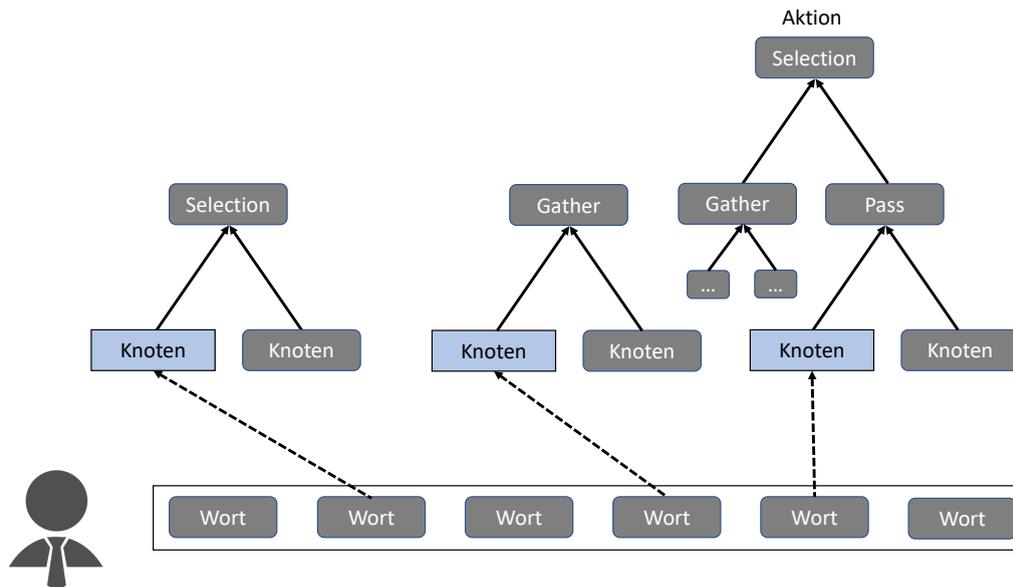


Abbildung 5.11: Schematischer Aufbau einer Aktivontologie nach [WKT16]

JustLingo NLUI führt mit Aktivontologie eine Bottom-Up Suche durch.

Die Bottom-Up Suche definiere ich wie folgt:

- Schritt 1: Suche nach einem Wort aus der Eingabe in den Knoten der Aktivontologie
- Schritt 2: Suche weiter nach oben bis eine Aktion getroffen wird oder die fehlenden Teile in der Eingabe zur validen Ausführung gefunden sind

Eine initiale Implementierung einer Aktiven Ontologie ist bei Guzzoni zu finden [Guz06]. Wir erweitern die Idee um einen weiteren Funktionsknoten Pass [WKT16]:

- Selection Knoten: der Knoten wählt unter allen aktivierten Kinderknoten, den mit der höchsten Zuverlässigkeitsbewertung, aus und gibt diesen an die Elternknoten weiter
- Gather Knoten: der Knoten feuert nach oben, wenn alle Kinder aktiviert sind
- Pass Knoten: der Knoten aggregiert die Menge aller aktivierten Kinderknoten und gibt diese als ein Event zu den Elternknoten.

Jede NLUI-Fähigkeit implementiert eine eigene Aktivontologie, da diese immer spezifisch zu der jeweiligen Fähigkeit konzipiert und erstellt wird. Für die Erstellung einer Aktivontologie steht der von uns implementierte AOEditor zur Verfügung.

5.3.3 Klassifikation

Im Verlauf der Forschungsarbeit wurde *JustLingo NLUI* durch mehrere NLUI-Fähigkeiten erweitert. Solange die natürlichsprachliche Schnittstelle bei der Bereitstellung der Funktionalität nur eine Aktivontologie zum Parsen benötigt, gibt es keine Notwendigkeit bei einer Benutzereingabe zwischen den NLUI-Fähigkeiten zu unterscheiden. Die Implementierung von NLUI-Fähigkeiten bei *JustLingo NLUI* sieht jedoch vor, dass die NLUI-Fähigkeiten unabhängige Funktionalität mit dafür vorgesehenen, spezialisierten Aktivontologien verwenden.

Die horizontale Dialogführung wird durch eine NLP-Fähigkeit *Klassifikation* ermöglicht. Dabei wird jede Eingabe eines Benutzers gegen während der Implementierung bereitgestellte Vergleichseingaben zu den jeweiligen NLUI-Fähigkeiten klassifiziert. Während des Dialogs ist es bei *JustLingo NLUI* möglich, eine Absichtserkennung mehrerer voneinander getrennten Eingaben sequenziell an die natürlichsprachliche Schnittstelle zu geben. Durch die Klassifikation kann zur Laufzeit pro Eingabe entschieden werden, zu welcher NLUI-Fähigkeit die Eingabe zur Bearbeitung geschickt wird. Die Klassifikation wird von *JustLingo.Dialogsystem* übersprungen, wenn ein aktiver Dialog bereits auf eine Rückmeldung des Benutzers wartet.

5.3.3.1 Erkennung der Absicht

Eine Schwierigkeit in der Verarbeitung natürlicher Sprache besteht darin, die richtige Absicht einer Eingabe zu erkennen. Abbildung 5.12 stellt das Problem vor, eine Benutzereingabe bei ähnlichen oder bisher unbekanntem natürlichsprachlichen Eingaben klassifizieren zu müssen [WEFT20]. Dabei stellt eine Benutzereingabe ein Klassifikationsproblem zu den bestehenden NLUI-Fähigkeiten *Datenanalyse*, *Algorithmus* und *Objektstruktur* dar. *JustLingo NLUI* kann in Abhängigkeit von der Eingabe des Benutzers und der zuständigen NLUI-Fähigkeit eine Beschreibung, ein Modell oder ein Algorithmus ausgeben.

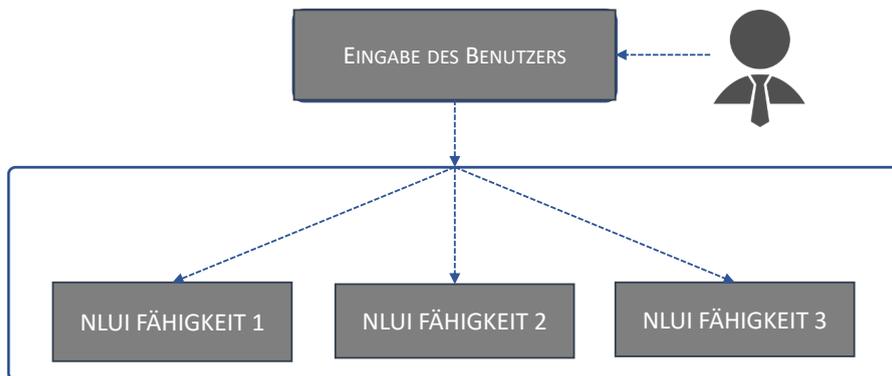


Abbildung 5.12: Klassifikationsproblem von einer Eingabe zu N Fähigkeiten aus [WEFT20]

5.3.3.2 Training der Klassifikatoren

Die lexikalischen und semantischen Klassifikatoren werden pro verfügbare NLUI-Fähigkeit trainiert und erhalten eine Gewichtung, die zeigt, wie hilfreich der Klassifikator für die Erkennung relevanter Sätze ist (Siehe Algorithmus 1).

Algorithmus 1 Training der Klassifikatoren für eine NLUI-Fähigkeit

```

1: for each classifier c do
2:   averageScores ← empty list
3:   for each sentence s of the input history of the current service do
4:     otherSentences ← all sentences of the
5:     input history of the current service which
6:     are unequal to s
7:     scores ← empty list
8:     for each sentence o in otherSentences do
9:       score ← c.getSimilarity(s, o)
10:      scores.add(score)
11:    end for
12:    averageScores.add(average of scores)
13:  end for
14:  service.classifierToAverageScore.put(c, averageScores)
15:  CalculateWeights(service)
16: end for

```

Bevor die Eingabe auf Ähnlichkeit überprüft wird, wird diese vereinfacht, was für die lexikalische Analyse besonders wichtig ist: Großbuchstaben werden durch Kleinbuchstaben ersetzt und Stoppwörter sowie Satzzeichen entfernt.

Die Implementierung des Verfahrens ist wie folgt:

- Die Berechnung des Ähnlichkeitsgrades erfolgt zwischen den sechs Klassifikatoren asynchron, so dass der aktuelle Berechnungsstand abrufbar ist.
- Die Historie einer NLUI-Fähigkeit wird verwendet, um die Relevanz der einzelnen Klassifikatoren zu bewerten.
- Die durchschnittliche Übereinstimmung von neuen Einträgen mit der Historie einer NLUI-Fähigkeit wird gewichtet und gemäß der Relevanz des Klassifikators addiert.
- Die berechnete Punktzahl der NLUI-Fähigkeit wird an das *JustLingo.Dialogsystem* weitergeleitet. Weitere Klassifikatoren können hinzugefügt werden.

Um die Relevanz und damit die Gewichtung der einzelnen Klassifikatoren abzuschätzen, wird ihr Anteil der summierten Durchschnittsbewertung für die jeweilige NLUI-Fähigkeit berechnet (Siehe Algorithmus 2).

Algorithmus 2 Berechnung der Gewichtung pro NLUI-Fähigkeit

```

1: sumOfAverageScores ← sum up the averages of
   the values of service.classifierToAverageScores
2: for each classifier c do
3:   service.putWeight(c, average(classifierToAverageScores.get(c)) /
4:   sumOfAverageScorePerClassifier)
5: end for

```

Die Implementierung unterstützt das inkrementelle, kontinuierliche Lernen. Das *JustLingo.Dialogsystem* liefert der Klassifikation die Information, ob die Klassifizierung korrekt war oder, falls nicht, welche NLUI-Fähigkeit ein Benutzer durch seine Korrektur gewählt hat. Auf diese Weise können diese Sätze zu der entsprechenden NLUI-Fähigkeit hinzugefügt und zum einen die Gewichte der Klassifizierer aktualisiert (Siehe Algorithmus 3), und zum anderen der neue Satz für zukünftige Ähnlichkeitsanalyse verwendet werden. Somit passt sich der Klassifikator der Ausdrucksweise des Benutzers an.

Algorithmus 3 Aktualisiere Gewichtung aufgrund neuer Eingabe

```

1: otherSentences ← all sentences of the input history
2: of the service which are unequal to the new sentence
3: for each classifier c do
4:   scores ← empty list
5:   for each sentence in history of service do
6:     score ← c.getSimilarity(s, o)
7:     scores.add(score)
8:   end for
9:   service.classifierToAverageScores.get(c).add(average of scores)
10: end for
11: CalculateWeights(service)

```

Im Falle einer Programmierung kann die Benutzereingabe auf die Erkennung eines Algorithmus oder zur Klasseninterpretation bezogen werden. Diese NLUI-Fähigkeiten haben ähnliche Eingaben, jedoch unterschiedliche Aktionen bei der Ausführung. Aus diesem Grund reicht es nicht aus, die Sprache auf syntaktischer Ebene zu überprüfen. Darüber hinaus sollte das Syntax-Matching auch um die Interpretation auf Meta-Ebene erweitert werden. Diese Methoden sollten Syntax-, Semantik- und Kontextklassifizierung kombinieren. Dafür verwenden wir verschiedene Klassifizierer. Das Ergebnis der einzelnen Klassifizierer bildet die Gesamtähnlichkeit der Eingabe für die spezifischen NLUI-Fähigkeiten.

5.3.3.3 Klassifizierung einer neuen Eingabe

Algorithmus 4 beschreibt die Prozedur zur Berechnung der Ähnlichkeitszuordnung einer neuen Eingabe. Alle NLUI-Fähigkeiten werden iteriert und für die verschiedenen Elemente wird ein Wert für die Übereinstimmung berechnet. Dieser Wert wird dann gewichtet summiert und ergibt die Gesamtbewertung der NLUI-Fähigkeiten. *JustLingo.Dialogsystem* erhält die Zuordnung der NLUI-Fähigkeiten zu den berechneten Bewertungen.

Algorithmus 4 Berechnung der Ähnlichkeitszuordnung einer neuen Eingabe

```

1: serviceToScore ← empty map
2: for each service parallel do
3:   relevance ← 0
4:   sort sentence of service according to their relevance for the
5:   classification result in past
6:   for classifier c do
7:     scores ← empty list
8:     for each sentence s in the history of service do
9:       score ← c.getScore(input, s)
10:      if score ≥ threshold then
11:        s.increaseRelevance()
12:      end if
13:      scores.add(score)
14:    end for
15:    aggregatedScore ← max(scores) - sd(scores)
16:    relevance ← relevance + (aggregatedResult * service.getWeight(c))
17:  end for
18:  serviceToScore.add(service, relevance)
19: end for
20: return serviceToScore

```

5.3.3.4 Implementierung von JustLingo.Classification

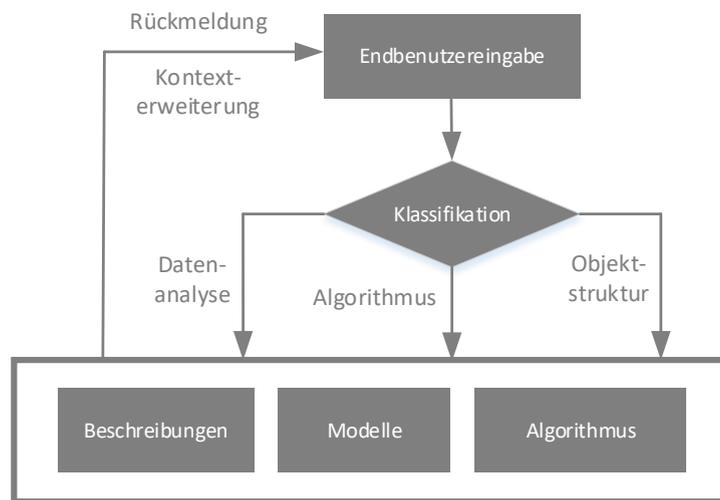


Abbildung 5.13: Klassifikation ähnlicher Benutzereingaben aus [WEFT20]

JustLingo.DialogManager erhält als zentrale Instanz eine Eingabe von einem Benutzer und delegiert es zu weiterer Verarbeitung an die jeweilige NLUI-Fähigkeit. Eine NLUI-Fähigkeit ist für eine bestimmte Tätigkeit konzipiert. Eine falsche Zuordnung des *JustLingo.DialogManager* führt zu keinem oder einem falschen Ergebnis. Damit die Zuordnung korrekt abläuft nutzt *JustLingo.DialogManager* einen Hilfsdienst *JustLingo.Klassifikation*, der eine neue Benutzereingabe auf Ähnlichkeit zu den vorhandenen Beispielen der NLUI Fähigkeit untersucht (Siehe Abbildung 5.13).

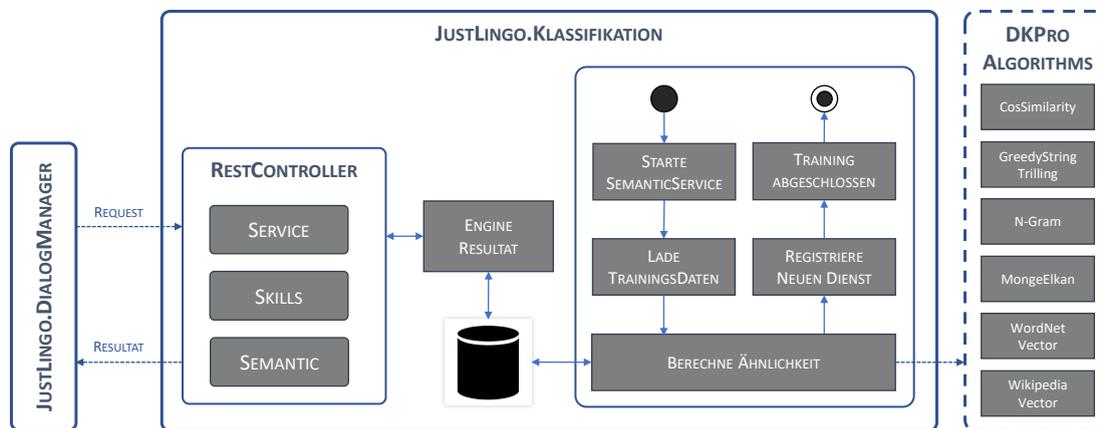


Abbildung 5.14: Aufbau von JustLingo.Klassifikation

Abbildung 5.14 stellt den Aufbau von *JustLingo.Klassifikation* vor. Innerhalb des Dienstes werden (i) die Klassifikatoren anhand bereitgestellter Beispielen trainiert und (ii) über die REST API ein Ähnlichkeitswert zu einer neuen Eingabe eines Benutzers asynchron ermittelt. Dabei handelt es sich beim Training um das überwachte Lernen, da die richtige Klassifikation der Sätze zum vorhandenen Dienst vor dem Training bereits bekannt ist. Jede neue Eingabe, die eine Zuordnung durch *JustLingo.Klassifikation* durchläuft, wird nachtrainiert und steht beim nächsten Mal dem Klassifikator zur Verfügung. Die Genauigkeit der Klassifizierung wächst somit kontinuierlich mit der Verwendung der natürlichsprachlichen Schnittstelle.

Aufgrund der Mehrdeutigkeit der natürlichen Sprache gibt es bei jeder neuen Benutzereingabe ein Klassifizierungsproblem zu den erforderlichen NLUI-Fähigkeiten. Wenn die Absicht hinter einer Eingabe nicht identifiziert werden kann, wird *JustLingo NLUI* ein falsches oder kein Ergebnis zurückliefern können. Damit dies nicht passiert, wird durch *JustLingo.DialogManager* eine NLP-Fähigkeit *JustLingo.Klassifikation* als Hilfsdienst angesteuert, der eine neue Benutzereingabe auf Ähnlichkeit zu den vorhandenen Beispieleingaben der vorhandenen NLUI-Fähigkeiten untersucht.

Das System lernt die Eingabe des Benutzers zur Laufzeit und erweitert durch ein Nachtrainieren die Bewertung neuer Eingaben. Durch das Dialogsystem wird die Eingabe des Benutzers vorverarbeitet und eine Antwort des Systems generiert. Dazwischen erfolgt die Ausführung einer gültigen Systemaktion durch die NLUI-Fähigkeiten.

5.3.4 Dynamische Wortschatzerweiterung

Die NLP-Fähigkeit der *dynamischen Wortschatzerweiterung* ermöglicht es, für unbekannte Wörter in einer Benutzeranfrage in externen Wissensbanken nach einer Bedeutung nachzufragen. Der Wortschatz von *JustLingo NLUI* definiert sich aus den aktiven NLUI-Fähigkeiten, die sich beim Dialogsystem angemeldet haben. Für den Fall, dass ein Benutzer einen anderen Wortschatz wählt, wird eine Möglichkeit benötigt, das ursprüngliche Grundwortschatz zu erweitern [WST19].

Für einen guten Dialog ist nicht nur das Verhalten der natürlichsprachlichen Schnittstelle entscheidend, sondern auch die Wissensbasis des System. Die Anforderungen an ein natürlichsprachliches System hängen von den Erwartungen eines Benutzers ab. Wird die Sprachdomäne nicht eingeschränkt, ist es für den Benutzer leicht, sich im gewohnten Sprachstil auszudrücken. Auf der anderen Seite ist es für das NLUI um so schwerer, eine nicht eingeschränkte Sprache zu verstehen. Hat das NLUI die Benutzereingabe korrekt verstanden, ist eine direkt Bearbeitung der Anfrage durch das Dialogsystem sowie die NLUI-Fähigkeiten möglich. In diesem Fall ist die Domänensprache des Benutzers gleich dem Grundwortschatz von *JustLingo NLUI* und die Verarbeitung bzw. die Erkennung der Eingaben ist eindeutig.

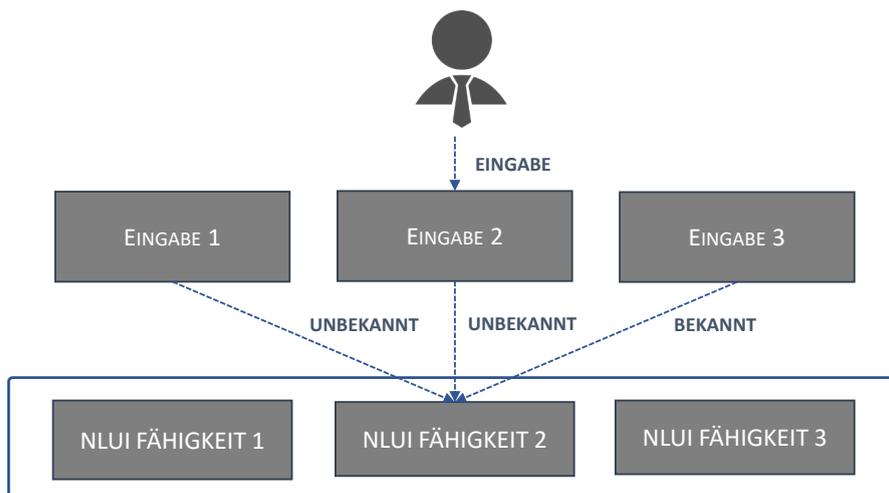


Abbildung 5.15: Mehrdeutigkeitsproblem von N Eingaben zu 1 Fähigkeit

Abbildung 5.17 skizziert einen Ablauf: bei (1) wird eine Benutzereingabe an den Dialogmanager mitgeteilt, die unter (2) gegeben die Aktivontologie geprüft wird. Sofern es unbekannte Wörter enthält und eine NLUI-Fähigkeit nicht direkt zugeordnet werden kann, erfolgt bei (3) eine Ausgabe an den Benutzer zur Umformulierung. Sofern die Eingabe immer noch unbekannt ist, wird unter (4) die Suche im lokalen Wortschatz eingestellt und eine Anfrage an externe Wissensquellen gestellt. Eine Rückmeldung bei (5) liefert Kandidaten $Wort_1$ bis $Wort_k$ zurück, die dem Benutzer zur Bestätigung angezeigt werden und die Antwort unter (6) bzw. (7) permanent in der Aktivontologie gelernt wird.

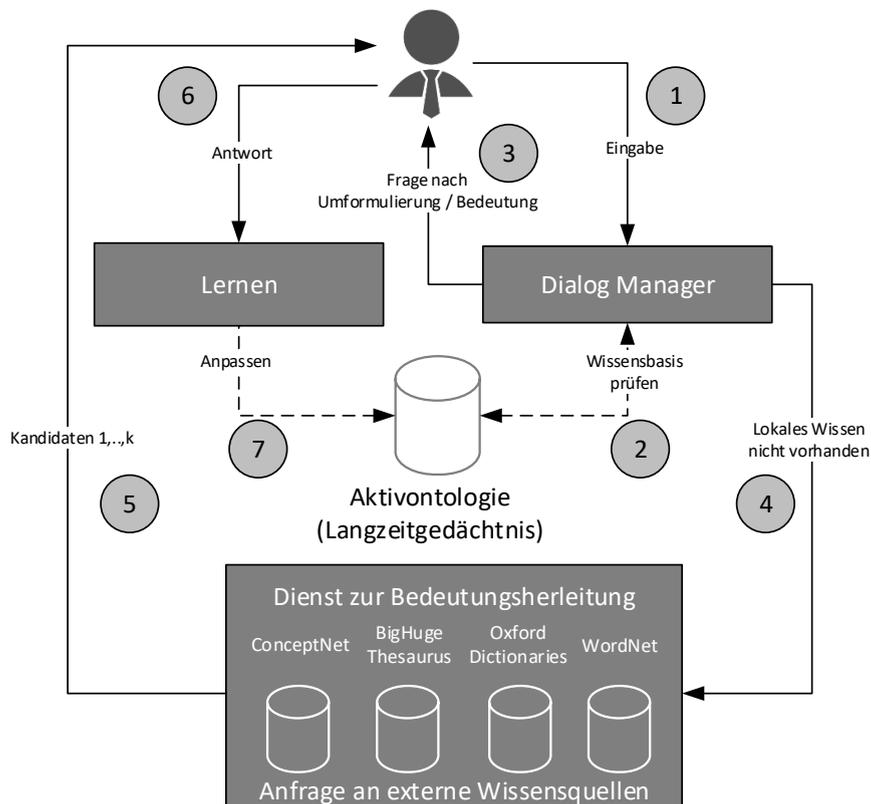


Abbildung 5.17: Dynamische Wortschatzerweiterung nach [WST19]

Ein Beispiel ist in der folgenden Auflistung zu sehen. Hierbei wurde aufgrund fehlender Information wegen Wort *larger*, die Aktion einer mathematischen Berechnung nicht erkannt. Als die NLP-Fähigkeit der dynamischen Wortschatzerweiterung eine Verbindung zwischen *larger* und *greater* hergeleitet hat, konnte die Aktion erfolgreich ausgeführt werden:

```
Nicht-erkannte Benutzereingabe:
Substract 10 from all cells in column C that are larger than 25.

Erkannte Benutzereingabe:
Substract 10 from all cells in column C that are greater than 25.
```

5.3.5 Fazit

Die vorgestellten NLP-Fähigkeiten sind für die korrekte Bearbeitung einer Benutzereingabe durch *JustLingo NLUI* unverzichtbar. Sofern im Wortschatz der Aktivontologie die eingegebenen Wörter nicht vorhanden sind, hilft eine dynamische Wortschatzerweiterung bei der Herleitung von ähnlichen Bedeutungen zu einer unbekanntem Eingabe und erweitert mit den bestätigten Kandidaten die Aktivontologie. Nicht zuletzt erfolgt durch die NLP-Fähigkeit *Klassifikation* eine Zuordnung einer Eingaben in Abhängigkeit der berechneten Absicht des Benutzers.

5.4 NLUI-Fähigkeiten

Das Kapitel stellt die für *JustLingo NLUI* bereitgestellten NLUI-Fähigkeiten der *Datenstrukturanalyse* sowie *Erfassung der Folgenalgorithmen* vor. Beide Fähigkeiten markieren den Anwendungsfall, der innerhalb der Forschungsarbeit untersucht wird.

5.4.1 Zweck

Eine natürlichsprachliche Beschreibung enthält eine Vielzahl an Objekten. Dabei können die Beschreibungen vorhandener Algorithmen als Methoden erkannt und auf den Objekten ausgeführt werden. Die Objektorientierung hat einige Vorteile, (i) Erfassung der Algorithmen für die Manipulation der Attribute, welche die Zustände der Objekte beschreiben (ii) erneutes Aufrufen der gespeicherte Methoden und (iii) Zuweisen der Methoden zu den jeweiligen Klassen bzw. deren Objekten. Sollten die Algorithmen nicht zu Klassen verlinkt sein, ist es nicht klar, auf welche Objekte die Algorithmen ausgeführt werden.

5.4.2 Datenstrukturanalyse

Die NLUI-Fähigkeit *Datenstrukturanalyse* unterstützt eine Erkennung von Klassenhierarchien aus natürlichen Beschreibungen sowie den nicht-zusammenhängenden Tabellen. In der Regel muss eine Information in den Knoten einer Aktivontologie modelliert sein, um eine Benutzereingabe zu erkennen. Wenn die Information neu ist, kann die Aktivontologie nicht weiterhelfen. Die NLUI-Fähigkeit verfolgt einen Ansatz, aus einer natürlichsprachlichen Beschreibung vorher noch unbekannte Informationen zu erfassen, können jedoch durch die NLUI-Fähigkeit *Datenstrukturanalyse* erfasst werden [WFPT19].

