# Programmierung in natürlicher Sprache


| | |
|---|---|
| Dokumentenart: | Exposé für eine Masterarbeit |
| Autor: | Viktor Kiesel |
| Matrikel-Nr.: | 1542010 |
| Studiengang: | Informatik Master |
| Betreuer: | Sebastian Weigelt |
| Datum: | 22. Januar 2020 |


## 1 Motivation

Die Kommunikation zwischen Mensch und Maschine wandelte sich im Laufe der Zeit. Während frühe Rechner noch mit Lochkarten betrieben wurden, gewinnt heutzutage die Sprachsteuerung immer stärker an Bedeutung. Intelligente Assistenten wie *Siri* und *Amazon Alexa* nehmen eine immer größere Rolle im Alltag vieler Menschen ein. Sie helfen bei einfachen Aufgaben, wie dem Vorlesen neuer Nachrichten oder der Suche nach einem Restaurant. Diese Assistenten werden zwar über sprachliche Kommandos gesteuert, ihre Möglichkeiten sind aber durch fest einprogrammierte Anweisungen begrenzt. Komplexe Befehlsfolgen können nicht erkannt und daher auch nicht ausgeführt werden. In naher Zukunft ist anzunehmen, dass die natürliche Sprache als Schnittstelle zwischen Mensch und Maschine weiter an Bedeutung gewinnt. Sprache stellt für Menschen die natürlichste Art der Kommunikation dar. Von Rechnersystemen wie Robotern wird zukünftig erwartet, dass sie komplexe Befehlsfolgen verstehen. Sie sollten daher in der Lage sein Anweisungen zu interpretieren und selbstständig durchzuführen. Die Rechnersysteme sollen sich an den Menschen anpassen, Anweisungen in natürlicher Sprache verstehen und daraus ausführbaren Quelltext generieren.

In der Forschung findet die Erzeugung von Quelltext aus natürlicher Sprache daher einige Beachtung. Ein Überblick über die verschiedene Ansätze gibt die Arbeit von Pulido-Prieto und Juárez-Martínez [PJ17]. Viele Lösungsansätze schränken die Problemstellung ein. Ein häufiger Ansatz ist es, nur einen festen Satz erkennbarer Begriffe vorzugeben. Dies verfolgt beispielsweise die Arbeit *NaturalJava* [PRZH00]. Andere Ansätze schränken hingegen die Allgemeingültigkeit der Ausgabe ein. Natürliche Sprache wird dort nur in eine domänenspezifische Programmiersprache übersetzt. Beispiele hierfür wären *SmartSynth* [LGS13] und die Arbeit von Desai et al. [DGH+16] Ein besonderer Fokus wird oft auf die Übersetzung von Anweisungen für Robotern gelegt [KTF, LBK+02, LV13].

