fachhochschule hamburg
FACHBEREICH ELEKTROTECHNIK UND INFORMATIK

Diplomarbeit

Chi Cuong Tat

Entwicklung und Aufbau einer Sprachsteuerungsschnittstelle für den Pioneer Roboter

Studiengang Softwaretechnik

Betreuender Prüfer: Prof. Dr. Christoph Klauck
Zweitgutachter: Prof. Dr. Kai von Luck
Abgegeben am 14.03.2002

Entwicklung und Aufbau einer Sprachsteuerungsschnittstelle für den Pioneer Roboter

Stichworte Sprachverarbeitung, Syntaxanalyse, DCG, Prolog/C++ Schnittstelle, Visual C++, Client/Server, Robotersteuerung

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Sprachverarbeitung. Dabei soll eine Sprachsteuerungsschnittstelle für den Pioneer Roboter an der Hamburger Hochschule für Angewandte Wissenschaft entwickelt werden. Das System wurde in der Programmiersprache C++ und Prolog entwickelt. Dadurch werden die Vorteile von C++ und Prolog miteinander verbunden. C++ als leistungsfähige objektorientierte Programmiersprache mit vielen Möglichkeiten der Datenbank- und Grafikunterstützung, und Prolog als bekanntes eingesetztes Regelwerk für die Syntaxverarbeitung. Dabei wird in Prolog der DCG Formalismus verwendet, um die Syntaxverarbeitung zu implementieren. Als C++ Bridge zur Prolog Maschine wurde die Prolog/C++ Schnittstelle von Jan Wielemaker verwendet. Das System ist als Client/Server System aufgebaut um von unterschiedlichen Räumen den Roboter steuern zu können.

Development of a speech control interface for the Pioneer robot

Keywords Speech processing, syntax analysis, DCG, Prolog/C++ interface, Visual C++, Client /Server, robot control

Abstract

This thesis is about speech processing. It shows a development of a speech control system that is deployed with a Pioneer robot at the university of applied sciences in Hamburg. The system is developed in the programming language c++ und prolog. The two languages join the advantages as follows. On the one hand c++ support a capable object oriented programming language with many possibilities for database connections and graphical user interfaces. And on the other hand prolog support a rule-based machine that is well known for syntax processing. The DCG formalism is used for implementing the syntax processing in prolog. A prolog/c++ interface from Jan Wielemaker is used for the c++ and prolog bridge. The system is developed as client/server architecture because the robot should controlled from different rooms.

INHALTSVERZEICHNIS

EINLEITUNG **1**
Ziel dieser Arbeit 1
ANALYSE **2**
1 EINFÜHRUNG IN DIE THEMATIK **2**
1.1 Sprachkorpus des Roboters 2
1.2 Grammatische Analyse des Sprachkorpus 3
1.2.1 Der Imperativsatz 4
1.2.2 Flektierbare und unflektierbare Wortarten 4
1.3 Spracherkennung 5
2 LÖSUNGSANSÄTZE **7**
2.1 Grammatiken und Parser 12
2.1.1 Sprachklassen und Grammatiken 12
2.1.2 Parser 15
2.2 Tiefen- vs. Breitenstrukturanalyse – Bewertung 22
3 ANFORDERUNGSANALYSE **23**
3.1 Verarbeitungskomponente 23
3.1.1 Vorverarbeitung 24
3.1.2 Syntax 24
3.1.3 Lexikon 25
3.1.4 Referenzauflösung (Mapping) 25
3.2 Verwaltungskomponente 27
DESIGN **29**
4 CLIENT/SERVER MODELL **29**
4.1 Client 30
4.2 Server 31
4.2.1 Verarbeitung 31
4.2.2 Verwaltung 33
5 DATENMODELL **36**
5.1 Schnittstellen 36
5.2 Verwaltung 37
5.3 Verarbeitung 39
IMPLEMENTIERUNG **43**
6 MFC UND PROLOG/C++ SCHNITTSTELLE **43**
6.1 Datenbankverbindung 43
6.2 Socket Verbindung 44
6.3 Lexikon 44
6.4 Parser 45
7 DCG PARSER MIT SWI PROLOG **47**
7.1 Syntax und Lexikon 47
7.2 Lexikon 48
8 PROGRAMMBESCHREIBUNG **49**
8.1 RSCServer 49

8.2 RSCClient 51
ZUSAMMENFASSUNG UND AUSBLICK 53
9 ZUSAMMENFASSUNG 53
10 AUSBLICK 54
GLOSSAR 57
ANHANG 59
Anhang A: Sprachkorpus 59
Anhang B: Korpus mit Konstituentenstruktur 61
Anhang C: Interface Datei zum Aktionsmodul 63
Anhang D: Mappingstruktur 64
LITERATURVERZEICHNIS 65

TABBELEN- UND ABBILDUNGSVERZEICHNIS

Tabellennummer *Seite*
Tab. 1: Chomsky Hierachie 12
Tab. 2: Tiefen- Vs. Breitenstrukturanalyse 22
Tab. 3: Eigenschaften der Eingabesätze 22
Tab. 5: Kommandoklassen 26

Abbildungsnummer *Seite*
Abb. 1: Ein einfacher Worthypothesegraph 6
Abb. 2: Einfache Architektur 7
Abb. 3: SPIDER Gesamtarchitektur 8
Abb. 4: Kombination flacher und tiefen Analyse in Verbmobil 10
Abb. 5: Architekturübersicht von Verbmobil 11
Abb. 8: DCG- und Prolog Regel 15
Abb. 9: Top Down Analyse 17
Abb. 10: Chart Algorithmus 19
Abb. 11: Chart als Graph 19
Abb. 12: Verarbeitungsleveln beim Chunk Parsing 21
Abb. 13: Klammerbaum 21
Abb. 14: Beispiele für einige Konstituentenstrukturen 25
Abb. 15: Anwendungsfalldiagramm 28
Abb. 16: Einsatzdiagramm 30
Abb. 17: Client GUI 31
Abb. 18: Einsatzdiagramm - Verarbeitung 33
Abb. 19: ERD 34
Abb. 20: Verwaltungskomponente 38
Abb. 21: Mappingkomponente 40
Abb. 22: Mappingbaum 41
Abb. 23: Sequenzdiagramm Verarbeitung 42
Abb. 24: RSCServer und Lexikon Maske 50
Abb. 25: Client - Server - Log 51

DANKSAGUNGEN

Für die Unterstützung dieser Arbeit möchte ich meinen herzlichen Dank aussprechen an meine Frau, die viel Verständnis für meine Abwesendheit gezeigt hatte und meiner Tochter, die in zwei Monaten das Licht der Welt erblicken wird, aber mir jetzt schon die Motivation verliehen hat diese Arbeit voranzutreiben.

Weiteren Dank für die Unterstützung der Arbeit gilt meinen beiden Betreuern Prof. Dr. Christoph Klauck und Prof. Dr. Kai von Luck, die mir durch Rat und Tat bei Seite standen. Einen herzlichen Dank gilt auch dem Prof. Dr. Bernd Bernd Owsnicki-Klewe, der, obwohl sichtlich erschöpft von seiner Reise aus den USA, mir dennoch in langen Gesprächen gute Ratschläge vermittelte.

Ferner möchte ich allen Kommilitonen danken, die durch fruchtbare Gespräche die Arbeit vorangetrieben haben.

ABKÜRZUNGSVERZEICHNIS

ADJ	Adjektiv
ADP	Adverbial Phrase
ADV	Adverb
APP	Adverbial Präpositionalphrase
DCG	Definit Clause Grammar
DET	Determiner
GP	Größen Phrase (für Kardinalzahlen und physikalische Einheit cm, grad, etc.)
HPSG	Head driven Phrase Structure Grammar
N	Nomen
NP	Nominalphrase
OPP	Objekt Präpositionalphrase
P	Präposition
PP	Präpositionalphrase
PRO	Pronomen
PS	Phrasenstruktur
PSG	Phrasenstrukturgrammatik
SAPI	Speech Application Programming Interface
SDK	Software Development Kit
SR	Sonderregel
TTS	Text To Speech
V	Verb
VP	Verbalphrase
MFC	Microsoft Fondation Classes

Einleitung

Ziel dieser Arbeit

Die Mensch-Maschine-Kommunikation beschränkt sich heutzutage weitestgehend auf das Lesen der Aktionsmöglichkeiten (Desktop Anwendung) des Systems und dem Klicken auf die Maustaste, um die Aktion auszuführen. Dieses Verhalten von Kommunikation ist nicht nur unergonomisch, sondern auch untypisch für den Menschen. Wenn wir unser Anliegen einem anderen Menschen äußern, so geschieht dies über die Sprache. Diese natürliche Kommunikationsform wünscht man sich auch auf die Mensch-Maschine-Kommunikation zu übertragen. Man denke dabei an Menschen, die durch körperliche Lähmung nicht mehr imstande sind Peripherien (Maus, Tastatur, Joystick) zu bedienen aber dennoch sprechen können. Diese Menschen könnten durch natürliche Sprache Maschinen (Roboter oder Computer) bedienen und wären so einigermaßen unabhängig. Für solch ein Szenario forscht ein Team am Institut für Automatisierungstechnik (IAT) in Bremen[FRIEND]. Eine ähnliche Anwendung ist der NURSEBOT von der University of Pittsburgh und Carnegie Mellon University [NURSEBOT]. Aber auch als Museumsführer, welcher durch die natürliche Sprache des Besuchers instruiert wird, bestimmte Artefakte anzufahren und darüber zu erzählen[MINERVA].

In dieser Diplomarbeit soll eine Sprachsteuerungsschnittstelle entworfen werden. Die Sprachkommandos sind in Textform einzugeben und werden in semantische Informationen transformiert. Da der Roboter nur eingeschränkte Aktionen bietet, ist ein Vokabular zu definieren, das der Roboter auch interpretieren kann. Der Sprachumfang ist auf folgende Aktionen beschränkt:

- vor- und rückwärts Kommandos in unterschiedlicher Variante
- Parametrisiertes vor- und rückwärts fahren ((...) bis Wand, (...) in das Zimmer)
- Richtungsänderungen
- Geschwindigkeitsänderungen
- Objekte transferieren

Dieses Vokabular und die dazugehörigen Regeln sollten später möglichst einfach erweiterbar sein, um das zukünftige Aktionsspektrum des Roboters erhöhen. Das System ist hinsichtlich der Wartbarkeit und Erweiterbarkeit modular und komponentenbasiert aufzubauen.

ANALYSE

In diesem Kapitel soll eine Einführung in die Thematik und Problematik im Zusammenhang mit der oben grob beschriebenen Aufgabenstellung vorgestellt werden. Die Beschreibungen einiger Sprachverarbeitungsprojekte sollen dem Leser erste Lösungsansätze und Verfahren vermitteln. Anschließend werden die Verfahren und die Anforderungen für das Ziel der Diplomarbeit gegeneinander abgewogen, um für die Designphase entscheiden zu können, wie und welche Komponenten entwickelt werden.

1 Einführung in die Thematik

Im folgenden soll eine Einführung in die Linguistik gegeben werden. Dabei sollen einige Begrifflichkeiten erklärt und die Problematiken, die mit der Sprachverarbeitung verbunden sind, aufgezeigt werden.

1.1 Sprachkorpus des Roboters

Bevor ein Sprachverarbeitungssystem entwickelt wird, muss vorher Klarheit darüber geschaffen werden, wie die Sprache bzw. Sätze auszusehen haben und welche semantische Kategorien diese Sätze haben sollen. In Absprache mit Herrn Prof. Kai von Luck soll zunächst der Roboter auf Sprachbefehle reagieren, die im Anhang A aufgelistet sind. Der Korpus wird in 6 Kategorien aufgeteilt:

Kategorie 1:
Diese Kategorie umfasst jene Kommandos, die dem Roboter zum Bewegen nach vorn und nach hinten veranlassen. Wird einer dieser Kommandos erkannt, soll sich der Roboter nach vorne oder nach hinten bewegen, bis dieser durch eine Wand gestoppt wird. Stünde dieser vor einer Wand oder einer geschlossenen Tür so wird er seine Aktion nicht ausführen können und weiterhin stehen bleiben. Werden hintereinander die Sätze nach vorne und hinten eingegeben, so wird die erste Aktion gestoppt und die zweite Aktion eingeleitet. Zusätzlich dürfen Adjektive wie *schnell, langsam* oder *normal* die Fahrgeschwindigkeit beeinflussen.

Kategorie 2:

Kategorie 2 erlaubt dem Roboter zu einem bestimmten Ziel zu fahren. Dieses Ziel kann eine beliebige Wand, ein Raum, eine Tür oder auch eine bestimmte Weglänge sein. Ferner soll die Fahrgeschwindigkeit (wie auch in Kategorie 1) durch Adjektive wie *schnell, langsam* oder *normal* zu dem Kommandosatz ergänzt werden können.

Kategorie 3:

Die dritte Kategorie erlaubt es die Richtung des Roboters zu verändern. Der Roboter befindet sich in einem ruhenden Zustand oder wird in diesen Zustand versetzt und wird erst dann nach links oder nach rechts drehen. Der Benutzer kann absolute Grad Winkeln vorgeben oder auch einfach den Roboter nach links oder rechts drehen lassen. Um die Drehung zu beenden, kann der Benutzer die Kommandos aus Kategorie 4 eintippen.

Kategorie4:

Kategorie 4 erlaubt eine aktuelle Aktion zu beeinflussen. Dabei kann der Benutzer die Aktion abbrechen oder die Geschwindigkeit relativ erhöhen bzw. reduzieren.

Kategorie 5:

Diese Kategorie wird zunächst für die Diplomarbeit von sekundärer Bedeutung sein, da der Roboter noch nicht über ein verlässliches Objekterkennungssystem verfügt. Gleichwohl soll dieser Anspruch des Objekttransfers für die Zukunft in der Architektur dieses Systems berücksichtigt werden.

Kategorie 6:

Kategorie 6 erlaubt dem Benutzer den Greifer des Roboters zu steuern. Allerdings setzt dies voraus, dass der Roboter über einen Greifer verfügt. Diese Kommandos beziehen sich auf Aktionen wie Greifer schließen, öffnen, heben und senken.

1.2 Grammatische Analyse des Sprachkorpus

In diesem Unterkapitel sollen die relevanten grammatischen Regeln erläutert werden, die für spätere Ausführung von Bedeutung sind. Es soll nicht versucht werden, die vielen Details der Grammatik im Korpus zu erläutern. Dafür ist der Duden der deutschen Grammatik [DUDGRAM] die passende Lektüre.

1.2.1 Der Imperativsatz

Wenn man sich den Korpus aus Anhang A ansieht, erkennt man, dass der Korpus hauptsächlich aus Befehlssätzen besteht.

Der Aufforderungssatz kennzeichnet sich [DUDGRAM] durch die Stellung des Verbs an erster stelle des Satzes. Dabei steht das Verb in der Singular/Imperativ oder Plural/Imperativ Form. Die Verwendung der imperativen Verbform schließt immer eine Hinwendung zu einem Gegenüber ein. In diesem Falle ist immer ein Roboter gemeint, wenn ein Imperativsatz in die Maske eingegeben wird. Bei Imperativsätzen treten charakteristische Partikel auf wie *eben, mal, bitte mal, einfach, halt, etc.* Diese Partikel sind jedoch aus referenzsemantischen Aspekt, also zur Ausführung eines Satzkommandos, nicht relevant. Sätze wie „*fahre mal nach vorne*" oder „*fahre doch mal bitte rückwärts*", sind aus steuerungstechnischer Sicht eines Roboters gleichbedeutend. Ein Imperativsatz kann aus einer Verb-, Substantiv-, Präpositions- und Adverbgruppe bestehen. Die Verbgruppe besteht aus einem Verb im Imperativmodus/2. Singular und optional aus einem Adjektiv in der Form Positiv. Die Substantivgruppe kann aus einem einzelnen Substantiv oder zusammen mit einem Artikel auftreten. Dabei müssen die beiden Wörter in Kasus, Genus und Numerus übereinstimmen (Kongruenz). Die Präpositionsgruppe besteht aus einer Präposition und einer Adverbgruppe oder einer Substantivgruppe. Die Präposition legt bei einer gefolgten Substantivgruppe den Kasus für diese Gruppe fest. Also für den Satz:„*fahre schnell* ***zu der Wand***", ist die Substantivgruppe („*der Wand*") im Kasus Dativ. Folgt nach einer Präposition ein Adverb ist keine Änderung am Adverb nötig, da Adverbien keine andere Formen besitzen. Eine Adverbgruppe kann nach einem Verb folgen oder wie beschrieben nach einer Präposition („*fahre* **[*vorwärts*]/[gerade aus]** *"*, „*fahre schnell nach* **[*vorne*]**"). Sie kennzeichnen meist eine Richtungsangabe(„*vorwärts, rückwärts, oben,unten*").

Die Form eines Aufforderungssatzes kann in einem Fragesatz umgewandelt werden wie zum Beispiel *„würdest du bitte nach vorne fahren?"*. Generell meint der Aufforderungssatz ein „ Ich will, dass du... tust". Dahinter kann im konkreten Fall eine Aufforderung, ein Vorschlag oder ein Fragesatz stecken. Diese Diplomarbeit befasst sich nur mit den Imperativsätzen.

1.2.2 Flektierbare und unflektierbare Wortarten

Unabhängig vom Korpus, lassen sich alle Wörter in sogenannte Wortarten einteilen bzw. zuordnen. Diese sind Nomen (*Wand, Büro, Zimmer etc.*), Verben (*laufe, gehe, fahre, etc.*), Adjektive (*schnell, langsam*), Präpositionen (*nach, bis, in, etc.*), Adverbien (*vorne, hinten, links, rechts etc.*) Artikel (*der, die, das, den, etc.*), Pronomen (*du, dich, sich*).

Aus der Schulgrammatik wissen wir, dass zwei Gruppen von Wortarten existieren. Die Gruppe der flektierbaren Wortarten (**Nomen, Adjektive, Verben, Artikel und Pronomen**) und die Gruppe der unflektierbaren Wortarten (**Adverb, Partikel, Präposition** und **Konjunktion**, die ich hier nicht behandeln werde).

Flektierbare Wortarten:
Die flektierbaren Wortarten [DUDGRAM] Artikel, Substantiv, Pronomen und Adjektiv werden bestimmt durch das **Genus, Kasus und Numerus**. Dabei spielt der Numerus in diesem Korpus eine untergeordnete Rolle, da immer nur ein Roboter gemeint ist. Im Korpus treten nur singuläre Formen auf, so dass eine Unterscheidung zwischen Singular und Plural bzw. in Personenzahl nicht notwendig ist. Bei Adjektiven kommt zusätzlich eine grammatische Kategorie hinzu, nämlich die **Vergleichsform** (siehe Anhang A, Satznummer 31-34). Das Verb hingegen wird durch **Tempus, Genus Verbi (Aktiv und Passiv), Modus (Indikativ, Konjunktiv und Imperativ), Person und Numerus** [DUDGRAM] bestimmt. Da das Verb im Satz (bezogen auf den Korpus) implizit eine Aufforderung zu einer Aktion für den Roboter enthält und der Benutzer indirekt mit dem Roboter kommuniziert, sind nur der Modus und die Person relevant. Alle anderen Modi werden in dieser Arbeit nicht berücksichtigt. Für die Arbeit ist also der Modus Singular/Imperativ und die Person ist 2. Form Singular.