Zum Beispiel wird der Satz *The result is a vector* zuerst in der Satzstruktur auf Beziehungen untersucht. Nach [Fuc18] wird aus der Aktivontologie der Datenstrukturanalyse (Anhang 9.2) eine *IS-A-Beziehung* erkannt und zwei Klassen *result* und *vector* erzeugt. Sowohl *result* als auch *vector* sind der Aktivontologie nicht bekannt.

Der folgende Auszug dient der Veranschaulichung (Vgl. [THM09]):

A house consists of logs, clay bricks, and nails. Every building material, whether it is a log, clay brick or nail, is part of exactly one house. Each house has a certain number of rooms.

Abbildung 5.18 zeigt eine Lösung für dieses Beispiel. In diesem Fall muss *JustLingo NLUI* die richtige Struktur mit Namen der Klassen und den Beziehungen herausfinden. Außerdem sollte eine Vererbung zwischen Baumaterial und Lehmziegel, Baumstamm und Nagel erkennen sowie eine Abhängigkeit zwischen Raum, Baumaterial und Haus finden.

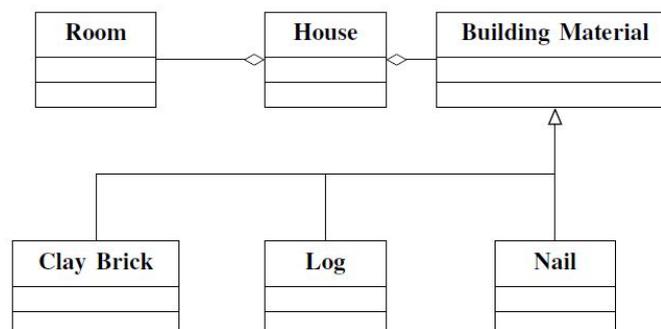


Abbildung 5.18: Modellierung einer Beschreibung als Klassenstruktur

Abbildung 5.19 zeigt eine Interpretation von Daten einer Tabelle. Dabei interpretiert *JustLingo NLUI* die vorhandene Struktur als Objekte, Attribute sowie Methoden. Als internes Metamodell wird die Beschreibung eines Objekts mit den zugehörigen Attributen und Methoden erfasst. In einer tabellarischen Form stellen die Objekte die Zeilen und die zugeordneten Attribute die Spalten dar. Die Daten in einer Tabelle repräsentieren eine Instanz des Objekts. Die Methoden sind Funktionen, welche einzelne Zellen der Tabelle manipulieren.

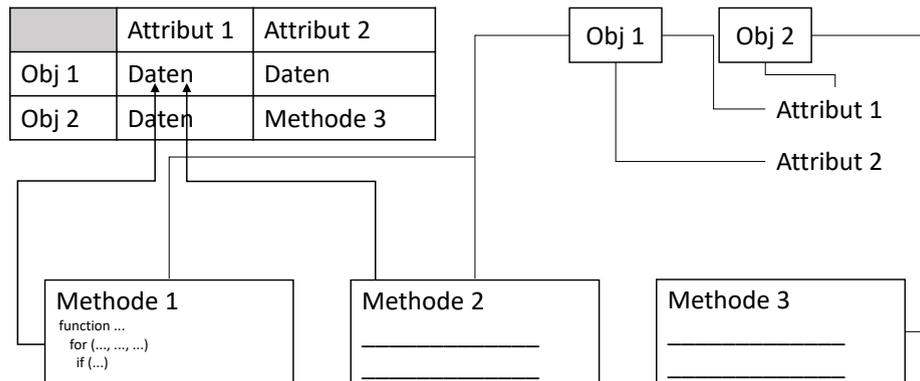


Abbildung 5.19: Interpretation einer Tabelle in der Datenstrukturanalyse

Bei *JustLingo NLUI* werden Objekte in einer Beschreibung als Klassen dargestellt. Zum Beispiel enthält die Beschreibung zu den Teilen eines Autos die Räder, Fenster und Türen. Des Weiteren erfüllt ein Auto bestimmte Funktionalität, wie zum Beispiel fahren, bremsen oder beschleunigen.

Durch die Datenstrukturanalyse werden Informationen zum Abfragen für einen Benutzer vorbereitet bzw. Strukturen der Daten gemerkt, so dass die Anfragen der Benutzers bearbeitet werden können. Erfolgt keine Datenstrukturanalyse, muss das NLUI im Falle einer Anfrage erstmal schauen, worum es in der Anfrage geht und was der Benutzer mit der Eingabe meinen könnte. Dies ist eine wertvolle Zeitspanne, denn sollte die Systemantwort zu lange dauern, sinkt die Akzeptanz bei dem Benutzer.

Alle gesammelten Informationen werden in Textform angezeigt. Da die Datenmenge mit jeder neuen Eingabe wächst, war eine leicht verständliche Darstellung im Text nicht möglich. In Anbetracht des Ziels der objektorientierten Programmierung wurde die bekannte UML-Darstellung verwendet. Nachdem alle Eingaben verarbeitet, analysiert und Klassifikationen bestimmt wurden, wird der aktuelle Status als UML-Klassendiagramm dargestellt. Das vollständige Diagramm wird dem Benutzer zusammen mit einer reduzierten Version der reinen Textdarstellung angezeigt.

Selbst eine solche einfache Benutzerbeschreibung enthält viele Informationen. Nicht nur Klassen mit ihren Attributen und Methoden müssen modelliert werden, sondern auch die Beziehungen dazwischen, z.B. Assoziations- und Aggregationsbeziehungen. Um das generierte Diagramm zu vereinfachen, fasst das System alle Kanten zusammen, die zu einer Beziehung zwischen *Instanz von*, *Vererbung* oder *Aggregation* gehören. Darüber hinaus kann das System Instanzen von Klassen erkennen. Diese Instanzen können entweder per Texteingabe beschrieben oder aus Datentabellen geladen werden.

5.4.2.1 Klassenmodelle aus Text

Bereits in der Pilotstudie war es bei *JustLingo NLUI* möglich, eine Funktionsbeschreibung abzugeben. Um eine Funktionsbeschreibung auf Daten auszuführen, wird eine neue Fähigkeit der natürlichsprachlichen Schnittstelle vorgestellt, die sowohl (i) die Verknüpfung von Attributen und Funktionen mit einer Klasse als auch (ii) die Erkennung von Beziehungen zwischen den Klassen ermöglicht [Wie18].

Dafür wurden mehrere Module implementiert:

- *JustLingo.Objects* führt eine Untersuchung nach Klassen innerhalb einzelner Sätze.
- *JustLingo.Relations* untersucht die natürlichsprachliche Beschreibung über die Satzgrenzen hinweg nach einer Beziehung zwischen den Klassen.
- *JustLingo.Shared* hält den geteilten Quellcode der beiden Module. Nicht zuletzt koordiniert *JustLingo.Supervisor* als eine zentrale Einstiegsstelle des Dienstes die Bearbeitung der Eingabe.

Untersuchung innerhalb der Sätze

Für die Erkennung und die Extraktion von Objekten wurde eine Grammatik entwickelt, die auf Omar und Razik [OR08] aufbaut. Beide schlugen eine Menge von Heuristiken vor, um Klassen, Methoden und Attribute in natürlicher Sprache zu bestimmen. Da bei Omar und Razik jedoch im Vordergrund eine abstrakte Erkennung eines zu einem Objekten gehörenden Beschreibungsblocks stand, musste bei *JustLingo NLUI* noch eine Erweiterung erfolgen, die eine Beziehung zwischen den Entitäten erkennt.

Im Allgemeinen erfolgt die Erkennung einer Klasse oder eines Attributs ähnlich. In *JustLingo NLUI* kann zur Laufzeit mit Hilfe des Dialogsystems das Ergebnis der Erkennung zwischen einer Klasse und einem Attribut getauscht werden [WFPT19]. Des Weiteren gibt es eine Menge an Wahrscheinlichkeitswerten. Die endgültige Ausgabe, die dem Benutzer präsentiert wird, ist von der wahrscheinlichsten Klassifizierung durch das *JustLingo NLUI* abgeleitet. Schließlich wurden vier Regeln auf oberster Ebene identifiziert und mit der zugehörigen Wertung umgesetzt.

Das Vorgehen ist an die Sprachverarbeitung aus Kapitel 4.3 angelehnt.

1. Stichwortsuche
2. Suche nach Beziehungszuordnung
3. Grammatikbasierte Übereinstimmung
4. Beziehungen zwischen zwei Entitäten

Untersuchung zwischen den Sätzen

Dieses Kapitel stellt die Erkennung von Beziehungen zwischen den Objekten vor. Abbildung 9.2 präsentiert eine Aktivontologie zur Erkennung von Relationen nach [Fuc18]:

- Ist-Ein-Beziehungen
- Assoziation
- Aggregation/Komposition

Im Allgemeinen gibt es zwei Ansätze zur Anpassung in einem Dialogsystem, das mit Daten arbeitet. Zum einen können wir das System an den Benutzer anpassen. Das heißt, wir ermöglichen es, Synonyme oder ähnliches aufzulösen (Siehe Kapitel 5.3.4).

Die andere Möglichkeit besteht darin, sich an die Daten anzupassen. Genau diese Anpassung der Daten wird im Folgenden durchgeführt. Wie wir oben gesehen haben, kann *JustLingo NLUI* Beschreibungen von Beziehungen, Klassen und Objekten verstehen. Wie sieht es aber mit Beziehungen aus, die in Modellen bereits vorhanden sind, beispielsweise in und zwischen Tabellen? Wir verwenden auch einen einfachen Ansatz, um aus diesen Tabellen zu extrahieren. Dazu abstrahieren wir die zugrundeliegende Daten und übertragen die Tabellen in ein gemeinsames Modell. Dann extrahieren wir die enthaltenen Elemente und ihre Beziehungen aus den Tabellen [WST19]. Der erste Schritt besteht darin, die verschiedenen Tabellentypen in ein vereinfachtes Tabellenmodell umzuwandeln. In diesem Modell werden Tabellen mit Zeilen und Spalten angenommen. In der ersten Zeile werden die Attribute (z.B. Name, Alter, Ruf,...) angezeigt. Jede weitere Zeile steht für ein Objekt. Jeder Tabelle ist eine Klasse zugeordnet. Jedes Attribut steht entweder für einen Primärschlüssel, einen Fremdschlüssel oder einfach für ein Attribut. Es gibt maximal eine Spalte pro Tabelle, die als Primärschlüssel ausgewählt ist. Die Umwandlung der verschiedenen Tabellen in das gemeinsame Format ist ein interaktiver Prozess. In einem ersten Schritt werden für jede Tabelle leere Dummy-Tabellen erstellt. Danach wird der Benutzer aufgefordert, einige Fragen zu beantworten. Zuerst werden die Namen der Elemente jeder Tabelle abgefragt. Dies liefert den zukünftigen Klassennamen. Je nach Plattform wird der Benutzer aufgefordert, die identifizierenden Eigenschaften für die Elemente der Tabelle anzugeben, wenn dies möglich ist. In einer letzten Frage kann der Benutzer angeben, mit welchen Spalten die Objektnamen erstellt werden sollen. Ein Objekt wird aus jeder Zeile der Tabelle erstellt. Die auf diese Weise erhaltenen Informationen werden verwendet, um in einem Folgeschritt Verbindungen zwischen den Tabellen herzustellen. Im nächsten Schritt werden mögliche Verknüpfungsbeziehungen gesucht und eine Join-Beziehung nach der Auflösung einer Fremdschlüssel-Beziehung gefunden.

Erläuterndes Beispiel:

Besitzer		
Name	Alter	Ruf
Smith	42	A+
Miller	53	A

Haus		
Straße	Parteien	Besitzer
2 Piper St	9	Smith
1 Piper St	6	Smith

Diese Tabellengruppe zeigt eine Gruppe von Besitzern in der ersten Tabelle. Diese sind mit Namen, Alter und Ruf aufgeführt. Die zweite Tabelle zeigt verschiedene Eigenschaften mit ihrer Straße, der Anzahl der Parteien und einem Eintrag für ihren Besitzer. Die Einträge in der Spalte *Besitzer* repräsentieren einen Fremdschlüssel für die Tabelle der Besitzer. Dies bedeutet, dass zwischen den Klassen *Besitzer* und *Haus* eine Beziehung besteht, die auf dieser Spalte basiert. Um solche Verknüpfungsbeziehungen zwischen Tabellen einzurichten, fragt das System den Benutzer beim Laden, ob eine solche Beziehung erstellt werden soll. Wenn dies der Fall ist, werden die Beziehungen automatisch auf Klassen- und Objektebene erstellt (Siehe Abbildung 5.20).

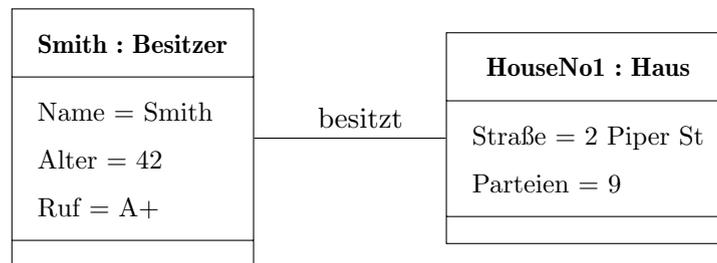


Abbildung 5.20: Klassen: Haus und Besitzer

Die Beschreibung des Benutzers, das UML-Diagramm sowie die Code-Fragmente bezeichnen wir als *NLUI Artefakt*.

5.4.2.2 Klassenmodelle aus Tabellen

In diesem Kapitel analysieren wir die vorhandenen Daten in der Tabellenkalkulation, um auf die Benutzeranfragen in natürlicher Sprache vorbereitet zu sein. Durch die Datenstrukturanalyse von *JustLingo NLUI* werden Anfragen des Benutzers auf den Daten ermöglicht. Dabei lernt die natürlichsprachliche Schnittstelle die Struktur der Daten und deren Beziehungen zueinander [WFPT19].

Interaktiver Aufbau eines Klassenmodells

Im folgenden wird aufgezeigt, wie ein Datenmodell aufgebaut werden kann:

```

Input: a state is part of a country
Output: The new class country was added.

Input: a state has a population
Output: The new unresolved population (class, attribute) was added.

Input: population is an attribute
Output: The class population is now an attribute.

Input: The USA and Canada are countries.
Output: What is the type of USA? Class or Object?

Input: object
Output: What is the type of Canada? Class or Object?

Input: object
Output: The new classes usa and canada were added.
  
```

```

    The class USA is now an instance. The class Canada is now an instance.

Input: California and Florida are states
Output: What is the type of California? Class or Object?
Input: object
Output: What is the type of Florida? Class or Object?
Input: object
Output: The new classes californi and florida were added.
       The class californi is now a instance.
       The class florida is now a instance.

Input: California and Florida are part of the USA
Input: Ontario and Quebec are states
Output: What is the type of Ontario? Class or Object?
Input: an object
Output: What is the type of Quebec? Class or Object?
Input: an object
Output: The new classes ontario and quebec were added. The class
       ontario is now a instance. The class quebec is now a instance.

Input: Ontario and Quebec are part of Canada
Output: Synced successfully

```

Diese Eingaben erzeugen das folgende Datenmodell:

Name	Rel:country-has-state	population
california	USA	null
florida	USA	null
ontario	Canada	null
quebec	Canada	null

Tabelle 5.1: Darstellung des Datenmodells

Des Weiteren erzeugen diese Eingaben zudem das folgende Klassenmodell:

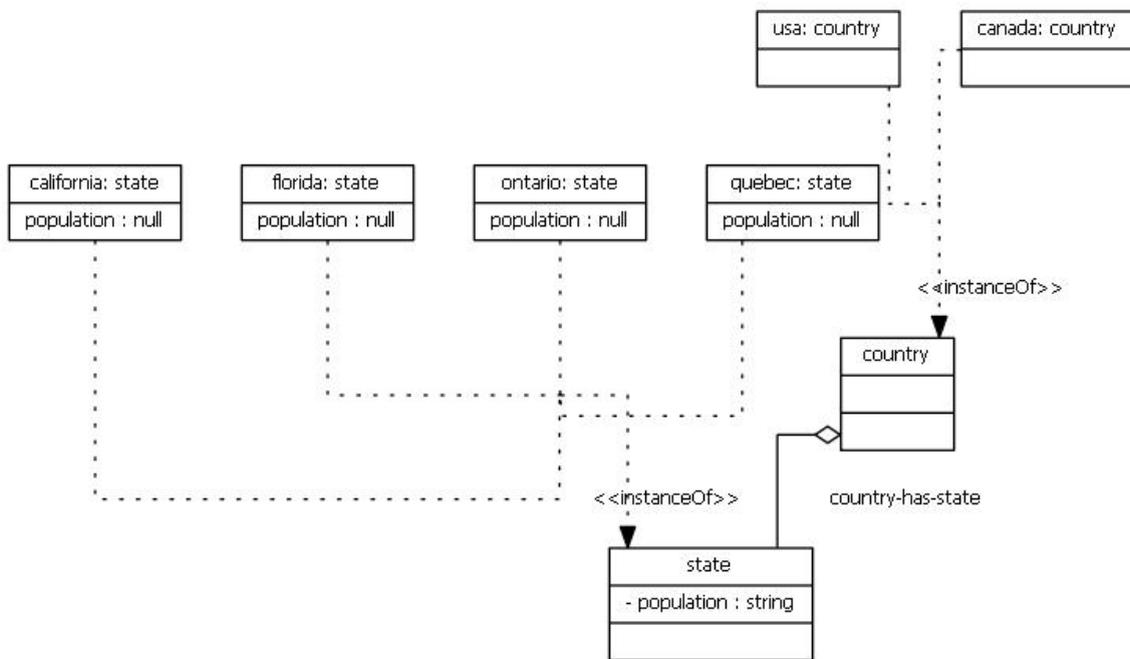


Abbildung 5.21: Das zugehörige Klassenmodell

Für das Abfragen und das Manipulieren der Daten werden alle Artefakte untersucht. Entweder über das Dialogsystem oder per Tabellenkalkulation kann ein Benutzer die Daten zu der Bevölkerung vervollständigen.

Folgende Artefakte liegen vor:

Name	Rel:country-has-state	population
california	USA	40
florida	USA	21
ontario	Canada	13
quebec	Canada	8

Tabelle 5.2: Darstellung des Datenmodells aus Tabellenkalkulation

Das zugehörige Datenmodell sieht wie folgt aus:

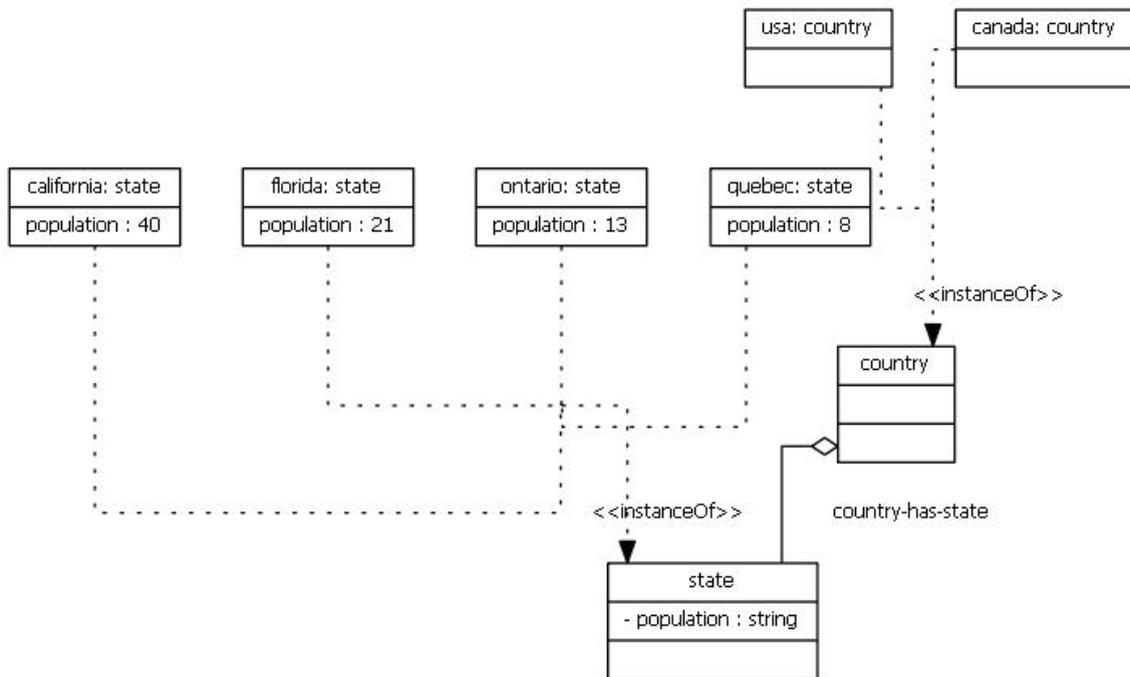


Abbildung 5.22: Das initialisierte Klassenmodell

In der folgenden Eingabe erkennt *JustLingo NLUI*, dass es sich um eine Verbindung zwischen *USA* und *population* handelt, die über *country-has-state -> florida* sowie *country-has-state -> california* Zustände kommt. Die Methode *average* ist eindeutig, daher fragt das System nicht mehr nach und liefert als Antwort 30,5.

Folgende Algorithmen sind für die Berechnung als Methoden hinterlegt:

- 1: switchingSort, 2: findMax, 3: selectionSort, 4: average,
- 5: minimum, 6: maximum, 7: sum, 8: toList, 9: count

```
Input:      What is the average population in the USA?
```

```
Output:    All connections found (2)
           usa->country-has-state->florida->population
           usa->country-has-state->california->population
```

```
Result found (1)
           30.5
```

Ähnliches Verhalten bei einer Abfrage der maximalen Bevölkerung in Kanada:

```
Input:      What is the maximum population in the states of Canada?
```

```
Output:    Best connections found (2)
           canada->country-has-state->quebec->population
           canada->country-has-state->ontario->population
```

```
Result found (1)
           13
```

Bei einer allgemeinen Abfrage nach der Bevölkerung der USA wird ein nicht mehr triviales Beispiel bearbeitet:

```

Input:      What is the population in the USA?

Output: I'm going to reduce the relation called country-has-state
        starting from country. Please specify the reduce function.

        Multiple Options possible, please choose one of them:
        1: switchingSort, 2: findMax, 3: selectionSort, 4: average,
        5: minimum, 6: maximum, 7: sum, 8: toList, 9: count

Input:      sum

Output: Best connections found (2)
        usa->country-has-state->florida->population
        usa->country-has-state->california->population

        Result found (1)
        61

```

In diesem Fall fragt das System nach welcher bekannten Methode die Berechnung durchgeführt werden sollte. Im Fall der Summe werden die Werte summiert und die Anzahl 61 ausgegeben.

Ein Benutzer hätte sich jedoch anders entscheiden können, wie das nachfolgende Beispiel zeigt. Dabei zielt die Anfangsfrage nach der Auflistung der Bevölkerung der USA und nicht nach der Summe. In diesem Fall wählt der Benutzer einen switchingSort-Algorithmus, um die Werte aufsteigend zu sortieren. Eine nicht sortierte Ausgabe der Werte erreicht ein Benutzer mit der Auswahl der toList Methode. Dabei kann sowohl der Name oder die zugehörige Zahl verwendet werden.

```

Input:      What is the population in the USA?

Output: I'm going to reduce the relation called country-has-state
        starting from country. Please specify the reduce function.

        Multiple Options possible, please choose one of them:
        1: switchingSort, 2: findMax, 3: selectionSort, 4: average,
        5: minimum, 6: maximum, 7: sum, 8: toList, 9: count

Input:      1

Output: Best connections found (2)
        usa->country-has-state->florida->population
        usa->country-has-state->california->population

        Result found (1)
        {21, 40}

// oder nicht sortiert, als Liste:

Input:      8

Output: Best connections found (2)
        usa->country-has-state->florida->population
        usa->country-has-state->california->population

        Result found (1)
        {40, 21}

```

5.4.2.3 Erkennen vorhandener Datenstrukturen

Neben dem Aufbau neuer Daten können die Benutzer im Dialog die vorhandenen Daten manipulieren. Dafür wird durch das *JustLingo NLUI* die Datenstruktur erfasst, in das UML-Modell überführt.

Folgende Daten liegen am Anfang vor:

Name	Owner	Reputation
MyHotel	Major	4+
FancyHotel	King	56+
DraculaHotel	King	100

Name	Age
King	23
Major	22

Tabelle 5.3: Eine Beispieldaten zum Einlesen der Daten

JustLingo NLUI überführt die Tabellenstruktur in das folgende Modell [Fuc18]:

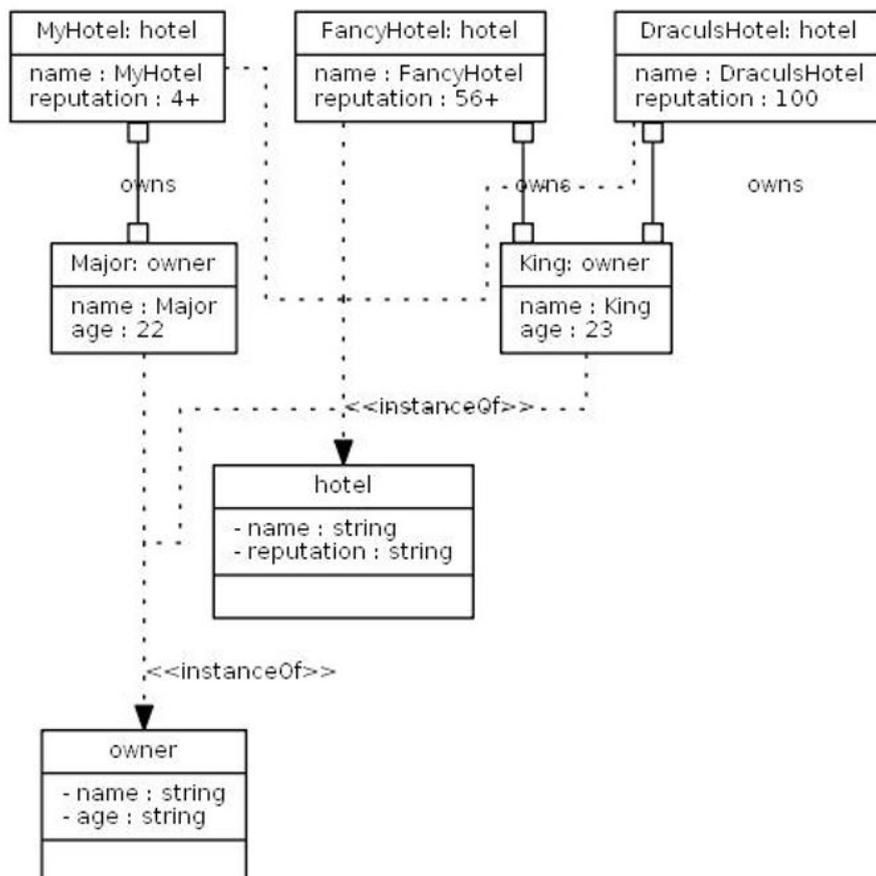


Abbildung 5.23: Modellierung einer Benutzerbeschreibung

Des Weiteren kann das UML-Diagramm in einen Code überführt werden:

```
class hotel
{
    private string name;
    private string reputation;
    private List<owner> myowner;

    public hotel (string newname, string newreputation, List<owner> newowner)
    {
        this.name = newname;
        this.reputation = newreputation;
        this.myowner = newowner;
    }
}

class owner
{
    private string name;
    private string age;
    private List<hotel> myhotel;

    public owner (string newname, string newage, List<hotel> newhotel)
    {
        this.name = newname;
        this.age = newage;
        this.myhotel = newhotel;
    }
}
```

5.4.2.4 Abfragen mehrerer Datenquellen

Eine Abfrage kann auf Daten aus unterschiedlichen Quellen stattfinden. Durch die Untersuchung wird der Satz in einzelne Teile für die Abfrage in der jeweiligen Datenquelle aufgeteilt und die jeweilige Teilabfrage mit der entsprechenden Datenquelle bearbeitet. Das Konzept der Datenstrukturanalyse ermöglicht JustLingo NLUI eine Struktur innerhalb einer textuellen Beschreibung sowie einer Tabelle zu erkennen und diese zum objektorientierten Modell zu transformieren. Wichtig dabei ist, dass die vorhandenen Objekte nicht in der Aktivontologie zur Spracherkennung vorhanden sein müssen.