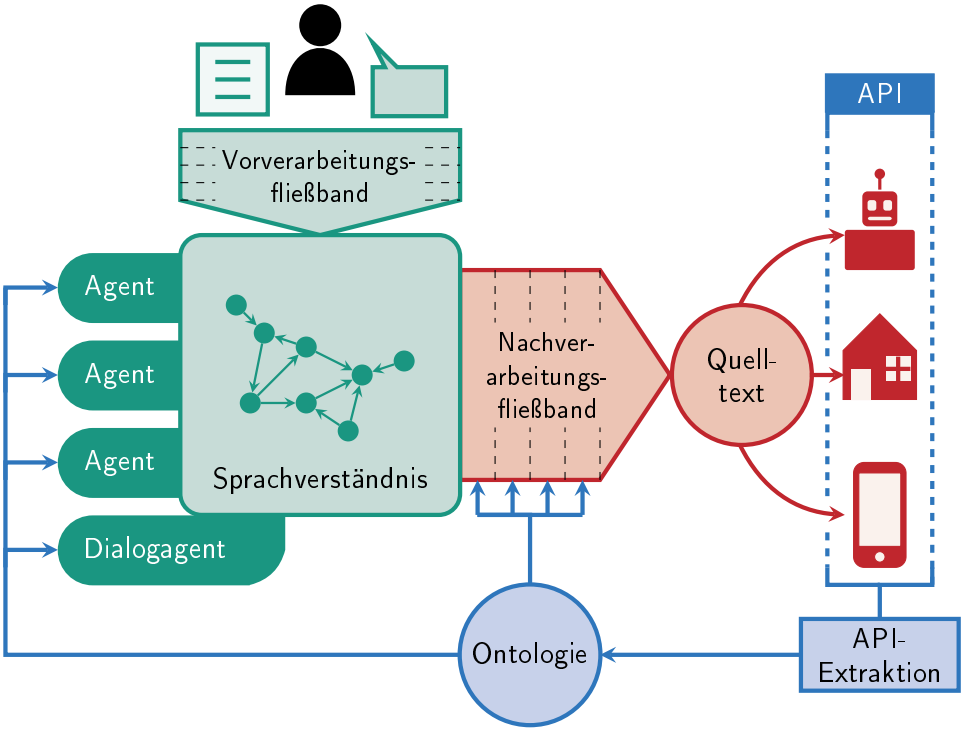


Abbildung 1: Dargestellt ist die PARSE-Architektur. In anderen Arbeiten implementierte Module sind grün. Die Umgebung ist blau gekennzeichnet. Sie enthält die Wissensrepräsentation und Schnittsstelle zum Zielsystem. Die Nachverarbeitung, die in dieser Arbeit bearbeitet wird, ist rot hervorgehoben.

Trotz verschiedener Ansätze existiert noch keine allgemeingültige, domänenunabhängige Lösung zur Übersetzung von natürlicher Sprache in Quelltext.

## 2 Projekt-Parse

Am IPD Tichy wird die *PARSE*-Rahmenarchitektur(**P**rogramming **AR**chitecture for **S**poken **E**xplanations) [WT15] entwickelt. Ziel des Projektes ist es ein domänenunabhängiges System zu entwickeln, welches in der Lage ist natürliche Spracheingaben eines Anwenders zu verarbeiten und in Quelltext zu übersetzen. Die Zielsystemunabhängikeit ist hierbei ein von Anfang an definiertes Ziel von PARSE. Damit soll das Problem, der „Programmierung in natürlicher Sprache“ allgemeingültig gelöst werden.

In Abbildung 1 ist die PARSE-Architektur dargestellt. In vorherigen Arbeiten wurden die grün dargestellten Module implementiert. Der Grund-

aufbau der Architektur besteht aus drei Strukturen. Zu Beginn wird in einem Vorverarbeitungsfließband die Spracheingabe durch einen Spracherkenner eingelesen und durch seichte Sprachverarbeitungswerkzeuge aufbereitet. Der für das Sprachverständnis zuständige Teil der Architektur basiert auf unabhängigen Agenten. Dabei dient ein Graph als zentrale Repräsentation der Spracheingabe und auch als Speicherstruktur für die Ergebnisse der verschiedenen Agenten. Der Graph wird inkrementell transformiert und mit neuen Informationen versehen. Zur Fehlerkorrektur lässt ein Dialogagent den Nutzer Fragen beantworten. In dieser Arbeit wird der rot markierte Teil der Architektur behandelt. In der Nachverarbeitung wird der Graph in lauffähigen Quelltext übersetzt. Die blau dargestellte Ontologie ermöglicht hierbei Zugriff auf Wissen über die Umgebung und Schnittstelle des Zielsystems. Der generierte Quelltext wird dann dem Zielsystem zur Verfügung gestellt. Dabei soll es möglich sein, dass dem Zielsystem neue Methoden beigebracht werden oder komplexe Anweisungen als Befehlsfolge zur Ausführung bereit gestellt werden. Im Rahmen dieser Arbeit sollen beispielhaft zwei Zielsysteme angesprochen werden. Als erstes System dient der Roboter *ARMAR-III* [ARA+06], welcher als Küchenhilfe fungiert. Ein Legoroboter soll als zweites System angesprochen werden. Er soll Hindernisparcours überwinden.

# 3 Zielsetzung

Ziel dieser Masterarbeit ist die Entwicklung eines Werkzeugs, welches die vollständig aufgebaute PARSE-Graphstruktur in Quelltext übersetzt. Der erzeugte Quelltext soll lauffähig, korrekt und vollständig sein. In vorherigen Arbeiten entwickelte Agenten annotieren den Graph mit zahlreichen Informationen. So wurden zum Beispiel bereits Aktionen aus einzelnen Satzteilen extrahiert. Danach wurden, mit Hilfe der Ontologie, diese Aktionen auf Methoden der Schnittstelle abgebildet. Auch existieren bereits weitere Annotationen, die beispielsweise Bedingungen und Ausführungsreihenfolgen markieren. Die Herausforderung liegt nun darin, die gesammelten Informationen des Graphen zusammenzusetzen und in Quelltext zu überführen. Dabei ist es unter anderem nötig sicherzustellen, dass die Variablenbenutzung konsistent ist und unvollständige Kontrollstrukturen vervollständigt werden. Weiterhin ist die softwaretechnische Erweiterbarkeit auf neue Eingabeinformationen und andere Zielsysteme von besonderer Bedeutung.