Unflektierbare Wortarten:
Die unflektierbaren Wortarten [DUDGRA] Adverbien, Partikel, und Präpositionen werden unverändert verwendet. Adverbien werden im Korpus verwendet als Beiwort zum Verb. „*Fahre vorwärts*“ oder „*fahre nach vorne*“ sind typische Beispiele für die Verwendung des Adverbs als Beiwort, weil das Adverb sich auf das Verb bezieht. Die Partikel (*eben, mal, bitte mal, einfach, halt,*) spielen keine Rolle in der Interpretation des Eingabesatzes. Wenn man sich den Korpus genauer anschaut, stellt man fest, dass Präpositionen einen großen Wortanteil haben. Neben dieser quantitativen Auffälligkeit haben Präpositionen eine weitere Eigenschaft. Sie bestimmen den Genus des Substantivs bzw. Artikel und Substantiv (z. B.: „*Fahre in das Zimmer*“, „*fahre zu der Wand*“). Steht allerdings ein Adverb hinter einer Präposition, so ist der Genus natürlich bedeutungslos, da Adverbien nicht flektierbar sind. Zum Beispiel der Kommandosatz „*fahre nach vorne*“.

1.3 Spracherkennung

Da der Schwerpunkt dieser Arbeit nicht in der Spracherkennung liegt, aber dieses Thema an dieser Arbeit angeknüpft werden kann (Ausbau des Prototyps mit einer Spracherkennungssoftware), möchte ich in diesem Kapitel einige wichtige Punkte zur Spracherkennung anmerken.

Spracherkennung bedeutet, ein akustisches Signal in einen oder mehrere Wörter umzuwandeln. Dabei treten beim Umwandlungsprozess mehrere Probleme auf[SSAHLT1], [WAHLSTERVM] :

- **Umgebungsrauschen:** Durch die Umgebungsgeräusche wird die Spracherkennung negativ beeinflusst. Deshalb ist zu beachten, dass die Umgebungsgeräusche so niedrig wie möglich gehalten werden.
- **Performanzphänomene:** Bei gesprochener Sprache werden häufig inhaltslose Wörter benutzt wie: „ähm“, „äh“ etc. Aber auch Räuspern, kurze Pausen, Satzabbrüche und Korrekturen können den Erkennungsprozess verschlechtern.
- **Sprecherabhängigkeit:** Noch immer sind einige Spracherkennungssysteme vom Sprecher abhängig, da die Software durch eine Trainingsphase an den Sprecher angepasst werden muss. Die Erkennungsrate wäre bei einer anderen Person, welche die Software bedienen würde, stark reduziert.

Alle diese geschilderten Probleme führen zu einer Vagheit oder unsicheren Aussage über die erkannten Wörter.

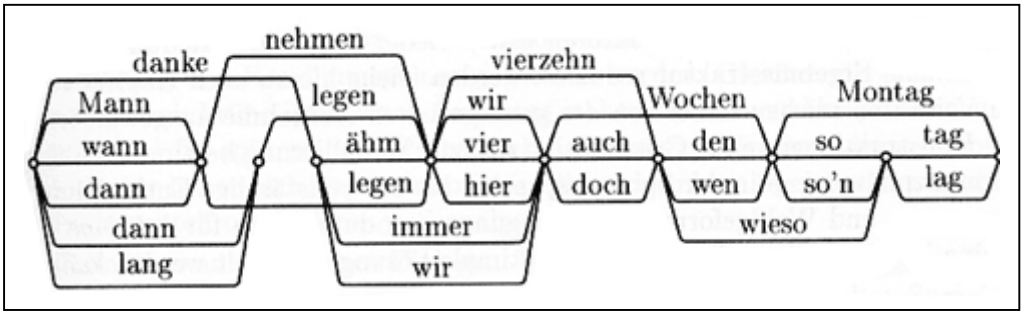


Abb. 1: Ein einfacher Worthypothesegraph

Am Beispiel des Satzes[HKI S.653]: „*Dann nehmen wir doch den Montag*“, sieht man auf Abbildung 1 wie die Erkennung ausschauen kann. Für diesen Beispielsatz gibt es unterschiedliche Wörter, die zu diesem Satz erkannt worden sind. Diese erkannten Wörter tragen eine statistische Zahl, die angibt wie hoch die Erkennungsrate zu dem gesprochenen Wort ist. Durch die oben beschriebenen Probleme der Spracherkennung kann die Software also nur eine wahrscheinliche Aussage darüber machen, welche Wörter vermeintlich erkannt werden. Ein sogenannter Worthypothesegraph wird verwendet, um eine Sequenz von wahrscheinlich erkannten Wörtern darzustellen.

Im Internet gibt es einige namhafte Anbieter von Spracherkennungssoftware. Einige davon bieten kostenlose SDK inklusive die Spracherkennungssoftware [PHILLIPS]/[DRAGON] und andere erlauben die SDK nur unter Einsatz einer bereits gekauften und installierten

Erkennungssoftware[IBM]. Alle diese drei Softwarepakete sind über die Microsoft Speech Application Programming Interface (SAPI) steuerbar. Leider enthält nur [PHILLIPS], von den drei Anbietern, Beispielprogramme, damit man die Spracherkennungssoftware ausprobieren kann. Die Qualität der Spracherkennung [PHILLIPS] lässt doch sehr zu wünschen übrig. Im übrigen gibt es nur die englische Sprache zum Testen des Spracherkenners.

2 Lösungsansätze

Pragmatisch betrachtet, ist die Aufgabe dieser Diplomarbeit, eine Menge von zusammengesetzten Zeichen zu analysieren und zu interpretieren. Abbildung 2 zeigt schematisch, wie die grobe Architektur eines solchen Systems aussehen kann. Auf dem Gebiet der Sprachverarbeitung existieren hauptsächlich zwei Lösungsansätze zur Verarbeitung von Sprache. Im Rahmen dieser Diplomarbeit werden diese zwei Lösungsansätze vorgestellt und ausführlich diskutiert. Diese Lösungsansätze sind Tiefenanalyse und Breitenanalyse. Um die Genauigkeit der Sprachanalyse zu erhöhen, werden auch die beiden Analyseformen zusammen in einem System verwendet (Verbmobil). Auf Basis dieser Überlegung wird die Entscheidung für die Entwicklung eines dieser Lösungsansätze fortgeführt.

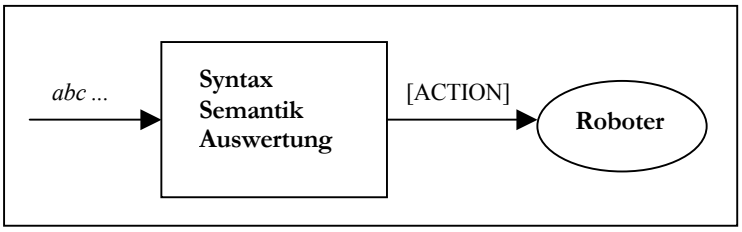


Abb. 2: Einfache Architektur

BotSpeak von ActivMedia[1]:

Bevor ich mich auf die Literatur gestürzt habe, habe ich mich über E-Mail an den Hersteller des Roboters gewandt, um zu erfahren, ob nicht vielleicht von Seiten ihrer in Richtung Sprachverarbeitung bereits etwas getan wurde. Doch bedauerlicherweise sagten sie mir, dass die Software BotSpeak zwar über ViaVoice[2] Spracherkennung verfügt und eine Integration in die Saphira[3] Umgebung einfach wäre, doch deren Leistungsfähigkeit sehr beschränkt sei. Es fehlt eine Grammatikprüfung und eine Möglichkeit das Vokabular, welches in Englisch ist, zu erweitern. Die Sprachverarbeitung funktioniert sehr rudimentär, in der Weise, dass statische Phrasen wie: „*hello*

[1] Hersteller des Pioneer Roboters ist ActivMedia

[2] Ein Spracherkennungssoftware Produkt der Firma IBM

[3] Spahira ist ein Simulationssystem für die Pioneer Roboter Modelle der Firma ActivMedia

there" oder „*greetings robot*", passende Aussprachemuster gesucht werden. Ferner haben Benutzer dieser Software bemängelt, dass die Erkennungsrate schon bei drei Wörtern erheblich schlecht wird. Neben der Spracherkennungsfunktion kann die Software auch Sprache synthetisieren (Text To Speech, kurz TTS). Man kann vage verstehen, was gesprochen wird, und doch klingt dies noch sehr mechanisch.

SPIDER [SPIDER]:

SPIDER ist ein Projekt an der Universität Hamburg, Fachbereich Informatik. Das Ziel dieses Projektes, das 1998 begann, war es einige Merkmale von Dialogen zwischen Menschen nachbilden zu wollen. Durch natürliche Sprache soll mit einem mobilen Roboter kommuniziert werden. Dabei beschränkt sich die Anwendung auf Aktionen wie das Fahren oder das Holen von Gegenständen. Die Anwendung lief nicht real mit einem mobilen Roboter, sondern auf einer Simulation.

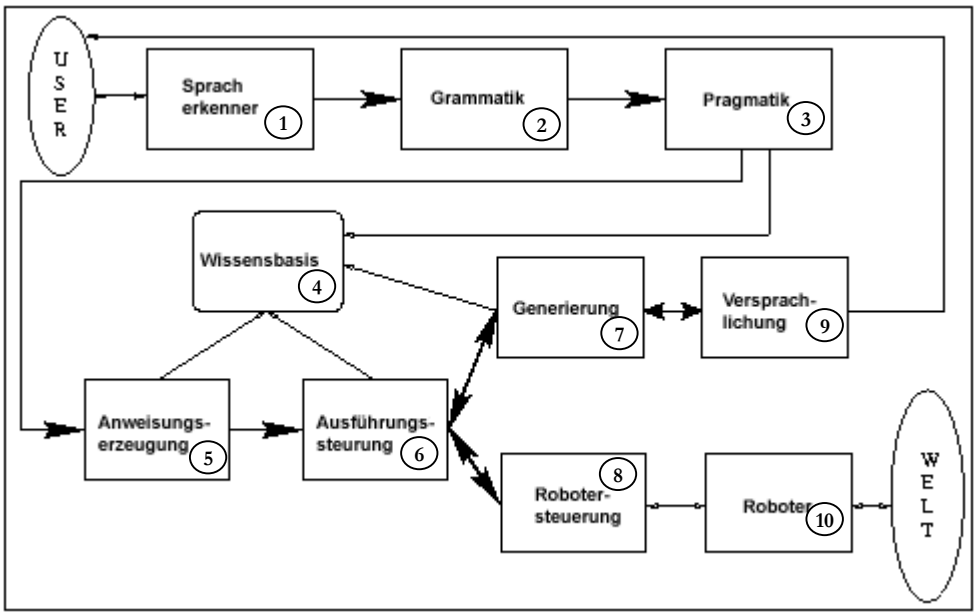


Abb. 3: SPIDER Gesamtarchitektur

Auf Abbildung 3 wird die Architektur des Systems dargestellt. Das System selbst besteht hauptsächlich aus der **Sprachverarbeitungs**-, **Planungs**- und **Robotersteuerungskomponente**. Der Verarbeitungsprozess gliedert sich in mehrere Phasen auf [SPIDER1]:

- Aufnahme des Sprachsignals (1)
- Analyse des Sprachsignals.(1)
- Parsing der erkannten Worte, Aufbau einer semantischen Repräsentation der Eingabe. (2)
- Interpretation der semantischen Repräsentation hinsichtlich der Sprecherintention. (3)(4)

- Erzeugung eines Planes, der im Endzustand der bzw. dem Ziel des Sprechers entspricht.(4)(5)(6)
- Schrittweise Ausführung des Planes.(6) (7) (8)

Da in dieser Diplomarbeit der Aspekt der textuellen Sprachverarbeitung im Vordergrund steht, ist nur Parsing und Interpretation der semantischen Repräsentation interessant (Abb. 2: Grammatik und Pragmatik Block).

Die grammatische Korrektheit der Eingabe wird durch einen Chart Parser gewährleistet. Dabei benutzt das SPIDER System ein Parser der HPSG[4] artige Grammatiken versteht.

> *„Die syntaktische Verarbeitung arbeitet nach dem Prinzip der Absorption. Jedes modellierte Inhaltswort besitzt eine Liste von Argumenten, die es benötigt, um zur ganzen Phrase zu werden. Diese Information ist direkt in den Lexikoneinträgen enthalten."* [SPIDER1]

Im Beispielsatz: „*Fahre in Kais Zimmer*", benötigen die ersten drei Wörter je ein Komplement. ‚*Fahre*' braucht eine Präposition PP, die semantisch eine Richtungsangabe ist. ‚*in*' verlangt eine rechts stehende Nominal Phrase (NP) in Akkusativ. ‚*Kais*' benötigt eine rechts stehende NP. Diese Form der Syntaxanalyse ist sehr auf Korrektheit bedacht, was zu folgender Konsequenz führt:

> *Unangemessen ist dagegen die äußerst strikte abstrakte Syntaxmodellierung. Eine Analyse kann nur gelingen, wenn sich alle beteiligten Wortformen in eine Gesamtstruktur unifizieren lassen. (...) Das ist genau das Gegenteil der Robustheit: Bei der kleinsten Unstimmigkeit schlägt die Analyse ohne Ergebnis fehl.* [SPIDER1]

Die Grammatikkomponente erzeugt bei dem Satz: „*Fahre in Kais Zimmer*" folgende semantische Struktur:

```
order((auth(user)
        prop( move( loc ( location (dest ( room( def(b)
                                                owner(kai))))))))
```

Diese Struktur wird der Pragmatikkomponente übergeben und dieser wiederum ermittelt durch sogenannte Schemata die Intention des Sprechers. Das Schema zu dem Satz: „*fahre zu Kai*", verlangt die Merkmale *action* (`move`), *destination* (`number`), *auth* (`speaker`), *adressee* (`robot`). Die Referenzauflösung findet durch eine Tabelle statt (z. B. `Jan` auf eine Raumnummer).

[4] siehe Kapitel Grammatiken und Parser

Verbmobil[VERBMOBIL]:

`Verbmobil` ist eines der ehrgeizigsten Sprachtechnologieprojekte in Deutschland, in der sowohl nationale als auch internationale Hochschulen und Industrien zusammengearbeitet haben. Das Ziel dieses siebenjährigen Projektes (1993-2000) war, ein sprecherunabhängiges Übersetzungssystem für spontansprachliche Verhandlungsdialoge zu entwickeln. Die Übersetzung war anfangs in den Sprachen Deutsch und Englisch und wurde später mit Japanisch erweitert. Durch die Kombination von Flachen- und Tiefeanalyse[5] kann `Verbmobil` mit einer korrekten Übersetzungsrate von mehr als 70% aufwarten, was eine Spitzenposition im internationalen Vergleich bedeutet. So kann `Verbmobil` folgenden nicht grammatisch, korrekten Satz: „*Ja, ich weil also würde mal äh vorschlagen, wir könnten uns am äh 7. treffen so im Mai*", interpretieren und ins Englische mit: „*How about the seventh of May*" übersetzen[WAHLSTERVM].

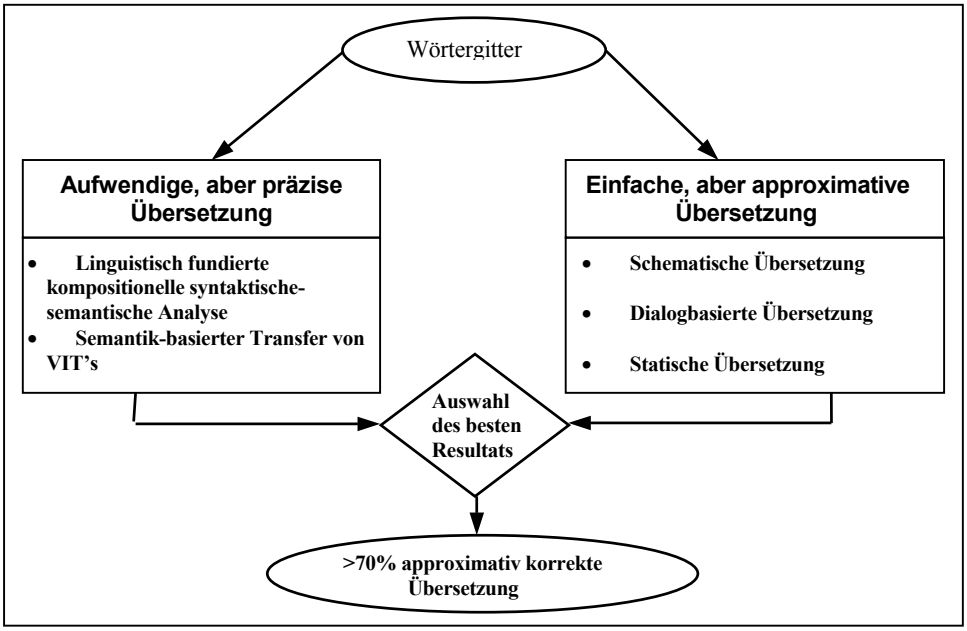


Abb. 4: Kombination flacher und tiefen Analyse in Verbmobil

Abbildung 4 zeigt den prinzipiellen Übersetzungsvorgang von `Verbmobil`. Das Spracherkennungsmodul liefert einen Worthypothesegraphen (Wortgitter), welches dem flachen und tiefen Analysemodul zur weiteren Verarbeitung übergibt. Die Tiefenanalyse verwendet eine HPSG artige Grammatik, wie auch im SPIDER Projekt und führt eine fundierte linguistische Analyse durch. Die Breitenanalyse verwendet eine sogenannte Chunk-Technik[6], die nicht sehr auf grammatische Korrektheit ausgerichtet ist.

[5] flache und tiefen Analyse siehe Kapitel Tiefen und Breitenanalyse

[6] siehe Kapitel Parser

Schon zu Beginn des Sprechens versucht `Verbmobil` durch die ersten Worte einen Dialogakt[7] festzustellen. Dadurch kann es Vorhersagen über das Gespräch treffen und somit die zu erwartenden Schlüsselworte einengen oder anders gesagt den Suchraum für die Schlüsselworte verkleinern. `Verbmobil` verwendet als einziges Sprachsystem eine Prosodiekomponente[8] der es ihm erlaubt durch Betonung des Sprechers die Interpunktion zu bestimmen und Aussagen, Zustimmung, Fragen oder andere Ausdrucksarten zu unterscheiden. Mit dieser Prosodiekomponente und eine kombinierte flachen und tiefen Analyse kann es an Fehlertoleranz und Robustheit in der Spracherkennung und Übersetzung gewinnen.