Durch dieses Modell ermöglicht *JustLingo NLUI* das Abfragen verteilter Wissensbasen. Das Abfragen einer Datenbank mittels natürlicher Sprache ist durch das Natural Language Interface to a DataBase (NLIDB) bereits bekannt [ART95]. In diesem Zusammenhang untersuchten wir, wie verschiedene Quellen durch ein NLUI System angebunden und mittels natürlicher Sprache zur Laufzeit die Informationen eines heterogener Umgebung für die Bearbeitung der Anfrage eines Benutzers zur Verfügung gestellt werden. Das JustLingo NLUI erweitert den bisherigen Ansatz, indem es erlaubt, mehrere Datenquellen als Wissensbasis zu verwenden.

Ein Szenario aus der Wirtschaft [WFPT19] hat die Forschungsarbeit unterstützt. Die Anfragen der Kunden liegen als Tickets in einer SQL Datenbank und die Daten der Buchhaltung mit den Einnahmen und Ausgaben der Firma als Tabellenkalkulation vor. In diesem Zusammenhang sollten die Anfragen *Wie viel Einnahmen pro Kundenanfragen gibt es beim Kunden K_1 ?* oder *Welche Ausgaben verursacht ein Mitarbeiter M_1 pro Ticket bei dem Kunden K_1 ?* ermöglicht werden. Insbesondere lassen sich dadurch zwei Zwischenergebnisse berechnen, die für die Berechnung der Einnahmen/Ausgaben pro Kunden notwendig sind.

Insgesamt sind Daten in verschiedenen Formaten verfügbar. Daher wurde im Rahmen dieser Arbeit begonnen, Daten ebenfalls aus anderen Quellen, u.a. anderen SQL-Datenbanken, zu verarbeiten. Durch das Sammeln von Verbindungen in SQL-Tabellen wurde erste Änderung der Wissensbasis erstellt. Damit kann gezeigt werden, dass das NLUI nicht nur auf die Excel-Domäne beschränkt ist. Insbesondere wird die Kombination der verschiedenen Grundlagen unterstützt und davon abstrahiert. Dazu wird ein Metamodell verwendet. Jede Datenänderung wird vom Konverter inkrementell synchronisiert. Die Implementierung arbeitet mit den abstrakten Daten. Aus diesem Grund ist es unabhängig von der zugrunde liegenden Datenquelle. Der Anwendungsfall ist weiterführend und geht über die Verwendung von Tabellenkalkulation hinaus, soll jedoch an dieser Stelle die Mächtigkeit von *JustLingo NLUI* verdeutlichen.

staff_id (PK)	username	firstname (N)	lastname (N)
1	lastname1	Firstname1	Lastname1
2	lastname2	Firstname2	Lastname2
3	lastname3	Firstname3	Lastname3

ticket_id (PK)	number (N)	staff_id (FK)	source
1	90102	1	PHONE
2	89456	2	PHONE
3	34574	1	EMAIL

Tabelle 5.4: Übersicht der Mitarbeiter und den bearbeitenden Kundenanfragen

Tabelle 5.4 zeigt einen Teil eines Helpdesk-Ticketsystems. In Tabellen sind *Staff* und *Tickets*, an denen der Staff arbeitet, vertreten. Der Konverter generiert aus den Daten ein Metamodell. Spalten, die kein Attribut eines Objekts sind, werden als Primärschlüssel (PK) oder Fremdschlüssel (FK) markiert, um das Verfahren zum Herstellen von Verbindungen zwischen zwei Klassentypen zu vereinfachen. Im Gegensatz zu Kalkulationstabellen sind die namensgebenden (N)-Attribute meistens keine Schlüsselspalten (siehe Abbildung 5.24).

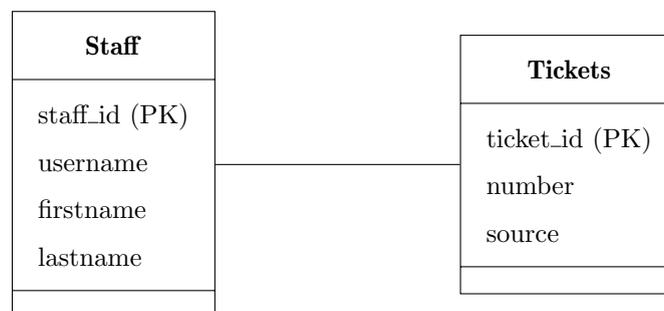


Abbildung 5.24: Schema der beiden Klassen in einer SQL-Datenbank

5.4.3 Erfassung der Folgealgorithmen

Die Benutzer beschreiben Algorithmen (Vgl. [CLLS09]) in ihrer natürlichen Sprache und erhalten eine Ausgabe durch das Dialogsystem. Ein Algorithmus besteht in der Regel aus Anweisungen, Schleifen sowie Bedingungen. Die Funktionalität richtet sich an Benutzer ohne Programmierkenntnisse, da mit dem System einfache Routinen ohne Vorkenntnisse programmiert werden können. Dies erleichtert Benutzern den Einstieg in die Programmierung. Das System veranschaulicht auch die Beziehung zwischen einer natürlichen Aussage und ihrer Codedarstellung.

Zunächst behandeln wir die Fragestellung, wie neue Folgealgorithmen anhand einer Ablaufbeschreibung in natürlicher Sprache erkannt und erlernt werden können. Basierend auf der Idee beschreibt das vorliegende Kapitel die Umsetzung und Integration einer natürlichsprachlichen Benutzerschnittstelle, welche die Funktion eines Kompilierers zwischen natürlicher und vom Computer verstandener Sprache veranschaulicht [WKT17].

Für dessen Umsetzung wurde eine linguistische Analyse entwickelt, welche in der Lage ist, grundlegende Werkzeuge der Programmierung, wie z.B. Variablen, Kontrollstrukturen und Zuweisungen in natürlicher Sprache zu erkennen. Des Weiteren wurde ein Interpretationsprozess entworfen, welcher ein von der Aktivontologie erkanntes Konzept in eine, ebenfalls als Teil dieser Arbeit, entworfene Datenstruktur überführt.

Beispiel der Überführung von einer Beschreibung zu einem Pseudocode [Eur17]:

```
The result is a vector. Initially it is empty. Find the minimal element of
the set and append it to the vector. Remove the element form the set. Then,
repeatedly find the minimum of the remaining elements and move them to the
result in order, until there are no more elements in the set.
```

↓

Algorithmus 5 SelectionSort

```
1: input as a set of numbers
2: result ← vector
3: while input IsNotEmpty do
4:   n ← length (input) - 1
5:   tmpMin ← 0
6:   for i ← 0 → n do
7:     if input[i] < input[tmpMin] then
8:       tmpMin ← i
9:     end if
10:  end for
11:  Add input[tmpMin] at the end of result.
12:  Remove element at index tmpMin from input.
13: end while
```

Als Sprachparser dient eine auf algorithmenspezifizierte Aktivontologie. Abbildung 5.25 zeigt die Struktur eines *WhileLoop*-Objektes in der Metasprache für die Benutzereingabe *Repeat all steps until the input is empty*. Wie zu sehen, besteht diese Objektstruktur aus drei Teilen: einem Schleifenindikator (weiter als *whileIndicator* verwendet), einer Schleifenaktion (weiter als *loopAction* verwendet) und einer Bedingung (weiter als *condition* verwendet).

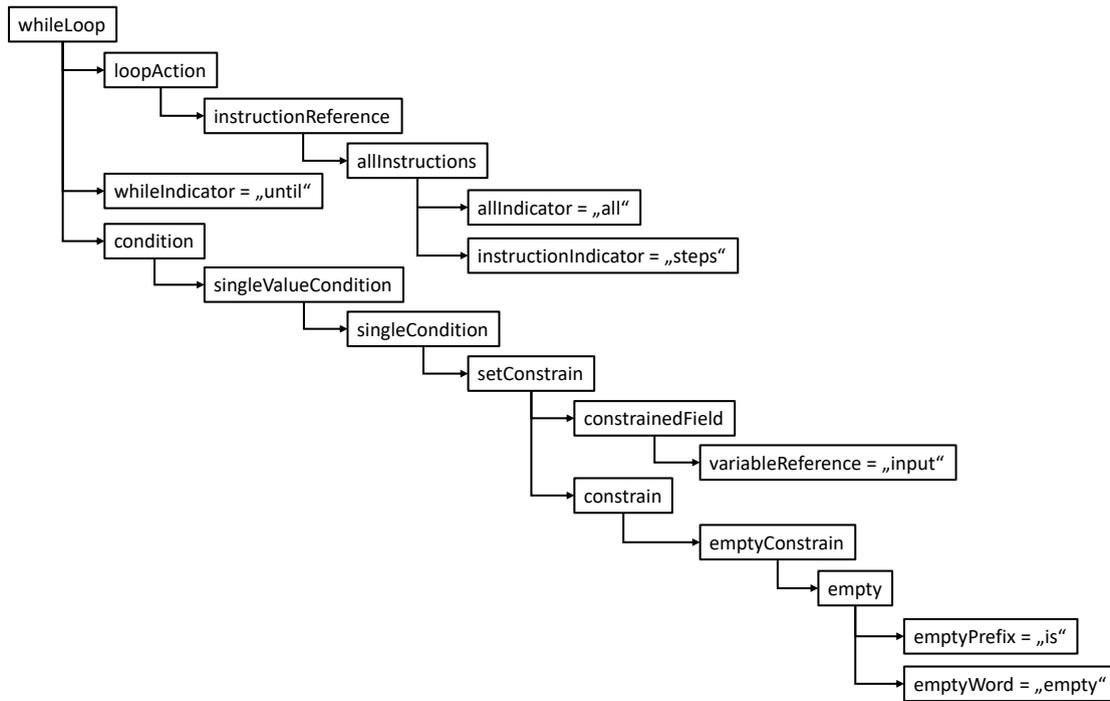


Abbildung 5.25: Auflösung einer While-Schleife in der Aktivontologie zur Benutzereingabe
“Visit all elements from the inputnach [WET18]

Die folgenden Abschnitte beschreiben die verschiedenen Schritte, wie das System die benötigten Objekte erstellt. Insgesamt identifiziert es die Routinen der unterstützten Konzepte, wie Anweisungen, Bedingungen und Schleifen, und bietet einen funktionalen Überblick über die linguistische Analyse.

5.4.3.1 Variablen

Der erste Schritt der Analyse ist die Referenzauflösung. Wie bereits im Kapitel *Dialogsystem* erläutert, müssen die sprachlichen Verweise der Benutzer aufgelöst werden. Im Allgemeinen verwenden die Benutzer in ihrer natürlichen Sprache implizite als auch explizite Verweise. Im Kontext der Algorithmenbeschreibungen können sie u.a. einen Verweis auf Variable als individuelles Pseudonym angeben, wie das Wort *Minimum* in der Eingabe *use 5 as Minimum*. Darüber hinaus kann eine implizite Referenz oder Anapher verwendet werden, z. B. *initialize it with 5*. Gepunktete Linien in der Abbildung 5.26 zeigen Sensor-knoten an, die reagieren, wenn eine der angehängten Wort- oder Wortklassen [TKMS03] in der Eingabe erkannt wurde.

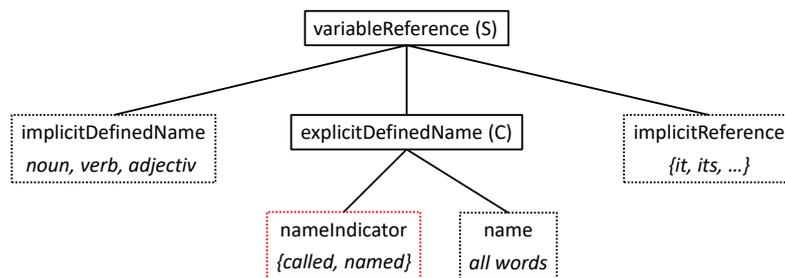


Abbildung 5.26: Struktur der Ontologie für die Auflösung der Variablenreferenzen
[WET18]

Eine rote Markierung um einen Knoten identifiziert einen Schlüsselknoten, dessen Auf-

treten für den Elternknoten wesentlich ist. Die Knotenarten zum Sammeln sind mit "C" und zum Auswählen mit "S" markiert. Die verschiedenen Arten von Knoten haben für die Bearbeitung eine unterschiedliche Bedeutung. So reagieren die Blattknoten auf die Wörter in der Benutzereingabe. Sammelknoten reagieren, wenn alle erforderlichen untergeordneten Knoten bereits aktiv reagiert haben. Dagegen werden die Auswahlknoten benutzt, sobald mindestens ein untergeordneter Knoten aktiviert wird. Bei der Auslösung wird das erkannte Konzept als neues Objekt an den übergeordneten Knoten übergeben.

5.4.3.2 Bedingungen

Bedingte Anweisungen sind aufgrund ihrer komplexen grammatikalischen und kontextuellen Struktur oft schwer erkennbar. Des Weiteren sind Referenzen in dieser Art von Sätzen kompliziert zu lösen. Der Vorteil von Bedingungen besteht darin, dass sie eine kleine Menge von Schlüsselwörtern enthalten (zum Beispiel *if*, *in case of*, *whenever*, *provided that*, *given*, usw.), die darauf hinweisen, dass der Benutzer eine bedingte Anweisung verwendet. Im Bereich der Tabellenkalkulationsmanipulation muss die Bedingung zu einer booleschen Operation aufgelöst werden, um diese verstehen zu können [WKT17]. Daher wurde eine Fähigkeit entwickelt, der sich mit bedingten Aussagen befasst und auf bestimmte Schlüsselwörter reagiert, um einen booleschen Ausdruck in der Benutzereingabe zu finden. Wenn keine boolesche Operation gefunden wird, fragt das Dialogsystem beim Benutzer nach. Nach der Antwort wird die Aktion ausgeführt. Dabei spielt wegen der Aktivontologie die Reihenfolge der Wörter keine Rolle.

Wie bereits erwähnt, wird *if* als Bedingungsschlüsselwort erkannt und JustLingo NLUI mit einer bedingte Anweisung zu Handeln animiert. *A1 ist größer als 3* ist eine boolesche Operation und kann als Bedingung verwendet werden. Die triviale Anweisung *save 5 in B1* kann als bedingungslose Aktion bzw. Anweisung erkannt und von unserem System verarbeitet werden (Siehe Abbildung 5.27).

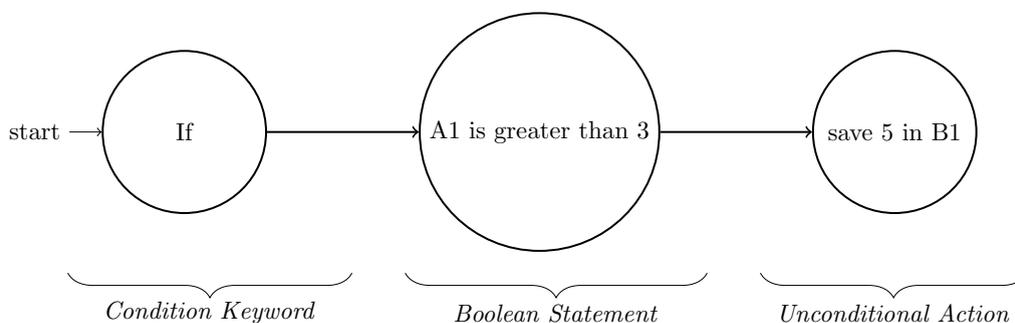


Abbildung 5.27: Beispiel einer bedingten Anweisung [WKT17]

Um Bedingungen erkennen zu können, erweitern wir die aktiven Ontologien, die das Wissen über die Struktur der natürlichen Sprache im Kontext von Programmieranweisungen darstellen. Kontrollstrukturen, wie Schleifen und Verzweigungen, sind wesentliche Elemente von Algorithmen. Durch die Berücksichtigung solcher Konzepte in der linguistischen Analyse können komplexere Algorithmen, wie Selection sort, implementiert werden. Abhängig von der Art der Referenz können unterschiedliche Bedingungen festgelegt werden [WET18]:

- eine Bedingung für einen einfachen Wert,
- eine Menge von Werten oder
- eine Mengenreferenz.

Abbildung 5.28 zeigt die Klassifizierung der Bedingungen. Zunächst wird zwischen den Zuständen unterschieden, die sich auf eine einzelne Referenz (`singleValueCondition`) beziehen oder eine zusammengesetzte Bedingung (`multipleValueCondition`) darstellen. Die zusammengesetzten Bedingungen stellen mehrere Bedingungen durch eine Kombination, wie *x ist kleiner als 3 und y ist gleich 5*, dar.

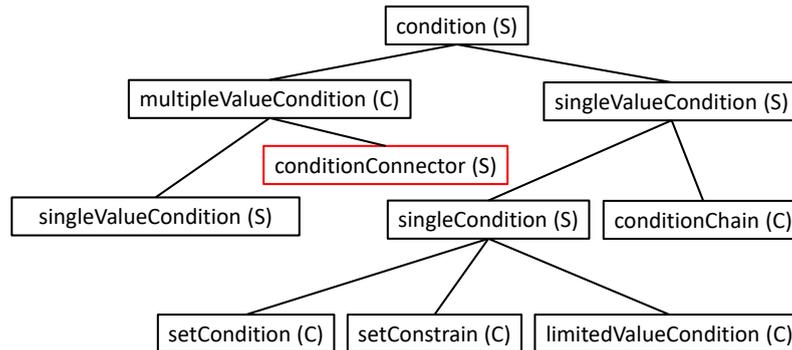


Abbildung 5.28: Vereinfachte Struktur einer Aktivontologie für die Bedingungsstypen

Darüber hinaus unterscheiden sich die einzelnen Bedingungen zwischen Anweisungen, die einem Wert eine Bedingung zuweisen (`singleCondition`) und solchen mit mehreren Voraussetzungen für einen Wert (`conditionChain`). Das Kind der Knoten `singleCondition` repräsentiert die drei unterstützten Typen einer Bedingung. `limitedValueCondition` erkennt Anweisungen, die zwei Werte vergleichen. Bedingungen, die sich auf eine Menge beziehen, zum Beispiel wenn das Ergebnis leer ist, werden durch `setConstrain` identifiziert. Zudem erkennt `setCondition` die Anforderungen einer Eigenschaft für bestimmte Elemente einer Menge, zum Beispiel wenn *alle Elemente des Ergebnisses größer als 4* sind.

5.4.3.3 Schleifen

Im Gegensatz zu Bedingungen muss die Schleife nicht unbedingt eindeutige Schlüsselwörter enthalten. Oft sind diese Schlüsselwörter im Satz versteckt, wie *für alle, für jeden, solange*. Es kann jedoch auch explizite Anweisungen wie *dreimal etwas tun* geben.

Sobald eines dieser Schlüsselwörter erkannt wurde, muss das System die entsprechende Aktion für die angegebenen Daten ausführen. Damit unsere aktive Ontologie die richtige Aktion erkennen kann, haben wir ein Looptarget eingeführt. Dies ist ein Operator, der sich wie eine normale Zelle verhält und durch normale Aktionen (wie Speichern, Multiplizieren) erkannt werden kann. Sobald der Schleifendienst durch ein Schleifenschlüsselwort aktiviert wurde, reagiert er auf Aktionen, die die betreffenden Ziele enthalten. In diesem Fall besteht der nächste Schritt darin, die Eingabe für jede Zelle des angegebenen Bereichs in mehrere Anweisungen aufzuteilen. Dies kann auch mit bedingten Anweisungen kombiniert werden. In der Regel besteht die Schleife bei *JustLingo NLUI* aus einem Rumpf, einer Anzahl der Wiederholungen und einer Aktion innerhalb der Schleife. Benutzereingabe *Save 3 in each cell of column A* wird vom System korrekt verarbeitet (Siehe Abbildung 5.29).

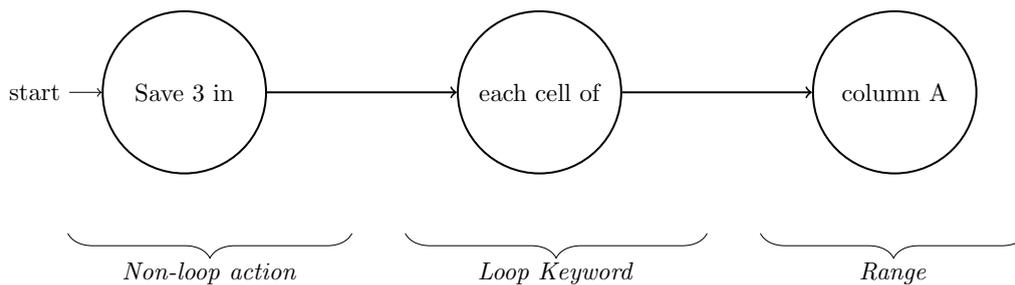


Abbildung 5.29: Beispiel einer wiederholenden Anweisung [WKT17]

Beispiel eines Dialogs Um die erweiterten Funktionen zu demonstrieren, wird ein Dialog zwischen *JustLingo NLUI* und einem Benutzer gezeigt. Für jede Iteration des Dialogs haben wir die Benutzereingabe, die intern erkannte Operation, die das System als Folge dieser Eingabe ausgeführt hat, und die vom System generierte Antwort aufgelistet:

```

Antwort:      How can I help you?

Benutzer:     Write 30 in all cells of column A

Aktion:       save(30, A:A)

Antwort:      Already done. Is there anything else i can do?

Benutzer:     Subtract 10 from each cell between A1 and A10

Aktion:       save(A1:A10 - 10, A1:A10)

Antwort:      Ok, I did that for you. Can I do something else for you?

Benutzer:     No

Antwort:      Alright
...
After a break, user turns back
...

Benutzer:     For each cell of column A which is greater than 25 add 2 to B2

Aktion:       conditionalAction(A:A$>$25, save(B2 + 2, B2))

Benutzer:     If A3 is greater than A2 write 2 in A3 else in B3

Aktion:       conditionalAction(A3$>$A2, save(2, A3), save(2, B3))

Antwort:      Ok, I did that for you.

Benutzer:     If A5 is greater than B5

Aktion:       NoAction

Antwort:      I see you used a condition.
              What do you want to do if the condition is true?

Benutzer:     Save 4 in B3

Aktion:       conditionalAction(A5 $>$ B5, save(4, B3))

Antwort:      Everything done as you wished.
  
```

Um eine Aktion zu wiederholen, kann der Benutzer diese mit einer Bedingung verknüpfen oder angeben, dass sie für jedes Element einer Gruppe ausgeführt werden soll. Daher erkennt die Analyse sowohl **for** als auch **while**-Schleifen. Das Ergebnis der Ontologie für eine erkannte **for**-Schleife ist in Abbildung 5.30 zu sehen.

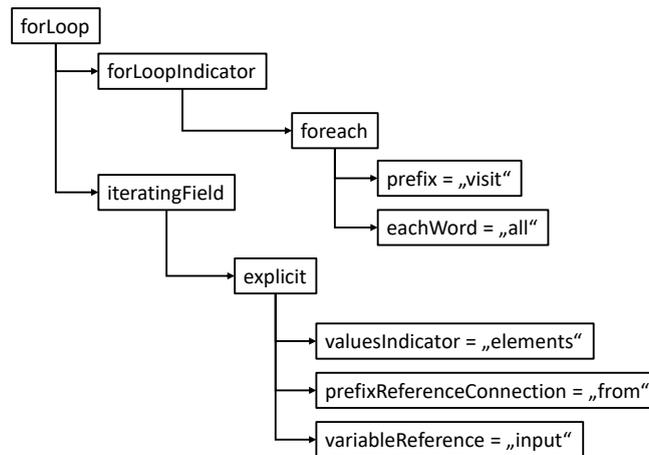


Abbildung 5.30: Ergebnis einer erkannten Eingabe: *visit all elements from input*

5.4.3.4 Implementierung von JustLingo.Algorithms

JustLingo NLUI erkennt verschiedene Programmierkonzepte in Benutzereingaben und konvertiert sie in eine Metasprache. Das Ergebnis der Analyse ist eine Baumstruktur aus Objekten, wobei ein Objekt ein erkanntes Konzept darstellt. Diese Bäume werden während der linguistischen Analyse von unten nach oben aufgebaut. Abbildung 5.31 zeigt den internen Aufbau des Dienstes zur Algorithmenerkennung:

- bei einer Auswahl des Dienstes durch den Dialogmanager wird im ersten Schritt eine linguistische Analyse der natürlichsprachlichen Eingabe durchgeführt.
- Als nächstes untersucht *InterpreterService* die vorverarbeitete Eingabe nach enthaltenen Aktionen, Referenzen und temporären Informationen.
- Nach einer Eingabe des Benutzers wird der Kontext und Programmzeiger mit der aktuellen Position im Pseudocode aktualisiert.

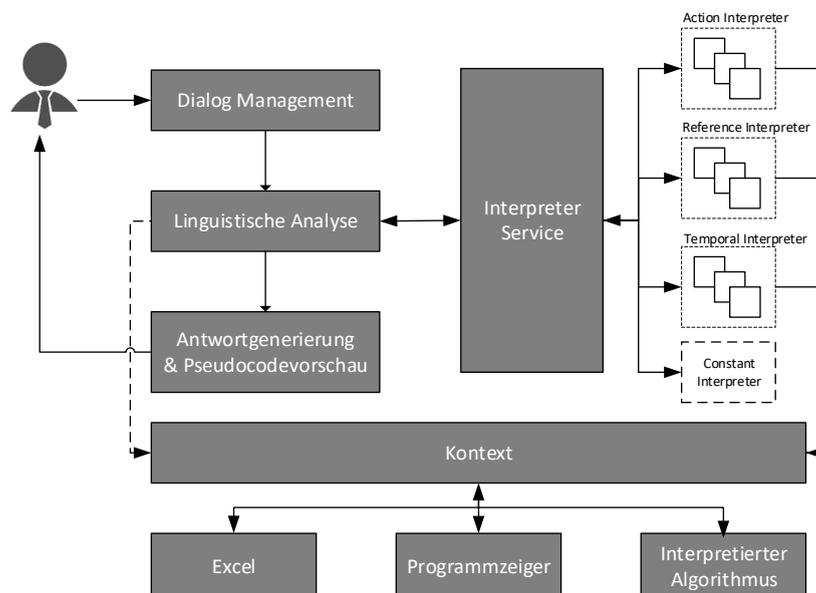


Abbildung 5.31: Interne Struktur von JustLingo.Algorithms

5.4.4 Fazit

Durch die NLUI-Fähigkeit Datenstrukturanalyse wird eine Beschreibung aus Text oder einer vorliegenden Tabelle als Klassenmodell modelliert. In diesem Fall erlaubt das Vorgehen einem Benutzer Anfragen auf diesen Daten, die *JustLingo NLUI* nicht vorher bekannt waren. Diese Fähigkeit zeichnet *JustLingo NLUI* aus, weil in der Regel die NLUIs nur mit vorbereiteten Daten umgehen können. Weitere NLUI-Fähigkeit ist die Erfassung von Folgealgorithmen aus textuellen Beschreibungen in natürlicher Sprache. Dabei erfolgt keine lineare Abbildung der natürlichen Sprache zum Code. Die Beschreibung wird zwar Satz-für-Satz bearbeitet, die resultierenden Zeilenreihenfolge im Code variiert in Abhängigkeit der bereitgestellten Information. Durch die bereitgestellten Funktionen haben die Benutzer eine Möglichkeit, die Eigenschaften eines Objekts zu manipulieren.