Beispiel

Die englische Anweisung „*Robo open the window but before that turn down the heating if it's on*“ soll in semantisch äquivalenten Quelltext übersetzt werden. Dazu nutzt PARSE eine Vielzahl von Agenten, die einen Graphen mit Informationen erzeugen. In dieser Arbeit wird nun ein Werkzeug entwickelt, welches aus einem gegebenen Graphen Quelltext generiert. Dazu müssen die Informationen des Graphen gesammelt und als Quelltext wiedergegeben werden. Als Zwischenrepräsentation wird aus dem Graph der Ablauf des Programms in Form eines abstrakten Syntaxbaum extrahiert. So ergibt im Beispielsatz die Kombination der Ausführungsreihenfolge mit der Bedingungserkennung die generelle Struktur des Syntaxbaums. Weiterhin müssen die gefundenen Methodenaufrufe der Schnittstelle an richtige Position eingefügt werden. Der fertige Syntaxbaum wird darauf folgend in konkreten Quelltext übersetzt.
Ein mögliches Ergebnis wäre:

```
if(robot.getACState().equals(ACState.ON))
    robot.deactivate(ac);
robot.open(window);
```

# 4 Vorgehen und Herausforderungen

Zuerst soll eine ausführliche Analyse der Problemstellung stattfinden. Die Aufgabenstellung lässt sich in folgende Teilprobleme zerlegen: Zuerst muss für das Zielsystem eine Ontologie erstellt werden. Sie enthält Informationen über die Umgebung und Schnittstelle des Systems. Darauf folgt das Abbilden der Befehlsbeschreibungen auf entsprechende Methoden der Schnittstelle über die Ontologie. Die Hauptaufgabe besteht darin den PARSE-Graphen auf Quelltext zu übersetzen. Über einen abstrakten Syntaxbaum als Zwischenrepräsentation kann die Aufgabe in zwei Teilschritte zerlegt werden. Im ersten Schritt wird der abstrakte Syntaxbaum aus dem PARSE-Graphen aufgebaut. Anschließen übersetzt man den erstellten Syntaxbaum in Quelltext.

In dieser Arbeit wird das erste Teilproblem der Erzeugung der Ontologie umgangen. Die Zielsysteme besitzen bereits vollständig erstellte Ontologien. Sollten weitere Zielsysteme eine neue Ontologie benötigen, so kann diese von Hand erstellt werden. Alternativ bietet sich ein halb-automatisches Verfahren an [LWT17]. Das zweite Teilproblem bezüglich der Ontologie ist die Abbildung der in der Eingabe erkannten Aktionen auf Methoden der Schnittstelle. Das Zuweisen passender Methoden zu Aktionsbeschreibungen wird in

einer anderer Arbeit erledigt.

Das direkte Übersetzen von PARSE-Graph in Quelltext führt dazu, dass viele Schritte für verschiedene Zielsysteme wiederholt werden müssen. Daher sollte mindestens eine Trennung zwischen der Aufbereitung des PARSE-Graphen und der Erzeugung von Quelltext stattfinden. So werden die einzelne Aufbereitungsschritte wiederverwendbar und die Quelltexterzeugung vom PARSE-Graphen separiert. Daher ist eine wichtige Herausforderung der Arbeit eine geeignete Zwischenrepräsentation zwischen Graph und Quelltext zu finden. Eine mögliche Form wäre ein abstrakter Syntaxbaum. Damit wird die Aufgabe der Quelltextübersetzung in zwei Schritte zerteilt. Der abstrakte Syntaxbaum soll im ersten Schritt aus dem PARSE-Graphen zusammengesetzt werden. Der Graph ist bereits in vorherigen Arbeiten durch Sprachverarbeitungswerkzeuge mit vielen Informationen angereichert worden. Eine Mustererkennung über die auf dem Eingabegraphen enthaltenen Informationen könnte dazu verwendet werden, um einen Syntaxbaum aufzubauen. Beim Aufbau des Syntaxbaums sollten verschiedene Punkte beachtet werden. Zum einen muss die semantische Korrektheit garantiert werden. Zum anderen ist es nötig sicherzustellen, dass der Syntaxbaum in lauffähigen Quelltext übersetzt werden kann. Der abstrakte Syntaxbaum sollte daher für die Übersetzung vollständig sein. Das Modul zur Quelltexterzeugung sollte nicht mehr auf den Graphen zugreifen müssen.