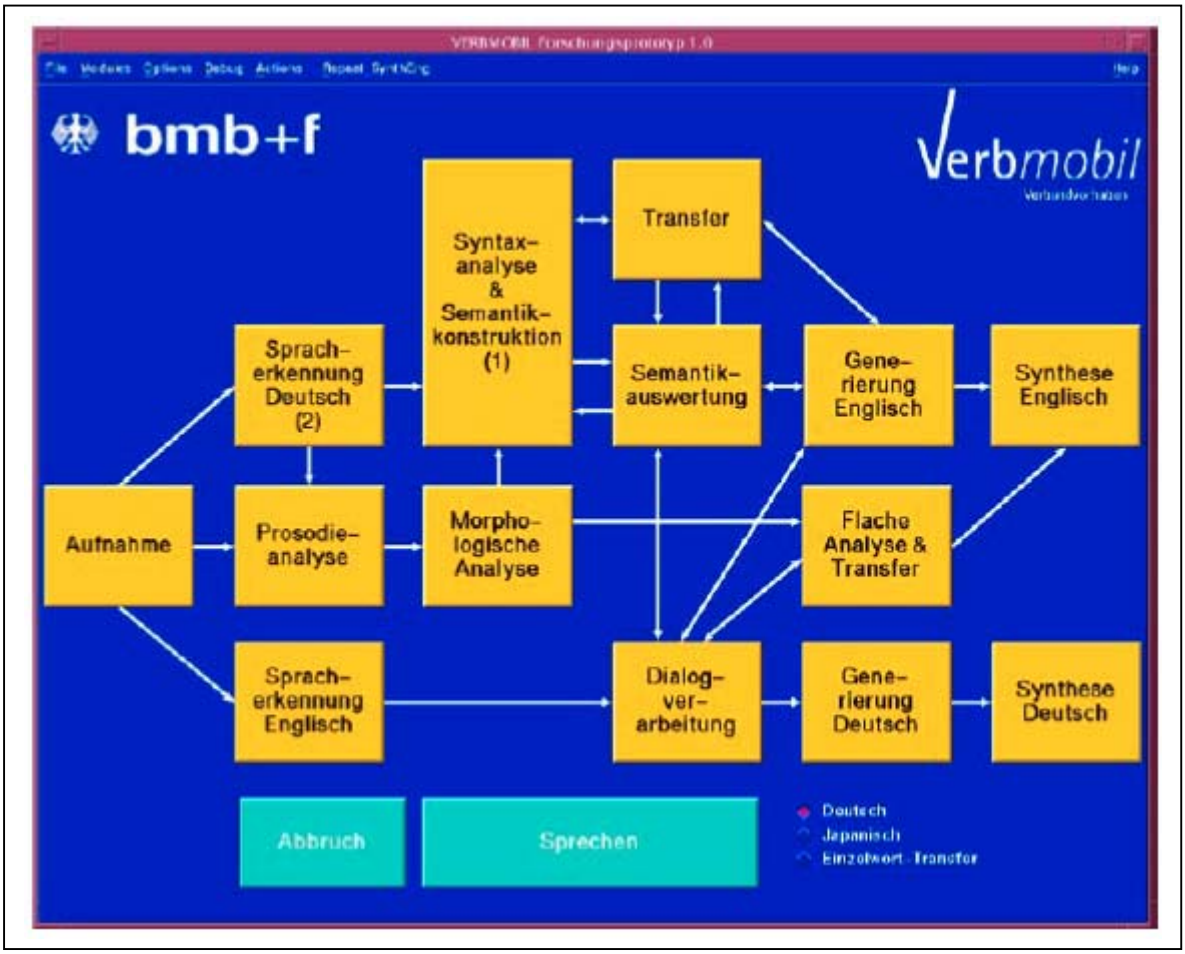


Abb. 5: Architekturübersicht von Verbmobil

Abbildung 5 zeigt das hochgradig nebenläufige System mit zahlreichen Kommunikationsstellen zwischen den Verarbeitungsmodulen. Sie ist aber auch gleichzeitig die Benutzungsoberfläche von `Vermobil`. Neben den Hauptmodulen, die man auf der Abbildung sieht, gibt es insgesamt dreiundvierzig Systemkomponenten.

---

[7] thematische Bezogenheit, Kontext

[8] siehe Glossar unter Prosodie

2.1 Grammatiken und Parser

Nachdem nun ein Überblick über einige interessante Projekte im Bezug auf diese Diplomarbeit gegeben wurde, möchte ich auf einige Begriffe und Techniken, die im vorigen Abschnitt erwähnt wurden, näher eingehen. Insbesondere sind Fragen zu klären wie: Zu welcher Sprachklasse kann man die natürliche Sprache einordnen? Welche Beschreibungsformalismen existieren dafür? Mit welcher Technologie kann die Spracheingabe verifiziert und in seine Struktur dargestellt werden? Welche konkrete Technik ist im Hinblick auf die Arbeit interessant?

2.1.1 Sprachklassen und Grammatiken

Im Nachfolgendem Kapitel soll gezeigt werden, in welcher Komplexitätsklasse Sprachverarbeitung einzuordnen ist. Dies ermöglicht eine adäquate Wahl eines Parsers, der mit einer Sprachklasse/Grammatik verknüpft ist.

In den 50er Jahren hat N. Chomsky die Chomsky-Grammatik oder auch Phrasenstrukturgrammatik (PSG) eingeführt ([ASBSSH], [GCRHAUSSER]). Mit dieser Grammatik hat er vier Typen von formalen Sprachen definiert, wobei die meisten Computersprachen in einem dieser Grammatiktypen gehört.

Grammatik	Auch bezeichnet als:	Komplexitätsgrad
Typ-3	Reguläre Sprachen	Linear - O(n/2n/3n/etc.)
Typ-2	Kontextfreie Sprachen	Polynominal - O(n^2/n^3/n^4/etc.)
Typ-1	Kontextsensitive Sprachen	Exponentiell - O(2^n/3^n/4^n/etc.)
Typ-0	Unbeschränkte Sprachen	Unentscheidbar - O(∞)

Tab. 1: Chomsky Hierachie

In der Tabelle 1 erkennt man, dass Grammatik Typ-3 und Typ-2 in der Informatikpraxis noch beherrschbar sind, wohingegen bei Typ-1 und Typ-0 Sprachen mit steigendem Bedarf an Grammatikregeln der Rechen- und Speicheraufwand enorm anwächst. In der Praxis allerdings verwendet man kontextfreie Grammatiken für natürliche Sprachen, weil sie folgende wichtige Eigenschaften besitzen[NLP S.17]:

- *Kontextfreie Sprachen sind mächtig genug, um die syntaktische Struktur natürlicher Sprachen fast vollständig zu beschreiben.*
- *Sie sind einfach zu entwickeln und zu modifizieren.*
- *Sie erlauben die Entwicklung effizienter Parsingalgorithmen.*

Auch sagt [GCRHAUSSER S.182], dass eine grammatische Analyse der natürlichen Sprache mindestens der Komplexität n^3 haben muss, was ein Ausschluss der Typ-3 (lineare Komplexität) bedingt.

Eine Grammatik für den Satz: „*Fahre in den Raum*“, sähe folgendermaßen aus:

V_T = {Fahre, in, den, Raum}. Terminale Symbole
V_{NT}= {VP,V,PP,P,NP,DET,N} Nicht Terminale Symbole
P ={S → VP, Produktion/Regel
VP → V,
PP → P NP,
NP → DET N,
DET → den,
N → Raum,
P → in,
V → Fahre}

VP: Verbalphrase, V: Verb,
PP: Präpositionsphrase, P: Präposition
NP: Nomnilaphrase, N:Nomen
DET: Determiner

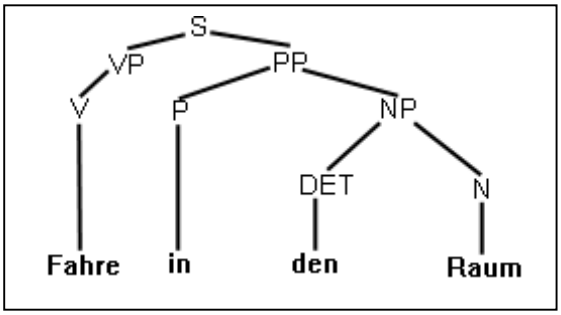


Abb. 6: Einfache Konstituentenstruktur

Diese nichtterminalen Symbole (S, VP, PP, NP etc.), die im Compilerbau verwendet werden, werden in der Computerlinguistik auch als Konstituenten bezeichnet. Die Idee, die auf dieser Konstituentenstruktur(siehe Abbildung 6) aufbaut ist [SNS] „*im wesentlichen ein Teil Ganzes Relation, wobei Linearität, d.h. sequentielle Anordnung, eine wichtige Rolle spielt.*“. Wörter oder Konstituenten, die semantisch zusammengehören, müssen direkt von einem Knoten dominieren[GCRHAUSSER S.169]. Die Phrase „*den Raum*“ ist eine Nominalphrase (NP) und wird durch die Präpositionalphrase (PP) dominiert, weil die Phrase „*den Raum*“ semantisch zu der Präposition „*in*“ gehört. Ist man sich nicht sicher ob die intuitive Segmentierung korrekt ist, kann man durch einen sogenannten Distributionstest etwas objektivieren. Dieser Distributionstest besteht aus einer Substitutionsprobe und einer Bewegungsprobe[GCRHAUSSER].

Substitutionsprobe:

Der Satz „*Fahre [nach vorne]*“, kann durch die Phrase *[vorwärts]* ersetzt werden. Wenn allerdings mit: „*Fahre [nach] vorne*“ segmentiert wird, kann mit *[vorwärts]* nicht ersetzt werden, da der Satz nicht wohlgeformt wäre.

Bewegungsprobe:

Der wohlgeformte Satz „*[der kleine Hund] sieht Julia*“ kann durch Verschiebung zu einem anderen wohlgeformten Satz verschoben werden: „*sieht* [*der kleine Hund] Julia?*“. Eine nicht akzeptable Bewegung eines Segments wäre von: „*der [kleine Hund] sieht Julia*“ zu „*der sieht [kleine Hund] Julia*“.

Neben dieser verbreiteten Phrasenstrukturgrammatik (PSG) existieren noch andere Formalismen, insbesondere die unifikationsbasierten Grammatiken, die in der letzten Dekade an Bedeutung gewonnen haben[SSAHLT3]. Bei den Unifikationsgrammatiken([SNS]/[VOLKM]/[MÜLLERS1]/[MÜLLERS2]) spielt die Unifikation von Merkmalen eine zentrale Rolle. Unifikation entspricht intuitiv einer Verträglichkeitsprüfung zweier Merkmale. Ein Merkmal ist dabei ein Attribut-Wert-Paar, das der Beschreibung der Eigenschaften linguistischer Objekte dient. Ein solches Attribut ist beispielsweise Numerus, der einen Wert aus dem Wertevorrat {Singular (`sg`), Plural (`pl`)}und einer Person, mit den Werten {`1, 2, 3`} enthält. Jede linguistische Einheit (Wort oder Phrase) ist durch eine Merkmalstruktur charakterisiert, d.h. durch eine Menge von Attribut-Wert-Paaren, deren Werte entweder atomare Symbole (z.b. `pl` oder `sg`) oder wiederum Merkmalstrukturen sein können. Merkmalstrukturen für syntaktische Einheiten werden auch als "komplexe Kategorien" bezeichnet.

$$\begin{bmatrix} \text{cat:} & \text{v} \\ \text{agr:} & \begin{bmatrix} \text{per:} & 3 \\ \text{num:} & \text{sg} \end{bmatrix} \end{bmatrix}$$

Abb. 7: Merkmalstruktur

Abbildung 7 zeigt eine sogenannte Merkmalsstruktur. Sie wird meist als zweispaltige Merkmalmatrix repräsentiert, in welcher die erste Spalte die Attribute, die zweite die zugeordneten Werte oder weitere Merkmale enthält. Einer der bekanntesten Unifikationsgrammatiken ist HPSG oder auch Head-Driver Phrase Structure Grammar genannt. Sie wurde zuerst beschrieben in [Pollard and Sag 87] und später grundlegend überarbeitet in [Pollard and Sag 94]. In `Verbmobil` und `SPIDER` wurde HPSG verwendet. Die zwei wichtigen Eigenschaften von HPSG sind die Integration von Syntax und Semantik und die knappe Formulierung von Grammatikregeln. Doch die letzte Eigenschaft führt dazu, dass man viele Informationseinträge in das Lexikon verlagert und diese dadurch sehr komplex wird. Unter der Internetadresse http://cl-www.dfki.uni-sb.de/~stefan/Babel/Interaktiv/ kann man einen HPSG Parser von Stefan Müller ausprobieren.

Neben HPSG gibt es noch die Defnite Clause Grammar (DCG). DCG ist historisch und systematisch eng mit der Programmiersprache Prolog verbunden, die u.a. zu Zwecken der natürlichen Sprache entwickelt wurde. Dies rührt daher, dass eine enge Verwandtschaft zwischen kontextfreier Grammatik und Hornklauseln besteht.

syntaktische Regel	**in DCG**	**in üblicher Prolog Schreibweise**
S → NP VP	s→ np, vp.	s(S) :- np(NP), vp(VP), append(NP,VP,S).
lexikalische Regel		
pro → wir	pro → [wir].	pro([wir]).

Abb. 8: DCG- und Prolog Regel

Abbildung 8 zeigt wie eine Syntax und eine lexikalische Regel jeweils in DCG und üblicher Prolog Schreibweise geschrieben wird. DCG ist im Sprachumfang der meisten Prolog-Implementierungen bereits enthalten, so dass der Theorembeweiser von Prolog als Parser dazu verwendet werden kann. Der DCG Formalismus erlaubt durch Unifikation weitere Möglichkeiten, die kontextfreie Grammatiken nicht bieten, nämlich der Behandlung von Kongruenz(z.b.: „*in den Raum*“ und nicht „*in der Raum*“*). DCG hat allerdings auch einige Nachteile[NLP]:

- DCG Regeln in großen und differenzierten Grammatiken werden wegen der hohen Stelligkeit unübersichtlich und fehlerträchtig.
- Modifikation an einzelnen Regeln führen unter Umständen zu erheblicher Modifikation der Gesamtgrammatik.
- Schnittstellendefinitionen (z.b. zum Lexikon) müssen an die Struktur der jeweils verwendeten DCG angepasst werden.

2.1.2 Parser

Bei der Verarbeitung von natürlicher Sprache ergeben sich Probleme, die beim Parsing im Bereich des Compilerbaus keine oder nur eine untergeordnete Rolle spielen. Diese Probleme resultieren direkt aus den besonderen Eigenschaften, die natürliche Sprache im Gegensatz zur künstlichen Sprache, wie z.b. Programmiersprachen, aufweisen. Die auffälligsten Eigenschaften sind höhere syntaktische Komplexität und zahlreiche Ambiguitäten (syntaktische, semantische, phonetische Mehrdeutigkeiten). Deshalb können Parsingkomponenten mehrere Aufgaben übernehmen[NLP]:

- Prüfung der Eingabe auf Wohlgeformtheit (syntaktisch korrekt).
- Aufbau der Strukturbeschreibung, welche die Weiterverarbeitung z.b. durch semantische Interpretationskomponenten oder Übersetzungsmodule unterstützen.
- Korrektur der Eingabe („robustes Parsing“).

Ein Parser besteht aus einer Menge von Syntaxregeln und dem Parsingalgorithmus. Je nach Architektur kann in einem Parser ein Lexikon integriert werden oder dieser hat Zugriff auf das Lexikon. In [SNS] und [NLP] werden Parser in folgende Merkmale klassifiziert:

- **Verarbeitungsrichtung:** Beim Parsen eines Satzes kann der Parser von links nach rechts oder von rechts nach links arbeiten. Diese Unidirektionale Verarbeitung mit links rechts Variante wird bei den meisten Parsern bevorzugt, weil sie der üblichen Leserichtung entspricht. Neben dieser unidirektionalen Verarbeitung gibt es noch die sogenannte bidirektionale Verabreitung. Bei einem bidirektionalen Verfahren gibt es mit Satzanfang und Satzende zwei feste Startpunkte für die Verarbeitung eines Satzes. Man kann aber auch nach bestimmten Kriterien mehrere Wörter (sogenannte Inseln) aus einem Satz als Startpunkte wählen und die Analyse entweder uni- oder bidirektional zu Ende zuführen. Die Entwicklung bidirektionaler Algorithmen wurde vor allem durch Probleme motiviert, die bei der Verarbeitung gesprochener Sprache auftreten können (siehe Kapitel Spracherkennung).
- **Analyserichtung:** Auch in der Analyserichtung gibt es unterschiedliche Varianten, wie die Struktur eines erkannten Satzes untersucht werden kann. Bei der Unidirektionalen Analyse unterscheidet man zwischen*Top Down* und *Bottom Up* Analyse (s Kapitel Top Down und Bottom Up Parser). Die Bidirektionale Analyse versucht die Vorteile beider Verfahren zu kombinieren (Left Corner Parser).
- **Deterministisch vs. nicht deterministisch:** Für bestimmte Klassen von kontextfreien Sprachen (z.b.: LL- & LR Sprachen)existieren deterministische Parser. Deterministisches Parsen bedeutet, wenn in jeder Situation die anzuwendende Regel eindeutig identifizierbar ist, d.h. nur eine Analysemöglichkeit betrachtet werden kann. Für die natürliche Sprache, wenn man sie der kontextfreien Sprachklasse zuordnen darf, kann es sicherlich keinen deterministischen Parser geben (Ambiguität). Nicht-deterministische Parser verfügen über eine Kontrollstruktur, die ihnen erlaubt bei bestimmten Situationen, in denen mehrere Regeln angewendet werden können, diese verschiedenen Analysewege zu verfolgen.

In den folgenden Unterkapiteln sollen einige Parsertechnologien erläutert werden, um die Unterschiede und die Einsatzmöglichkeiten besser einschätzen zu können. Es sei darauf hingewiesen, dass diese Diplomarbeit nicht zum Ziele hat detaillierte Parseralgortihmen zu erklären. Zu diesem Zweck bietet sich die Literatur [NLP] an.

2.1.2.1 Top Down und Bottom Up Parser

Charakteristisch für Top Down Analyse ist der von oben nach unten gerichtete Aufbau syntaktischer Strukturbeschreibungen. Ausgehend von dem Startsymbol der Syntax wird nach einer Ableitung gesucht, die zu einer oder mehreren Strukturbeschreibungen der Eingabe führt.

Abbildung 9 zeigt beispielhaft wie so ein Top Down Parser arbeitet. Ausgehend vom Startsymbol S gibt es eine Ableitung `NP` und `VP`. Für `NP` wird wieder eine Ableitung gefunden mit `det` und `N1`. Zu `det` gibt es keine weitere Ableitungen, sondern das Wort (Terminales Symbol) ‚Der'. Der Stack wird um die Konstituente `det` reduziert und es wird weiter nach einer Ableitung gesucht bis der Stack abgearbeitet ist oder eine Ausnahmehandlung eintritt, weil eine Regel oder ein Wort nicht erkannt wurde.