5.5 Zusammenfassung

Dieses Kapitel stellt die Kernkomponenten der natürlichsprachlichen Benutzerschnittstelle *JustLingo NLUI* vor. Im Mittelpunkt stehen das Dialogsystem sowie die einzelnen Fähigkeiten. Die NLP-Fähigkeiten verantworten die korrekte Interpretation einer Benutzeranfrage. Die Modellierung der Klassen und die Erkennung eines Algorithmus erfolgt aus natürlichsprachlichen Beschreibungen durch die NLUI-Fähigkeiten. Benutzer beschreiben ständig unbekannte Informationen. Eine Aktivontologie kann nicht auf alles mögliche vorbereitet werden. Mit Hilfe der Datenstrukturanalyse werden vorher unbekannte Daten modelliert. Bei einer vorliegenden Beschreibung des Benutzers können die einzelnen Sätze zu den jeweiligen Codefragmenten transformiert werden.

6. Evaluation

Die Forschungsarbeit an der natürlichsprachlichen Schnittstelle *JustLingo NLUI* beinhaltet unabhängig durchgeführte Studien, welche die einzelnen Bestandteile untersuchen und einen Nachweis der Funktionalität einzelner Konzepte und deren Implementierung liefern.

Die Evaluation unterteilt sich in fünf Studien:

- Die erste Studie wird im Kapitel 6.1 vorgestellt, indem ein Vergleich des natürlichsprachlichen Parsers erfolgt. Dabei werden die Parser mit einem musterbasierten Ansatz in Kapitel 6.1.1 und einer Aktivontologie in Kapitel 6.1.2 gegenübergestellt.
- Kapitel 6.2 präsentiert eine Studie, in der eine Klassifikation der Benutzereingaben zu den implementierten NLUI-Fähigkeiten erfolgt. Dabei ziehen wir den Vergleich einer Klassifikation mit balancierten Trainingsmengen in Kapitel 6.2.1 sowie mit unbalancierten Trainingsmengen in Kapitel 6.2.2.
- Die Datenstrukturanalyse steht im Kapitel 6.3 in Vordergrund. Dabei evaluiert Kapitel 6.3.1 die Modellierung von Klassen, Attributen, Methoden und Beziehungen in den natürlichsprachlichen Beschreibungen. Die Erstellung der Klassenmodelle aus Tabellen wird in Kapitel 6.3.2 vorgestellt.
- Kapitel 6.4 evaluiert eine Erweiterung der Funktionalität einer natürlichsprachlichen Schnittstelle mit Hilfe einer Benutzereingabe. Dabei wird die Erkennung von Kontrollstrukturen in Kapitel 6.4.1 beschrieben. Anschließend wird die Erfassung einfacher Folgenalgorithmen in Kapitel 6.4.2 untersucht.
- Nicht zuletzt führt Kapitel 6.5 eine Studie zur Akzeptanz von JustLingo NLUI durch.

Zum Schluss werden in Kapitel 7.1 die erreichten Ergebnisse evaluiert und in einen Bezug zu den verfassten Forschungsfragen aus dem Kapitel Kapitel 3.5 gesetzt.

6.1 Studie I: Vergleich der Parser

Diese Studie evaluiert die eingesetzten Parser.

6.1.1 Studie I.1: Musterbasierter Parser

Der musterbasierte Ansatz eines Parsers ist häufig in der Implementierung einer natürlichsprachlichen Schnittstelle zu finden. Aus diesem Grund wollten wir eine Evaluation zur Performanz eines XML-basierten Parsers durchführen [WWT15].

6.1.1.1 Aufbau

An der Evaluation eines musterbasierten Parsers nahmen 17 Teilnehmer teil, die vorher keine Erfahrung mit *JustLingo NLUI* hatten. 15 der Teilnehmer haben Microsoft Excel mindestens fünf Jahre lang gelegentlich verwendet, zwei Teilnehmer dagegen hatten es nie im Einsatz. In der Evaluation nutzten die Teilnehmer die natürlichsprachliche Schnittstelle, um mehrere Aufgaben in einer Tabellenkalkulation zu erledigen. Einige Aufgaben bauen aufeinander auf, um die Verwendung von Referenzen auf bereits mitgeteilte Information zu fördern.

Die Aufgabenstellungen zielen auf mathematische Berechnungen und forcierten eine natürlichsprachliche Formulierung einer Mehrdeutigkeiten, eines Rückbezügen oder einer komplexen Operation zu einem gegebenen Szenario in Tabellenkalkulation [Voi14].

Anleitung für Teilnehmer (Mehrdeutigkeit):

Arbeiten Sie zur Lösung der folgenden Aufgaben selbst mit dem Dialogsystem. Fassen Sie es weiterhin als menschlichen Ansprechpartner auf, den Sie zur Lösung der Aufgaben um Hilfe bitten. JustLingo wird abhängig von der Eingabe ggf. mit einer Rückfrage antworten. Reagieren Sie darauf wie Sie es in einem gewöhnlichen Gespräch täten.

Anleitung für Teilnehmer (Rückbezug):

Es werden kleinere Prämien in einer Gesamthöhe von 3000 Euro an weitere Mitarbeiter ausgezahlt. Addieren Sie 3000 Euro auf das vorherige Ergebnis. Hinweis: Beziehen Sie sich dabei auf die Lösung aus Schritt 1.

Anleitung für Teilnehmer (Komplexe Berechnung):

Gehen Sie davon aus, dass 68% der Gewinne (nach Abzügen) im Jahre 2013 für Investitionen verwendet werden sollen. Berechnen Sie diesen Wert. Hinweis: Beziehen Sie sich auf das Ergebnis in Zelle E21.

6.1.1.2 Ergebnisse

Das System löst nicht nur erfolgreich Verweise auf die zuvor bereitgestellten Informationen auf, sondern handelt auch aktiv und stellt relevante Rückfragen:

```
JustLingo NLUI: How can I help you?

Benutzer:      Calculate 1000 multiplied by the sum of E6 and E7

JustLingo NLUI: For computing 1000 multiplied by the sum of E6 and E7 you
                require the formula (1000*(E6+E7)) which results in 9000.
                I can store this somewhere in the worksheet if you like?

Benutzer:      now add 3000

JustLingo NLUI: For computing 1000 multiplied by the sum of E6 and E7
                plus 3000 you require the formula (1000*(E6+E7))+3000
                and the output is 12000.
                I can write that somewhere in your worksheet?

Benutzer:      Save that formula in E8

JustLingo NLUI: Alright, something else then. It was put to E8.
```

Am interessantesten ist die Gesamterfolgsquote sowie die Ergebnisse in Bezug auf die drei Hauptziele: Anzahl der Rückfragen, Erkennung der Rückfragen und Erkennung der Mehrdeutigkeiten (Siehe Tabelle 6.1).

Insgesamt	
Anzahl der Aufgaben	170
Anzahl der Dialogiterationen	316
Erfolgreich gelöst	79,5%
Ziele	
Rückfragen des Systems	17,1%
Erkennung der Rückbezüge	82%
Erkennung der Mehrdeutigkeiten	34,8%

Tabelle 6.1: Auswertung der Evaluation nach [WWT15]

Die Teilnehmer wurden gebeten, die Gesamtleistung des Systems subjektiv zu bewerten. Über 80% bewerteten das System als intuitiv bedienbar und 70% empfanden einen deutlichen Mehrwert für sich. Die Abfragen des Dialogsystems galten in 68% der Fälle als hilfreich. Besonders Anfänger profitierten vom Dialog. Im Hinblick auf die zukünftige Arbeit ist auch der beobachtete Effekt bemerkbar, dass Benutzer, obwohl das System keine Einschränkungen aufweist, sich innerhalb weniger Anforderungen an den Prototyp anpassen kann, indem Formulierungen verwendet werden, die sie als vom System ordnungsgemäß verarbeitet erfahren haben.

6.1.2 Studie I.2: Parser mit Aktivontologien

Nach dem musterbasierten Ansatz wurde ein Parser durch eine Aktivontologien [WKT16] eingesetzt. Der Vorteil einer Aktivontologie im Vergleich zu einem Muster ist, dass die Reihenfolge der Wörter in einer Benutzereingabe für das Sprachverständnis nicht mehr wichtig ist.

6.1.2.1 Aufbau

Wir haben 16 Teilnehmer gebeten, mehrere Aufgaben in Microsoft Excel ausschließlich mit unserem Dialogsystem auszuführen und Feedback zur Leistung des Dialogsystems zu geben. Die Aufgaben dienten der Evaluation mehrerer komplexer Probleme wie das Lösen von Referenzen, Mehrdeutigkeiten und Bedingungen [Kla15].

1. Eintragung der Werte:

- Tragen Sie eine zufällig generierte Zahl zwischen 1000 und 1500 Euro für den Januar ein (C2).
- Tragen Sie bei schlechtesten Monat und Monatsdurchschnitt jeweils die fehlenden Felder ein.
- Tragen Sie in D10 die für diesen Monat anfallende Umsatzsteuer ein.

2. Einfache Operation:

- Im April wurde der Nettogewinn nicht berechnet. Tragen Sie dort (B5) die entsprechenden Netto-Einnahmen ein, indem Sie sich auf die Brutto-Einnahmen des selben Monats beziehen (C5).

3. Zellenübergreifende-Operationen:

- Der Chef möchte eine Gesamtbilanz des Jahres ziehen. Berechnen Sie die Summe aller Netto-Einnahmen des Jahres und tragen sie diese anschließend in B15 ein.

4. Mehrdeutigkeiten:

- Berechnen Sie ausgehend von den Brutto-Einnahmen des Jahres die Gesamtabgaben inklusive Umsatzsteuer und Bearbeitungsgebühr und speichern Sie diese in E15.

5. Bedingungen:

- Tragen Sie in E5 und E13 jeweils wie viele Abgaben anfallen, ohne dies selbst zu überprüfen (100 bei Brutto-Einnahmen über 1500 Euro, 0 sonst).

Arithmetische Operationen:

- Einfache Funktion: Berechnen Sie den Mittelwert der Erträge eines Kunden (Hinweis: Erträge befinden sich in den Zellen B17 bis E17).
- Einfache Rechenoperation: Berechnen Sie die Dividende, die insgesamt pro Anteil (steht in der Zelle A6) einer Dividende (Dividende pro Anteil ist in B6 angegeben) ausgezahlt wird.
- Speichern des Ergebnisses: Subtrahieren Sie das vorherige Ergebnis von 1 und speichern Sie es in B7.

Folgende Auflistung zeigt eine Beispielkonversation:

```
JustLingo NLUI: How can I help you?
Benutzer:      Please calculate the product of A6 and B6.
JustLingo NLUI: When I calculate A6 multiplied by B6 the formula
                is (A6*B6) and the result is 0.19872.
JustLingo NLUI: Should I store this somewhere in your document?
```

6.1.2.2 Ergebnisse

Die Studie zeigt, dass die Probanden fast 90% der Aufgaben lösen konnten. Dabei wurde das NLUI bei 76% der Aufgaben als hilfreich bewertet. Die Teilnehmer bewerteten das NLUI insgesamt in 15 von 16 Fällen als gut und sehr gut. Mehr als 80% der Teilnehmer gaben an, das NLUI zumindest gelegentlich zu verwenden. Nicht zuletzt waren die Antworten des NLUI nach Angaben unserer Teilnehmer in 88% der Fälle angemessen, und 75% der Testpersonen empfanden die Antworten als natürlich oder sehr natürlich. Die Benutzererfahrung mit dem NLUI ist sehr positiv. 75% der Teilnehmer gaben an, das Dialogsystem als natürlich oder sehr natürlich wahrzunehmen. Die Gesamtqualität unseres Prototyps wurde von 15 der 16 Teilnehmer als gut oder sehr gut bewertet. Tabelle 6.2 zeigt die Gesamtauswertung der Ergebnisse.

Insgesamt	
Anzahl der Aufgaben	112
Anzahl der Dialogiterationen	376
Erfolgreich gelöst	89,5%

Tabelle 6.2: Auswertung der Evaluation nach [WKT16]

6.1.3 Fazit

Die Evaluation zeigt, dass ein musterbasierter Ansatz für die Erkennung einer Benutzereingabe möglich ist, jedoch einige Nachteile aufweist. Die Benutzereingabe wird erfolgreich verarbeitet, solange das Muster bereits in der Wissensdatenbank vorhanden ist. Wenn das

Muster fehlt, kann das System die Benutzereingabe nicht erkennen. Ein Nachteil der Muster ist, dass *zu viele Permutationen pro zu erkennende Aktion* benötigt werden. Dabei müssen spezifische bis allgemeine Muster der natürlichsprachlichen Schnittstelle zur Verfügung gestellt werden. Dies lässt einem Benutzer keine Möglichkeit, die Eingabe sowohl im Vokabular als auch in der Reihenfolge der Wörter zu variieren.

Die Studie offenbart, dass ein AO-basierter Erkenner gegenüber einem musterbasierten Ansatz bessere Ergebnisse in der Spracherkennung liefert. Das liegt daran, dass die Aktivontologie sämtliche Treffer aus der Eingabe in den Knoten sammelt und zur Verarbeitung weiterverwendet. Die Reihenfolge der Wörter spielt keine Rolle.

In der Nutzerstudie wurde die Erkennung mit einer musterbasierten Grammatik und einer Aktivontologie verglichen. Nach der Evaluation eines Dialogsystems mit einem AO-basierten Ansatz ist eine deutliche Verbesserung messbar (Siehe Tabelle 6.3):

- Obwohl die Anzahl an Teilaufgaben kleiner war, verzeichnet *JustLingo NLUI* mehr Dialogiterationen und kann die Lösungsrate von 79,5% auf 89,5% steigern
- Durch den Einsatz der AO kann *JustLingo NLUI* mehr Eingaben verstehen und beantworten, so dass sich die Bewertung der Antwortgenerierung des Dialogsystems von 40% auf 75% verbessert
- Die Gesamtzufriedenheit der Benutzer bei der Verwendung von *JustLingo NLUI* stieg von 70% auf 80%

Studie	Studie I.1: musterbasiert	Studie I.2: AO-basiert
Anzahl der Teilnehmer	17 Personen	16 Personen
Anzahl der Aufgaben	170	112
Anzahl der Dialogiterationen	316	376
Gesamtbewertung		
Lösungsrate	79,5%	89,5%
Antwortgenerierung	40%	75%
Zugriedenheit	70%	80%

Tabelle 6.3: Gegenüberstellung beider Evaluationen [WKT16]

Beispiel eines Dialogs aus der Evaluation:

Benutzer:	Please calculate the product of A6 and B6.
JustLingo NLUI:	I believe you need (A6 * B6) which results in 0,19872. What else can I help you with?
Benutzer:	No, calculate C5 minus D5 multiplied with C5.
JustLingo NLUI:	Your input may be interpreted in various ways: a) (C5 - product(D5, C5)) b) product((C5 - D5), C5). Which one should I use?
Benutzer:	First one.
JustLingo NLUI:	Where do you want me to save that in the worksheet?
Benutzer:	B5
JustLingo NLUI:	Ok, I did that for you.

6.2 Studie II: Klassifikation

Die Evaluation untersucht die Zuordnung einer Benutzereingabe zu den implementierten NLUI-Fähigkeiten. Es ist außerdem zu untersuchen, ob es eine Auswirkung des Ergebnisses bei unbalancierter Verteilung in den Trainingsdaten gibt.

6.2.1 Studie II.1: Balancierte Datensätze

Diese Evaluation präsentiert die Ergebnisse mit balancierten Datensätzen, also gleichverteilten Trainingssätzen pro NLUI-Fähigkeit. Zur Evaluierung standen zu dem damaligen Zeitpunkt folgende NLUI-Fähigkeiten zur Verfügung: mathematische Berechnungen (Math), Klassenstruktur (Objects), Datenstrukturanalyse (Data) sowie Algorithmenerkennung (Algo).

6.2.1.1 Aufbau

In [WEFT20] haben einen gleich verteilten Datensatz von 200 Proben aus verschiedenen Büchern des betreffenden Fachgebiets genommen:

- Basic Engineering Mathematics [Bir10]
Beispielsatz: what is the square root of 49?
- The State of the Art in End-User Software Engineering [KAB⁺11]
Beispielsatz: Researchers have also extended their focus from perfective activities to design, including work on requirements, specifications, and reuse.
- Data Analysis Using SQL and Excel [Lin08]
Beispielsatz: The definition of the tables, the columns, and the relationships among them constitute the data model for the database.
- Introduction to Algorithms [CLLS09]
Beispielsatz: Typically, we use the loop invariant along with the condition that caused the loop to terminate.

Aus jedem Buch wurden 50 zufällige Sätze entnommen und jeweils zu einem Datensatz zusammengefasst. Als nächstes wurden die Sätze aus jedem Datensatz nach einem Zufallsprinzip auf in 80% Trainings- und 20% Testdaten geteilt. Wie üblich, wird der Klassifikator mit Trainingsdatensätzen trainiert und anschließend die Klassifizierung mit Testdatensätzen bewertet.

Beispielsätze:

```
do the previously steps until the input is empty

whenever an element of the result is larger than the auxiliary variable
iterate over x

x should be contained in result or the first value of the output is equal 10

set all elements of result less than 3 to minimum

remove the first value of numbers from the the input

perform this if x is smaller than 5, otherwise do the previous step
```

Die vollständige Auflistung der Datensätze sind im Anhang 9.4.2 zu finden.

6.2.1.2 Ergebnisse

Die Konfusionsmatrix bewertet die Ergebnisse in der Tabelle 6.4. Für diese Evaluation berechnen wir die Werte für Genauigkeit, Präzision, Sensitivität sowie F-Wert ausführlich.

		Aktuelle Klasse			
		Objects	Math	Data	Algo
Vorhersage	Objects	46	4	0	0
	Math	3	44	3	0
	Data	1	6	43	0
	Algo	3	3	3	41

Tabelle 6.4: Konfusionsmatrix bei balancierten Datensätzen nach [WEFT20]

$$\text{Genauigkeit} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{46 + 44 + 43 + 41}{200} = 0,87 \quad (6.1)$$

$$\text{Präzision}_{Obj} = \frac{TP}{TP + FP} = \frac{46}{46 + 4} = 0,92 \quad (6.2)$$

$$\text{Ausbeute}_{Obj} = \frac{TP}{TP + FN} = \frac{46}{46 + 3 + 1 + 3} = 0,87 \quad (6.3)$$

$$F - \text{Wert}_{Obj} = 2 * \frac{\text{Präzision}_{Obj} * \text{Ausbeute}_{Obj}}{\text{Präzision}_{Obj} + \text{Ausbeute}_{Obj}} = 2 * \frac{0,92 * 0,87}{0,92 + 0,87} = 0,89 \quad (6.4)$$

Element	Präzision	Ausbeute	F-Wert
Objects	0.92	0.87	0.89
Math	0.88	0.77	0.82
Data	0.86	0.88	0.87
Algo	0.82	1.0	0.90
Gesamt	0.87	0.88	0.87

Tabelle 6.5: Ergebnisse der Auswertung

6.2.2 Studie II.2: Unbalancierte Datensätze

Für die Evaluation des Klassifikators [WEFT20] wurden weitere Tests mit unbalancierten Datensätzen durchgeführt.

6.2.2.1 Aufbau

Aus den Datenquellen wurden zufällig, jedoch nicht mehr gleichverteilt, 233 Proben entnommen und in 80% Trainings- und 20% Testdatensätze per Zufall aufgeteilt. Zunächst wurde der Klassifikator mit dem Trainingsdatensatz von 189 Proben trainiert und anschließend auf dem Testdatensatz evaluiert. Insgesamt sah die Verteilung der Proben wie folgt aus: Algorithmerkennung (kurz: Algo, 48 Trainingsätze | 12 Testsätze), Datenstrukturanalyse (kurz: Data, 24 | 5), Erstellung von Diagrammen (kurz: Chart, 8 | 2), Mathematische Berechnungen (kurz: Math, 56 | 13), und Klassenmodelle (kurz: Objects, 53 | 12).

6.2.2.2 Ergebnisse

Obwohl die Anzahl der Stichproben pro NLUI-Fähigkeit nicht ausgeglichen ist, gab es keine gravierenden Auswirkungen auf das Ergebnis der Klassifikation (Genauigkeit = 0,95).

		Aktuelle Klasse				
		Algo	Data	Chart	Math	Objects
Vorhersage	Algo	11	0	0	0	0
	Data	0	5	0	0	0
	Chart	1	0	1	0	0
	Math	0	0	0	13	0
	Objects	0	0	1	0	12

Tabelle 6.6: Konfusionsmatrix bei unbalancierten Datensätzen nach [WEFT20]

Element	Präzision	Ausbeute	F-Wert
Algo	1	0.92	0.96
Data	1	1	1
Chart	0.5	0.5	0.5
Math	1	1	1
Objects	0.92	1	0.96
Gesamt	0.88	0.88	0.88

Tabelle 6.7: Ergebnisse der Auswertung

6.2.3 Fazit

Der gewählte Ansatz der Klassifikation lässt als NLP-Fähigkeit eine zuverlässige Zuordnung einer Benutzereingabe zu den implementierten NLUI-Fähigkeiten auch bei kleinen und unbalancierten Trainingsmengen des Klassifikators zu. Durch die zusammengesetzte asynchrone Berechnung der Klassifikatoren trägt diese Studie zur Evaluation der horizontalen Dialogführung bzw. der zugehörigen Forschungsfrage bei und kann auch mit kleinen Trainingsmengen gute Ergebnisse erzielen.

6.3 Studie III: Datenstrukturanalyse

Die Studie evaluiert die Fähigkeiten Klassenmodelle zu erkennen.

6.3.1 Studie III.1: Klassenmodelle aus Text

Dieses Kapitel evaluiert die Erkennung von Klassen mit Attributen und Methoden sowie den Beziehungen zwischen den Klassen. Großer Teil der Evaluation wurde zusammen mit [WFPT19], [Fuc18] und [Wie18] durchgeführt.

6.3.1.1 Aufbau

An neun verschiedenen Aufgaben aus der Vorlesung Softwaretechnik wurden die UML-Modellierungsfähigkeiten von *JustLingo NLUI* getestet. Die vollständige Auflistung ist in [Wie18] zu finden.

Beispiel einer Eingabe:

```
Every station has a big display which shows the actual departure times of the
trains. Furthermore the display shows the track number, the destination and
the type of the train, for example a ICE. The information shown on the displays
can also be observed on monitors of the station tracks. At each track there are
additional track boards which only show the data of the next train departing
on it. The central station control manages all necessary data about the current
train connections of a train station. In addition it registers departing trains
and informs all connected display media. The display media will delete this
train and replace it with a later departing train. Track boards show the next
train departing on this track. The internal station network will be relieved
by exchanging as little data as possible between the displays and the station
control system. The station control only transmits the track on which a train
has departed. The connected displays must then request additional information
from the station control system. Track displays only trigger reactions if a
train have left on their station track.
```

Abbildung 6.1 zeigt die dazugehörige Ausgabe als UML-Klassenmodell:

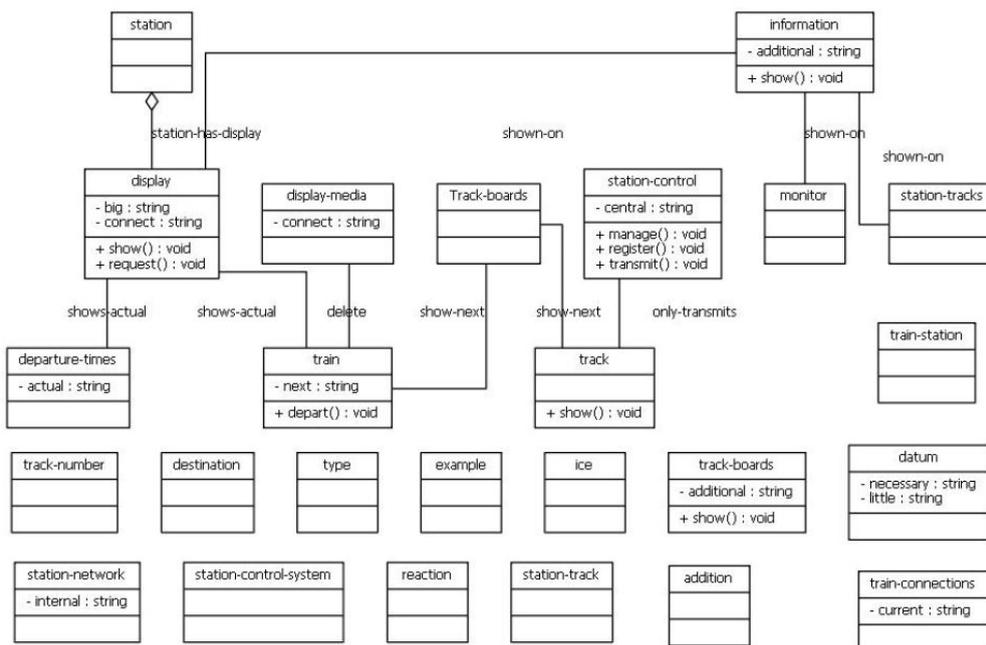


Abbildung 6.1: Erzeugung einer Struktur

6.3.1.2 Ergebnisse

Insgesamt enthielt jede der neun Aufgaben unterschiedliche Beschreibungen, einschließlich Klassen und Beziehungen zwischen Klassen, aber auch Attribute und Methoden. Die zusammengefassten Ergebnisse werden in Tabelle 6.8 dargestellt.

Element	Precision	Recall	F-Wert
Klassen	0.76	0.90	0.83
Attribute	0.48	0.58	0.52
Methoden	0.51	0.88	0.64
Beziehungen	0.60	0.44	0.51
Gesamt	0.59	0.70	0.63

Tabelle 6.8: Ergebnisse der Auswertung

6.3.2 Studie III.2: Klassenmodelle aus Tabellen

In dieser Studie wird die Datenstrukturanalyse untersucht.

6.3.2.1 Aufbau

Die Benutzerstudie aus [WFPT19] zeigt, inwieweit JustLingo NLUI Benutzeranfragen beantworten kann. Zunächst erhalten die 13 Teilnehmer die Aufgabe, dem System Fragen zu stellen. Als Datenbasis wird hier das Modell verwendet, das die Beziehung zwischen Ländern und Bundesländern sowie Österreich mit seinen Bundesländern als Daten enthält. In der Auswertung erhielten die Teilnehmer dann Aufgaben wie *Bestimmen Sie die durchschnittliche Einwohnerzahl aller österreichischen Bundesländer*, und mussten das System verwenden, um die Antwort zu erhalten. Die Tabelle 6.9 dient hierbei als Datenbasis.

Name	Capital	Population	Area
Burgenland	Eisenstadt	292.638	3.965
Carinthia	Klagenfurt	561.227	9.536
Lower Austria	Sankt-Poelten	1.670.299	19.179
...