Der letzte Schritt ist die eigentliche Erzeugung des Quelltextes. Dazu soll der Syntaxbaum in Quelltext übersetzt werden. Hierfür gibt es bereits verschiedene Ansätze aus dem Bereich der Programmtransformation. Ein Anwendungsbereich dabei ist die Transpilierung, d.h. die Übersetzung von Quelltext einer Programmiersprache in den Quelltext einer anderen. Hierbei wird ebenfalls als Zwischenrepräsentation eine Form von Syntaxbaum verwendet. Dieser Baum wird in der Regel dann über das Besucher-Muster abgelaufen und in Quelltext übersetzt. Ein solcher Ansatz soll zur Lösung des Teilproblems betrachtet werden. Beim Entwurf des Quelltexterzeugers sollte beachtet werden, dass sich Zielsysteme in bestimmten Eigenschaften unterscheiden können. Eine solche Eigenschaft wäre zum Beispiel die Programmiersprache.

Eine softwaretechnische Herausforderung an den Entwurf ist die Variabilität der Eingabe. Der in PARSE erstellte Graph kann nachträglich, durch die Implementierung weiterer Agenten, um neue Informationen ergänzt werden. Da das in dieser Arbeit entwickelte Werkzeug Teil der Nachverarbeitung ist, ist es wichtig diese neuen Informationen leicht einpflegen zu können.

# 5 Evaluation

Für die Evaluation existieren Korpora für zwei Zielsysteme. Im Rahmen einer vorherigen Arbeit [Gü15] wurde für ARMAR-III ein Korpus mit verschiede-

ne Szenarien angelegt. Der Umfang des Korpus beträgt über 4000 Texte und aufgrund der Größe findet die eigentliche Evaluation der Arbeit auf diesem Korpus statt. Die Ontologie des ARMAR-III wurde von Hand aufgebaut und gepflegt. Für den Legoroboter wurden im Rahmen verschiedener Praktika Hindernisparcours und entsprechende Wegbeschreibungen entworfen. Dabei wurden auch entsprechende Ontologien halb-automatisch erstellt. Der Legoroboter dient dazu zu zeigen, dass das Ziel der Domänenunabhängigkeit erreicht wurde.

Bei der Evaluation des Werkzeuges gibt es einige Herausforderungen. Ideal wäre es die Module des Werkzeugs individuell zu evaluieren. Hierfür wären Korpora nötig, die aus einem Goldstandard für Eingabegraphen, Syntaxbaum und Quelltext bestehen. Der ARMAR-Korpus besitzt aber nur Eingabetexte ohne Muster- und Zwischenlösungen. Für die separate Auswertung der Module, müssten die Texte von Hand um Lösungen erweitert werden. Dies wäre mit hohem manuellem Aufwand verbunden und nur für Quelltext von einzelnen Szenarien durchführbar. Da das Werkzeug auf PARSE aufbaut, führt dies zusätzlich dazu, dass Folgefehler aus den vorherigen Arbeitsschritten übernommen werden. Daher kann das Werkzeug nur in Verbindung mit der gesamten PARSE-Rahmenarchitektur evaluiert werden. Da PARSE auf universelle Anwendbarkeit ausgelegt ist, bestünde die Möglichkeit der Verwendung öffentlicher Korpora mit Musterlösungen. Für die neuen Korpora müssten aber entsprechende Ontologien erzeugt werden. Falls die Korpora in anderen Arbeiten verwendet werden, würde dies auch die Vergleichbarkeit mit anderen Arbeiten verbessern.

Bei der Evaluation der Korrektheit gibt es eine weitere Herausforderung. Der generierte Quelltext kann vom Musterquelltext abweichen und trotzdem korrekt sein. Ein direkter Vergleich ist daher wenig sinnvoll. In einigen Forschungsarbeiten wird daher die BLEU-4 Metrik [PRWZ02], die zum Vergleich von Übersetzungen dient, verwendet. Diese Metrik hat aber den Nachteil, dass pro Szenario eine Vielzahl an Musterlösungen benötigt werden. Eine andere Möglichkeit wäre die Verwendung von Plagiatskontrollsoftware wie *jPlag*, die Quelltext auf Ähnlichkeit überprüft. Für einzelne Fälle könnte eine manuelle Analyse durchgeführt werden.