Satz : Den Greifer öffnen.
Syntax: {1: S → NP VP, 2: NP → det N1, 3:N1 → n, 4:VP → v}

	Satz	ABLEITUNG	Operation	Regel
(1)	[Der Greifer öffnen]	[S]		
(2)	[Der Greifer öffnen]	[NP VP]	EXPANDIERE	1
(3)	[Der Greifer öffnen]	[det N1 VP]	EXPANDIERE	2
(4)	[Greifer öffnen]	[N1 VP]	EXPANDIERE	3
(5)	[Greifer öffnen]	[n VP]	REDUZIERE	
(6)	[öffnen]	[VP]	EXPANDIERE	4
(7)	[öffnen]	[v]	REDUZIERE	
(8)	[]	[]	REDUZIERE	

Abb. 9: Top Down Analyse

Bei einem Top Down Parser ist zu beachten, dass die Syntax nicht linksrekursiv (oder rechtrekursiv) ist, wenn die Verarbeitungsrichtung von links nach rechts (bzw. von rechts nach links) erfolgt. Wird diese Restriktion nicht beachtet, ist nicht gewährleistet, dass der Parser bei der Analyse eines Satzes terminiert, da durch die rekursive Regel eine Endlosschleife entstehen kann. Diese Top Down Strategie kann in der Suche mit depth first, breadth first oder auch mit backtracking kombiniert werden.

Der Bottom Up Parser verfolgt eine andere Strategie als der Top Down Parser. Bottom Up Parser arbeiten *datengesteuert*, d.h. sie versuchen, ausgehend von den Wörtern durch einen oder mehrere Schritte auf das Startsymbol der Syntax zurückzuführen. Die Probleme, die bei Top Down Parsern auftreten können, treten bei Bottom Up Parsern nicht auf. Stattdessen verursachen Tilgungsregeln

(z.b.: die Regel A→A a wird vor der Regel A→a angewendet) und Zyklen(A→B, B→C, C→A) eine Nichtterminierung des Algorithmus.

2.1.2.2 Chart Parser

Unabhängig davon welche Variante man wählt, weisen die skizzierten Algorithmen im letzten Abschnitt zwei erhebliche Nachteile auf. Erstens formulieren sie relativ starke Restriktionen bzgl. der Klasse der Syntaxen, welche sie in der Lage sind zu verarbeiten (keine links- rechtsrekursive Regeln bei Top Down Parsing; keine Tilgungsregeln und Zyklen bei Bottom Up Parser). Und zweitens sind sie ineffizient, da unabhängig von der Kontrollstrategie(depth first mit Backtracking/ breadth first Suche) nicht verhindert werden kann, dass in vielen Fällen ein Satzabschnitt mehrfach auf dieselbe Art und Weise analysiert wird [NLP]/[HKI]. Um diese Nachteile zu umgehen wurde in den 60er Jahren der sogenannte Chart Parser entwickelt, der heute in vielen Sprachverarbeitungssystemen eingesetzt wird. Der Chart Parser basiert auf der Idee, die durch die Analyse erzeugten Teilergebnisse in einem *Chart* zu speichern. Dabei ist ein Chart ein abstrakter Datentyp, der meist in Form eines Vektors, Matrix oder Graphen auftritt. Die gespeicherten Objekte in einem Chart werden *Kanten* genannt. Man unterscheidet zwischen *aktiven* und *passiven Kanten*. Eine aktive Kante repräsentiert eine partiell erkannte Konstituente des Satzes (z.b. ein Verb oder ein Nomen erkannt); eine passive Kante repräsentiert eine vollständig erkannte Konstituente (S → NP VP). Eine Chart verfügt zudem über Operationen, die zum einen, passive Kanten auf Grundlage des verfügbaren lexikalischen Wissens in die Chart eintragen und zum anderen durch das verfügbare syntaktische Wissen der Chart weitere aktive/passive Kanten erzeugt werden.

Ein kleines Beispiel soll verdeutlichen wie ein Chart Parser arbeitet. Angenommen der Parser hat als Beispielsatz: „*Er sah das Mädchen mit dem Skateboard*“, zu parsen, und die entsprechende Syntax sähe wie folgt aus:

`G={S → NP VP, NP → pro, NP → det n, NP → NP PP, VP → v NP, VP → v NP PP, PP → p NP}`.

Der Algorithmus würde folgendermaßen ablaufen: Zunächst werden für alle Wörter des Satzes auf Grundlage des Lexikons lexikalische (aktive) Kanten gebildet. Also für ‚er’ → pro[nomen]’, ‚sah’ → v[erb] usw. Anschließend werden dann- gesteuert durch die Regel der Syntax – solange bereits gefundene Kanten zu neuen syntaktischen (passive) Kanten kombiniert, bis sich schließlich keine weiteren Kanten mehr bilden lassen.

{ [0, pro, 1], [1, v, 2], [2, det, 3], [2, relpro, 3], [3, n, 4], [4, p, 5], [5, det, 6], [5, relpro, 6], [6, n, 7] }

(1)	[0, NP, 1]	Aus: NP → pro	&	[0, pro, 1]
	[2, NP, 4]	Aus: NP → det n	&	[2, det, 3], [3, n, 4]
	[5, NP, 7]	Aus: NP → det n	&	[5, det, 6], [6, n, 7]
(2)	[1, VP, 1]	Aus: VP → v NP	&	[1, v, 2], [2, NP, 4]
	[4, PP, 4]	Aus: PP → p NP	&	[4, p, 5], [5, NP, 7]
(3)	[1, VP, 7]	Aus: VP → v NP PP	&	[1, v, 2], [2, NP, 4], [4, PP, 7]
	[2, NP, 7]	Aus: NP → NP PP	&	[2, NP, 4], [4, PP, 7]
(4)	[0, S, 7]	Aus: S → NP VP	&	[0, NP, 1], [1, VP, 7]

Abb. 10: Chart Algorithmus

Abbildung 10 zeigt wie der Satz: „*Er sah das Mädchen mit dem Skateboard*“, von einem Chart Parser arbeitet. In Abbildung 11 ist der Chart grafisch dargestellt. Man erkennt wie der Parser sukzessiv Bottom-Up arbeitet, bis die große Kante von Index 0 bis Index 7 verbunden ist. In der Grammatik des Satzes sieht man auch eine linksrekursive Regel (NP → NP PP), die dem Chart Parser keine Probleme bereitet.

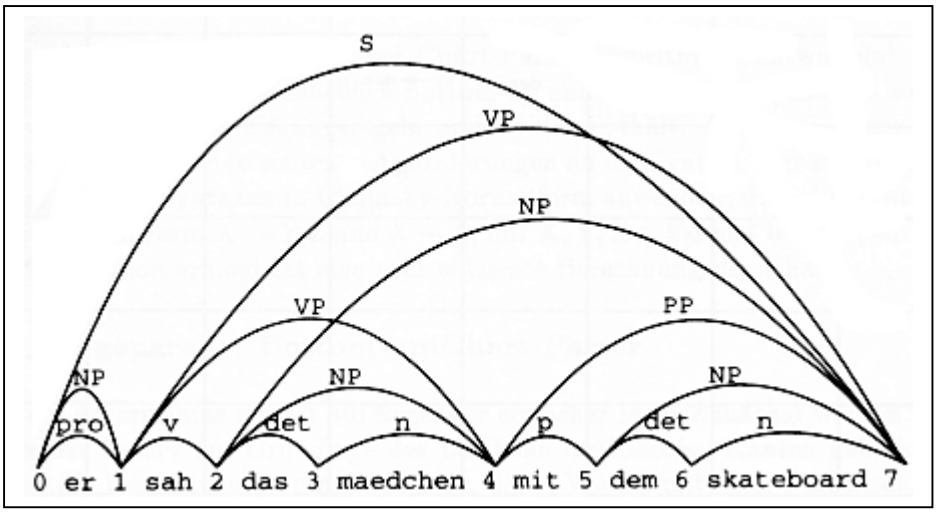


Abb. 11: Chart als Graph

Neben diesem allgemeinen Chart Parser gibt es noch andere Chart Parser (CKY-, Earley Parser[9]), die durch Syntaxeinschränkung und unterschiedlichem Informationsgehalt des Charts, den Algorithmus effizienter gestalten. Ein anderer Vertreter des Chart Parsers ist der Insel Parser, welcher auf den Earley Algorithmus basiert. Der Algorithmus arbeitet sich nicht von links nach rechts durch den Satz, sondern sucht zunächst nach hoch bewerteten (also recht sicher erkannten) Wörtern oder Wortfolgen. Diese Wortfolgen werden auch *Inseln* genannt, die er dann mit Hilfe von

[9] in [NLP] sind die Algorithmen für CKY, Earley und Insel Parser näher beschrieben

Grammatikregeln in beide Richtungen zu vergrößern versucht, bis schließlich der gesamte Satz analysiert worden ist. Dieser Inselparser wird im Zusammenhang mit dem spracherkennungssoftwaregeneriertem Worthypothesengraphen verwendet. Seine Priorität liegt in der Robustheit und nicht in der Korrektheit, wie die zuvor beschrieben Parser es tun.

2.1.2.3 Chunk Parser

Ein weiterer Parser, der auf Robustheit und Effektivität ausgelegt ist und in `Verbmobil` eingesetzt wird, ist der sogenannte Chunk Parser oder auch Partial Parser [KübHin]/[BADE]. Steven Abney [Abey2] definiert ein Chunk als ein Informationssegment in einem Satz. Für den Satz: „*[I begin] [with an intuition] [when i read] [a sentence], [I read it] [a chunk] [at a time]*“, sind die intuitiv markierten Chunks in eckigen Klammern dargestellt. Chunks bestehen aus einem Inhaltswort und den sogenannten Funktionswörtern, die das Inhaltswort umschließen. Eine einfache kontextfreie Grammatik ist ausreichend zur Beschreibung der Struktur von Chunks. Ein Chunk Parser besteht aus zwei Teilen: Einem POS (Part-of-Speech Tagger) und einem aneinandergereihten endlichen Automaten. Im folgenden soll der Vorgang des Parsens skizziert werden[BADE]:

- **Part-of-Speech Tagger:**
  Die POS Komponente wandelt jedes Wort des Eingabesatzes in syntaktische Kategorien. Für den Beispielsatz:

„*Oder*	*haben*	*Sie*	*kulturelle*	*Ereignisse*	*ins*	*Auge*	*gefasst*“
kon	***vafin***	***pper***	***adja***	***nn***	***apprat***	***nn***	***vvpp***

  wird eine Sequenz von POS Tags generiert. Wobei die Abkürzungen kon - Konjunktion, vafin – Hilfsverb, pper – Personalpronomen, adja – attributives Adjektiv, nn – Substantiv, apprat – Präposition mit Artikel und vvpp als Partizip Perfekt. Diese Sequenz von POS Tags dienen nun als Eingabe für den Chunk Parser.
- **Chunk Parser**: Den Chunkparser kann man sich als eine Kette von endlichen Automaten vorstellen, die in mehreren nacheinander stehenden Levels angeordnet sind. Pro Level wird normalerweise ein bestimmter Typus eines Chunks ausgeklammert, d.h., die Wortarten werden hinsichtlich ihrer grammatikalischen Struktur im (Teil-)Satz beschrieben. Die Ausgabe eines Levels dient hierbei als Eingabe des nachfolgenden Levels. Durch die einmalige Ausführung der Levelkette, sind Rekursionen in den Chunks ausgeschlossen. Wenn ein Automat ein POS Tag erfolgreich identifiziert, wird in der Ausgabe an dessen Stelle seine Ableitung geschrieben. Abbildung 11 zeigt die Eingaben für den Teilsatz „*oder*

haben Sie kulturelle Ereignisse ins Auge gefasst" in fünf verschiedene Levels. Im ersten Level wird das Teilstück, welches das Hilfsverb *haben* enthält, erkannt und durch das Label für Verbalchunks (vxfin) ersetzt. Der nächste Level identifiziert die drei Substantivausdrücke (nx2) *Sie*, *kulturelle Ereignisse* und *Auge*. Der Präpositionalchunk (px) wird im darauffolgenden Level konstruiert, indem die Präposition mit Artikel *ins* und dem Substantivchunk *Auge* zusammengefasst werden. Als Nächstes kann aus den Ausdrücken ein einfacher Satz (simpx0) erstellt werden, dem letztendlich die Konjunktion *oder* vorangestellt wird (simpx). Für die Ausgabe des Chunkparsers werden die Chunks als letztes in einen Klammerbaum umstrukturiert. Ein Beispiel eines solchen Klammerbaumes ist in Abbildung 13 dargestellt.

```
Level 1: kon vafin pper adja nn apprart nn vvpp
Level 2: kon vxfin pper adja nn apprart nn vvpp
Level 3: kon vxfin nx2 nx2 apprart nx2 vvpp
Level 4: kon vxfin nx2 nx2 px vvpp
Level 5: kon simpx0
```

Abb. 12: Verarbeitungsleveln beim Chunk Parsing

```
[simpx      [kon oder]
            [simpx0
                  [vxfin  [vafin      haben]]
                  [nx2    [pper       Sie]]
                  [nx2    [adja       kulturelle]
                          [nn         Ereignisse]]
                  [px     [apprart    ins]
                          [nx2        [nn     Auge]]]
                  [vvpp gefasst]]]
```

Abb. 13: Klammerbaum

Aus Abbildung 12 wird ersichtlich, dass nicht erkannte Teilchunks unverändert in das nächste Level übernommen werden. Die Konjunktion wird beispielsweise bis in den fünften Level unverändert weitergereicht bis sie an den Satz angefügt wird.

Der Chunk Parser kann jederzeit gefundene Teilergebnisse liefern. Beispielsweise wenn der Zeitrahmen für das Parsen abgelaufen ist, wie es bei `Verbmobil` als Anforderung der Fall ist. Auch für nicht wohlgeformte natürliche Ausdrücke ist der Chunk Parser geeignet. Besonders für große Textmengen bietet der Chunk Parser einen erfolgversprechenden Ansatz [ABNEY1]/ [ABNEY2].

2.2 *Tiefen- vs. Breitenstrukturanalyse – Bewertung*

Zum Parsen von natürlicher Sprache ist, wie aus den Lösungsansätzen und Parser Kapiteln zu ersehen, dass nur zwei Analysestrategien zu nennen sind. Auf der einen Seite, die Flache- bzw. Breitenstrukturanalyse mit dem Ziel einer breiten Datenabdeckung und auf der anderen Seite, die Tiefenstrukturanalyse, die eine möglichst vollständige grammatische Abbildung erreichen möchte. Systeme wie `Verbmobil` versuchen durch Kombination der beiden Analysestrategien ein besseres Analyseresultat zu bekommen. In Tabelle 2 sind die Gegensätzlichkeiten der beiden Strategien zusammengefasst.

Tiefenstrukturanalyse (Chart Parser(HPSG), PSG(DCG))	**Breitenstrukturanalyse (Insel- u. Chunk Parser)**
Hohe Korrektheit	Robust, aber weniger korrekt
Geringe Ambiguität	Hohe Ambiguität
Umfangreiche Syntaxregeln	Geringe Syntaxregeln
Meist komplexer Grammatikformalismus	Einfache Kontextfreie Grammatik
Geeignet für wohlgeformte kurze Sätze	Geeignet für ungeformte und große Satzstrukturen

Tab. 2: Tiefen- Vs. Breitenstrukturanalyse

Eigenschaften und Art der Eingabesätze
Eingabe der Sätze per Tastatur
Die Sätze sind meist wohlgeformt
Die meisten Sätze sind kurze Imperativsätze

Tab. 3: Eigenschaften der Eingabesätze

Im Kontext dieser Diplomarbeit ist die Tiefenstrukturanalyse der Breitenstrukturanalyse vorzuziehen, weil sie folgende Vorteile verbindet:

- Die ‚Spracheingabe' erfolgt über die Tastatur und die Eingabesätze sind meist wohlgeformt und auf wenige Worte beschränkt. Es bietet aufgrund des hohen Korrektheitsgrades der Syntaxanalyse eine genauere semantisch pragmatische Analyse.
- Durch die relativ begrenzte Anwendungsdomäne sind die Syntaxregeln noch verhältnismäßig gering, sodass von Seiten der Grammatikbeschreibung weniger Probleme zu erwarten sind.

Aus diesem Grund werden die Überlegungen im Hinblick des Ziels dieser Diplomarbeit auf die Tiefenstrukturanalyse gerichtet sein. Ferner wird als Grammatikbeschreibung DCG verwendet, da dieser Formalismus völlig ausreicht für die Beschreibung der Syntaxregeln.

3 Anforderungsanalyse

Die [SPIDER] und [KANTRA] (sprachgesteuerter Roboter zum Greifen von Gegenständen) Projekte zeigen, dass die Architektur bei Sprachgesteuerten Robotern aus einer Kommunikations-, Interpretations-, Planungs- und Ausführungsschicht besteht. Diese Arbeit konzentriert sich auf die Kommunikations- und Semantikschicht. Die Kernaufgaben der Kommunikations- und Semantikschicht sind:

- Prüfung der Eingabe auf syntaktische Korrektheit und Ausgabe einer Syntax-Semantik Struktur zur Weiterverarbeitung.
- Anhand der syntaktischen und semantischen Struktur wird eine Anweisung erzeugt.

Daraus ergeben sich zwei weitere Anforderungen:

- Man benötigt eine Verwaltungskomponente, die es erlaubt syntaktische, lexikalische und semantische Informationen zu pflegen.
- Eine Verarbeitungskomponente, welche die Eingabe des Satzes aufnimmt, analysiert und interpretiert.

Die Verwaltungs- und die Verarbeitungskomponenten werden nicht ganz unabhängig sein, da beide auf lexikalische und syntaktische Komponenten zugreifen. Im folgenden werden die Anforderungen an der Verarbeitungs- und Verwaltungskomponente herausgearbeitet.

3.1 Verarbeitungskomponente

Die Kernaufgabe der Verarbeitungskomponente besteht darin, die Eingabe des Benutzers aufzunehmen, zu analysieren und zu interpretieren. Sie benötigt ein Regelwerk(Parser), welches die Eingabe auf Wohlgeformtheit des Satzes überprüft. Ferner braucht sie einen Zugriff durch den Parser auf definierte Syntaxregeln und lexikalisch-semantisches Wissen, um eine Syntax-Semantik Struktur aufzubauen. Im folgenden wird der Korpus genauer analysiert um die Satzsyntax zu definieren. Anschließend werden die Anforderungen des Lexikons ermittelt.

3.1.1 Vorverarbeitung

Wenn man sich den Korpus ansieht, stellt man fest, dass es Wörter gibt, die keine Bedeutung enthalten (mal, bitte, usw.). Oder es treten meist doppelte Wörter auf, die immer in Kombination auftauchen(bis zu, gerade aus etc.). Für solche und anderen Fälle, soll eine Vorverarbeitung in die Verarbeitungskomponente integriert werden. Die Aufgaben der Vorverarbeitung sind folgende:

- Für Umlaute und 'ß' bietet sich an, sie in andere Buchstabenfolgen wie ae, ue, oe und ss umzuwandeln.
- Die Verschmelzung von Präposition (*ins, zur, zum*) und Artikel wird aufgelöst. Zum Beispiel wird die Präposition ‚*zur*' zu ‚*zu der*'.
- Bestimmte Wortfolgen treten meist gemeinsam auf z.b.: *bis zu, ein bisschen, ein wenig, gerade aus u.s.w.*, werden mit einem Unterstrich verbunden also bis_zu, ein_bisschen, ein_wenig, gerade_aus.
- Ferner wird hier bereits eine Rechtschreibprüfung durchgeführt um eventuell unbekannte Wörter auszufiltern und eine weitere Verarbeitung zu stoppen. Eine angemessene Meldung soll den Benutzer informieren, warum seine Eingabe nicht ausgeführt werden konnte. Der Vorteil liegt darin, dass in der eigentlichen Syntaxverarbeitung nicht bekannt ist, ob der Satz grammatische Fehler hat oder ein Wort im Lexikon nicht enthalten ist.
- Stoppwörter filtern: Stoppwörter wie *ähm, öhm, ehm, mal, bitte* werden aus dem Eingabestring entfernt, da sie keine relevanten Information enthalten. Diese Maßnahme soll zur Robustheit des Parsens beitragen.