Tabelle 6.9: Übersicht der Gemeinden in Österreich

6.3.2.2 Ergebnisse

Für die Auswertung der Ergebnisse werden in Zusammenarbeit mit [Fuc18] zwei Maßnahmen definiert. Zunächst wird die *Query Error Rate (QER)* für einen Abfragesatz Q und ein Satz von Antworten A wie folgt definiert:

$$QER(A, Q) = \frac{\#(\text{falscheAntwort}) + \#(\text{keineAntwort})}{|Q|}$$

Wir teilen die Summe der falschen Antworten und die Fragen ohne Antworten durch die Gesamtzahl der Fragen. Zweitens betrachten wir die Anzahl der *nicht verstandenen Eingaben*, kurz $\#(NUI)$, für das JustLingo NLUI. In der Studie konnten neun Teilnehmer alle sechs Aufgaben dieses ersten Teils lösen. Die restlichen vier Teilnehmer konnten eine der Aufgaben nicht lösen. Insgesamt wurde ein *QER* von 0,051 erzielt. Der Durchschnitt für $\#(NUI)$ betrug 2,62.

Im zweiten Teil der Studie wurde untersucht, wie sich diese Ergebnisse ändern, wenn mehrere Tabellen verknüpft werden. Außerdem sollten die Teilnehmer ihre eigene Aufgabe definieren, nachdem sie alle gestellten Aufgaben gelöst haben. Für diesen Teil der Bewertung wurden von Eurostat zwei Tabellen angenommen und die Beziehungen dazwischen festgelegt [Sta17, Sta18]. Insgesamt erhielten die Teilnehmer drei Aufgaben, für die sie eine Lösung formulieren mussten, und sie durften selbst eine weitere Aufgabe erstellen. Eine vordefinierte Aufgabe in diesem Zusammenhang war: *Bestimmen Sie die durchschnittliche Bevölkerungszahl in Belgien in den Jahren 2012 und 2013*. Hier betrug der *QER* 0,255 und der $\#(NUI)$ 2,31. Diese Werte basieren auf der Tatsache, dass alle Teilnehmer, wenn sie ihre eigene Aufgabe formuliert hatten, eine komplexere Anforderung stellten, die das System nicht verstand. Diese komplexen Fragen waren durch Vergleiche oder Datenreihenfolgen gekennzeichnet. Zum Beispiel: *Wessen Bevölkerung ist zwischen 2007 und 2010 am stärksten gewachsen?*

6.3.3 Fazit

Zusammenfassend erstellt JustLingo NLUI valide Klassenmodelle auf Basis von Benutzerbeschreibungen oder erfasst diese aus Tabellen. Die Daten müssen somit nicht zwingend beim Start der Benutzung vorhanden sein, sondern können während der Verwendung von JustLingo NLUI eingegeben bzw. bereitgestellt werden. Für eine Erstellung einer Klassenhierarchie muss ein Benutzer in der Beschreibung die Objekte, deren Eigenschaften und Funktionen mitteilen. Benutzer bei der Erstellung von UML-Modellen und Codefragmenten durch die natürlichsprachliche Schnittstelle JustLingo NLUI unterstützt werden können und trägt damit Erkenntnisse zur Forschungsfrage bei. Darüber hinaus zeigt die Anwenderstudie, dass einfache Anfragen zu nicht verbundenen Daten, zum Beispiel aus zwei Excel-Tabellen, beantwortet werden können. Es gibt eine Korrelation zwischen den mitgeteilten Beschreibungen und deren Verwendung für die natürlichsprachliche Benutzerprogrammierung.

6.4 Studie IV: Erweiterung der Funktionalität

Diese Studie besteht aus mehreren Benutzerstudien, die das *JustLingo NLUI* bei der Erfassung von Algorithmen aus natürlichsprachlichen Beschreibungen evaluieren.

6.4.1 Studie IV.1: Kontrollstrukturen

Im Bezug auf die natürlichsprachliche Benutzerprogrammierung haben wir uns als Erstes mit der Erfassung von Kontrollstrukturen beschäftigt [WKT17].

6.4.1.1 Aufbau

Für die Benutzerstudie baten wir zwölf Teilnehmer, bestimmte Aufgaben mithilfe von *JustLingo NLUI* zu lösen. Die Teilnehmer waren nicht englische Muttersprachler und die meisten haben unser System noch nie benutzt. Des Weiteren gaben die meisten an, dass sie Excel schon vorher kannten, jedoch nicht regelmäßig benutzten.

Im Detail wurden die Teilnehmer nach jeder Aufgabe gefragt,

- ob die Aufgabe gelöst ist,
- ob das System ihnen half, die Aufgabe zu lösen,
- ob die Ausgabe natürlich war,
- und ob das System die Eingabe verstanden hat.

Neben der Reaktionszeit von *JustLingo NLUI* wurden die Aufgaben darauf ausgelegt, die neuen Funktionen bei der Erfassung von Kontrollstrukturen von *JustLingo NLUI* zu testen. Folgende Aufgabenstellung wurden evaluiert:

Bereichsaufgaben

Tragen Sie in alle Zellen der Spalte A den Wert 10 ein.

Multiplizieren Sie alle Zellen zwischen B1 und B10 mit 3.

Ziehen Sie von allen Zellen der Spalte C, die größer als 25 sind, 10 ab.

Addieren Sie für jeden Wert in Spalte C, der größer als 15 ist 2 zu Zelle B1 hinzu.

Bedingungen

Falls B5 größer als 3 ist, tragen sie die Wurzel von A1 in B2 ein.

Tragen Sie 5 in B3 ein, falls A1 kleiner als A2 ist und sonst in B4.

6.4.1.2 Ergebnisse

Die Aufgabestellung haben die Teilnehmer in natürlicher Sprache auf Englisch der natürlichsprachlichen Schnittstelle *JustLingo NLUI* mitgeteilt. Die folgende Auflistung zeigt Beispiellösungen eines Teilnehmers zu der zuvor beschriebenen Aufgabestellung:

Insert 10 into all fields in column A.
 Multiply all cells from B1 to B10 by 3.
 Subtract 10 from all cells in column C that are greater than 25.
 For each value in column C that is greater than 15, add 2 to B1.
 If B1 is bigger than 3, insert the squareroot of A1 in B2.
 If A1 is smaller than A2, insert 5 in B3, otherwise insert 5 in B4.

Die Ergebnisse zeigen, dass die Benutzer mehr als 60% der Aufgaben zumindest teilweise lösen konnten. Fast 70% der Systemantworten wurden von den Teilnehmern als natürlich bewertet. Insgesamt wurde die Qualität des Systems mit 3,33 von 5 Sternen bewertet. Bis auf einen sagten alle Teilnehmer, dass sie die natürlichsprachliche Schnittstelle zur Lösung der Aufgabe verwendet hatten.

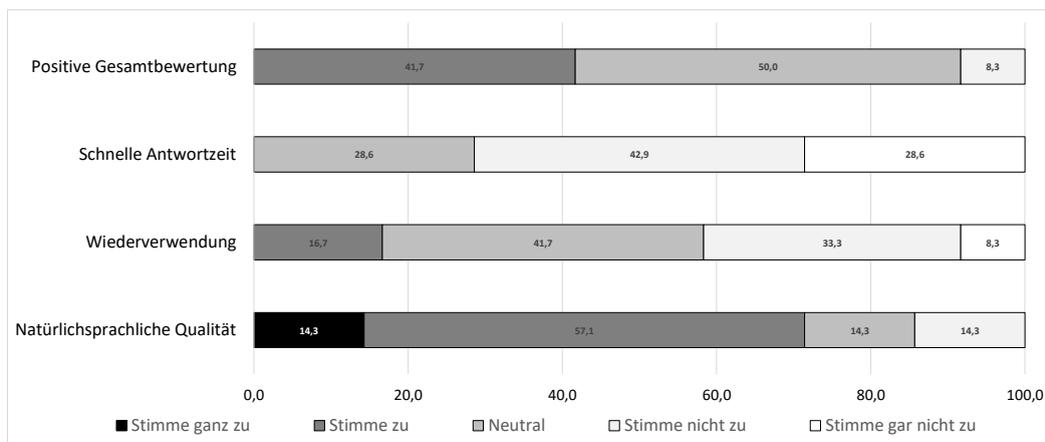


Abbildung 6.2: Auswertung der Evaluation von Kontrollstrukturen

Obwohl *JustLingo NLUI* bei der langsamsten Antwort mit 561 ms weit unter der gesetzten Höchstgrenze von 2 Sekunden antwortete, lag eine negative Bewertung der Geschwindigkeit bei 70% der Teilnehmer vor. Auch wenn *JustLingo NLUI* im Durchschnitt bei der Evaluation unter 280 ms eine Antwort ausgab, gab es Ausreißer, wenn die Interpretation der Eingabe mit Rückbezügen auf die vorherigen Eingaben kommuniziert wurde:

```

Benutzer:      for each cell in C

JustLingo NLUI: What would you like to do for those cells? (302ms)

Benutzer:      if cell is greater than 15

JustLingo NLUI: I see you used a condition.
                What do you want to if the condition is true? (477ms)

Benutzer:      add 2 to B2 and save in B2

JustLingo NLUI: I am sorry, but I am having issues to get you right.
                What are you refering to? (561ms)

Benutzer:      add 2 to B2

JustLingo NLUI: Ok i did that for you. (501ms)

```

In über 50% der Eingaben verstand *JustLingo NLUI* die Benutzereingabe nicht. Dieses Ergebnis zeigt zwar, dass *JustLingo NLUI* weit davon entfernt ist, perfekt zu sein, dennoch gibt es bei der Verwendung speziell für unerfahrene Benutzer einen Mehrwert. Die Hälfte der Eingaben wurde aufgrund unbekannter Synonyme nicht gelöst:

Folgende Auflistung zeigt ein Beispiel:

```

Nicht-erkannte Benutzereingabe:
Substract 10 from all cells in column C that are larger than 25.

Erkannte Benutzereingabe:
Substract 10 from all cells in column C that are greater than 25.

```

Im Nachgang wurde der Fehler durch die NLP-Fähigkeit der dynamischen Wortschatzerweiterung behoben.

6.4.2 Studie IV.2: Einfache Folgenalgorithmen

Die Erkennung von Kontrollstrukturen wurde als nächstes zu einer dynamischen Erfassung von Algorithmen erweitert und anschließend evaluiert [WET18].

6.4.2.1 Aufbau

Um einen Eindruck von der Zuverlässigkeit des Systems in Bezug auf unbekannte und uneingeschränkte Formulierungen zu erhalten, wurde eine Nutzerstudie mit insgesamt 11 Teilnehmern durchgeführt. Die Teilnehmer waren Studenten der Informatik und nicht-englische Muttersprachler. Mit sechs Personen bezeichnete sich mehr als die Hälfte der Teilnehmer als Anfänger in der Programmierung. Fünf Teilnehmer bezeichneten ihre Fähigkeiten als mindestens fortgeschritten. Sieben von ihnen haben das System noch nie benutzt. Vier der Teilnehmer haben bereits frühere Versionen unseres Prototyps in Bewertungen verwendet. Einen Vorteil bei der Evaluation war für diese nicht zu sehen, da sich die Implementierung des NLUI geändert hat.

Die Aufgaben der Studie umfassen sowohl das Erkennen einzelner Konzepte als auch die Verarbeitung ganzer Algorithmusbeschreibungen. Im ersten Teil wurden die Teilnehmer gebeten, atomare Beschreibungen der unterstützten Programmelemente einzureichen. Dazu gehören Definitionen, Zuweisungen, Bedingungen und Schleifen. Der zweite Teil enthielt zwei Aufgaben. Im ersten Fall sollten die Benutzer das größte Element in einer Menge von Zahlen finden und eine schrittweise Beschreibung aufschreiben. Die zweite Aufgabe beinhaltete einen vorgegebenen Pseudocode eines Sortieralgorithmus und beanspruchte dessen Realisierung mit Hilfe des NLUI in natürlicher Sprache. Interessante Fragestellung ist hierbei, welchen Einfluss die Antwort und die Vorschau des Algorithmus auf den Verlauf der Beschreibung und das Erreichen des Ziels haben. Für die Evaluation wurden die folgenden Aufgaben aufgelistet und den drei Komplexitätsklassen zugewiesen [Eur17]:

1. Komplexitätsklasse: Deklarationen, Aktionen & For-Schleifen
 - Zuweisen einer Zellreferenz zu einer Variable
 - Entfernen eines Wertes aus einer Menge
 - Speichern eines Referenzwertes in eine Zellreferenz
 - Iterieren über alle Elemente einer Menge
2. Komplexitätsklasse: Bedingungen & While-Schleifen
 - Den Wert an einer Position gleich einer Zahl prüfen
 - Den Wert einer Variablen in einer Menge prüfen
 - Etwas ausführen, bis eine bestimmte Menge leer ist
 - Den Wert einer Variablen kleiner als eine Zahl oder größer oder gleich einer anderen Zahl prüfen
3. Komplexitätsklasse: Schleifen & Bedingte Aktionen
 - Wenn der Wert einer Variablen kleiner als eine Zahl ist, diesen in eine Zelle, andernfalls in eine andere Zelle speichern
 - Alle Elemente unter einer Zahl aus einer Menge entfernen

6.4.2.2 Ergebnisse

Insgesamt wurde eine Erfolgsrate von 57% in der ersten, 64% in der zweiten und 59% in der dritten Komplexitätsklasse erreicht. Dies lag daran, dass Teilnehmer Anweisungen an das System gaben, die ein noch unbekanntes Wort enthielten. Dabei handelte es sich um ein Synonym zu einem bereits vorhandenen Wort.

Erfolgreich durch JustLingo NLUI erkannt:

```

create new function
i want to create a new function

initialize input with every value in column A
use as input the values of column A
please take the values from column A as input
use column A as input
set the input to column A
set the parameter input to column A
set input to A

iterate over the input
for each element in input
iterate over all elements from input
iterate over input
for all values in input do

check if the value of the element of indexA is 5
check if the element of the input at indexA is 5
check if the value of the input at indexA is 5

remove the value at indexA from the input
delete the element at index 3 from the input
remove element at 3
remove the element at 3 of the input
remove the element at index 3

write the value of input in B1 at the end of the function
at the end write input into B1
at the end save the value of the input in B1
at the end write values of input to B1

```

Für eine abschließende Bewertung der umgesetzten Fähigkeiten der Erweiterung, wurde eine Studie durchgeführt, bei der Teilnehmer zu verschiedenen komplexen Anweisungen und Abläufen natürlichsprachliche Beschreibungen abgegeben haben. Hierbei zeigte sich, dass das System bei 60% (mit der dynamischen Wortschatzerweiterung bei 73,6%) der Aufgabenstellungen erfolgreich war und den Teilnehmern u.a. möglich war, *JustLingo NLUI* bspw. einen Sortieralgorithmus zu erklären, welchen das System später auf beliebige Daten korrekt ausführt. Durch die dynamische Wortschatzerweiterung konnten die Lösungsraten in der ersten Komplexitätsklasse von 57% auf 80% sowie in der zweiten Komplexitätsklasse von 64% auf 82% verbessert werden [WST19].

Nicht erfolgreich durch *JustLingo NLUI* erkannt:

```

i want to make a function

input is in column A
let input be column A

iterate over every value in VARIABLE input
please loop over the input
iterate over the parameter input
take a look at each value of the input

go through all values in input

is the value at indexA 5
is the element number indexA in input equal to 5
is the input at position indexA 5

```

```

check wether the value of input on the index indexA is equal to 5
check if a value of indexA in array input equal 5
does element of input at indexA equal 5

remove the third element
remove element 3 from input (entfernt 3 vom input)
delete the third element of input
delete element of input at index 3
remove the third elment from the input values

put all values of input into B1
in the end of the function write input into B1
copy all elements of input to B1
go to end, write input in B1

set B1 to input at the end
at the end print the values of input in cell B1

```

6.4.3 Fazit

Diese Studie zeigt den Grad der Erkennung von formulierten Algorithmeingaben durch eine natürlichsprachliche Schnittstelle. Bis zu 82% der gestellten Formulierungen können durch das *JustLingo NLUI* erkannt und erfolgreich zu Code verarbeitet werden. Diese Studie zeigt damit, dass mit dem gewählten Ansatz bei *JustLingo NLUI* die gestellte Forschungsfrage erfüllt werden kann.

6.5 Akzeptanz von JustLingo NLUI

Dieser Abschnitt stellt eine Studie zur Akzeptanz von JustLingo NLUI dar.

6.5.1 Aufbau

Insgesamt nahmen 20 Teilnehmer teil: 14 Benutzer und sechs Entwickler. Unter den Benutzern waren neun Studenten, drei Auszubildende und zwei Berufstätige. Alle sechs Entwickler waren Studenten. Die Teilnehmer haben zunächst *JustLingo NLUI* benutzt und wurden im Anschluss zur Auswertung zufällig in zwei Gruppen, in Benutzer und in Entwickler mit jeweils spezifischen Fragestellungen, aufgeteilt.

Folgende Punkte wurden für die Benutzer evaluiert:

- (E1) Welche Erfahrung haben Sie im Umgang mit den (natürlichsprachlichen) Schnittstellen im Allgemeinen?
- (E2) Worin sehen Sie den Mehrwert, einen Benutzer in den Entwicklungsprozess traditioneller Software einzubeziehen?
- (E3) Würden Sie den Prozess in Ihrer Nutzung verwenden?
- (E4) Sehen Sie Vorteile in der Verwendung? Wenn ja, welche?
- (E5) Sehen Sie irgendwelche Nachteile? Wenn ja, bitte angeben.
- (E6) Halten Sie es für sinnvoll, Schnittstellen in natürlicher Sprache in herkömmliche Software zu integrieren?

Folgende Punkte wurden für die Entwickler evaluiert:

- (D1) Haben Sie zuvor Benutzer direkt in den Entwicklungsprozess einbezogen? Wenn ja, wie waren die Benutzer beteiligt?
- (D2) Würden Sie den Softwareentwicklungsprozess für Benutzer in natürlicher Sprache verwenden, um Ihre vorhandene Software zu erweitern?

- (D3) Nennen Sie die Vorteile des Software-Engineering-Prozesses für Benutzer in natürlicher Sprache.
- (D4) Nennen Sie die Nachteile des Programmierprozesses für Benutzer in natürlicher Sprache.
- (D5) Denken Sie, dass Schnittstellen in natürlicher Sprache zunehmend entwickelt werden?

6.5.2 Ergebnisse

Die Studie zeigt, dass 65% der Benutzer die Frage E3 als positiv oder neutral bewerteten. Von allen 14 Benutzern haben elf angegeben, dass die Frage E6 positiv beantwortet werden sollte. Zwei bewerten diese jedoch negativ und eine Antwort war neutral.

E2:
Benutzer haben oft einen anderen Blickwinkel. Die Einbindung eines Benutzers macht die Software unter Umständen benutzerfreundlicher und leichter verständlich.

Neue Software ist häufig kein positives Erlebnis für Benutzer, da diese sich in eine neue Technologie einarbeiten müssen. Nutzer direkt mit einzubinden könnte Akzeptanz für die Software steigern und die Software verbessern, da die Entwicklung näher am Benutzer ist.

Abbildung 6.3 zeigt die Ergebnisse für sechs Entwicklerteilnehmer zu Frage D1, D2 und D5. Dabei bezieht sich 1 auf den negativsten und 5 auf den positivsten Wert.

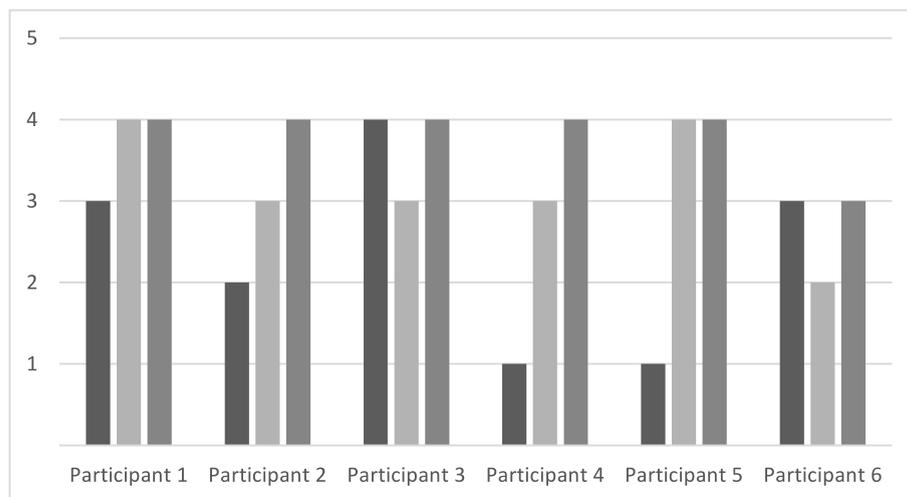


Abbildung 6.3: Aussagen der Entwickler auf D1 (links), D2 (mitte), D5 (rechts)

D3:
Kein etwaiges Kommunikations-Bottleneck zwischen Endnutzer und Entwicklern. Die Initialhürde ist deutlich niedriger.

Endnutzer fühlen sich stärker eingebunden.

D4:
Eventuell viel Code-review und viele eingereichte duplizierte Codestellen bei mehreren Endnutzern.

6.5.3 Fazit

Die Endstudie untersucht die Akzeptanz der Benutzer und Entwickler. Mittels JustLingo NLUI können die Teilnehmer in natürlicher Sprache kommunizieren und die gewünschte Funktionalität mitteilen. Dabei werden NLUI-Artefakte erzeugt, die mit anderen JustLingo NLUI Benutzern geteilt werden können. Durch den Austausch der lokal erzeugten

NLUI-Artefakte können die Benutzer untereinander die neuen Funktionalitäten zum vorhandenen System teilen. Dadurch sinken die Anfragen der gewünschten Änderungen an den Softwareentwickler.

6.6 Zusammenfassung

Das Kapitel der Evaluation stellte die verschiedenen Benutzerstudien der einzelnen Komponenten von *JustLingo NLUI* vor. Dabei ist zu beachten, dass die Anfangsimplementierung wenig robust war und viele Eingaben nicht verstanden hat. Mit der Zeit wurden mehr NLP-Fähigkeiten implementiert, die den Kontextwechsel sowie die dynamische Wortschatzweiterung ermöglichten. Dadurch stieg die Erfolgsrate der korrekten Erkennung von Benutzereingaben. Nicht zuletzt die NLUI-Fähigkeiten ermöglichten dem *JustLingo NLUI* eine Gesamtfunktionalität, um die Zielsetzung dieser Arbeit zu erfüllen.

7. Diskussion

Kapitel 7.1 fasst die Erkenntnisse der Forschungsarbeit zusammen und führt eine Diskussion, inwieweit das ursprüngliche Vorhaben erreicht werden konnte. Insbesondere beurteilt das Kapitel die in der Evaluation erreichten Ergebnisse und setzt diese zum Schluss in einen Bezug zu den verfassten Forschungsfragen aus dem Kapitel 3.5. Des Weiteren werden im Kapitel 7.2 die Empfehlungen bei der Implementierung zukünftiger NLUI zusammengefasst. Eine erfolgreiche Erprobung der aufgestellten Empfehlungen durch die Implementierung eines ChatBots für Azure wird im Kapitel 7.3 vorgestellt. Abschließend wird in Kapitel 7.4 die Diskussion bei der Bereitstellung natürlichsprachlicher Bibliotheken für andere Benutzer angeschaut.

7.1 Erfüllung der Forschungsfragen

Forschungsfrage I: Wie kann das NLUI eine nicht-perfekte oder nicht vollständige Eingabe eines Benutzers erkennen?

JustLingo NLUI Lösung: **Aktivontologie & Aktives Dialogmanagement**

In der Evaluation konnte gezeigt werden, dass die natürlichsprachliche Schnittstelle *JustLingo NLUI* in der Lage ist durch den Einsatz einer Aktivontologie die fehlende Information in einer Benutzeranfrage zu erkennen und gezielt Rückfragen zu stellen.

Durch das Aktive Dialogmanagement lässt sich ein flexibler Dialogfluss modellieren. Des Weiteren spielt die Reihenfolge der mitgeteilten Information für die natürlichsprachliche Schnittstelle keine Rolle, um die Anforderung zu erfüllen, einen Benutzer in der Formulierung nicht einzuschränken.

Durch die geteilte Initiative hat *JustLingo NLUI* die Fähigkeit, einen zielgerichteten Dialog mit einem Benutzer aufzubauen. Nach dem Signal der Aktivontologie, welche Teile zur erfolgreichen Ausführung der Aktion notwendig sind, kann die Dialogkomponente die Information beim Benutzer erfragen.

Forschungsfrage II: Kann das NLUI auf spontanen Kontextwechsel durch einen Benutzer adäquat reagieren?

JustLingo NLUI Lösung: **Klassifikation & horizontale Dialogführung**

Die Evaluation der Klassifikation hat gezeigt, dass der Ansatz mit balancierten sowie unbalancierten kleinen Datenmengen gut funktioniert. Erfolgt ein Kontextwechsel durch eine Benutzereingabe, welche die Bearbeitung durch eine andere NLUI-Fähigkeit verlangt, wird die vertikale Ausführung des Dialogsystems angehalten und horizontal der gewünschte NLUI-Fähigkeit gesucht. Bei einer horizontalen Dialogführung sind alle Dialoge parallel aktiv und können gleichzeitig angesteuert werden. Dieses Konzept ermöglicht *JustLingo NLUI* einen reibungslosen Kontextwechsel, den die Benutzer öfters in den Dialogen ausführen. In unseren Nutzerstudien haben wir festgestellt, dass die Benutzer durch die Eingaben öfters einen Kontextwechsel erzwingen, weil sie an der vorherigen Aufgabe nicht mehr arbeiten und schon zur nächsten übergehen.

Forschungsfrage III: Wie erfasst das NLUI die Beschreibungen und die Daten, um Benutzeranfragen zu ermöglichen?

JustLingo NLUI Lösung: **Datenstrukturanalyse**

Die Evaluation zeigte, dass die Strukturuntersuchung der Beschreibungen sowie der Tabellendaten der natürlichsprachlichen Schnittstelle *JustLingo NLUI* durchgeführt werden kann. Die Modellierung der Klassen, Attribute sowie Methoden können zum späteren Zeitpunkt wiederverwendet werden. Eine Aktivontologie kann nicht auf alles mögliche vorbereitet werden. Mit Hilfe der Datenstrukturanalyse werden vorher unbekannte Daten erfasst und modelliert.

Forschungsfrage IV: Können mittels natürlichsprachlicher Beschreibung einfache Algorithmen durch das NLUI erfasst werden?

JustLingo NLUI Lösung: **Algorithmenerfassung**

Im Allgemeinen unterstützen die Programmiersprachen atomare Anweisungen sowie Kontrollstrukturen, wie Schleifen bei sich wiederholenden Aufgaben sowie Bedingungen bei vorhandenen Abzweigungen [CP11]. Die Evaluation zeigt, dass die Erfassung von atomaren Bestandteilen eines Algorithmus mit *JustLingo NLUI* möglich ist. Dabei werden die einzelnen Bestandteile des Algorithmus aus der natürlichen Sprache extrahiert und keine Fähigkeiten zu programmieren bei Benutzern vorausgesetzt. Die erfassten Folgealgorithmen können abgespeichert und zum späteren Zeitpunkt wiederverwendet werden.

7.2 Empfehlungen für die Entwicklung von NLUI

Insgesamt erfüllt die vorliegende Forschungsarbeit die gesetzte Zielsetzung mit den Erkenntnissen zum Dialogsystem, der Algorithmenerkennung sowie bei Text- und Datenstrukturanalyse. Es lassen sich nach diesen Erkenntnissen diverse Anforderungen für eine neue Implementierung einer natürlchsprachlichen Schnittstelle formulieren.