Eine Metrik, welche keine Musterlösung benötigt, wäre der Anteil an lauffähigem, erzeugten Quelltext. Hierbei soll anhand des großen ARMAR-Korpus überprüft werden, ob grundsätzlich kompilierbarer Quelltext entsteht. Eine weitere Metrik wäre die Abdeckung des Textes, d.h. wie viele Wörter des Eingabetextes wurden zur Erzeugung des Syntaxbaumes oder Quelltextes verwendet. Niedrigere Abdeckungswerte können darauf hinweisen, dass Teile der Anweisungen nicht beachtet wurden. Damit sollte ein genereller Überblick über die grundsätzliche Qualität des Werkzeuges gewonnen werden können.

## Literatur


[ARA+06] Asfour, T. ; Regenstein, K. ; Azad, P. ; Schroder, J. ; Bierbaum, A. ; Vahrenkamp, N. ; Dillmann, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, S. 169–175

[DGH+16] Desai, Aditya ; Gulwani, Sumit ; Hingorani, Vineet ; Jain, Nidhi ; Karkare, Amey ; Marron, Mark ; R, Sailesh ; Roy, Subhajit: Program Synthesis Using Natural Language. In: *Proceedings of the 38th International Conference on Software Engineering*. New York, NY, USA : ACM, 2016 (ICSE '16). – ISBN 978–1–4503–3900–1, S. 345–356

[Gü15] Günes, Zeynep: *Aufbau Eines Sprachkorpus Zur Programmierung Autonomer Roboter Mittels Natürlicher Sprache*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelor's Thesis, Mai 2015

[KTF] Khayrallah, Huda ; Trott, Sean ; Feldman, Jerome: Natural Language For Human Robot Interaction.

[LBK+02] Lauria, S. ; Bugmann, G. ; Kyriacou, T. ; Bos, J. ; Klein, E.: Converting Natural Language Route Instructions into Robot Executable Procedures. In: *11th IEEE International Workshop on Robot and Human Interactive Communication Proceedings*, 2002, S. 223–228

[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*. New York, NY, USA : ACM, 2013 (MobiSys '13). – ISBN 978–1–4503–1672–9, S. 193–206

[LV13] Lincoln, Nicholas K. ; Veres, Sandor M.: Natural Language Programming of Complex Robotic BDI Agents. In: *Journal of Intelligent & Robotic Systems* 71 (2013), August, Nr. 2, S. 211–230. `http://dx.doi.org/10.1007/s10846-012-9779-1`. – DOI 10.1007/s10846–012–9779–1. – ISSN 1573–0409

[LWP+] Lin, Xi V. ; Wang, Chenglong ; Pang, Deric ; Vu, Kevin ; Zettlemoyer, Luke ; Ernst, Michael D.: Program Synthesis from Natural Language Using Recurrent Neural Networks.

[LWT17] Landhäusser, Mathias ; Weigelt, Sebastian ; Tichy, Walter F.: NLCI: A Natural Language Command Interpreter. In: *Automated Software Engineering* 24 (2017), Dezember, Nr. 4, S. 839–861. http://dx.doi.org/10.1007/s10515-016-0202-1. – DOI 10.1007/s10515–016–0202–1. – ISSN 1573–7535

[MML+15] Mou, Lili ; Men, Rui ; Li, Ge ; Zhang, Lu ; Jin, Zhi: On End-to-End Program Generation from User Intention by Deep Neural Networks. In: *arXiv:1510.07211 [cs]* (2015), Oktober

[OMK18] Otter, Daniel W. ; Medina, Julian R. ; Kalita, Jugal K.: A Survey of the Usages of Deep Learning in Natural Language Processing. In: *arXiv:1807.10854 [cs]* (2018), Juli

[PJ17] Pulido-Prieto, Oscar ; Juárez-Martínez, Ulises: A Survey of Naturalistic Programming Technologies. In: *ACM Comput. Surv.* 50 (2017), September, Nr. 5, S. 70:1–70:35. http://dx.doi.org/10.1145/3109481. – DOI 10.1145/3109481. – ISSN 0360–0300

[PRWZ02] Papineni, Kishore ; Roukos, Salim ; Ward, Todd ; Zhu, Wei-Jing: BLEU: A Method for Automatic Evaluation of Machine Translation. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2002 (ACL '02), S. 311–318

[PRZH00] Price, David ; Rilofff, Ellen ; Zachary, Joseph ; Harvey, Brandon: NaturalJava: A Natural Language Interface for Programming in Java. In: *Proceedings of the 5th International Conference on Intelligent User Interfaces*. New York, NY, USA : ACM, 2000 (IUI '00). – ISBN 978–1–58113–134–5, S. 207–211

[WT15] Weigelt, S. ; Tichy, W.F.: Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language. In: *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference On* Bd. 2, 2015, S. 819–820