3.1.2 Syntax

In Kapitel 1 wurde beschrieben, dass der Korpus aus drei Satztypen besteht. Es soll exemplarisch gezeigt werden, wie der Imperativsatztyp, der mehrheitlich im Korpus auftritt, in Konstituenten zerlegt wird. Der Imperativsatz (im Korpus) wird meist mit einem Verb im Imperativ Modus eingeleitet und wird von einem Adverb oder einer Präpositionalgruppe gefolgt. Der Satz: „f*ahre/laufe/gehe nach vorn*", ist eine adverbiale Präpositionalphrase [DUDGRAM S.647]. Eine andere Variante der Präpositionalphrase, ist die Phrase mit einem Präpositionalobjekt („*fahre zu der Wand*"). Dabei legt die Präposition den Kasus für das Objekt fest.

Eingabesatze:	**Syntax**
Fahre vorwärts	VP(V) Adv
Fahre schnell nach vorn	VP(V Adj) APP
Fahre zu der Wand	VP OPP(PP NP(Det N))

Tab. 4: Satz als Konstituenten

```
VP = Verbalphrase, bestehen aus einem Verb und/oder gefolgt von einem Adjektiv
APP = Adverbiale Präpositionalphrase, bestehen aus einer Präposition und einem Adverb
OPP = Objekt Präpositionalphrase, bestehen aus einer Präposition und einer NP
NP = Nominalphrase, besten aus einem Nomen oder Artikel und Nomen
Det = Artikel
V = Verb
N = Nomen
Adj = Adjektiv
```

Abb. 14: Beispiele für einige Konstituentenstrukturen

In Abbildung 14 werden einige Konstituentenstrukturbeispiele gezeigt. In Anhang B sind Phraseregeln für die Sätze aus Anhang A dargestellt. In der Implementierung wird durch Einsatz von sogenannten Agreements die Kongruenz gewährleistet [PP]/[GPL]/[LuckLP].

3.1.3 Lexikon

Das Lexikon muss die relevanten syntaktischen und semantischen Informationen für den Parser zur Verfügung stellen. Nachfolgend werden die Anforderungen für die Verarbeitungskomponente an einem Lexikon gestellt:

- Bereitstellung der syntaktischen und semantischen Informationen für den Parser
- Schnittstelle zur Prüfung und eines Wortes für die Vorverarbeitung und Parser. Ferner muss eine Log-Datei geführt werden, wenn ein Wort nicht bekannt ist. Diese unbekannten Wörter sollen durch einen Administrator später in das Lexikon eingepflegt werden.

3.1.4 Referenzauflösung (Mapping)

Nachdem der Eingabesatz auf syntaktische Korrektheit analysiert und mit semantischen Informationen ergänzt wurde, muss aus dem Strukturbaum eine Struktur erzeugt werden, die dem Aktionsmodul dazu dient Kommandos für den Roboter zu generieren. Mit diesem Aktionsmodul befassen sich die Kommilitonen Diekmann und Gossen aus dem Fachbereich Technische Informatik in ihrer Studienarbeit[Diekmann/ Gossen].

Die Befehlssätze aus dem Korpus lassen sich in sechs Roboter Aktionen einteilen. Dabei können diese Kommandos Parameter besitzen, welche die Kommandos genauer spezifizieren. Der Befehlssatz: „*fahre in den Raum eins eins acht null*“ z.B. besitzt die Aktion `move` und den Parameter `target` mit 1180 als Wert. Allerdings gibt es Kommandos mit `move`, in der andere Parameter wie `speed` (`fast, slow, default`) oder `distance(number)` eine Rolle spielen. Diese Sätze wären etwa: „*fahre schnell nach vorn*“ oder „*fahre eins eins null Zentimeter vorwärts*“.

Kommandoklasse	Aktion	**Parameter**	**Attribute**
MOTION	move	speed	fast, slow, default
		quantity	more, less
		direction	forward, backward
		target	number (4 ziffrige Nummer)
		distance	number (in Zentimetern)
MOTION	turn	direction	left, right,
		degree	number (0 – 360)
MOTION	gripper	direction	up, down
		do	close, open
MOTION	transfer	object	cup(z.Z. nur Becher)
		target	number
MOTION	get	object	cup (z.Z. nur Becher)
		target	number
		color	red, yellow, blue
MOTION	stop	nil	nil

Tab. 5: Kommandoklassen

In Tabelle 5 sind alle Kommandoklassen und deren relevanten Parameter aufgeführt. Diese Tabelle stellt das Wissen des Roboters dar. **Kommandoklasse** beschreibt welche Kommandoarten der Roboter ausführen kann. Momentan kann der Roboter nur Bewegungsaktionen ausführen. Es wäre durchaus denkbar sein Kommandospektrum zu erweitern, indem der Roboter nicht bewegt wird, sondern bestimmte Statusvariablen setzt bzw. abfragt. Das Feld **Aktion** beschreibt alle Aktionen, die mit Bewegungen zu tun haben(`move`, `turn`, `gripper`, etc.) bzw. indirekt auch der Befehl Stop, da er eine negative Beschleunigung zur Folge hat. Im Feld Parameter sieht man, dass einige Begriffe wiederholt aufkommen. Dies liegt daran, dass zum Beispiel eine Aktion `move` und `turn` einen Parameter `direction` verlangt. Aber die Richtung zur Aktion `move` hat eine andere semantische Bedeutung als die Aktion `turn`, die links oder rechts erwartet und nicht vor- oder rückwärts. Das Feld Attribut definiert die Attributmenge die einem Parameter, einer Aktion und einer Kommandoklasse zugeordnet sind. Attribute stellen also Werte für einen Parameter dar.

3.2 Verwaltungskomponente

Die Kernaufgabe der Verwaltungsaufgabe ist die Pflege des Lexikons und der Syntaxregeln. Zur Pflege des Lexikons sind Aufgaben wie:

- **Wartung und Pflege des Wortbestandes:**
  Dabei sind Aufgaben zu nennen wie einfügen, entfernen, aktualisieren von Lexemen in das Lexikon. Das Speichern der Lexeme basiert auf Vollformen, d.h. alle flektierten Lexeme werden einzeln gespeichert. Ein Stammlexikon würde den Speicheraufwand reduzieren, da er nur mit einem Wortstamm arbeitet (z.B: fahr[e]/[en]). Allerdings benötigt es eine morphologische Komponente, welche in dieser Arbeit nicht behandelt werden soll. Das Lexikon kennt folgende Wortklassen:
  - Verben (Infinitiv, Imperativ(Singular Form)) beschreiben meist eine Aktion bzw. ein Kommando (fahre, drehe, hole etc.). Bei bestimmten Verben verlangt das Verb ein Reflexivpronomen (sich bewegen). Da der singuläre Imperativ die zweite Numerusform verlangt, wird aus ‚sich' ein ‚dich', also „bewege dich"
  - Nomen in Genus, Singularform und den vier Kasusformen
    - Personennamen (Kai, Christoph, Klauck)
    - Greifbare Objekte (Becher, Dose)
    - Nicht Greifbare Objekte (Wand, Raum, Büro etc.)
    - Einheiten (Zentimeter, Grad)
  - Adjektive (schnell, langsam) keinen Kasus, Numerus, Genus
  - Adverbien (links, rechts, vorwärts, vorne, etc.) spielen im Korpus eine entscheidende Rolle. Sie geben meist eine Richtung an
  - Kardinalzahlen (eins, zwei, etc.)
  - Präpositionen geben bei Objekt Präpositional Phrasen den Kasus an. Zum Beispiel die Präposition ‚zu' legt den Dativ für das Objekt fest („zu der Wand"). Bei Adverbial Präpositionalphrasen wie nach vorne, nach hinten, nach links, nach rechts spielt der Kasus keine Rolle
  - Pronomen
  - Artikeln: Man unterscheidet bestimmte(definit) und unbestimmte (indefinit) Artikeln. Dabei sind Numerus, Kasus und Genus zu berücksichtigen. Wobei im Korpus nur der Kasus und der Genus eine Rolle spielt (Nominalphrase mit vorgehender Präposition)

- Zuordnen von Semantischen Begriffen (wie f*ahren, gehen*, u.s.w. zu `move` oder *nach vorne, vorwärts* zu `forward`)
- Stoppwörter definieren (ehm, äm, öhm, mal, bitte) um in der Eingabe Vorverarbeitung diese aus dem Eingabestring zu filtern.
- Kombi-Wörter definieren wie bis_zu, ein_bisschen, ein_wenig, gerade_aus u.s.w für die Vorverarbeitungskomponente

- Erzeugung einer Lexikondatei zur Trennung von lexikalischen Informationen und syntaktischen Regeln.

Die Verwaltungskomponente muss eine Pflegemöglichkeit der Syntaxregeln erlauben. Die Syntaxregeln sollten untergliedert werden in Phrasenstrukturen. Dies ermöglicht einen besseren Überblick über die Regeln. Die Regeln sollen einfach einzugeben sein.

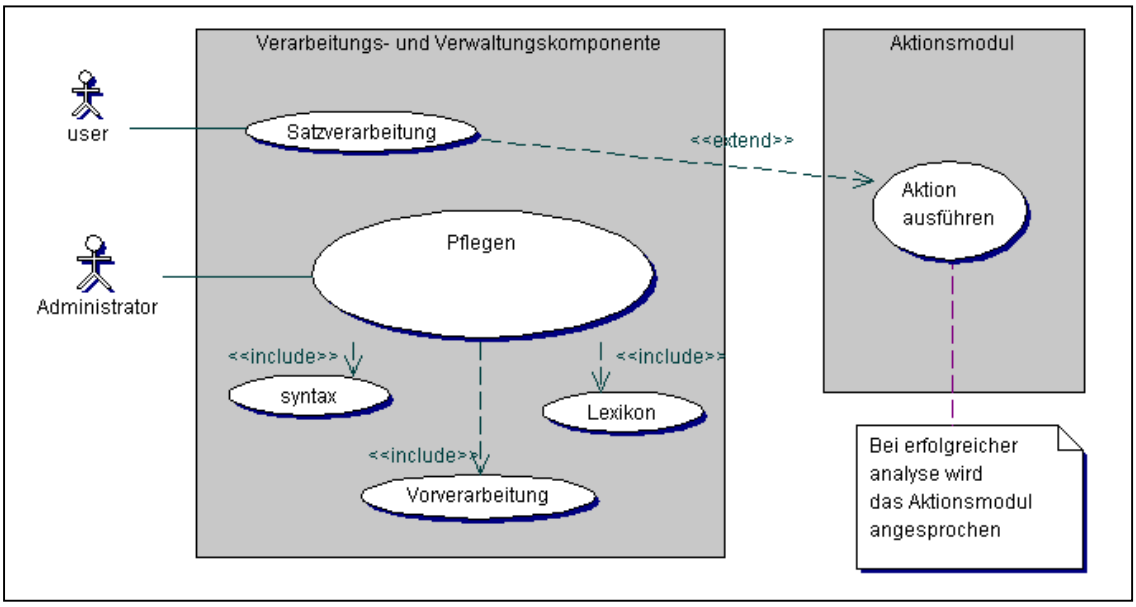


Abb. 15: Anwendungsfalldiagramm

In Abbildung 15 sind die beschriebenen Aktionen nochmals grafisch dargestellt. Es werden zwei Benutzer unterschieden, den „einfachen Benutzer“ und den Administrator mit besonderen Rechten zur Pflege des Systems.

DESIGN

Nachdem betrachtet wurde, welche Anforderung an die Sprachverarbeitungsschicht gestellt werden, und wie die einzelnen Aufgaben funktionieren sollen, wird in diesem Kapitel das Design des Systems entwickelt. Zunächst werden ausgehend von der Architektur des Systems, die Komponenten, die Schnittstellen und dann die Klassen beschrieben.

4 Client/Server Modell

Als Anforderung für diese Diplomarbeit sollte eine Client/Server Architektur verwendet werden (Vorschlag von Hr. Prof. Dr. Klauck). Es bietet meines Erachtens für diese Anwendung mehrere Vorteile:

- Der Client hat eine geringe Komplexität. Die Hauptaufgaben des Clients bestehen darin, den Client beim Server anzumelden und die Eingabe des natürlichsprachlichen Textes zu versenden.
- Durch diese Client/Server Architektur hat man die Option, dass mehrere Benutzer mit mehreren Robotern kommunizieren können. Die Clientsoftware kann an einem ‚beliebigen' Rechner installiert und mit dem Server verbunden werden.
- Eine Migration auf ein heterogenes Netz wäre durch diese Grundarchitektur ebenfalls gegeben. Durch Erweiterung der CORBA/DCOM Architektur ist es denkbar diese Anwendung auf ein heterogenes Netz zu übertragen.
- Durch die Zentralhaltung des Servers ergibt sich eine leichte Anpassung der Schnittstelle an dem Aktionsmodul[Diekmann/Gossen].
- Die Zentralhaltung des Servers bietet den Vorteil, dass Änderungen der Syntaxregeln des Lexikons oder der Verarbeitungskomponenten (z.B. Parser), relativ geringe Auswirkung auf das Gesamtverhalten des Systems haben.

Als Konsequenz für die geringe Komplexität des Clients, wird die Komplexität beim Server entsprechend höher.

Das Einsatzdiagramm (Abbildung 16) verdeutlicht grafisch die einzusetzenden Systemkomponenten. Der Server besteht aus den Komponenten Verarbeitung und Verwaltung. Diese beiden Komponenten werden später näher beschrieben. Am Server kann ein Client angemeldet sein oder nicht. Wenn am Server Änderungen vorgenommen werden (Lexikon, Syntax und Datenbank), muss sichergestellt werden, dass die Clients nicht verbunden sind. Dies soll vor inkonsistenten Daten schützen, wenn der Administrator am Lexikon oder an den Syntaxregeln Änderungen vornimmt.

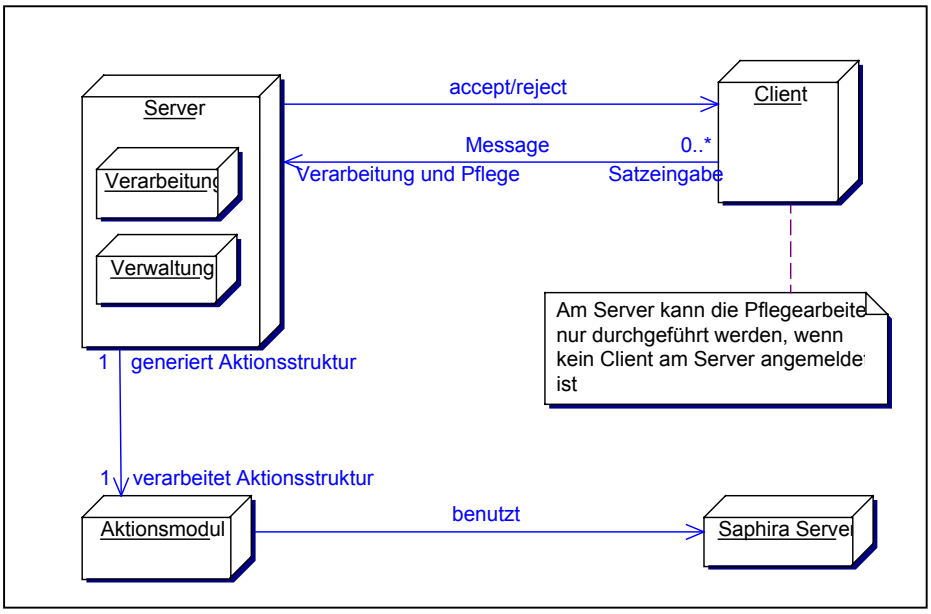


Abb. 16: Einsatzdiagramm

4.1 Client

Mit dem Client kann der Benutzer die natürlichsprachlichen Kommandosätze eingeben. Dabei muss er seinen Benutzer- und Rechnernamen(Host) eingeben, bevor er Kommandosätze an den Server verschicken kann.. Für den ersten Prototypen soll nur ein Roboter angesteuert werden, nämlich der Roboter mit dem Greifer (Pioneer 1 -S.P.O.T).

Die Eingaben, welche der Benutzer tätigt, sollen alle aufgezeigt werden. Alle nicht erfolgreich analysierten Sätze werden auf dem Server aufgezeichnet. Dieser Prototyp verfügt noch nicht über eine Rückmeldungsmöglichkeit, so dass der Client kein Feedback erhält, ob der Roboter die Aktion ausgeführt hat oder nicht. Alle Daten wie Eingabestring, Benutzername und Rechnername werden in einem Nachrichtenobjekt an den Server verschickt.

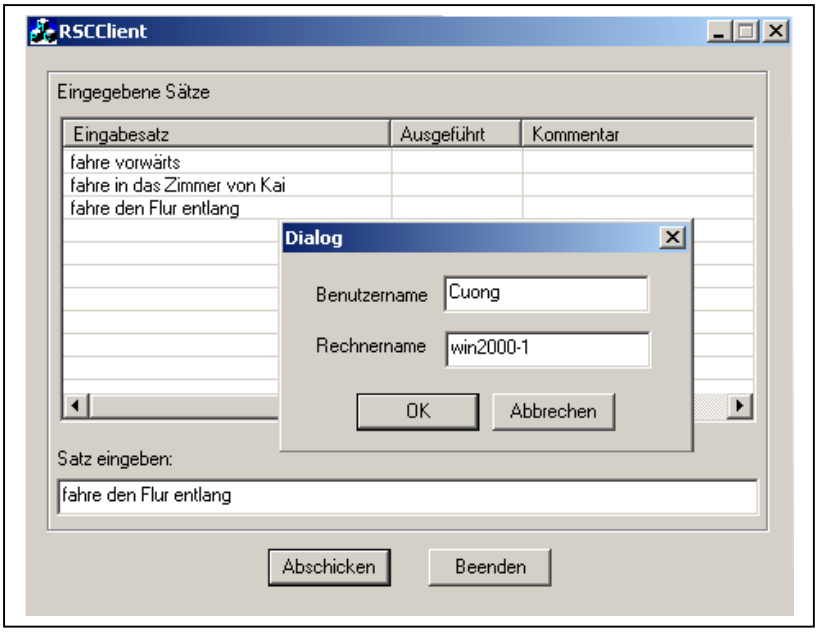


Abb. 17: Client GUI

Abbildung 17 zeigt die grafische Benutzeroberfläche des Client. Wird der Benutzer- und Rechnername akzeptiert, sieht man den eigentlichen Clientfenster. Wenn das Anmelden fehlschlägt, wird die Möglichkeit gegeben die Eingabe zu korrigieren. Auf dem Bild sieht man, das die Felder „Ausgeführt“ und „Kommentar“ noch leer stehen. Aus zeitlichen Gründen werden diese Funktionen nicht implementiert.