Abbildung 7.1 zeigt die einzelnen Entwicklungsschritte vor, während und nach der Entwicklung von NLUI auf:



Abbildung 7.1: Zusammenfassung der Empfehlungen

Empfehlungen vor der Entwicklung:

- 1. Definition der NLUI-Funktion in einem messbaren Anwendungsfall**
 Es ist wichtig, die Komplexität bei dem NLUI während der Anforderungsermittlung nicht zu hoch zu bestimmen. Die Komplexität kommt mit dem Fortschritt der Entwicklung.
- 2. Definition einer kleinen Aktivontologie/Datenset**
 In der Anforderungsermittlung wird die Grundfunktionalität des NLUI festgelegt. Es erfolgt parallel dazu die Definition einer kleinen, qualitativ guten Datenmengen. Die Datenset umfasst zunächst die Sprachdomäne des ersten NLUI-Fähigkeit.
- 3. Definition der NLUI-Fähigkeit**
 In Abhängigkeit des Anwendungsfalls und der Gesamtfunktion des NLUI erfolgt die Definition der ersten NLUI-Fähigkeit.
- 4. Definition der NLP-Fähigkeiten:**
 In Abhängigkeit der NLUI-Fähigkeit muss bestimmt werden welche NLP-Fähigkeiten implementiert werden müssen. Alternativ kann für NLP-Aufgaben auf externe, bereits implementierten Werkzeuge zurückgegriffen werden.
- 5. Definition eines Frontend/GUI:**
 Verwendung von einem einfachen und nach Möglichkeit bekannter GUI damit die Benutzer sich die Verwendung nicht erst aneignen müssen.

Empfehlungen während der Entwicklung:

1. **Aktives Dialogmanagement:**

Basierend auf dem Datenset erfolgt die Implementierung der aktiven Dialoge. Dabei sollte die geteilte Initiative im Antwortverhalten der natürlichsprachlichen Schnittstelle forciert werden. Eine Fehlerbehandlung im Dialog bzw. das Korrigieren der letzten Eingabe soll ermöglicht werden.

2. **Implementierung der Fähigkeiten:**

Die jeweiligen NLP- & NLUI-Fähigkeiten werden voneinander getrennt implementiert. Dabei unterstützen die NLP-Fähigkeiten um die Anfrage des Benutzers zu verarbeiten und können durch externe Werkzeuge oder Eigenimplementierung erfolgen. Bei den NLUI-Fähigkeiten muss auf eine zielorientierte Dialogführung bis zur Ausgabe des gewünschten Ergebnisses geachtet werden.

3. **Erweiterung des Datensets durch Entwickler:**

Während der Implementierung der Aktiven Dialoge wird durch den Entwickler iterativ das Datenset um neue Formulierungen erweitert.

4. **Fähigkeiten ins Frontend integrieren:**

Damit eine Evaluierung mit Pilotbenutzern erfolgen kann, muss eine Integration des Dialogsystems ins Frontend erfolgen. Während der Verwendung wird eine Analyse des Frontends bei Benutzeranfragen für spätere Verbesserungen durchgeführt.

Empfehlungen nach der Entwicklung:

1. **Klassifikation der Fähigkeiten:**

Sind mehrere Fähigkeiten in der Verwendung, muss zwischen diesen pro Benutzereingabe klassifiziert werden, um die Absicht des Benutzers zu erfahren. Dafür können externe Werkzeuge, die als Bündel eine neue Eingabe zu den vorhandenen Fähigkeiten untersuchen, eingesetzt werden.

2. **Benutzer erweitern das Datenset:**

Das NLUI muss sich an das Sprachmuster des Benutzers anpassen. Somit sollte es während der Verwendung eine Möglichkeit geben, für Benutzer im Dialog eine Erweiterung der Sprachdomäne durchzuführen.

3. **Erweiterung auf mehrere Kanäle:**

Die Benutzer sind heutzutage gewohnt, den für Sie geeigneten Kanal für die Kommunikation zu verwenden. Das Frontend des NLUI kann somit über eine Webseite, einen Kanal in Microsoft Teams oder einer anderen Anwendung bereitgestellt werden.

4. **Durchgehende Evaluierung:**

Um die Funktionalität zu messen, sollten Benutzer und Keyplayer frühzeitig für Evaluation einbezogen. Für einen Erfolg der Implementierung wird es in einer durchgehende Evaluierung durchgeführt.

7.3 Implementierung eines NLUI nach verfassten Empfehlungen

Die Erkenntnisse der Dissertation wurden bei der Implementierung eines ChatBots für die Industrie erprobt. Aus diesem Grund haben wir ein open-source Framework vorgestellt, das aufbauend auf dem Microsoft LUIS die Dialoge und die Benutzersitzungen verwaltet [WFST19]. Dabei legen wir Richtlinien für die Modellierung einer dialogbasierten Konversation mit einem Chatbot fest. Dadurch ermöglichen wir einem Entwickler in einem Visual Studio-Projekt durch das Framework, ein eigenes Dialogverhalten zu erstellen.

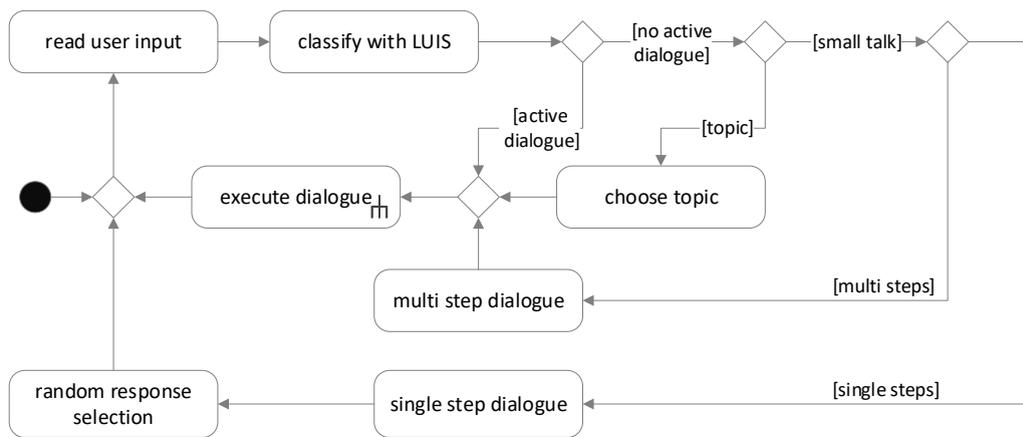


Abbildung 7.2: Aktives Dialogmanagement aus [WFST19]

Abbildung 7.2 zeigt das Aktive Dialogmanagement nach Vorbild von *JustLingo NLUI*. Sobald eine neue Eingabe durch einen Benutzer vorliegt, muss es zuerst vom LUIS interpretiert bzw. klassifiziert werden. Anschließend überprüft die Dialogverwaltung, ob aktive Dialoge vorhanden sind, die auf eine Reaktion des Benutzers warten. Ist ein aktiver Dialog gefunden, wird es ausgeführt. Wenn kein aktiver Dialog vorhanden ist, initiiert die Dialogverwaltung einen neuen Dialog. Zu diesem Zweck klassifizieren wir zunächst die Eingabe mit LUIS. Es gibt zwei verschiedene Arten von Dialogen. Einerseits die Smalltalk-Dialoge und andererseits die Themendialoge. Themendialoge sind die Feature-Dialoge des Bots. Sie sind Absichten zugeordnet und repräsentieren die Funktionalität des Bots.

7.4 Bereitstellung der NLUI-Bibliotheken

Die Erfassung der NLUI-Artefakte erlaubt die Identifizierung der Beziehungen zwischen den Objekten und ermöglicht eine erweiterte Interpretation der Benutzereingabe. *JustLingo NLUI* hilft dabei, durch die Benutzer Quellcode zu generieren, ohne dabei Programmierkenntnisse bei einem Benutzer zu verlangen oder als Voraussetzung zu setzen. Demnach wächst die Anzahl der erstellten Bibliotheken und andere Benutzer können einfach die gewünschte Funktionalität bei einem zentralen NLUI-Repository laden. Da viele der Benutzereingaben sich wiederholen, kann ein intelligentes Sprachsystem die Routineaufgaben übernehmen, so dass sich die professionellen Entwickler auf andere, komplexere Aufgaben konzentrieren können. Die durch *JustLingo NLUI* erfassten Daten können verwendet werden, um gezielt neue Lösungsansätze zu automatisieren und die natürlichsprachliche Schnittstelle intelligenter zu machen. Komplexere Sprachschnittstellen können einen menschenähnlichen Gesprächsverlauf mit dem Benutzer anbieten. Dafür müsste noch weitere Forschungsarbeit durchgeführt werden.

7.4.1 Lokale Benutzerprogrammierung

Nachdem die Klassen mit den beschreibenden Attribute sowie den Algorithmen als Funktionen erkannt sind, beschreiben die Beziehungen eine Interaktion zwischen den Klassen.

Ein Benutzer des Systems kann durch die natürliche Sprache nicht nur neue Funktionen beschreiben, sondern auch die zugehörigen Objekte sowie Beziehungen zwischen den Objekten zur Verfügung stellen. Die NLUI-Artefakte stehen jedoch nur dem Benutzer lokal zur Verfügung.

Um die Zielsetzung der Forschungsarbeit professionelle Entwickler durch die Benutzer zu unterstützen, müssen lokal erzeugte Bibliotheken aus natürlicher Sprache zur Verfügung gestellt werden.

Die Struktur der Objektorientierung hat hierbei einige Vorteile: (i) die Erfassung der Algorithmen für die Manipulation der Attribute, welche die Zustände der Objekte beschreiben (ii) das erneute Aufrufen der gespeicherten Methoden und (iii) die Zuweisung der Methoden zu den jeweiligen Klassen bzw. deren Objekten. Im Allgemeinen können Klassen Entitäten aus der realen Welt darstellen. Daneben geben Attribute die unterschiedliche Eigenschaften einer Entität an und Methoden sind Algorithmen, mit denen die Attribute manipuliert werden. Des Weiteren stellt Abbildung 7.3 die Kommunikation zwischen *JustLingo NLUI* mit einem Benutzer sowie einem Entwickler dar.

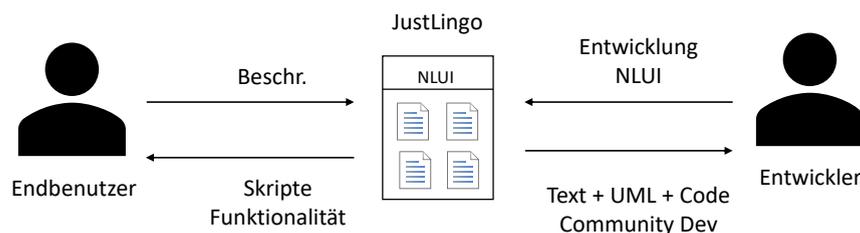


Abbildung 7.3: Lokale Erweiterung der Funktionalität durch das *JustLingo NLUI*

7.4.2 Globale Bereitstellung

Abbildung 7.4 stellt einen neuen Arbeitsablauf in *JustLingo NLUI* vor, bei dem die lokalen NLUI-Artefakte als Daten mit anderen Benutzern geteilt werden können [WFT20]. Als Ergebnis liegt dem Entwickler die Beschreibung des Benutzers sowie der Code, der durch das System aus dieser Beschreibung extrahiert wurde, in der objektorientierter Form dar. Für einen Entwickler des Systems ist es einfach den Code zu überprüfen und es in das globale Repository bereitzustellen.

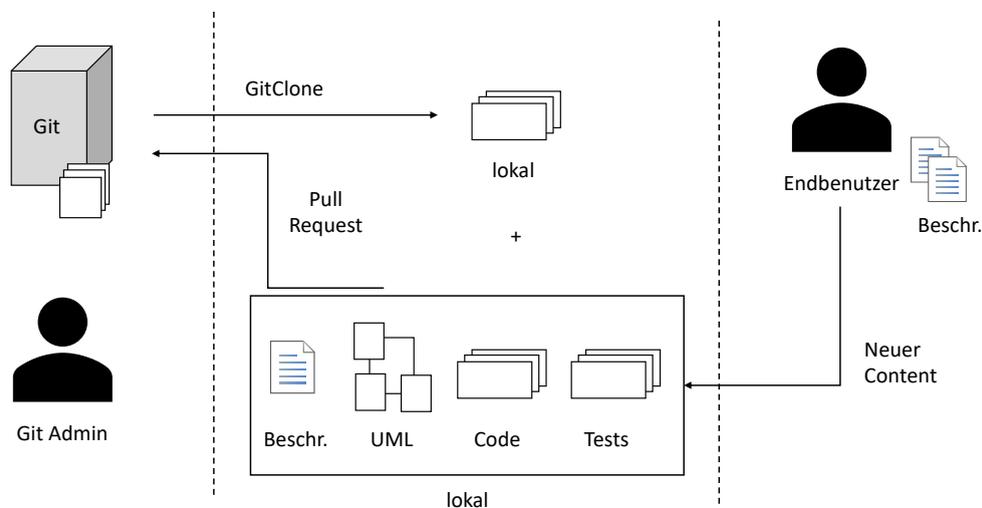


Abbildung 7.4: Erweiterung globaler Systeme mit neuem Inhalt

Neben den Vorteilen für die Benutzer in der lokalen Verwendung, bietet das *JustLingo NLUI* eine Möglichkeit die Benutzer in den Entwicklungsprozess einzubeziehen sowie eine Erweiterung des Endsystems durch die Entwickler. Durch *JustLingo NLUI* ist es möglich, dass die Benutzer die für sich gewünschte Funktionalität durch natürliche Sprache in objektorientierter Form bereitstellen, so dass diese von einem Entwickler nach der Prüfung übernommen werden können. *JustLingo NLUI* könnte somit durch einen Benutzer und einen Softwareentwickler verwendet werden. Der Benutzer erhält mittels *JustLingo NLUI* eine Hilfestellung und für eine Anpassung einer anderen NLUI mit anderen Benutzern die NLUI Artefakte teilen. Dadurch können die Benutzer selbst Funktionalität in einer natürlichsprachlichen Schnittstelle bereitstellen, so dass die Last für diese Implementierung bei einem Softwareentwickler sinkt.

Ein Beispiel für ein NLUI Artefakt:

```
{
  "interactions":
  [
    {
      "Type": "User",
      "Text": "there is a robot"
    },
    ...
  ],
  "algo":
  [
    {
      "Name": "Function Min",
      "Code": "void min()\n{\nInt32 min = INPUT[0]; ... RESULT = min}",
      "Module": "Algorithms"
    }
  ],
  "algoData": "BASE64-STRING",
  "data":
  {
    "Classes": ["robot", "head"],
    "Class2Base": {"robot": null, "head": null},
    "Class2Attr": {"robot": [], "head": ["eye"]},
    "Class2Method": {"robot": [], "head": []},
    "ClassRelations":
    [
      {
        "Name": "robot-has-head",
        "A": "robot",
        "B": "head",
        "RelationType": "Aggregation"
      }
    ],
    "Objects": [],
    "Object2Class": {},
    "Object2Attribute": {},
    "ObjectRelations": [],
    "Module": "Data"
  }
}
```


8. Zusammenfassung und Ausblick

Dieses Kapitel fasst die Forschung an der natürlichsprachlichen Schnittstelle *JustLingo NLUI* zusammen und wagt einen Ausblick auf die zukünftige Forschung.

8.1 Zusammenfassung

Die Arbeit legt Methoden in der Implementierung einer natürlichsprachlichen Benutzerschnittstelle dar und konzentriert sich auf einer Kommunikation in geschriebener Sprache. Im Vordergrund steht dabei die Fähigkeit der Schnittstelle zu nicht-ingeschränkter Natürlichsprachlichkeit. Aus diesem Grund schränkt *JustLingo NLUI*, auf der einen Seite, die natürliche Sprache eines Benutzers nicht ein und versucht dadurch, die Ausdrucksweise jeder Person zu unterstützen. Auf der anderen Seite wird durch die NLUI-Fähigkeit Datenstrukturanalyse die natürlichsprachliche Schnittstelle auf mögliche Benutzeraktionen auf dem Zielsystem vorbereitet. Als Zielsystem wird im Rahmen der Forschungsarbeit die Tabellenkalkulationen Microsoft Excel mit den enthaltenen Daten in einer gewöhnlichen Arbeitsmappe verstanden. Dadurch bildet *JustLingo NLUI* eine Brücke zwischen dem Benutzer und dem Zielsystem.

In der Pilotstudie haben wir herausgefunden, dass Benutzer i.d.R. nicht in der Lage sind, die komplette Information zu einer Aufgabe in einer einzelnen Anweisung mitzuteilen. Vielmehr versuchen sie, eine komplexe Aufgabe schrittweise zu lösen. Demnach wird eine natürlichsprachliche Schnittstelle *JustLingo NLUI* benötigt, die einen Benutzer anleitet und den Aufbau einer komplexen Anfrage Schritt-für-Schritt ermöglicht. Eine Benutzerinteraktion wird durch ein Aktives Dialogmanagement, mit u.a. geteilter Initiative, einer vertikalen und horizontalen Dialogführung sowie einer Fehlerbehandlung ermöglicht. Insgesamt interpretiert das *JustLingo NLUI* die natürlichsprachlichen Anfragen des Benutzers, fragt das korrekte Anliegen des Benutzers bei Mehrdeutigkeiten oder Unklarheiten nach und führt zum Schluss die gewünschte Aktion aus. Durch die Anleitung des Benutzers übernimmt *JustLingo NLUI* eine aktivere Rolle und reagiert nicht nur passiv auf Benutzereingaben, wie ein gewöhnliches Frage-Antwort-System.

Um die Zielsetzung der Forschungsarbeit zu erreichen, sind neben einem Dialogsystem die NLP-Fähigkeiten und NLUI-Fähigkeiten implementiert. Die Fähigkeiten hängen vom Anwendungsfall des NLUI ab und stellen Hilfswerkzeuge zur Verfügung. Somit ist es möglich, dass *JustLingo NLUI* eine textuelle Beschreibung eines Benutzers nach den Methoden der objektorientierten Programmierung untersucht. Dabei kann die Beschreibung entweder

Satz-für-Satz oder als Block eingegeben werden. Als Erstes sucht *JustLingo NLUI* nach den vorhandenen Objekten und erstellt aus diesen eine grundlegende Klassenstruktur. Die Zustände eines Objekts werden als Attribute in der jeweiligen Objektklasse abgelegt sowie die Funktionalität aus den Beschreibung als Algorithmus erfasst und zur weiteren Verwendung als Methode gespeichert. Das Ergebnis ist ein UML-Diagramm, das diese Struktur enthält und für *JustLingo NLUI* einen Zwischenschritt zur Erstellung des C#-Quellcode darstellt.

8.2 Ausblick und zukünftige Arbeiten

In den letzten Jahren wurden bedeutende Fortschritte in den Verarbeitung der natürlichen Sprache gemacht. Als Ursache ist die gestiegene Rechenkapazität zu sehen, mit der die NLP-Systeme in der Lage sind, die natürliche Sprache effizient zu verarbeiten, ein Modell zu trainieren und robust auf die Anfragen an großen Datenmengen zu reagieren.

Am Beispiel einer Tabellenkalkulation wollten wir lernen, wie allgemeine Schnittstellen zu implementieren sind. Dieses Kapitel diskutiert einen nächsten Schritt, wie die Ergebnisse der Implementierung einer natürlichsprachlichen Benutzerschnittstelle in der Zukunft genutzt werden können. Die Verbesserungen für die natürlichsprachliche Schnittstelle sprengen den Umfang der Dissertation und werden daher als zukünftige Arbeiten diskutiert.

8.2.1 Lernende Systeme

Im Rahmen der Forschungsarbeit haben wir eine interaktive Schnittstelle implementiert, die jedoch nicht als KI-System bezeichnet werden kann. Als Künstliche Intelligenz (KI) wird die Wissenschaft der intelligenten Systeme verstanden, die ihre Umgebung wahrnehmen, Aktionen ausführen und aus diesen Aktionen lernen können. Insbesondere können die Systeme aus dem bereits bekannten zu einem unbekanntem Problem eine Schlussfolgerung ziehen oder die Lösung zum unbekanntem Problem abschätzen. Interessant ist die Erforschung eines *hybriden Sprachsystems*, das regelbasiert mit wenig Daten im Wortschatz arbeiten kann und das Maschinelle Lernen an geeigneten Stellen eingebaut hat.

8.2.2 Bedeutungsschlaue KI-Systeme

Die zukünftige Entwicklung der Sprachsysteme, wie OpenAI's ChatGPT, lässt die Möglichkeiten dieser Systeme in Zukunft nur erahnen. Dennoch basiert das aktuelle Sprachsystem, wie andere aktuelle Implementierungen, auf dem Training mit großen Datenmengen zum Large Language Model (kurz: LLM).

Die KI-Systeme sind datenschlau, der Mensch ist bedeutungsschlau.

Demnach fehlen den aktuellen Systemen sowie *JustLingo NLUI* noch die intuitiven Fähigkeiten eines Menschen. Wie muss das Dialogsystem reagieren, wenn bisher gespeicherte Daten nicht zur Lösung führen, sondern eine komplexe Bedeutungsherleitung aufgrund von gesammelten Erfahrungen bereitgestellt werden sollte. Die Erfahrungswerte sammeln die Menschen über die Lebensjahre an, ein KI-Sprachsystem bekommt diese Erfahrungswerte entweder gar nicht oder in kürzester Zeit durch die bereitgestellten Datenmengen unzureichend beigebracht. Dadurch ist es einem Menschen möglich, eine neue Erfahrung auf das bisher Erlebte zurückzuführen, die Bedeutung herzuleiten und eine adäquate Reaktion durchzuführen.

8.2.3 Komplexere Algorithmen

Bei der Algorithmenerkennung konzentriert sich *JustLingo NLUI*, welches als Ziel die natürlichsprachliche Programmierung für Benutzer erforscht, zunächst auf einfachen Folgenalgorithmen. Um die Idee einer natürlichsprachlichen Benutzerprogrammierung zu untersuchen, ist dies ausreichend gewesen, verlangt jedoch weitere Forschungsarbeit, wenn komplexere Algorithmenfassung ermöglicht wird.

Hierbei muss untersucht werden, wie z.B. eine Rekursion in natürlicher Sprache beschrieben wird. Die meisten Benutzer müssten damit ihre Schwierigkeiten haben. Insbesondere ist es dann zu erforschen, ob die Limitierung bei der Beschreibung komplexer Ausdrücke in der natürlichen Sprache selbst, in der Ausdrucksweise eines Benutzers oder bei der natürlichsprachlichen Schnittstelle festzustellen ist. Des Weiteren ist es interessant zu erforschen, wie ein Graph durch einen Benutzer in natürlicher Sprache beschrieben werden kann und welche Komponenten noch bei der Algorithmenfassung implementiert werden müssen, um zum Beispiel ein A*-Algorithmus zu ermöglichen.

9. Anhang

9.1 Pilotstudie

In der Pilotstudie werden XML-basierte Annotationen verwendet: *WordDatabase.xml*, *CommandDatabase.xml*, *SyntaxRuleDatabase.xml* [Voi14].

Abbildung 9.1 zeigt ein Beispiel zum Kommando *sum*. Hierbei werden die Parameter, die zugehörige Excel-Formel sowie eine Ausgabe definiert:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <CommandDatabase>
3      <Commands>
4          <Command Key="sum">
5              <Type>Operation</Type>
6              <Parameters>
7                  <Min>1</Min>
8                  <Max>X</Max>
9                  <Parameter Use="required" Type="Any" />
10                 <Parameter Use="following" Type="Any" />
11             </Parameters>
12             <Formula>
13                 <Sign Position="between">+</Sign>
14                 <en-US>SUM</en-US>
15                 <de-DE>SUMME</de-DE>
16             </Formula>
17             <Output>
18                 <en-US>the sum of %1</en-US>
19             </Output>
20         </Command>
21         ...
22     </Commands>
23 </CommandDatabase>
```

Abbildung 9.1: Auszug aus der *CommandDatabase.xml*

Das Dialogsystem stützt sich auf ein Grundvokabular, das speziell für die Sprachdomäne der mathematischen Berechnungen entwickelt ist.

Auszug aus der *WordDatabase.xml*:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <WordDatabase>
3      <Words>
4          <Word Name="hello">
5              <Substitution>hello</Substitution>
6              <Meaning>Greetings</Meaning>
7              <Type>noun</Type>
8          </Word>
9          <Word Name="hi">
10             <Substitution>hello</Substitution>
11             <Meaning>Greetings</Meaning>
12             <Type>noun</Type>
13         </Word>
14         <Word Name="yes">
15             <Substitution>yes</Substitution>
16             <Meaning>Answer</Meaning>
17             <Type>noun</Type>
18         </Word>
19         <Word Name="add">
20             <Substitution>add</Substitution>
21             <Meaning>Operator</Meaning>
22             <Type>verb</Type>
23         </Word>
24         <Word Name="set">
25             <Substitution>set</Substitution>
26             <Meaning>Action</Meaning>
27             <Type>verb</Type>
28         </Word>
29         <Word Name="calculate">
30             <Substitution>calculate</Substitution>
31             <Meaning>Calculation</Meaning>
32             <Type>verb</Type>
33         </Word>
34         <Word Name="to">
35             <Substitution>to</Substitution>
36             <Meaning>Direction</Meaning>
37             <Type>binding</Type>
38         </Word>
39         ...
40     </Words>
41 </WordDatabase>

```

Abbildung 9.2: Auszug aus der *WordDatabase.xml*

Aufgaben in der Pilotstudie [Voi14]:

- Funktionen in Excel-Formeln:
In der Spalte „Mittelwert“ soll für jeden Schüler der Durchschnitt aller seiner Leistungen während des Schuljahrs berechnet werden. Es soll nur der Mittelwert ohne Rücksicht auf die Gewichtung der schriftlichen und mündlichen Leistungen berechnet werden. Die entsprechende Berechnung soll speziell für den Schüler Marcel durchgeführt und in Zelle F4 geschrieben werden.