4.2 Server

Der Server hat, wie im Kapitel 3 beschrieben, die Kernaufgaben Satzverarbeitung und Datenpflege. Da die Aufgaben des Servers sehr umfassend sind, wird dieses Kapitel aufgeteilt in Verarbeitung und Verwaltung.

4.2.1 Verarbeitung

Nachdem nun der Client sein Nachrichtenobjekt versendet hat, muss auf der Serverseite die Nachricht entpackt und verarbeitet werden. Dabei durchläuft die Nachricht folgende Stationen:

- **Vorverarbeitung**: In der Vorverarbeitung wird der Eingabetext auf unbekannte Wörter hin durchsucht bzw. auf korrekte Eingaben geprüft. Das führt zu einer schnellen Antwort für den Anwender, da der Satz nicht analysiert werden braucht. Jedes Wort wird mit Hilfe einer Lexikonschnittstelle zur Vorverarbeitung geprüft. Ferner wird im Satz nach Wortkombinationen, welche unmittelbar benachbart sind, gesucht und miteinander durch ein Unterstrichzeichen verknüpft. Eine weitere Aufgabe der Vorverarbeitung ist das

transformieren von Umlauten und dem ‚ß' Zeichen. Diese Aufgabe ist notwendig, da bestimmte Regelwerke keine Umlaute akzeptieren. Die Notwendigkeit des Vorverarbeitungsmoduls liegt in der Entkopplung des Lexikons und einer Transformation des Eingabetextes. Zumal die Kompetenz des Lexikons nicht darin besteht Wörter zu transformieren, sondern Wörter und deren Bedeutung nachzuschlagen. Die Vorverarbeitungskomponente könnte auch in Zukunft nicht nur als Worttransformation genutzt werden, sondern auch als grammatikalische Transformation (Transformationsgrammtik [GL s.179 ff.]).

- **Parser**: Nachdem die Vorverarbeitungskomponente den Eingabetext aufbereitet hat, wird er dem Parser übergeben. Der Parser hat die Aufgabe auf syntaktische Korrektheit des Satzes zu prüfen und eine Syntax-Semantik-Struktur für das Mappingmodul zu generieren. Wie im Kapitel 2.2 bereits erwähnt, wird der DCG Formalismus verwendet, da er eine ausreichende Mächtigkeit für die Beschreibung der Grammatiken im Korpus besitzt. Das Designziel beim Parser ist die Trennung von Regeln und lexikalischem Wissen. Die Vorteile liegen in der Wartbarkeit von Parser(Austausch oder Erweiterung eines anderen Parsers) und Lexikon. Das lexikalische Wissen wird in einer Datenbank abgelegt. Der Vorteil liegt darin, dass das Datenbank-Managementsystem-System (DBMS) bei Datenverlust die Daten wiederherstellen könnte[10]. Ferner werden zwei Dateien für den Prologparser verwendet. Die eine Datei ist für die Syntaxbeschreibung und die andere Datei definiert das Lexikon.
- **Mapping**: Wenn der Eingabesatz durch den Parser erfolgreich analysiert wurde, bedeutet dies, dass der Satz syntaktisch korrekt ist. Nun muss die Ausgabe des Parsers analysiert werden, ob semantisch der Satz einen Sinn hat. Wenn dies erfolgreich analysiert wurde, wird die Aktionsdatei in dem Verzeichnis geschrieben, in der das Aktionsmodul seine Aktionsinformationen liest. Ob ein Satz semantisch korrekt ist oder nicht wird in der Tabelle Mapping (siehe Tabelle 5) definiert.

[10] Unter der Voraussetzung es wird regelmäßig ein Backup durchgeführt

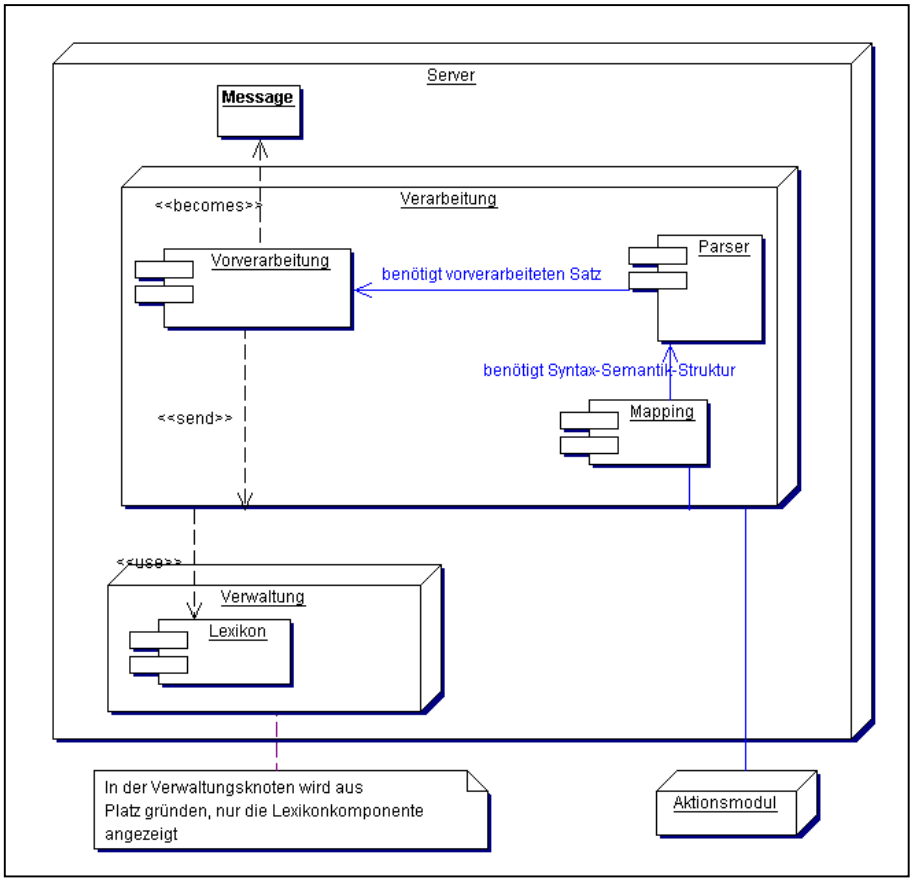


Abb. 18: Einsatzdiagramm - Verarbeitung

Das Einsatzdiagramm 18 verdeutlicht die Beziehung der einzelnen Komponenten des Servers, die oben beschrieben worden sind.

4.2.2 Verwaltung

Die Verwaltungskomponente muss Funktionen für die Pflege des Lexikons und der Syntax bereitstellen, die in Kapitel 3.2 erläutert worden sind. Ferner muss sie aber auch Schnittstellen für die Verarbeitung zur Verfügung stellen, welche die Vorverarbeitung und das Mapping benötigt. Das Lexikon enthält neben den Lexemen (Verben, Adjektive, Substantive etc.) auch Bedeutungseinheiten, die die Tabelle 5 (Kommandoklassen) abbildet. Jedes Lexem erhält eine Bedeutungsinformation, die vorher definiert wird. Es gibt vier Bedeutungskategorien, die zu verwalten sind: Kommandoklasse, Aktion, Parameter und Semantik. Die Semantikkategorie ist eine Art Übersetzung der eingegebenen Wörter einer Anfrage des Benutzers. Zum Beispiel bekommen die Lexeme *„fahre, gehe, laufe"* die Bedeutungsinformation `move` aus der Kategorie Semantik. Die Kategorie Parameter gibt sozusagen den Namen der zu akzeptierenden Attribute. Die Attribute gehören zu der Kategorie Semantik. Ein Beispiel für einen Parameter wäre `direction`, welches die Attribute `left, right, forward, backward` kennt. Diese Attribute sind in der

Semantik festgelegt, da sie ja aus den Befehlssätzen indirekt kommen. Um diese Informationen verwalten zu können, wird eine relationale Datenbank verwendet.

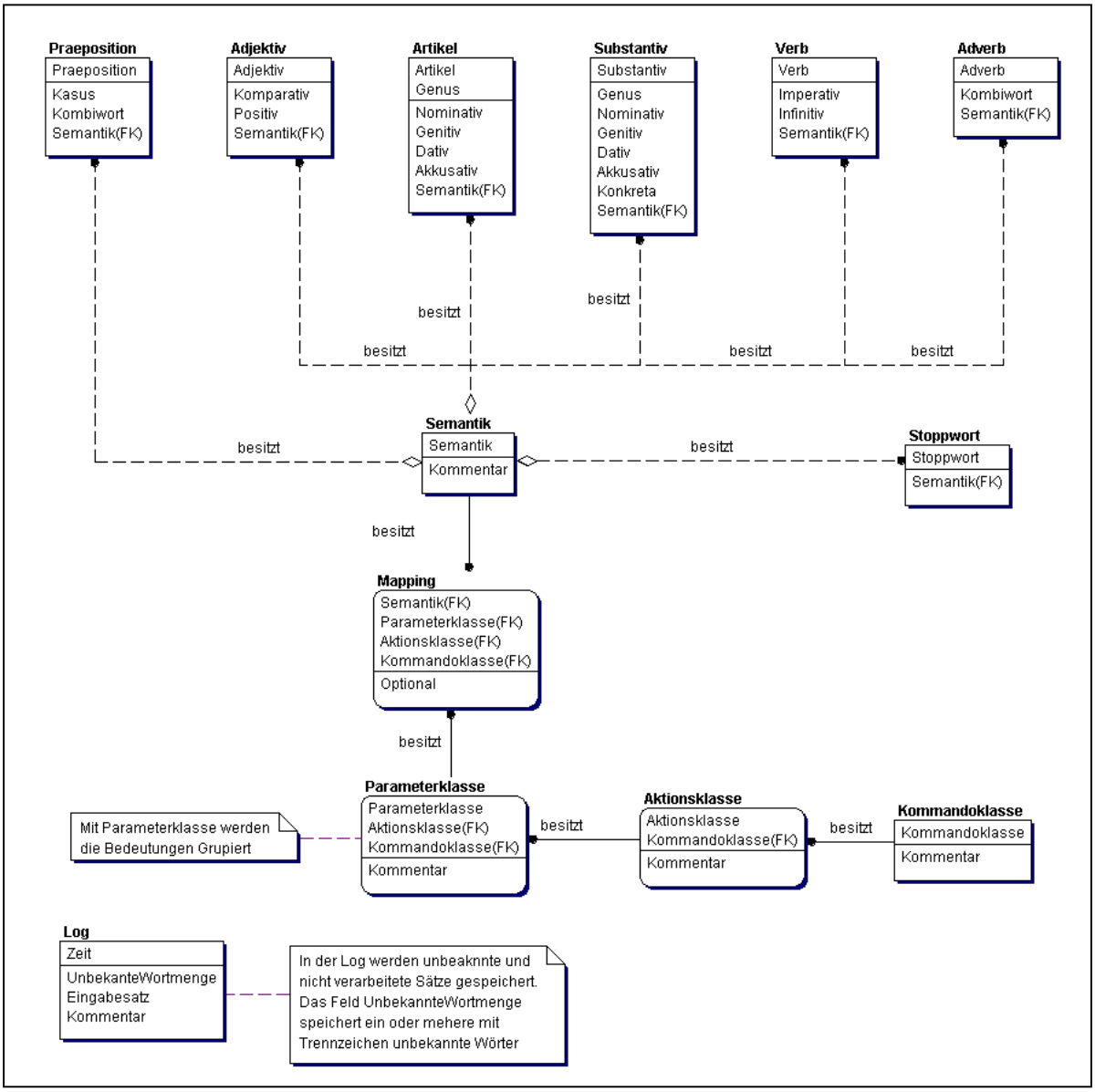


Abb. 19: ERD

Abbildung 19 zeigt ein Entity Relation Diagramm des Lexikons und des Logs. Alle Lexeme (Adjektiv, Präposition, etc.) haben eine Semantik. Die Semantik kann aber auch bedeutungslos sein und wird mit `nil` definiert. Als Primärschlüssel für alle Lexeme wird das Grundwort verwendet. Für das Verb wird also die Infinitivform benutzt und beim Substantiv der Nominativ. Beim Artikel wurde zusätzlich das Genus neben der Grundform des Artikels zum Primärschlüssel hinzugefügt, da bei dem unbestimmten Artikel „ein“ nicht unterschieden werden konnte, ob es sich um ein Maskulinum oder Neutrum indefiniten Artikel handelt. Wie aus dem Diagramm zu entnehmen ist gehört die Aktionsklasse zu einer Kommandoklasse und die Parameterklasse zu einer

Aktionsklasse. Wenn man sich die Tabelle 5 anschaut, wird dadurch die Beziehung widergespiegelt. Die Mapping Tabelle ist der Kern der semantischen Verarbeitung. Sie legt fest was der Roboter verstehen soll. Semantisch ist folgender Satz zwar richtig: „*fahre nach links*“, aber pragmatisch falsch, da definitionsgemäß der Roboter nicht nach links fährt, sondern nur um sich selbst drehen kann. Auch der Satz „*fahre eins eins grad nach vorne*“ ist zwar syntaktisch korrekt, aber semantisch falsch, da die Größenangabe beim Fahren sich auf eine Längeneinheit bezieht und nicht einen Winkel. Das Optionalfeld kennzeichnet die Parameterklasse, ob es sich um ein optionales Attribut oder obligatorisches Attribut handelt. So ist zum Beispiel für die Aktionsklasse `MOVE` obligatorisch die Parameterklasse `direction` notwendig, die bestimmt in welche Richtung gefahren wird. Um die Mappingstruktur abzubilden, enthält sie eine Kommandoklasse, Aktionsklasse, Parameterklasse und eine Semantik. Die Kommandoklasse ist als Oberklasse der Aktionsklasse zu sehen. Sie bestimmt welche Aktionsmöglichkeiten der Roboter hat (alle Bewegungsformen). Jeder Aktionsklasse ist einer Kommandoklasse zugeordnet. So zum Beispiel alle Bewegungsaktionen (MOVE, TURN oder GET) gehören zu der Kommnadoklasse (MOTION). Eine Aktionsklasse beschreibt, welche Parameter sie annehmen können. Zum Beispiel akzeptiert die MOVE Aktionsklasse die Parameter `direction, speed, distance`, etc. Diese Parameterklassen definieren wiederum Attribute (Semantik), welche sie akzeptieren können. Zum Beispiel gehören `fast, slow` Attribute zu der Parameterklasse `speed`. In Anhang D wird die aktuelle Mappingstruktur gezeigt.

Die Tabelle Log speichert die Zeit, die unbekannten Wörter, den Eingabesatz und einen Kommentar. Ein Logtupel wird erstellt, wenn der Eingabesatz unbekannte Wörter enthält oder die Syntax zu dem Eingabesatz nicht bekannt ist. Im Feld Kommentar wird die Information - unbekannte Wörter oder fehlende eine Syntax- abgelegt. Die Zeit wurde als Primärschlüssel verwendet, da ein unbekannter Eingabesatz während einer Sitzung mehrfach eingegeben werden kann und das generierte Fehlermeldungsprotokoll würde identisch sein.

Wie man sieht handelt es sich um ein Vollformenlexikon. Ein Vollformenlexikon kann viele hunderttausend oder Millionen Einträge enthalten, da sie alle Flexionsformen und Kompositionen enthalten [MorikNS]. Um ein großes Lexikon zu vermeiden wird die Morphologie verwendet. Dabei wird der Wortstamm mit Hilfe von Flexions- und Kompositionsregeln erweitert. Da die Morphologie gutes linguistisches Wissen abverlangt und die Anwendungsdomäne eingeschränkte Wörtermengen verwendet, habe ich das Vollformenlexikon verwendet.

5 Datenmodell

Im folgenden wird das Datenmodell beschrieben, welches alle wichtigen Klassen zeigen soll. Es werden Funktionen und Beziehungen der Klassen gezeigt. Das Datenmodell ergibt sich aus dem Anforderungsprofil aus Kapitel 3 und den Funktionsbeschreibungen aus Kapitel 5.

5.1 Schnittstellen

Bevor die konkreten Klassen beschrieben werden, soll ein Überblick über die Schnittstellen gegeben werden. Neben den Schnittstellendefinitionen sind noch diverse Methoden deklariert, die in den konkreten Klassen implementiert werden. Doch dazu mehr im nächsten Kapitel . Insgesamt sind sieben Schnittstellen definiert. Diese sind IWrite, IRead, ILexikonPflege, ILexikonVerarbeitung, IVorverarbeitung, IParser, IMapping.

IWrite:

Das `IWrite` Interface legt fest, wie und auf welche Weise geschrieben wird. Alle von `CLexemeinheit` und `CBedeutungseinheit` abgeleiteten Klassen wissen wie und was sie in die Datenbank schreiben. Zusätzlich legt diese Schnittstelle fest, welche Datenbankoperationen die konkreten Klassen implementieren müssen. Diese wären das aktualisieren, speichern und löschen von Datensätzen. Neben den Datenbankschreiboperationen müssen die Lexeme ihre Daten auf einer Datei speichern, um der Prologmaschine das lexikalische Wissen zu vermitteln. Auch diese Methode wird in der `IWrite` Schnittstelle definiert. Um die Daten der konkreten Instanzen zur Laufzeit zu setzen wird die Methode `SetData` verwendet. Diese Methode initialisiert die Datamembers. Dabei muss die Reihenfolge der übergebenen Daten beachtet werden. Alle Daten sind in einer `CPtrList` Objekt gesichert.

IRead:

Die Schnittstelle ist das Gegenstück zu IWrite. Es bietet statt Schreiboperationen Leseoperationen an. Dabei wird allerdings nur aus der Datenbank gelesen, nicht aber aus dem Lexikonfile, da die Daten aus der Datenbank generiert werden. Statt `SetData` heißt die Lese Methode dieser Schnittstelle `GetData`. Auch diese Methode verwendet ein `CPtrlList` Objekt um die Datamembers nach außen zu geben.

ILexikonPflege:

Die `ILexikonPflege` Schnittstelle hat im wesentlichen die Aufgabe die konkreten `CLexemeinheit` und `CBedeutungseinheit` Objekte zu erzeugen. Zusätzlich muss das

`CLexikon` Objekt, dass es implementiert, eine Methode zur Verfügung stellen um die DCG Lexikondatei zu erzeugen. Denn nur das `CLexikon` Objekt kennt alle Lexeme und delegiert das Schreiben an die Lexemobjekte (`CAdverb`, `CAdjektiv`, etc.).

ILexikonVerarbeitung:

Die `ILexikonVerarbeitung` Schnittstelle bietet der `IVorverabeitung` Schnittstelle die Möglichkeit ein oder mehrere Wörter daraufhin zu prüfen, ob sie im Lexikon enthalten sind. Wenn das Wort oder die Wörter nicht enthalten sind wird in der Tabelle Log ein neuer Tupel geschrieben.

IVorverarbeitung:

Diese Schnittstelle transformiert den Eingabetext des Benutzers. Es werden die Aufgaben erfüllt, die in der Anforderungsanalyse im Kapitel 3.1.1 gestellt worden sind.

IParser:

IParser legt die Schnittstelle für einen Parser fest. Ein Parser hat nur die Parse-Methode. Die konkrete Klasse muss in dieser Methode die abstrakte Methode implementieren und eine Syntaxstruktur zurückliefern.

IMapping:

Diese Schnittstelle legt fest, wie die Syntaxstruktur des Parsers verarbeitet wird. Sie kennt die Methode map, die als Eingabe die Syntax-Semantik-Struktur enthält, und als Ausgabe eine Knotenobjektliste zurückliefert.

5.2 Verwaltung

Die Verwaltung besteht im wesentlichen aus dem Lexikon, den Lexemeinheiten, Bedeutungseinheiten, Log und Datenbankverbindungsklassen (siehe Abbildung Verwaltungskomponente).

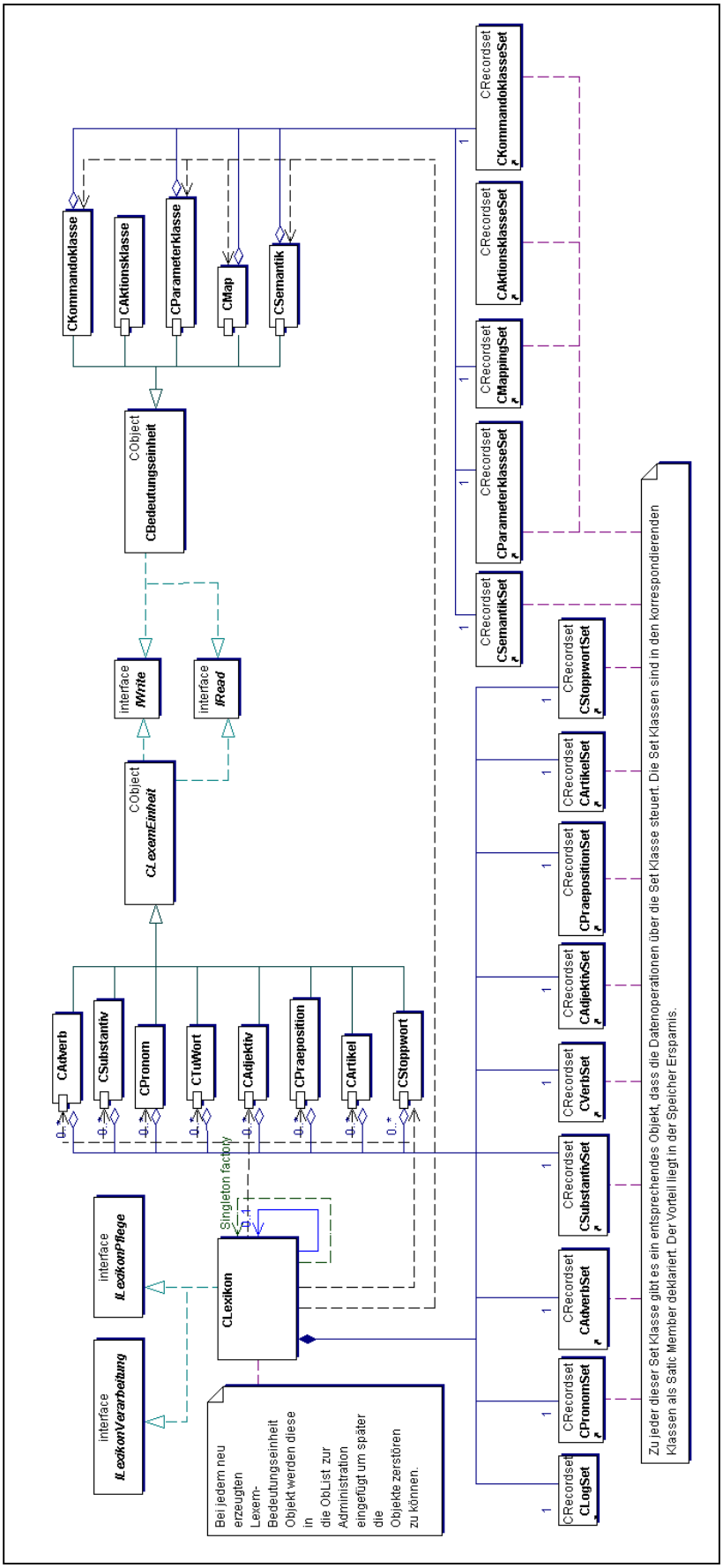


Abb. 20: Verwaltungskomponente

Im nachfolgenden werden diese Klassen näher beschrieben.

Lexikon:

Im Mittelpunkt der Verwaltungskomponente steht die Klasse `CLexikon`. Sie wird von den Schnittstellenklassen `ILexikonVerarbeitung` und `ILexikonPflege` abgeleitet. In ihr werden alle Verwaltungsaufgaben erledigt. Diese umfassen das Verwalten von Lexemeinheiten und Bedeutungseinheiten, Protokollierung der Anfragesitzung und bietet eine geeignete Schnittstelle für die Verarbeitungskomponente. Das Lexikon ist nach der Fabrikmethode- und Singeltonmuster [EM] aufgebaut. Über eine Konstante kann das `CLexikon` Objekt zur Laufzeit ein konkretes `CLexemeinheit` oder ein `CBedeutungseinheit` Objekt erzeugen. Dabei wird eine CLexemeinheit oder eine CBedeutungseinheit Zeiger zurückgegeben. Da das Lexikon wegen der Datenkonsistenzprobleme nur einmal instanziiert werden darf, wurde es auch als Singleton Pattern realisiert. Dieses Muster verhindert eine mehrfache Instanzeierung des `CLexikons`-Objektes. Das Lexikon hat Beziehungen zu den konkreten `CLexemeinheit`- und `CBedeutungseinheit`-Klassen. Es ist eine `Use-a` Beziehung.

Das Lexikon erzeugt auch die Prologlexikondatei. Bei der Schreiboperation wird immer die gesamte Datei überschrieben. Der Vorteil ist, dass beim Schreiben der gesamten Datei die Dateischreiboperation relativ schnell ist. Eine Suche nach dem ändernden Wort in der Datei wäre umfangreich und zeitintensiv.

CAdverb, CAdjektiv, CAktionsklasse, CParameterklasse etc.

Diese Klassen sind vom Aufbau sehr ähnlich. Deshalb wird an dieser Stelle exemplarisch nur eine Klasse beschrieben. Diese konkreten Klassen implementieren die Schnittstellen IWrite, IRead (siehe Kapitel Schnittstellen). Jeder dieser Klassen enthält eine statische Instanz der korrespondierenden CRecordset Klasse. Das heißt zum Beispiel, dass die Klasse CAdverb eine statische Instanz der Klasse CAdverbSet besitzt. Diese CAdverbSet-Instanz ermöglicht alle CAdverb Instanzen Datenbankoperationen auszuführen, um ihre Datamembers zu sichern bzw. aus der Datenbank zu lesen. Dieses CAdverbSet existiert für alle CAdverbSet Instanzen nur einmal. Dadurch reduziert man den enormen Speicherbedarf.

5.3 Verarbeitung

Die Verarbeitung besteht im wesentlichen aus dem der Komponente Vorverarbeitung, Parser und Mapping. Die Komponenten kommunizieren untereinander nur mit den Schnittstellen. Die Klasse CVorverarbeitungDCG die von der Schnittstelle IVorverarbeitung abgeleitet wird, implementiert

die Methode `EingabeTextTransformieren`. Die Klasse CVorverarbeitungDCG erfüllt die Aufgaben, die in Kapitel 3.1.1 beschrieben worden sind. Sie hat eine Beziehung zu der Schnittstelle ILexikonVerarbeitung, um die Eingabewörter des Textes zu überprüfen.

Die Klasse CDCGParser wird abgeleitet von der Schnittstelle IParser. Sie implementiert die Methode `Parse`. Diese Methode baut, nachdem die Vorverarbeitung den Text aufbereitet hat, ein Anfrageobjekt und ruft die Prologengine auf. Diese liefert nach einer erfolgreichen Unifikation die Syntax-Semantik-Struktur zurück. Der CDCGParser ist als Singleton Muster definiert, da nur eine Prologengine pro Prozess laut der Prolog/C++ Schnittstelle von Jan Wielemaker [SWICPP] erlaubt ist.

Die Mappingkomponente ist etwas komplexer aufgebaut als die vorherigen beschriebenen Komponenten.

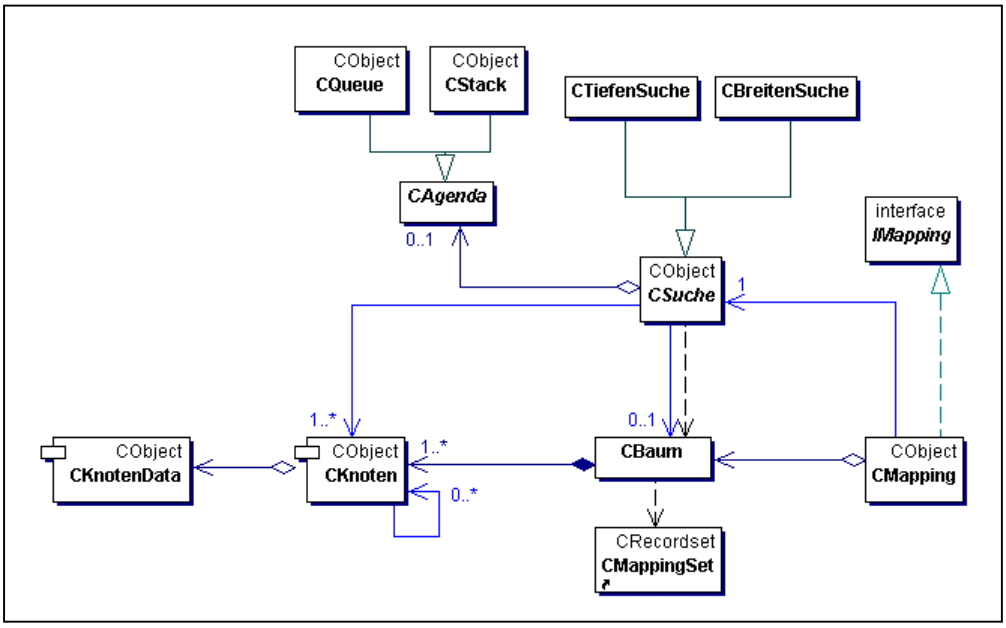


Abb. 21: Mappingkomponente

Abbildung 21 zeigt die Klassen und Beziehungen von der Mappingkomponente. `CMapping` implementiert das Interface `IMapping`, welche die Methoden `SchreibeInDatei` und `map` festlegt. Die letzte Methode ist die interessantere. Sie liefert eine Liste von Knotenobjekten zurück, die einen Teilbaum darstellt, welche die Aktionsklasse als Wurzel und die Parameter und Attribute als Kindknoten enthalten. Beim Erzeugen des `CMapping` Objektes, wird ein Baum nach den Daten der Mappingtabelle erstellt (siehe Abbildung 22). Ein Baum enthält mindestens ein Knoten (Wurzel). Ein Knoten besitzt eine Instanz `CKnotenObjekt`. Diese Instanz besitzt zwei CString Datamember, um den Wert und das Feld der Tabelle zu sichern.

Nachdem nun ein Baum während des Erzeugen des `CMapping` Objektes erstellt wurde, kann durch Aufruf der Methode `map`, die Syntax-Semantik-Struktur verarbeitet werden. Dazu benötigt die `CMapping` Klasse ein Suchobjekt, um nach den Wörtern im Baum zu suchen. Die Breitensuche [WissVer] wird verwendet, um eine Aktion zu suchen. Wie aus der Abbildung 22 ersichtlich wird, liegen die Aktionsknoten unter den Kommandoknoten. Das ermöglicht eine schnellere Suche, da bei der Breitensuche die direkt unteren Knoten von der Wurzel zuerst gesucht werden. Jede Syntax-Semantik-Struktur besitzt durch das Verb einen Kennzeichner „a-" (a für Aktion). Dieser Kennzeichner dient der `CMapping` Klasse das Verb zu identifizieren, um im Baum nach einem `CKnotenData` Objekt zu suchen, dass gleich ist mit dem Semantikwort. Wurde ein Knoten gefunden, werden alle Wörter mit dem Kennzeichner mit „v-"(v für Value) unter dem zuvor gefundenen Aktionsknoten gesucht (Tiefensuche [WissVer]). Wenn alle Wörter mit diesem Kennzeichner gefunden worden sind, wird eine Liste von CKnoten Objekten zurückgeliefert. Sonst wird eine leere Liste zurückgegeben.

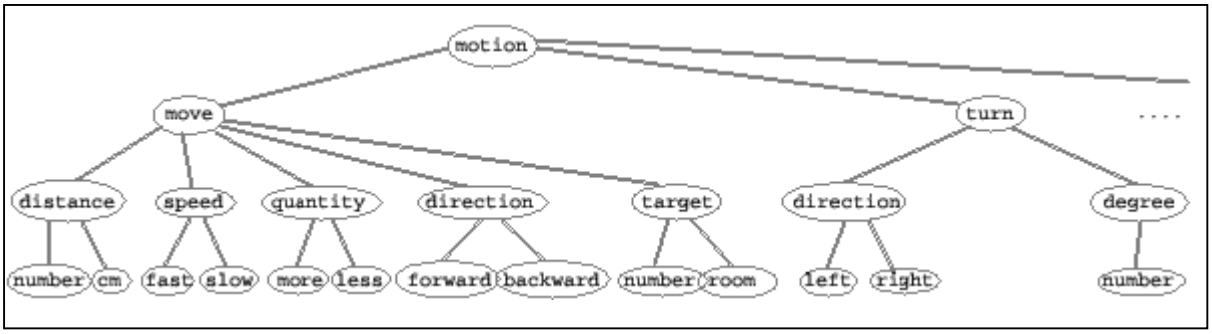
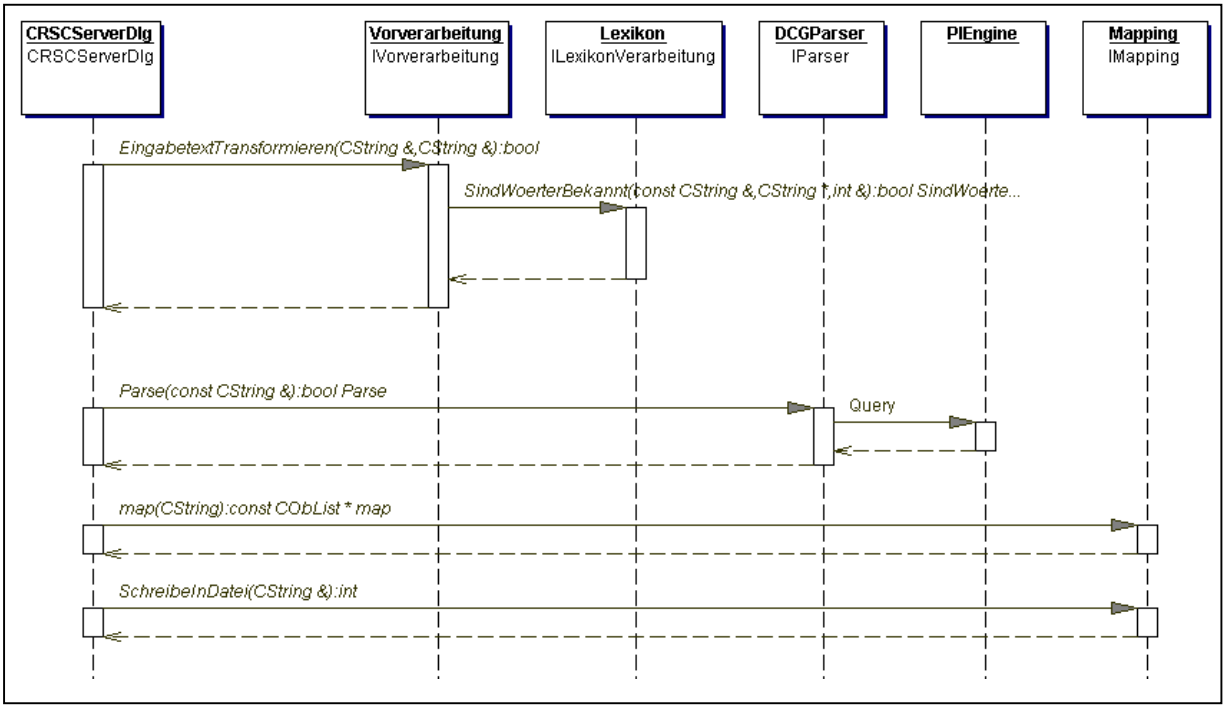


Abb. 22: Mappingbaum

Abbildung 22 zeigt nur einen Teilbaum, dass mit Hilfe der `CMappingSet` Klasse, welche die Datenbankverbindung zu der Tabelle Mapping bereitstellt, erzeugt wurde.

Um den zeitlichen Verarbeitungsablauf zu verdeutlichen, wird ein Sequenzdiagramm verwendet. In der Abbildung 23 ist ein solches Sequenzdiagramm zu sehen. Der Benutzer gibt ein Eingabetext in der Client Anwendung ein. Eine Socket Klasse benachrichtigt die Methode Message der Instanz `CRSCServerDlg`. Diese Instanz besitzt die Interfaces (`IVorverarbeitung`, `ILexikonVerarbeitung`, `IParser`, `IMapping`). Zunächst wird der Eingabetext durch Aufruf der Methode `EingabetextTransformieren` des Interfaces `IVorverarbeitung` vorbereitet. Nachdem im Eingabetext überprüft wurde, ob alle Wörter dem Lexikon bekannt sind und Stoppwörter vorher gefiltert wurden, wird der transformierte Text der Schnittstelle `IParse` durch die Methode `Parse` aufgerufen. Diese Methode bekommt zwei Stringsreferenzen, nämlich Eingabestring (von der `IVorverarbeitung`) und Ausgabestring

(Syntax-Semantik-Struktur). In der Methode Parse wird der sequentielle String in Tokens verwandelt, sodass jedes einzelne Token zum Aufbau der Prologliste vorbereitet wird. Nachdem die Liste aufgebaut wird, wird ein `PlQuery` Objekt erzeugt, welche eine Anfrage an die Prologmaschine stellt. Diese Anfrage befindet sich in einem `try` und `catch` Exceptionblock, welcher dafür sorgt, dass Ausnahmen der Prologengine abgefangen werden. Wenn die Syntaxanalyse erfolgreich war, wird dieser String der `map` Methode (von der `IMapping` Schnittstelle) übergeben. Diese sucht, wie im Mappingbaum beschrieben, nach allen gekennzeichneten Wörtern (mit a- bzw. v-), die im Baum in der richtigen Beziehung zueinander stehen. Enthält die `map` Methode als Rückgabewert eine Menge von `CKnoten` Objekten, dann wird die Aktionsdatei geschrieben. Diese Datei kann jedoch nur geschrieben werden, wenn das Aktionsmodul diese Datei nach Abarbeitung des Befehls freigegeben, also den Schreibschutz der



Datei auf write gesetzt hat.

Abb. 23: Sequenzdiagramm Verarbeitung

IMPLEMENTIERUNG

Dieses Kapitel beschreibt einige Aspekte der Implementierung des Systems. Zunächst werden kurz die Tools beschrieben mit der das System entworfen wurde, dann einige technische Beschreibungen zum DCG Parser und die C++ Schnittstelle und zum Schluss wird das Programm erläutert. Dabei wird insbesondere auf die Bedienoberfläche eingegangen.