- **Zusammensetzen von Funktionen:**
Der Mittelwert in Zelle F4 aus der vorherigen Fragestellung berücksichtigt nicht die unterschiedliche Gewichtung aus schriftlichen (60%) und mündlichen (40%) Leistungen, dies soll nun jedoch geschehen. Es soll auch hier die entsprechende Berechnung speziell für den Schüler Marcel durchgeführt und in Zelle F4 geschrieben werden.
- **Übertragen einer Excel-Formel in mehrere Zellen:**
Unabhängig davon, ob Sie für die Lösung der vorherigen Fragestellung Unterstützung in Anspruch hätten nehmen müssen, wird nun in Zelle F4 der korrekte gewichtete Mittelwert des Schülers Marcel berechnet. Dieselbe Berechnung soll nun für alle übrigen Schüler in den Zellen F5 bis F15 durchgeführt werden, ohne jedoch für jeden Schüler explizit eine Formel schreiben zu müssen.
- **Weitere Funktionen:**
In Spalte F befindet sich nun für jeden Schüler der Durchschnitt seiner Leistungen. Die jeweilige Endnote ergibt sich durch auf- bzw. abrunden des Durchschnitts. Sie sollen speziell die Endnote des Schülers Marcel bestimmen und in Zelle G4 schreiben. Beachten Sie, dass sich die Endnote automatisch ändern soll, wenn nachträglich einzelne Noten verändert werden.
- **Bedingte Funktion:**
Geben Sie an, wie viele Schüler bestanden haben, d.h. eine Endnote zwischen 1 und 4 erreicht haben. Beachten Sie, dass sich diese Anzahl automatisch ändern soll, falls später eine Endnote verändert wird.
- **Einfügen einer zusätzlichen Zeile:**
Ein neuer Kunde ist gewonnen und soll in die Tabelle in alphabetischer Reihenfolge eingefügt werden. Ihre Aufgabe ist es, eine zusätzliche Zeile zwischen Oracle und SAP einzufügen.
- **Erzeugen einer Tabelle innerhalb eines Dokuments:**
Die unformatierten Daten zu den Erträgen aus Kundenverträgen sollen in Form einer Tabelle dargestellt werden, um durch Rahmen und Farbgebung die Übersichtlichkeit zu erhöhen. Sie möchten die Tabelle außerdem „Kundenverträge“ nennen und dafür sorgen, dass die bestehenden Spalten-Namen bestehen bleiben.
- **Sortieren nach einer Spalte:**
Die Erträge aus Kundenverträgen werden nun übersichtlich in Form einer Tabelle dargestellt. Sie möchten nun die Kunden nach den Erträgen im Jahr 2013 in Spalte E absteigend sortieren, sodass die umsatzstärksten Kunden zu Oberst dargestellt werden.
- **Erstellen eines Diagramms aus bestehenden Daten:**
Ihre Aufgabe ist es nun, ein Liniendiagramm für die drei umsatzstärksten Kunden d.h. für die drei obersten Zeilen 5 bis 7 der sortierten Tabelle zu zeichnen. Nach rechts soll dabei die zeitliche Entwicklung von 2010 bis 2013 und nach oben die zugehörigen Erträge dargestellt werden. Jeder der drei Kunden soll auf einer separaten Linie dargestellt werden. Der Diagrammtitel soll „Umsatzstärkste Kunden“ lauten.
- **Benennen einer Zelle mit einem Namen:**
Für die Berechnung weiterer Kennzahlen benötigen Sie nun mehrfach die Summe der Erträge aus dem Jahr 2013 in Zelle E15. Sie halten es für übersichtlicher, dieser Zelle einen Namen zu geben, um sie in diesem Zuge nicht nur durch die Angabe „E15“ sondern auch mit dem Namen „Gesamtertrag2013“ verwenden zu können. Sie möchten die Zelle deshalb nun „Gesamtertrag2013“ nennen.

9.2 Datenstrukturanalyse

Die Aktivontologie für die Darstellung von Beziehungen nach [Fuc18].

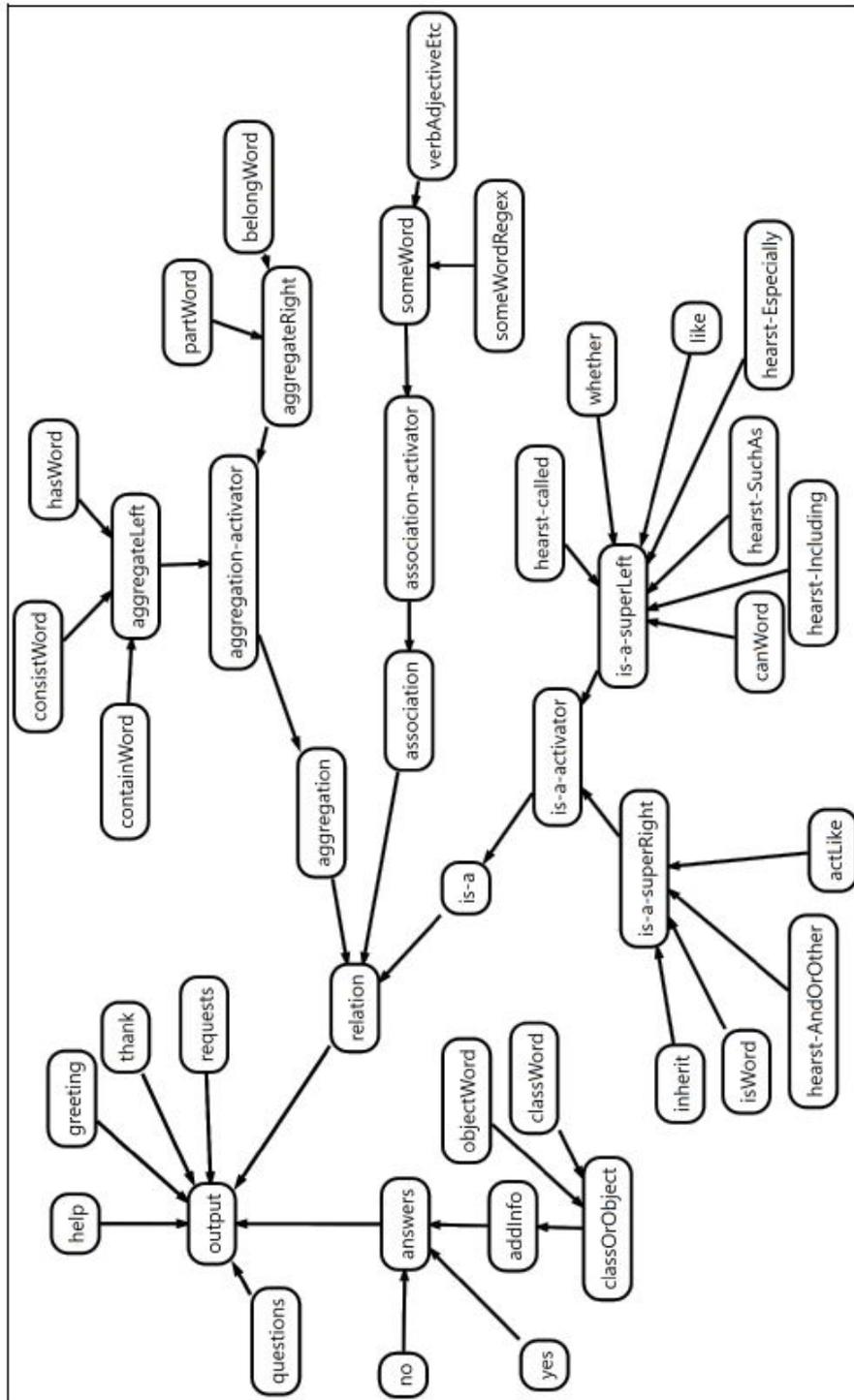


Abbildung 9.3 zeigt eine beispielhafte Interaktion und Ausgabe in *JustLingo NLUI*.

JustLingo

How can I help you?
4:13:49 PM

A timbered house consists of 5 to 10 logs, 200 to 400 bricks and 1000 to 2000 nails. Every building-material, whether log, brick or nail, is part of exactly one timbered house. Each timbered house has a certain number of rooms and floors. For the construction of a timbered house at least one carpenter is responsible, that has a name and an individual hourly wage. To build the timbered house, every carpenter uses his own tool, consisting of exactly one hammer and exactly one saw. Each carpenter can build a maximum of one half-timbered house at the same time.
4:14:28 PM

Multiple Options possible .. please choose one of them ..
0: cancel
1: construction has a name and an individual hourly wage .
2: house has a name and an individual hourly wage .
3: carpenter has a name and an individual hourly wage .
4:14:28 PM

3
4:14:40 PM

I could not identify the relation. Please try to reformulate your task. I guess there is a connection between construction, house, carpenter.
I could not identify the relation. Please try to reformulate your task. I guess there is a connection between house, carpenter, tool, hammer, saw.
4:14:41 PM

show classes, objects and relations
4:14:52 PM

Known Classes: house, log, brick, nail, number, room, floor, carpenter, maximum, time, building-material, name, wage
Known Objects: NONE

Known Relations:
[Relation: Name=house-has-log Type=Aggregation View=ClassView Aggregate=house Part=log]
[Relation: Name=house-has-brick Type=Aggregation View=ClassView Aggregate=house Part=brick]
[Relation: Name=house-has-nail Type=Aggregation View=ClassView Aggregate=house Part=nail]
[Relation: Name=house-has-number Type=Aggregation View=ClassView Aggregate=house Part=number]
[Relation: Name=house-has-room Type=Aggregation View=ClassView Aggregate=house Part=room]
[Relation: Name=house-has-floor Type=Aggregation View=ClassView Aggregate=house Part=floor]
[Relation: Name=build Type=Association View=ClassView Left=carpenter Right=maximum]
[Relation: Name=build Type=Association View=ClassView Left=carpenter Right=house]
[Relation: Name=build Type=Association View=ClassView Left=carpenter Right=time]
[Relation: Name=house-has-building-material Type=Aggregation View=ClassView Aggregate=house Part=building-material]
[Relation: Name=carpenter-has-name Type=Aggregation View=ClassView Aggregate=carpenter Part=name]
[Relation: Name=carpenter-has-wage Type=Aggregation View=ClassView Aggregate=carpenter Part=wage]
Generalization log<building-material
Generalization brick<building-material
Generalization nail<building-material
4:14:53 PM

Abbildung 9.3: Gesamtablauf der Aufgabe in *JustLingo NLUI*

9.3 Algorithmenerkennung

Aufgabestellung:

Sortierung einer Sequenz an Zahlen (a_1, \dots, a_n)



Die natürlichsprachliche Eingabe eines Benutzers:

The result is a vector. Initially it is empty. Find the minimal element of the set and append it to the vector. Remove the element form the set. Then, repeatedly find the minimum of the remaining elements and move them to the result in order, until there are no more elements in the set.



Überführung der natürlichen Sprache zum UML Modell:

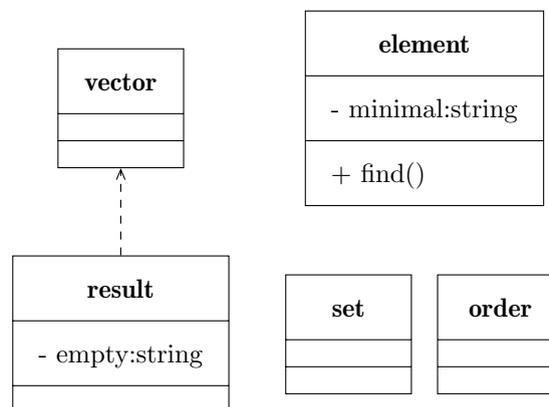


Abbildung 9.4: Überführung der natürlichen Sprache zum UML Modell



```

namespace Name
{
  class result : vector
  {
    private string empty;

    public result(string newempty)
    {
      this.empty = newempty;
    }
  }

  class vector
  {
    public vector() {}
  }
}
  
```

```

class element
{
    private string minimal;

    public element(string newminimal)
    {
        this.minimal = newminimal;
    }

    public void find() { }
}

[...]

class order
{
    public order() { }
}
}

```

Quelltextausschnitt 9.1: Quellcodeausgabe aus dem UML Modell

Im weiteren Beispiel wird eine Algorithmenerkennung schrittweise gezeigt [Eur17].

Die natürlichsprachliche Eingabe eines Benutzers:

create a new function called min
 set val to first element of the input
 iterate over the remaining elements
 if the current value is less than val use it as val
 at the end save val to A2
 set input to A1

Die Algorithmenerkennung ergibt dabei schrittweise den folgenden Quellcode:

Schritt 1:

create a new function called min

```

min( dynamic input = null )
{
}

```

Schritt 2:

create a new function called min
 set val to first element of the input

```

min( dynamic input = null )
{
    Int32 val = input[ 0 ];
}

```

Schritt 3:

create a new function called min
 set val to first element of the input
 iterate over the remaining elements

```
min( dynamic input = null )
{
    Int32 val = input[ 0 ];

    for ( Int32 indexA = 0; indexA < input.Size(); indexA++)
    {

    }
}
```

Schritt 4:

create a new function called min
 set val to first element of the input
 iterate over the remaining elements
 if the current value is less than val use it as val

```
min( dynamic input = null )
{
    Int32 val = input[ 0 ];

    for ( Int32 indexA = 0; indexA < input.Size(); indexA++)
    {
        if ( input[ indexA ] < val )
        {
            val = input[ indexA ];
        }
    }
}
```

Schritt 5a mit Fehler:

create a new function called min
 set val to first element of the input
 iterate over the remaining elements
 if the current value is less than val use it as val
 save val to A2

```
min( dynamic input = null )
{
    Int32 val = input[ 0 ];

    for ( Int32 indexA = 0; indexA < input.Size(); indexA++)
    {
        if ( input[ indexA ] < val )
        {
            val = input[ indexA ];
            A2 = val;
        }
    }
}
```

Schritt 5b mit Korrektur und verbesserter Ausführung:

create a new function called min
set val to first element of the input
iterate over the remaining elements
if the current value is less than val use it as val
undo
at the end save val to A2

```
min( dynamic input = null )
{
    Int32 val = input[ 0 ];

    for ( Int32 indexA = 0; indexA < input.Size(); indexA++)
    {
        if ( input[ indexA ] < val )
        {
            val = input[ indexA ];
        }
    }
    A2 = val;
}
```

Schritt 6:

create a new function called min
set val to first element of the input
iterate over the remaining elements
if the current value is less than val use it as val
at the end save val to A2
set input to A1

```
min( List<Int32> input = A1 )
{
    Int32 val = input[ 0 ];

    for ( Int32 indexA = 0; indexA < input.Size(); indexA++)
    {
        if ( input[ indexA ] < val )
        {
            val = input[ indexA ];
        }
    }
    A2 = val;
}
```

9.4 Evaluation

9.4.1 Aktivontologie

Aufgabestellung:

Die Einnahmen für den Januar wurden vergessen in die Buchführung einzutragen. Tragen Sie eine zufällig generierte Zahl zwischen 1000 und 1500 Euro als Netto-Einnahmen für den Januar ein (B2).

Im April wurde der Bruttogewinn nicht berechnet. Tragen Sie dort (C5) die entsprechenden Brutto-Einnahmen ein, indem Sie sich auf die Netto-Einnahmen des selben Monats beziehen (B5).

Im September wurde der Mehrwertsteueranteil nicht korrekt übernommen. Tragen Sie in D10 die für diesen Monat anfallende Mehrwertsteuer ein.

Der Chef möchte eine Gesamtbilanz des Jahres ziehen. Berechnen Sie die Summe aller Netto-Einnahmen des Jahres und tragen sie diese anschließend in B15 ein.

Außerdem soll festgestellt werden wie groß die Schwankungen waren. Tragen Sie dafür bei Bester Monat, Schlechtester Monat und Monatsdurchschnitt jeweils die fehlenden Felder ein.

Zusätzlich zur Mehrwertsteuer kommt eine Bearbeitungsgebühr von 500 Euro pro Jahr. Berechnen Sie Ausgehend den Netto-Einnahmen des Jahres die Gesamtabgaben inklusive Mehrwertsteuer und Bearbeitungsgebühr und speichern Sie diese in E15.

Wenn in einem Monat mehr als 1500 Euro Nettogewinn gemacht wurde fallen zusätzliche abgaben von 100Euro an. Tragen Sie in E5 und E13 jeweils ein wie viele Angaben anfallen ohne dies selbst zu überprüfen (100 bei Netto-Einnahmen über 1500 Euro, 0 sonst).

Beispieldialog:

User - please put a randomn number between 1000 and 1500 in cell C2

System - Everything done as you wished for. Your input results in the following formula: 1000 Can I help you with anything else?

User - please put 93 percent of cell C5 in cell B5

System - Already done. This is the formula i put together: product(93/100, C5) Is there anything else i can do for you?

User - put 7 percent of cell C10 in cell D10

System - Already done. Your input results in the following formula: product(7/100, C10) Is there anything else i can do for you?

User - Put total sum from cell B2 o cell B13 in cell B15

System - Already done. Your input results in the following formula: sum(B2, B13) Is there anything else i can do for you?

User - put minimum from C2 to C13 in C18

System - Already done. This is the formula i put together: min(C2:C13) Is there anything else i can do for you?

User - put average from D2 to D13 in D19

System - Everything done as you wished for. This is the formula i put together: avera-ge(D2:D13) Can I help you with anything else?

User - put 7 percent of cell C15 plus 500 in cell E15

System - Your input is ambiguous. Following interpretations are possible:

a) `product(7/100, sum(C15, 500))`

b) `sum(product(7/100, C15), 500)`

Please choose which one you were trying to tell me.

User - b)

System - Ok i did that for you. Can I do something else for you?

User - if in cell C2 to C13 the number is higher than 1500 put 100 in the cell right of it

System - I get you right. Since I found too few arguments in your input i assumed you tried to reference the last formula and used it. This is the formula i put together: `miss` Where exactly do you want me to save that in the worksheet?

User - put 100 in cell E5

System - Ok i did that for you. This is the formula i put together: `100` Can I do something else for you?

User - if the number in C13 is higher than 1500 put 100 in the cell E13

System - Excuse me but I can not compute your input with the given operands because there are too few arguments.

User - if the number in C13 is higher than 1500 put 100 in the cell E13 otherwise put 0 in cell E13

System - Everything done as you wished for. Can I help you with anything else?

9.4.2 Klassifikation

Algorithmenfassung

Trainingsdaten

- If the first value of numbers is not smaller than 10
- If the size of x is greater than 1 remove all elements less than the first from it
- Whenever an element of the result is larger than the auxiliary variable
- Add the element at index 3 from the set called numbers to the end of the input
- Create a new variable
- While the size of the result is 0 remove the last element from the result
- Check if maximum is smaller than minimum and otherwise set minimum to 5
- If there is a value less than the minimum in the input, use it as minimum
- Do the previously steps until the input is empty
- At the end write it into A1
- If x is smaller than the size of y
- Save it in an auxiliary variable
- Remove all elements less than the first from it
- For each element of the input check if it is equal to 0 and add it to the result
- Define a new set called xy with a size of 10
- Set the 3 element of it to 10
- If size of numbers is greater than 0
- Set maximum to 12 after initialize minimum with the first element of numbers
- Create a function for sorting
- First of all set x to 12
- Set the input to A3
- Add minimum to the result after removing it from the input

- The input value of the function should be column A
- Add each element of the input to the result
- Take the values from column A as input
- Whenever an element is larger than the variable, store it in the variable
- Check if maximum is smaller than minimum set minimum to 9 otherwise
- Iterate over x
- If numbers does not contain x
- Set x to the size of numbers
- Set minimum to 5 if maximum is contained in result
- Take A3 as input
- Do this for each element of the input
- X should be contained in result or the first value of the output is equal 10
- If there is a value less than the minimum in the input
- Define a new set with 3 fields
- Whenever an element is larger than 3, initialize the size of alpha with it
- Use 1 after finished iterating over the set
- While x is greater 5
- Do the action before before x was set to 4
- Iterate until the input isn't empty
- Remove the first element from numbers if numbers contains minimum
- Set all elements of result less than 3 to minimum
- Then visit all the remaining elements
- Do the previously steps till the size of the input is 0
- Perform this if x is smaller than 5, otherwise do the previous step
- Remove the first value of numbers from the the input
- Add the minimum to the input

Testdaten

- The first value of numbers should be smaller than 3
- Do everything before after setting x to 3
- If maximum is smaller than minimum set minimum to 5
- Remove last element of result
- Set abc to its value
- Initialize the variable copy with the input at index 3
- Take the values from column A as input for the new function named sort
- Remove minimum from the input
- X should has a smaller size than the input
- Do all steps until the input is empty
- After all previous steps use 3 as minimum
- After setting the maximum to 12, initialize the minimum with 6

Mathematische Berechnungen

Trainingsdaten

- Calculate the difference of the previous result and E7
- Could you please calculate the sine of 1?
- Give me the negation of -3 please
- Could you give me the root of 42?
- Add to the last value 3000
- Give me the sum of D11 and D12 times 9
- What is the product of 5 and 9?

- What is the square root of 16?
- Are you able to calculate the sum of 20 and 54?
- What is the average value of all the classes' grades?
- What is the mean of E1 to F8?
- Give me 68% of E21
- Divide the product of 42 and 16 by 2
- Now subtract E8
- Calculate 1000 multiplied by the sum of E6 and E7
- Calculate 3 times 500 then add 17
- Count the number of entries in 2010
- Now calculate the root of that number
- Subtract 7 from G5 then multiply by 19
- Calculate the average of column X
- Give me the sum of cell A1 to cell F6
- I need you to calculate the difference between 10 and 9
- Calculate Pi with 3 digits
- Subtract E8 from the last value
- What is the product of 45 and 67?
- Calculate the average of D minus 5
- Write the value to E21
- Multiply the last value by B7
- Add E6 to E7 and multiply it with 1000
- Calculate the product of the previous result with B7
- Calculate 8 minus 3% multiplied by the value of cell D8
- What is the square root of 49?
- Where is the maximum of this function?
- Calculate the sum from E11 to E20
- Please calculate cell A1 minus cell B1
- Count the number of entries in row 3
- Give me 68% of 200
- Would you tell me the Minimum of the equation $y=x$
- Round Pi to 3 decimals.
- Save it in E21
- Calculate PI rounded to 3 decimals
- Please give me a random number.
- Would you calculate the square root of 64
- I want to know the root of 4
- Now write it in E21
- Save that formula in E8
- Multiply that result by the value of B7
- Put the result in cell E8
- Round PI to 3 decimals
- Calculate the sum of E6 and E7 and multiply it by 1000
- What is the average of 3, 5, 7 and 9?
- The maximum of -3, 4 and -5 please.
- Calculate the product of 7 and 8 please.
- Now add 3000
- Write the last value in E8.
- Calculate 68% of E21

Testdaten

- Round pi on three decimals

- Please calculate the product of 3 and 89
- Calculate the average of column E to column H
- Add 3000 to the previous result
- tell me the number of customers in 2010
- Calculate 68% of E21
- Get the number of filled lines in 2010
- Now subtract C8
- Add 3000 to the last product
- Save it to cell C7
- Please add e6 to e7 and multiply it with 1000
- Add 5 times the mean of column D to 25
- Sum up the revenue of 2013
- Could you calculate the average of 3,6,9,11,78,5 and 121

Klassenmodelle

Trainingsdaten

- The key belonging to a content is calculated by the player from the stored and secured DeviceKey and the MediaKey stored on the medium
- What is the type of?
- The information shown on the displays can also be observed on monitors of the station tracks
- The teleconverters can be combined with all lenses
- A recording device must therefore generate a separate key for the content for each blank medium, which depends on the MediaKey of the medium and the DeviceKey
- The extension factor is specified for the teleconverters
- The station control only transmits the track on which a train has departed
- The internal station network will be relieved by exchanging as little data as possible between the displays and the station control system
- ashes that only fit the professional camera cases, and three amateur
- The photo system also contains
- There are two types of buildings - a house and a castle
- Each business transaction generates a booking record, which consists of a date, a booking text, the debit account, the credit account and the amount
- The focal length value must always be greater than 0
- There is a compatible sun visor for each lens; the diameter of the filter thread is provided for both
- Castles and Houses are both buildings
- Copies of prerecorded media all have the same MediaKey
- Each of the accounts for double-entry book-keeping has a name and a current value
- The connected displays must then request additional information from the station control system
- Of these five camera cases, two are intended for professional use and three for amateur use
- Each train has one or two locomotives
- Can you show relations between the objects?
- The debit and credit sides are ordered lists of entries, each consisting of an amount and a date
- At each track there are additional track boards which only show the data of the next train departing on it
- Ships are either passenger or container ships
- A diesel engine consists of several cylinders

- Every ship has at least one diesel engine
- In addition to the five camera cases, there are a total of 20 different lenses that can be used on both professional and amateur camera cases
- The maximum focal length of a zoom lens is always at least 5 mm larger than the minimum focal length of the zoom lens
- Among the lenses offered are two teleconverters and 18 different lenses
- The photo system consists of five different camera cases
- The display media will delete this train and replace it with a later departing train
- This amount must always be positive!
- Every passenger wagon has an air conditioning system that can be switched on and off
- Passenger wagons consist of up to one wider area and one or more compartments
- The operands can be either a number or another binary arithmetic expression
- Each of these units has an owner, any number of residents and if necessary a renter and a landlord
- All booking records are stored in the same central journal
- Track boards show the next train departing on this track
- The central station control manages all necessary data about the current train connections of a train station
- Please sync the tables
- At least one of the occupants of an apartment must own or rent it
- All content is encrypted using either the AES-128 or Triple-DES method
- Each account has exactly one debit and one credit side and can be used both as a debit account or as a credit account
- A passenger ship can have one or more restaurants, each of which provides a certain number of seats
- Track displays only trigger reactions if a train have left on their station track
- The wagons of a passenger train are at least one passenger wagon and at most one dining car
- A cargo train is a train whose wagons are exclusively cargo wagons
- Zoom lenses have a minimum and a maximum focal length, while fixed focal lengths have only one focal length
- Each air conditioner may only be operated up to a certain maximum outside temperature
- In addition it registers departing trains and informs all connected display media
- Each person lives in at least one accommodation unit, but exactly one of these residential units is their main residence
- Every person should be able to enter into or terminate a rental, ownership or residential relationship at any time

Testdaten

- Furthermore the display shows the track number, the destination and the type of the train, for example a ICE
- In case of blank media, each medium has its own individual MediaKey; there are no two blank media with the same MediaKey
- A building consists of one or more residential, business and/or office units.
- There are two professional
- The lenses are divided into fixed focal lengths and zoom lenses
- Each cylinder has valves and a combustion chamber
- Debit account is the account where the amount is taken on the specified date, credit account is the account where the amount is added on the specified date
- Can control any number of
- If there is a renter, of course there must also be a landlord and vice versa
- Every station has a big display which shows the actual departure times of the trains

- Ashes that only fit the amateur camera cases
- A binary arithmetic expression contains an operand, an operator such as sum, subtraction, multiplication or division, and another operand

Datenstrukturanalyse

Trainingsdaten

- Mark the 5 persons with the highest age
- Mark the five persons with the highest age
- Concatenate all the entries in job in Table Persons and save the result in F1
- Mark persons with age greater than
- Mark persons with age 18
- Mark all persons with age over 50
- Mark all persons age 18
- Mark all entries in Persons with an age of 18
- Mark all entries in Persons with an age of 18
- Add age of all persons with title Dr.
- Mark all rows in Persons with an age of 18
- Sum of all persons where title is dr. and save result to F1
- Calculate how many entries in Table Persons have the title doctor and an age of less than 40 and save it
- Sort by mail
- Sum of all persons with title Dr. , save in F1
- Sum age of all persons where title is dr. and save result to f1
- How many entrie in last names of table person start with A
- Mark every person age > 50
- How many entries in Table Persons have the title doctor and an age of less than 40
- Sum age of all persons with title Dr. , save sum in F1
- Group credit cards by provider
- Sort D3 to D52
- Sort Companies by mail
- Sort mails

Testdaten

- Group by proband
- Sort by proband
- Mark persons with
- Mark the top ten persons with the highest age
- Group same provider

9.5 Publikationsliste

- [WEFT20] Alexander Wachtel, Felix Eurich, Dominik Fuchß, Walter F. Tichy. *Context Classification in Dialog-Based Interaction*. 2020, February. The 14th IEEE International Conference on semantic computing (ICSC), San Diego, California.
- [WFST19] Alexander Wachtel, Dominik Fuchß, Sophie Schulz, Walter F. Tichy. *Approaching natural conversation chatbots by interactive dialogue modeling & Microsoft LUIS*. 2019, December. IEEE 1th International Conference Conversational Data & Knowledge Engineering (CDKE). San Diego, California.
- [WFPT19] Alexander Wachtel, Dominik Fuchß, Matthias Przybylla, Walter F. Tichy. *Natural Language Data Queries on Multiple Heterogenous Data Sources*. 2019, July. The Seventh International Symposium on End-User Development , Hertfordshire, UK.
- [WST19] Alexander Wachtel, Sophie Schulz, Walter F. Tichy. *Dialog-based Meaning Derivation Service For Technical Language Domains*. 2019, January. The 13th IEEE International Conference on semantic computing (ICSC), Newport Beach, California.
- [WET18] Alexander Wachtel, Felix Eurich, Walter F. Tichy. *Programming In Natural Language: Building Algorithms From Human Descriptions*. 2018, March. The Eleventh International Conference on Advances in Computer-Human Interaction, Rom, Italy.
- [WET17] Alexander Wachtel, Felix Eurich, Walter F. Tichy. *Programming Spreadsheets in Natural Language: Design of a Natural Language User Interface*. 2017, December. The International Journal on Advances in Software, Vol. 10, no. 3&4.
- [WPT17] Alexander Wachtel, Jonas Paczia, Walter F. Tichy. *An Interactive Action Set Detection In Natural Language Hybrid System*. 2017, July. The Eleventh International Conference on Interfaces and Human Computer Interaction, Lisbon, Portugal.
- [WKT17] Alexander Wachtel, Jonas Klamroth, Walter F. Tichy. *Natural Language User Interface For Software Engineering Tasks*. 2017, March. The Tenth International Conference on Advances in Computer-Human Interactions, Nice, France.
- [WKT16] Alexander Wachtel, Jonas Klamroth, Walter F. Tichy. *A Natural Language Dialog System Based on Active Ontologies*. 2016, April. The Ninth International Conference on Advances in Computer-Human Interactions, Venedig, Italy.
- [WWT15] Alexander Wachtel, Sebastian Weigelt, Walter F. Tichy. *Initial implementation of natural language turn-based dialog system*. 2015, October. The Seventh International Conference on Intelligent Human Computer Interaction, Allahabad, India.

- [WWVT15] Alexander Wachtel, Sebastian Weigelt, Philipp Voigt, Walter F. Tichy. *Prototyp einer natürlchsprachlichen Schnittstelle für Tabellenkalkulation*. 2015, März. Multikonferenz Software Engineering & Management. Dresden.

9.6 Programmausschuss und Gutachter

- INTERACT'23 Technical Review (Full Paper), the 19th IFIP TC.13 International Conference on Human-Computer Interaction
- AIKE'21 Program Committee Co-Chair, the 4rd IEEE International Conference on Artificial Intelligence & Knowledge Engineering
- RAISE'21 Program Committee, the 9th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering
- ASE'21 Program Committee (Artifact Evaluation), the 36th IEEE/ACM International Conference on Automated Software Engineering
- AIKE'20 Program Committee Co-Chair, the 3rd IEEE International Conference on Artificial Intelligence & Knowledge Engineering
- CC'20 Program Committee Co-Chair, the 1st IEEE International Conference on Conversational Computing
- RAISE'20 Program Committee, the 8th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering
- ICSC'20 Workshop Co-Chair, the 14th IEEE International Conference on Semantic Computing
- CDKE'19 Program Committee Co-Chair, the 1st IEEE International Conference on Conversational Data and Knowledge Engineering
- INTERACT'19 Technical Review (Short Paper), the 17th IFIP TC.13 International Conference on Human-Computer Interaction
- RAISE'19 Program Committee @ RAISE'19, the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering
- CHI'19 Technical Review Case Studies @ ACM CHI'19, the ACM CHI Conference on Human Factors in Computing Systems
- JSS'19 Technical Review, Journal of Systems and Software

Literaturverzeichnis

- [ACF07] ALLEN, James ; CHAMBERS, Nathanael ; FERGUSON, George: PLOW: A Collaborative Task Learning Agent. In: *Association for the Advancement of Artificial Intelligence* (2007)
- [AE04] ABRAHAM, Robin ; ERWIG, Martin: Header and Unit Inference for Spreadsheets Through Spatial Analyses. In: *IEEE Symposium on Visual Languages - Human Centric Computing* (2004)
- [Ale20] letzter Aufruf: 9. April 2024, <https://developer.amazon.com/de/alexa>
- [ALS⁺20] ADIWARDANA, Daniel ; LUONG, Minh-Thang ; SO, David R. ; HALL, Jamie ; FIEDEL, Noah ; THOPPILAN, Romal ; YANG, Zi ; KULSHRESHTHA, Apoorv ; NEMADE, Gaurav ; LU, Yifeng ; LE, Quoc V.: *Towards a Human-like Open-Domain Chatbot*. 2020. – <https://arxiv.org/abs/2001.09977>
- [ART95] ANDROUTSOPOULOS, Ion ; RITCHIE, Graeme D. ; THANISCH, Peter: Natural Language Interfaces to Databases – An Introduction. In: *Natural Language Engineering, 1(1):29-81, Cambridge University Press* (1995)
- [Ass20] letzter Aufruf: 9. April 2024, <https://www.ibm.com/de-de/cloud/watson-assistant>
- [Aut20] letzter Aufruf: 9. April 2024, <https://docs.microsoft.com/en-us/power-virtual-agents/advanced-flow>
- [BB79] BALLARD, Bruce W. ; BIERMANN, Alan W.: Programming in natural language: NLC as a prototype. In: *Association for Computing Machinery (ACM), Volume 10* (1979)
- [BB80] BIERMANN, Alan W. ; BALLARD, Bruce W.: Toward Natural Language Computation. In: *American Journal of Computational Linguistics, Volume 6, Number 2* (1980)
- [BBS83] BIERMANN, Alan W. ; BALLARD, Bruce W. ; SIGMON, Anne H.: An experimental study of natural language programming. In: *International Journal of Man-Machine Studies* (1983)
- [BBW17] BORDES, Antoine ; BOUREAU, Y-Lan ; WESTON, Jason: Learning end-to-end goal-oriented dialog. (2017)
- [BD10] BERG, Markus M. ; DÜSTERHÖFT, Antje: Interaction with Text-based Natural Language Dialog Systems. In: *7. Wismarer Wirtschaftsinformatiktag* (2010)
- [Beg05] BEGEL, Andrew: Spoken Language Support for Software Development. In: *Ph.D. Thesis, Berkeley* (2005)
- [Ber12] BERG, Markus M.: Natürlichsprachlichkeit in Dialogsystemen. In: *Stand der Technik in Industrie und Forschung, Springer-Verlag* (2012)

- [Ber14] BERG, Markus M.: Modelling of Natural Dialogues in the Context of Speech-based Information and Control Systems. In: *Dissertation, Technischen Fakultät der Christian-Albrechts-Universität zu Kiel* (2014)
- [BHL⁺14] BUSSE, Johannes ; HUMM, Bernhard ; LÜBBERT, Christoph ; MOELTER, Frank ; REIBOLD, Anatol ; REWALD, Matthias ; SCHLÜTER, Veronika ; SEILER, Bernhard ; TEGTMEIER, Erwin ; ZEH, Thomas: Was bedeutet eigentlich Ontologie? - Ein Begriff aus der Philosophie im Licht verschiedener Disziplinen. In: *Informatik Spektrum* 37 (2014), Nr. 4, 286–297. <http://dx.doi.org/10.1007/s00287-012-0619-2>. – DOI 10.1007/s00287-012-0619-2
- [Bir10] BIRD, John: Basic Engineering Mathematics. In: *Elsevier* (2010)
- [BKL09] BIRD, Steven ; KLEIN, Ewan ; LOPER, Edward: Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. (2009)
- [BMR⁺20] BROWN, Tom B. ; MANN, Benjamin ; RYDER, Nick ; SUBBIAH, Melanie ; KAPLAN, Jared ; DHARIWAL, Prafulla ; NEELAKANTAN, Arvind ; SHYAM, Pranav ; SASTRY, Girish ; ASKELL, Amanda ; AGARWAL, Sandhini ; HERBERT-VOSS, Ariel ; KRUEGER, Gretchen ; HENIGHAN, Tom ; CHILD, Rewon ; RAMESH, Aditya ; ZIEGLER, Daniel M. ; WU, Jeffrey ; WINTER, Clemens ; HESSE, Christopher ; CHEN, Mark ; SIGLER, Eric ; LITWIN, Mateusz ; GRAY, Scott ; CHESS, Benjamin ; CLARK, Jack ; BERNER, Christopher ; MCCANDLISH, Sam ; RADFORD, Alec ; SUTSKEVER, Ilya ; AMODEI, Dario: *Language Models are Few-Shot Learners*. 2020. – <https://arxiv.org/abs/2005.14165>
- [BZG13] BÄR, Daniel ; ZESCH, Torsten ; GUREVYCH, Iryna: DKPro Similarity: An Open Source Framework for Text Similarity. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2013, S. 121–126
- [Car17] CARSTENSEN, Kai-Uwe: Sprachtechnologie - ein Überblick. In: *Universität Siegen* (2017)
- [CEE⁺10] CARSTENSEN, Kai-Uwe ; EBERT, Christian ; EBERT, Cornelia ; JEKAT, Susanne J. ; KLABUNDE, Ralf ; LANGER, Hagen: Computerlinguistik und Sprachtechnologie - eine Einführung. In: *Spektrum Akademischer Verlag Heidelberg* (2010)
- [CFM⁺12] CUNHA, Jácome ; FERNANDES, Joã. ; MENDES, Jorge ; PACHECO, Hugo ; SARAIVA, João: Bidirectional transformation of model-driven spreadsheets. In: *International Conference on Theory and Practice of Model Transformations* Springer, 2012, S. 105–120
- [CLLS09] CORMEN, Thomas H. ; LEISERSON, Charles E. ; L.RIVEST, Ronald ; STEIN, Clifford: *Introduction to algorithms*. MIT Press, 2009
- [Cox14] COX, Gunther: *Chatterbot Documentation*. <https://chatterbot.readthedocs.io/>. Version: 2014
- [CP11] CHONG, Stephen ; PUCELLA, Riccardo: A Framework for Creating Natural Language User Interfaces for Action-Based Applications. In: *Cornell University* (2011)
- [CX20] letzter Aufruf: 9. April 2024, <https://dialogflow.com>
- [Cyp93] CYPHER, Allen: Watch what I do: programming by demonstration. (1993)

- [DCLT19] DEVLIN, Jacob ; CHANG, Ming-Wei ; LEE, Kenton ; TOUTANOVA, Kristina: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. – <https://arxiv.org/abs/1810.04805>
- [DSW08] DORNER, Christian ; SPAHN, Michael ; WULF, Volker: *End User Development: Approaches Towards A Flexible Software Design*. (2008)
- [EC93] ENGESSER, Hermann ; CLAUS, Volker: *Duden Informatik - ein Sachlexikon für Studium und Praxis*. In: *Dudenverlag* (1993). ISBN 978-3-411-05232-5
- [Eur17] EURICH, Felix: *Automatisiertes Erkennen und Erlernen neuer Programmabläufe im Kontext natürlichsprachlicher Beschreibungen von Algorithmen*. Karlsruher Institut für Technologie (KIT), Bachelorarbeit, 2017
- [Fer10] FERRUCCI, David: *Building Watson: An Overview of the DeepQA Project*. In: *Association for the Advancement of Artificial Intelligence* (2010)
- [Fry13] FRYE, Curtis D.: *Microsoft Excel 2013, Step by Step*. In: *O'Reilly Media* (2013)
- [Fuc18] FUCHSS, Dominik: *Verarbeitung natürlich-sprachlicher Beziehungsbeschreibungen zwischen Objekten*. Karlsruher Institut für Technologie (KIT), Bachelorarbeit, 2018
- [GBM13] GRAÇAS BRUNO MARIETTO, Maria das: *Artificial Intelligence Markup Language: A Brief Tutorial*. In: *CoRR* 1307.3091 (2013)
- [Gel10] GELHAUSEN, Tom: *Modellextraktion aus natürlichen Sprachen: Eine Methode zur systematischen Erstellung von Domänenmodellen*. Karlsruher Institut für Technologie (KIT), Dissertation, 2010
- [GM07] GABRILOVICH, Evgeniy ; MARKOVITCH, Shaul: *Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis*. In: *20th International Joint Conference on Artificial Intelligence* (2007)
- [GM14] GULWANI, Sumit ; MARRON, Mark: *NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation*. In: *SIGMOD* (2014)
- [Gul11] GULWANI, Sumit: *Automating string processing in spreadsheets using input-output examples*. (2011)
- [Guz06] GUZZONI, Didier: *Active: A unified platform for building intelligent web interaction assistants*. (2006)
- [Har84] HARRIS, Larry R.: *Experience with INTELLECT: Artificial Intelligence Technology Transfer*. In: *The AI Magazine* (1984)
- [HS97] HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: *Long Short-Term Memory*. 9 (1997), S. 1735–1780. <http://dx.doi.org/https://doi.org/10.1162/neco.1997.9.8.1735>. – DOI <https://doi.org/10.1162/neco.1997.9.8.1735>
- [Hur00] HURST, Matthew: *The interpretation of tables in texts*. In: *University of Edinburgh, Ph.D.* (2000)
- [Hü06] HÜTTNER, Tanja: *Einführung in die Sprachverarbeitung*. Universität Potsdam, 2006
- [KAB⁺11] KO, Andrew J. ; ABRAHAM, Robin ; BECKWITH, Laura ; BLACKWELL, Alan ; BURNETT, Margaret ; ERWIG, Martin ; SCAFFIDI, Chris ; LAWRENCE, Joseph ; LIEBERMAN, Henry ; MYERS, Brad ; ROSSON, Mary B. ; ROTHERMEL, Gregg ; SHAW, Mary ; WIEDENBECK, Susan: *The State of the Art in*

- End-user Software Engineering. In: *ACM Comput. Surv.* 43 (2011), April, Nr. 3, 21:1–21:44. <http://dx.doi.org/10.1145/1922649.1922658>. – DOI 10.1145/1922649.1922658. – ISSN 0360–0300
- [KFM⁺18] KOPLOWITZ, Rob ; FACEMIRE, Michael ; MINES, Christopher ; SJOBLUM, Sara ; LYNCH, Diane ; HARRISON, Peter: *The Forrester New Wave: Conversational Computing Platforms.* (2018)
- [Kla15] KLAMROTH, Jonas: *Implementierung des natürlichsprachlichen Dialogsystems JustLingo, basierend auf Active Ontologies.* Karlsruher Institut für Technologie (KIT), Bachelorarbeit, 2015
- [Ko04] KO, Andrew J.: Six Learning Barriers in End-User Programming Systems. In: *Visual Languages and Human Centric Computing (VLHCC)* (2004)
- [KPB⁺12] KALYANPUR, Aditya ; PATWARDHAN, Siddharth ; BOGURAEV, BK ; LALLY, Adam ; CHU-CARROLL, Jennifer: Fact-based question decomposition in DeepQA. (2012)
- [Kur12] KURZWEIL, Ray: How to Create a Mind: The Secret of Human Thought Revealed. In: *Viking Penguin* (2012). ISBN ISBN 978–0670025299
- [Kö14] KÖRNER, Sven J.: *Werkzeugunterstützung in der Anforderungserhebung.* Karlsruher Institut für Technologie (KIT), Dissertation, 2014
- [Lan16] LANDHÄUSSER, Mathias: *Eine Architektur für Programmsynthese aus natürlicher Sprache.* Karlsruher Institut für Technologie (KIT), Dissertation, 2016
- [LGS13] LE, Vu ; GULWANI, Sumit ; SU, Zhendong: SmartSynth: Synthesizing Smartphone Automation Scripts from Natural Language. In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys)* (2013)
- [Lin08] LINOFF, Gordon S.: Data analysis using sql and excel. (2008)
- [LL05] LIU, Hugo ; LIEBERMAN, Henry: Metafor: Visualizing stories as code. In: *10th international conference on Intelligent user interfaces* (2005)
- [LLX⁺20] LEPIKHIN, Dmitry ; LEE, HyoukJoong ; XU, Yuanzhong ; CHEN, Dehao ; FIRAT, Orhan ; HUANG, Yanping ; KRIKUN, Maxim ; SHAZEER, Noam ; CHEN, Zhifeng: *GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding.* 2020. – <https://arxiv.org/abs/2006.16668>
- [LPKW06] LIEBERMAN, Henry ; PATERNÒ, Fabio ; KLANN, Markus ; WULF, Volker: End-User Development: An Emerging Paradigm, *Human-Computer Interaction Series, Volume 9.* (2006)
- [ME01] MONGE, Alvaro E. ; ELKAN, Charles: An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. (2001)
- [Mic20a] MICROSOFT: *Language Understanding Intelligent Service.* 2020. – letzter Aufruf: 9. April 2024, www.luis.ai
- [Mic20b] MICROSOFT: *Power Virtual Agents Bots.* 2020. – letzter Aufruf: 9. April 2024, <https://docs.microsoft.com/de-de/power-virtual-agents/authoring-first-bot>
- [MKB06] MYERS, Brad A. ; KO, Andrew J. ; BURNETT, Margaret M.: Invited Research: Overview End-User Programming. In: *CHI* (2006)
- [MSB⁺14] MANNING, Christopher D. ; SURDEANU, Mihai ; BAUER, John ; FINKEL, Jenny R. ; BETHARD, Steven ; MCCLOSKEY, David: The stanford corenlp natural language processing toolkit. In: *ACL System Demonstrations* (2014)

- [OR08] OMAR, Nazlia ; RAZIK, Nomariani A.: Determining the basic elements of object oriented programming using natural language processing. In: *2008 International Symposium on Information Technology* Bd. 3, 2008. – ISSN 2155–8973, S. 1–6
- [Pat13] PATERNÒ, Fabio: End user development: Survey of an emerging field for empowering people. In: *ISRN Software Engineering* (2013)
- [PMP00] PRECHELT, Lutz ; MALPOHL, Guido ; PHILIPPSEN, Michael: JPlag: Finding plagiarisms among a set of programs. In: *Universität Karlsruhe, Technical Report 2000-1* (2000)
- [PNI⁺18] PETERS, Matthew E. ; NEUMANN, Mark ; IYER, Mohit ; GARDNER, Matt ; CLARK, Christopher ; LEE, Kenton ; ZETTLEMOYER, Luke: *Deep contextualized word representations*. 2018. – <https://arxiv.org/abs/1802.05365>
- [Ras20] letzter Aufruf: 9. April 2024, <https://rasa.com/docs>
- [RHW86] RUMELHART, David E. ; HINTON, Geoffrey E. ; WILLIAMS, Ronald J.: Learning representations by back-propagating errors. (1986). <http://dx.doi.org/https://doi.org/10.1038/323533a0>. – DOI <https://doi.org/10.1038/323533a0>
- [RNSS18] RADFORD, Alec ; NARASIMHAN, Karthik ; SALIMANS, Tim ; SUTSKEVER, Ilya: Improving Language Understanding by Generative Pre-Training. In: *Technical report, OpenAI* (2018)
- [RSR⁺20] RAFFEL, Colin ; SHAZEER, Noam ; ROBERTS, Adam ; LEE, Katherine ; NARANG, Sharan ; MATENA, Michael ; ZHOU, Yanqi ; LI, Wei ; LIU, Peter J.: *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. – <https://arxiv.org/abs/1910.10683>
- [RT18] RAMAN, Anand ; TOK, Wee H.: A Developer’s Guide to Building AI Applications. (2018)
- [RWC⁺18] RADFORD, Alec ; WU, Jeffrey ; CHILD, Rewon ; LUAN, David ; AMODEI, Dario ; SUTSKEVER, Ilya: Language models are unsupervised multitask learners. In: *Technical report, OpenAI* (2018)
- [Sam66] SAMMET, Jean E.: The use of English as a programming language. In: *Communications of the ACM* 9 (1966), Nr. 3, S. 228–230
- [Sch17] SCHULZ, Sophie: *Erweiterung der Sprachdomäne durch das Erlernen von Synonymen basierend auf dem Dialogsystem JustLingo*. Karlsruher Institut für Technologie (KIT), Bachelorarbeit, 2017
- [Sen02] SENEFF, Stephanie: Response planning and generation in the MERCURY flight reservation system. In: *Computer Speech & Language* 16 (2002), Nr. 3-4, S. 283–312
- [Sin01] SINGHAL, Amit: LModern Information Retrieval: A Brief Overview. (2001). <http://dx.doi.org/http://singhal.info/ieee2001.pdf>. – DOI <http://singhal.info/ieee2001.pdf>
- [SJN05] SNOW, Rion ; JURAFSKY, Daniel ; NG, Andrew Y.: Learning Syntactic Patterns for Automatic Hypernym Discovery. Version: 2005. <http://papers.nips.cc/paper/2659-learning-syntactic-patterns-for-automatic-hypernym-discovery.pdf>. In: SAUL, L. K. (Hrsg.) ; WEISS, Y. (Hrsg.) ; BOTTOU, L. (Hrsg.): *Advances in Neural Information Processing Systems 17*. MIT Press, 2005, 1297–1304

- [Spe20] letzter Aufruf: 9. April 2024, <https://developer.amazon.com/de-DE/docs/alexa/alexa-voice-service/speechrecognizer.html>
- [SS13] SESTOFT, Peter ; SØRENSEN, Jens Z.: Sheet-defined functions: implementation and initial evaluation. In: *International Symposium on End User Development* Springer, 2013, S. 88–103
- [Sta17] STATISTICAL OFFICE OF THE EUROPEAN UNION (EUROSTAT): *Population on 1 January by age and sex*. appsso.eurostat.ec.europa.eu/nui/show.do?dataset=demo_pjan. Version: 2017
- [Sta18] STATISTICAL OFFICE OF THE EUROPEAN UNION (EUROSTAT): *House price index - annual data (2015 = 100)*. appsso.eurostat.ec.europa.eu/nui/show.do?dataset=prc_hpi_a. Version: 2018
- [THM09] TICHY, Walter F. ; HÖFER, Andreas ; MEDER, David: *Klausur: Softwaretechnik I*. Karlsruher Institut für Technologie (KIT), 2009
- [TKMS03] TOUTANOVA, Kristina ; KLEIN, Dan ; MANNING, Christopher D. ; SINGER, Yoram: Feature-rich part-of-speech tagging with a cyclic dependency network. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1* Association for Computational Linguistics, 2003, S. 173–180
- [TLK13] TICHY, Walter F. ; LANDHÄUSSER, Mathias ; KÖRNER, Sven J.: Universal Programmability - How AI Can Help. *Artificial Intelligence Synergies in Software Engineering*. (2013), May
- [Voi14] VOIGT, Philipp: *Datenverarbeitung in Tabellenkalkulationen durch das natürlichsprachliche Dialogsystem JustLingo*. Karlsruher Institut für Technologie (KIT), Bachelorarbeit, 2014
- [VSP⁺17] VASWANI, Ashish ; SHAZEER, Noam ; PARMAR, Niki ; USZKOREIT, Jakob ; JONES, Llion ; GOMEZ, Aidan N. ; KAISER, Lukasz ; POLOSUKHIN, Illia: *Attention Is All You Need*. 2017. – <https://arxiv.org/abs/1706.03762>
- [WEFT20] WACHTEL, Alexander ; EURICH, Felix ; FUCHSS, Dominik ; TICHY, Walter F.: Context Classification in Dialog-Based Interaction. In: *The 14th IEEE International Conference on semantic computing (ICSC)* (2020)
- [Wei21] WEIGELT, Sebastian: *Eine agentenbasierte Architektur für Programmierung mit gesprochener Sprache*. Karlsruher Institut für Technologie (KIT), Dissertation, 2021
- [WET17] WACHTEL, Alexander ; EURICH, Felix ; TICHY, Walter F.: Programming Spreadsheets in Natural Language: Design of a Natural Language User Interface. In: *The International Journal on Advances in Software, Vol. 10, no. 3&4* (2017)
- [WET18] WACHTEL, Alexander ; EURICH, Felix ; TICHY, Walter F.: Programming Spreadsheets in Natural Language: Design of a Natural Language User Interface. In: *The Eleventh International Conference on Advances in Computer-Human Interaction* (2018)
- [WFPT19] WACHTEL, Alexander ; FUCHSS, Dominik ; PRZYBYLLA, Matthias ; TICHY, Walter: Natural Language Data Queries on Multiple Heterogenous Data Sources. In: *The 7th International Symposium on End-User Development (IS-EUD)* (2019)

- [WFST19] WACHTEL, Alexander ; FUCHSS, Dominik ; SCHULZ, Sophie ; TICHY, Walter: Approaching natural conversation chatbots by interactive dialogue modeling & Microsoft LUIS. In: *IEEE 1th International Conference Conversational Data & Knowledge Engineering (CDKE)* (2019)
- [WFT20] WACHTEL, Alexander ; FUCHSS, Dominik ; TICHY, Walter F.: End User Software Engineering By Programming In Natural Language. In: *The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)* (2020)
- [WGLM17] WANG, Sida I. ; GINN, Samuel ; LIANG, Percy ; MANNING, Christopher D.: Naturalizing a programming language via interactive learning. In: *arXiv preprint arXiv:1704.06956* (2017)
- [Wie14] WIEST, Fabian: *Interaktive Mustergenerierung als Funktionserweiterung des natürlichsprachlichen Dialogsystems JustLingo*. Karlsruher Institut für Technologie (KIT), Bachelorarbeit, 2014
- [Wie18] WIEST, Fabian: *Objektorientierte Programmierung im natürlichsprachlichen Dialogsystem JustLingo*. Karlsruher Institut für Technologie (KIT), Masterarbeit, 2018
- [Wil09] WILLIAMS, Jason D.: Spoken dialogue systems: Challenges, and opportunities for research. In: *ASRU*, 2009, S. 25
- [WKT16] WACHTEL, Alexander ; KLAMROTH, Jonas ; TICHY, Walter F.: A Natural Language Dialog System Based on Active Ontologies. In: *The Ninth International Conference on Advances in Computer-Human Interactions* (2016)
- [WKT17] WACHTEL, Alexander ; KLAMROTH, Jonas ; TICHY, Walter F.: Natural Language User Interface For Software Engineering Tasks. In: *Tenth International Conference on Advances in Computer-Human Interactions* (2017)
- [WPT17] WACHTEL, Alexander ; PACZIA, Jonas ; TICHY, Walter F.: An Interactive Action Set Detection In Natural Language Hybrid System. In: *Eleventh International Conference on Interfaces and Human Computer Interaction* (2017)
- [WRB01] WAHLSTER, Wolfgang ; REITHINGER, Norbert ; BLOCHER, Anselm: Smart-Kom: Multimodal Communication with a Life-Like Character. (2001)
- [WST19] WACHTEL, Alexander ; SCHULZ, Sophie ; TICHY, Walter F.: Dialog-based Meaning Derivation Service For Technical Language Domains. In: *The 13th IEEE International Conference on semantic computing (ICSC)* (2019)
- [WWT15] WACHTEL, Alexander ; WEIGELT, Sebastian ; TICHY, Walter F.: Initial implementation of natural language turn-based dialog system. In: *International Conference on Intelligent Human Computer Interaction (IHCI)* (2015)
- [WWVT15] WACHTEL, Alexander ; WEIGELT, Sebastian ; VOIGT, Philipp ; TICHY, Walter F.: Prototyp einer natürlichsprachlichen Schnittstelle für Tabellenkalkulation. In: *Multikonferenz Software Engineering & Management* (2015)
- [YDY⁺20] YANG, Zhilin ; DAI, Zihang ; YANG, Yiming ; CARBONELL, Jaime ; SALAKHUTDINOV, Ruslan ; LE, Quoc V.: *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2020. – <https://arxiv.org/abs/1906.08237>
- [ZMG08] ZESCH, Torsten ; MÜLLER, Christof ; GUREVYCH, Iryna: Using Wiktionary for Computing Semantic Relatedness. In: *AAAI* Bd. 8, 2008, S. 861–866

- [ZSG⁺20] ZHANG, Yizhe ; SUN, Siqu ; GALLEY, Michel ; CHEN, Yen-Chun ; BROCKETT, Chris ; GAO, Xiang ; GAO, Jianfeng ; LIU, Jingjing ; DOLAN, Bill: *DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation*. 2020. – <https://arxiv.org/abs/1911.00536>