Alle Bezüge auf Quellcode und Dokumentationen zur Prolog/C++ Schnittstelle sind auf der beiliegenden CD der Diplomarbeit zu finden.

6 MFC und Prolog/C++ Schnittstelle

Nach Vorgaben der Kapitel Analyse und Design, wurde mit Hilfe des UML Tools, `Together` 5.01, das Datenmodell entworfen. Mit `Together` wurde auch das relationale Datenmodell entworfen und in der Access 2000 DBMS die Datenbank erzeugt. `Together` unterstützt mehrere Datenbank Managementsysteme so z.B. Oracle 7-8 , SQL Server 7.x und DB2 6.3. Die Hauptarbeit wurde mit der Entwicklungsumgebung Visual C++ geleistet. Da in der Roboterprogrammierung häufig diese Sprache verwendet wird(Saphira Schnittstelle), wurde diese Sprachumgebung nicht verlassen, um anderen Leuten das Lesen des Quellcodes zu erleichtern und die Einarbeitung des Systems zu verkürzen. Da diese Anwendung keine hohen Anforderungen an die Datenbank stellt, habe ich mich für Access 2000 entschieden.

6.1 Datenbankverbindung

Als Datenbank wurde wie gesagt Access 2000 verwendet. Im Prinzip könnte auch jedes andere relationale DBMS verwendet werden, da das `Together` Tool ein DDL (Data Definition Language) Skript erstellt, welches über eine ODBC Verbindung die Datenbanktabellen erzeugt. Die RSCServer Anwendung benötigt zum Aufbau des Lexikons eine Verbindung zur Datenbank. Dies geschieht über die ODBC Verbindung, welche durch Treiber der Datenbankhersteller zur Verfügung gestellt wird. Der Aufbau einer Verbindung durch den Methodenaufruf `Open()` der Basisklasse CRecordset (MFC) verbraucht sehr viel Zeit, sodass ein vielfacher Aufruf dieser Methode vermieden werden sollte. Um diese Problematik zu entgehen erzeugt man ein CDatabase Objekt (MFC), stellt dann eine Verbindung zur Datenbank her und übergibt es dann dem

Konstruktor des `CRecordset` Objekts. Dadurch können nun die Klassen `CAdverbSet`, `CAdjketivSet`, `CSemantikSet`, usw. eine schnelle Verbindung aufbauen.

6.2 Socket Verbindung

Um die Client /Server Architektur in MFC umzusetzen, bietet es sich an mit der `CSocket` Klasse von MFC zu arbeiten. Die `CSocket` Klasse abstrahiert das zugrundeliegende Netzwerk, damit der Programmierer keine Kenntnisse über das Netzwerk benötigt und die Anwendung in jedem Netzwerk ausgeführt werden kann [MSDN]. Um Sockets zu verwenden, müssen folgende Dinge erledigt werden:

- Ein CSocket-Objekt erstellen.
- Für ein CSocket-Client-Objekt sollten die Standardparameter für `Create` verwendet werden. Für ein CSocket-Server-Objekt sollte ein Anschluß im `Create`-Aufruf angegeben werden.
- Wenn der Socket ein Client ist, wird die `Connect` Methode aufgerufen, um das Socket-Objekt mit einem Server Socket zu verbinden.
  Wenn der Socket ein Server ist, wird die `Listen` Methode aufgerufen, um mit dem Abhören von Verbindungsversuchen durch einen Client zu beginnen.
- Wenn die Sockets nicht mehr verwendet werden, also beim Schließen der Anwendung, werden die Sockets mit `Close` geschlossen.

In der Regel werden beim Client/Server Modell die zugriffe auf gemeinsame Ressourcen kontrolliert. In dieser Anwendung hat diese Anforderung noch keine hohe Priorität, da mir ein Grundgerüst für eine Client/Server Architektur wichtiger erscheint. Eine Erweiterung der Anwendung auf Zugriffskontrollen und Zugriffsbelastbarkeit ist ohne großen Aufwand möglich. Die zu schützenden gemeinsamen Ressourcen wären die Aktionsausgabedatei und der Parser, da dieser die Regelunifizierung abschließen muss, um eine neue Anfrage zu bearbeiten. Die Aktionsausgabedatei darf nur kontrolliert geschrieben werden, da sie die Schnittstelle zum Aktionsmodul darstellt.

6.3 Lexikon

Die Klasse CLexikon besitzt eine CMapStringToString Klasse um alle Lexeme zu speichern. Dies ist notwendig, um in der Vorverarbeitungskomponente die Wörter daraufhin zu überprüfen, ob sie dem Lexikon bekannt sind oder nicht. Wenn nun die Datenbank aktualisiert wurde, müssen die

geänderten Daten in einer Datei namens lexikon.pl erzeugt werden, damit Prolog die aktuellen Daten zum Parsen zur Verfügung hat. Beim Hochfahren des Servers werden alle Lexemdaten in einer Dictionary abgelegt. Wird nun die Datenbank aktualisiert, wird aber das Dictionary nicht mit aktualisiert. Man muss dabei aufpassen, dass, wenn in der Datenbank aktualisiert wurde, der Server neu gestartet werden muss. Dies ist eine Schwäche des Systems, das verbessert werden muss. Es muss auch möglich sein, Daten zu aktualisieren ohne dass danach der Server neu gestartet werden muss.

Ein Dictionary (CMapStringToString) bietet, wenn zu Anfang genügend Speicher allokiert wird, eine gute Performanz beim Zugriff auf die gesuchten Elemente. Wird jedoch die Speichergrenze allmählich erreicht, so fällt die performante Suche zu einer fast linearen Suche. Und das ist mit Abstand das schlechteste groß O Verhalten. Im Hinblick auf das Performanzverhalten und die gute Speicherausnutzung kann vielleicht ein anderer abstrakter Datentyp verwendet werden, der diesen Anforderungen gerecht wird, nämlich der Trie. In [MorikNS] wird auf einen Trie verwiesen um eine geringe Speicherauslastung zu erreichen. Das Prinzip ist, dass man sich durch einen Buchstabenbaum durchhangelt in der das Wort gespeichert ist. Ein Beispiel soll dies verdeutlichen: Für die beiden Worte „einen“ und „einer“ entspricht der Suchpfad e-i-n-e-r und e-i-n-e-n. Beide Worte haben fast den selben Suchpfad. Sie unterscheiden sich nur vom letzten Buchstaben. Man hat also für die Suche von zwei Worten einen fast gemeinsamen Suchpfad. Leider ließ sich in der Literatur keine Studien finden, die eine Aussage darüber machen welche abstrakte Datenstruktur die vorteilhaftere für eine Sprachverarbeitungsanwendung ist. In vielen Systemen wird jedoch eine morphologische Verarbeitung in die Sprachanalyse integriert [KANTRA], [VERBMOBIL]. Dabei wird versucht Worte in ihre Bestandteile zu zerlegen. Zum Beispiel steckt in dem Wort ‚Robotersteuerung’ die Worte ‚Roboter’ und ‚Steuerung’. Statt nun Robotersteuerung zu speichern, hat man nur die Worte Roboter und Steuerung im Lexikon gespeichert. Diese Form der Wortzusammensetzung nennt man auch Komposition. Das Ziel der Morphologie ist aber nicht primär den Umfang des Lexikons zu reduzieren, sondern die „Worte in den Wörtern zu erkennen“, um eine genauere Interpretation der Aussage des Sprechers machen zu können.

6.4 Parser

Als Parser wird Prolog verwendet. Konkret handelt es sich um den SWI Prolog Compiler in der Version 4.09. Unter [SWI] kann man nach der Registrierung kostenlos diese Version für akademische Zwecke runterladen und nutzen. Diese Version beinhaltet auch eine C++ Schnittstelle, mit der man von Prolog auf C++ zugreift und umgekehrt. Für diese Sprachverarbeitungsschnittstelle ist nur die Richtung von C++ nach Prolog interessant. Um die

Schnittstelle zu nutzen muss in der Entwicklungsumgebung von Visual C++ unter Extras->Optionen->Verzeichnisse die Pfade für Includes, Library und ausführbare Dateien angegeben werden.

Ich möchte an dieser Stelle nicht auf alle Klassen der SWI Prolog / C++ Schnittstelle eingehen, sondern nur auf die, die auch in der Diplomarbeit behandelt werden. Um den vollen Umfang der SWI Prolog/C++ Schnittstelle zu erfahren, empfiehlt sich die Dokumentation [SWICPP]. Bevor das Interface verwendet wird, muss man sie mit „`include <swi-cpp>`“ in das Projekt einbinden. Um Prolog als ‚Logikserver' zu verwenden, muss eine Instanz der Klasse `PlEngine` erzeugt werden. Dabei darf diese Instanz nur einmal pro Prozess existieren. In einer Multi-Threading Anwendung muss sicher gestellt werden, dass durch kontrollierten Ressourcenzugriff, nur ein Thread zur Zeit eine Prologanfrage starten darf. Nehmen wir an, die Prologdatei hieße „lexikon.pl“, dann würde der Aufruf so aussehen: `PlEngine("lexikon.pl")`. Wenn der Prozess zerstört wird, wird der Destruktor des `PlEngine` aufgerufen. Man braucht also nicht explizit die Engine zu schließen oder zu zerstören. Der folgende Code Ausschnitt zeigt wie eine Prologanfrage von C++ aufgerufen wird:

```
1    Tokenizer(strInput, strToken); //Eingabe in Token wandeln
2    try {
3       PlTermv ParameterFunktor(3);
4       PlTail liste(ParameterFunktor[1]);
5
6       pos = strToken.GetHeadPosition();
7       CString strTokenTmp;
8       while(pos) {
9         strTokenTmp = strToken.GetNext(pos);
10        liste.append(strTokenTmp.GetBuffer(10));
11        if (pos==NULL)
12          liste.close();
13      }
14      PlQuery q("s", ParameterFunktor);
15      if ( q.next_solution()) {
16        strOutput += (char *)ParameterFunktor[0];
17        bAnalysiert = true;
18      }
19   }
20   catch ( PlException &ex ){
21     err = (char *) ex ;
22     AfxMessageBox(CString(err));
23     return false;
24   }
```

Die Anweisung in Zeile 3 legt fest, dass eine Anfrage mit 3 Parametern verwendet wird. Die Klasse `PlTermv` ist null basierend, d.h. dass die erste Indexstelle mit null beginnt und die dritte Stelle für den Terminator NULL reserviert ist. Zeile 4 definiert nun eine Prologliste, welche an der Stelle eins des Bezeichners ParameterFunktor steht. In Zeile 6-13 werden die Token(Worte) zu einer

Prologliste aufgebaut. Also in der Prologform z.B.:„`[fahre,in,das,zimmer,von,kai]`“. Zeile 14 startet die eigentliche Anfrage an die Prolog Engine und Zeile 15,16 holt das Ergebnis der Anfrage aus der Engine und speichert es in einem CString Objekt. Alle Prologanweisungen sollten in einem try-catch Block geschrieben werden, um die Ausnahmen abzufangen. Wenn zwei Prologdateien verwendet werden sollen, kann man dies über die Anweisung `PlCall(“consult('syntax.pl').“)` direkt aufrufen.

Da ich nicht alle Möglichkeiten dieser Prolog/C++ Schnittstelle genutzt habe, kann ich keine gesamte Bewertung abgeben. Aber die Klassen, die ich verwendet habe, zeigen einen zuverlässigen Eindruck der Prolog/C++ Schnittstelle.

7 DCG Parser mit SWI Prolog

Dieses Kapitel soll ein wenig das DCG Programm dokumentieren, dass in der Datei synatx.pl und lexikon.pl zu finden ist. Es werden nicht alle Regeln ausführlich beschrieben, sondern nur die, die dem Verständnis des Programms dienen.

Die Datei syntax.pl beschreibt die Regeln, also die Syntax der Eingabe.. Dabei sind Sätze von der Gruppe Selbstlokalisierung und Statusabfrage aus dem Korpus noch nicht implementiert. Die lexikon.pl Datei wird aus den Daten der Datenbank generiert. Jede Klasse, die von `CLexemEinheit` abgeleitet wird, muss die abstrakte Methode `SchreibeInDatei` implementieren. In der Methode `SchreibeInDate` muss die konkrete Klasse (`CAdverb`, `CAdjektiv`, `CSubstantiv` etc.) festlegen, wie sie ihre Daten in die lexikon.pl Datei schreibt.

7.1 Syntax und Lexikon

Die Syntaxdatei umfasst zwei Gruppen von Regeln: Regeln mit Agreement (Kongruenz) und Regeln ohne Agreement. In dem oberen Block werden zwar alle S-Regeln mit Agreement definiert, aber in den unteren Grammatikdefinitionen, wo die Agreements keine Rolle spielen, werden sie einfach mit der Dummy Variable verbunden (ausgeblendet). Folgende zwei Prädikate werden zwar angegeben, aber in den weiteren Grammatikregeln nicht verwendet.

```
s(s(VP,APP)) --> vpimp(VP,AGR), app(APP,AGR).
s(s(VP,ADP)) --> vpimp(VP,AGR), adp(ADP,AGR).
```

Diese beiden Prädikate werden für adverbiale Präpositionalphrasen (app) und Adverbialphrasen (adp) verwendet. Also, für Sätze wie „*fahre nach vorne*“ oder „*fahre vorwärts/rückwärts*“, die keine

Kongruenz verlangen. Sätze, welche den Imperativ Verb einleiten und von einer Adverbialphrase oder adverbiale Präpositionalphrase gefolgt werden, werden mit den beiden Prädikaten unifiziert. Für die Terme `vpimp`, `app` und `adp` werden natürlich weitere Regeln festgelegt. So sieht etwa der `adp` Term folgendermaßen aus:

```
adp(adp(ADV),AGR) --> adv(ADV,AGR).
adv(adv(Root),_) --> [ADV], {lex(ADV,Root,adv,_)}.
```

Wird der Eingabesatz „*fahre vorwärts*“, und ‚*fahre*' als imperatives Verb unifiziert, wird in der Faktenbasis nach einem Wort ‚*vorwaerts*' gesucht, welches an der dritten Stelle des vierstelligen Prädikats das Attribut ‚adv' besitzt. Zwischen den äußeren Klammern des Regelkopfes wird die Syntaxstruktur sukzessiv bis an die Wurzel (s-Prädikat) aufgebaut. Die Syntaxstruktur hat dann folgendes aussehen: `s(vp(vimp(move)), adp(adv(forward)).`

Ähnlich verhält es sich mit den Agreement - Regeln. Dabei wird in der untersten Regel festgelegt, dass Genus und Kasus zweier Wörter übereinstimmen müssen. Für die Nominalphrase „das Zimmer“ sieht die Abarbeitung folgendermaßen aus:

```
1.  np(np(DET,N),AGR) --> det(DET,AGR), n(N,AGR).
2.  n(n(Root),agr(G,_,K)) --> [N],{lex(N,Root,nom,agr(G,_,K))}.
3.  det(det(Root),agr(G,_,K)) --> [DET], {lex(DET,Root,def,agr(G,_,K))}.
```

Eine Nominalphrase besteht aus einer Determiner und einem Nomen (1). Dabei wird verlangt, dass der Agreement-Ausdruck in Genus und Kasus übereinstimmen muss(1), was auch von (2) und (3) erfüllt wird. Wenn also in der Faktendatenbank für die beiden Wörter ‚das' und ‚Zimmer' die Agreements übereinstimmen, wird Prolog erfolgreich unifiziert.

7.2 Lexikon

Das Lexikon wird, wie bereits oben erwähnt, aus den Daten der Datenbank generiert. Die Lexikondatei enthalt lex Prädikate von der Stelligkeit 4 und 5. Nur Verben und Adjektive haben die Stelligkeit 5, da sie neben den anderen 4 Attributen angegeben, ob sie Imperativ/Infinitiv bzw. Komparativ/Positiv sind. Alle lex Prädikate haben den Aufbau [lexem, semantik, wortart, agr] bzw. für Verben und Adjektive [lexem, semantik, wortart, form, agr]. lexem bezeichnet das Wort, dass nachgeschlagen werden soll. semantik die Bedeutung des Wortes. Die Wörter, die keine Semantik besitzen werden mit `nil` bezeichnet. wortart gibt an, um welche Wortart es sich handelt, also `verb, adj, adv` etc. form gibt wie gesagt an, ob es sich um ein Imperativ/Infinitiv

bzw. Komparativ/Positiv handelt. agr ist eine Struktur mit folgendem Aufbau [genus, numerus, kasus]. Ein Codeausschnitt für einen Artikel aus der lexikon.pl Datei soll den Aufbau des Lexikons verdeutlichen:

```
lex(das, nil, def, agr(neut,_, n)).
lex(des, nil, def, agr(neut,_, g)).
lex(dem, nil, def, agr(neut,_, d)).
lex(das, nil, def, agr(neut,_, a)).
```

Der Neutrum-Artikel ‚das' wird in vier Deklinationsformen im Lexikon gespeichert. Man erkennt, dass dadurch die lexikon.pl Datei stark anwächst, wenn viele Lexeme in dieser Form eingetragen werden. Dies ist auf langer Sicht eine kritische Stelle, wo Handlungsbedarf ist.

8 Programmbeschreibung

Für die Diplomarbeit wurden zwei Programmprojekte angelegt. Nämlich RSCServer und RSCClient. Beide Projekte sind so eingestellt, dass die ausführbaren Dateien im Verzeichnis Robot zu finden sind. In diesem Verzeichnis sind auch die Dateien lexikon.pl, syntax.pl und RSCData.mdb zu finden. Die anderen vier Dateien werden beim Erzeugen der Execute Dateien mitgeneriert und sind Visual C++ spezifisch. Im folgenden wird die Programmbedienung beschrieben.

8.1 RSCServer

Der RSCServer sollte gestartet werden, bevor der Client gestartet wird. Andernfalls kann der Client Prozess sich nicht mit dem Server verbinden. Die Anwendung hat zwei Hauptmenüs, nämlich Sitzung und Administration. Im Menü Sitzung kann der Server auf Empfangsmodus geschaltet werden, um die Anfragen vom Client zu empfangen. Mit „Beenden" wird die Anwendung und die Sitzung beendet. Im Menü „Administration" können die Aktionen

- Lexikon bearbeiten
- Syntax bearbeiten
- Verzeichnis des Aktionsmoduls angeben
- Lexikon generieren
- Sitzungslog ansehen

ausgeführt werden. Der Server enthält ein List-Fenster, dass den Benutzernamen, die Eingabe des Benutzers, den Status des Syntax- und Semantikanalysevorgangs anzeigt. Das Feld Syntaxanalyse kommentiert, ob der Satz erfolgreich analysiert wurde oder, ob unbekannte Wörter im Text enthalten sind (Lexikon anpassen) oder, ob die Syntax der vorhandenen Grammatik erweitert werden muss. Das Feld Semantik Analyse kommentiert, ob nach der Syntaxanalyse eine Semantik-Pragmatik Analyse durchgeführt werden konnte. Ist dieser Vorgang erfolgreich, wird eine Aktionsdatei geschrieben, und es erscheint ein Text. Andernfalls wird angezeigt, dass die Semantik-Pragmatik-Analyse fehlgeschlagen ist.

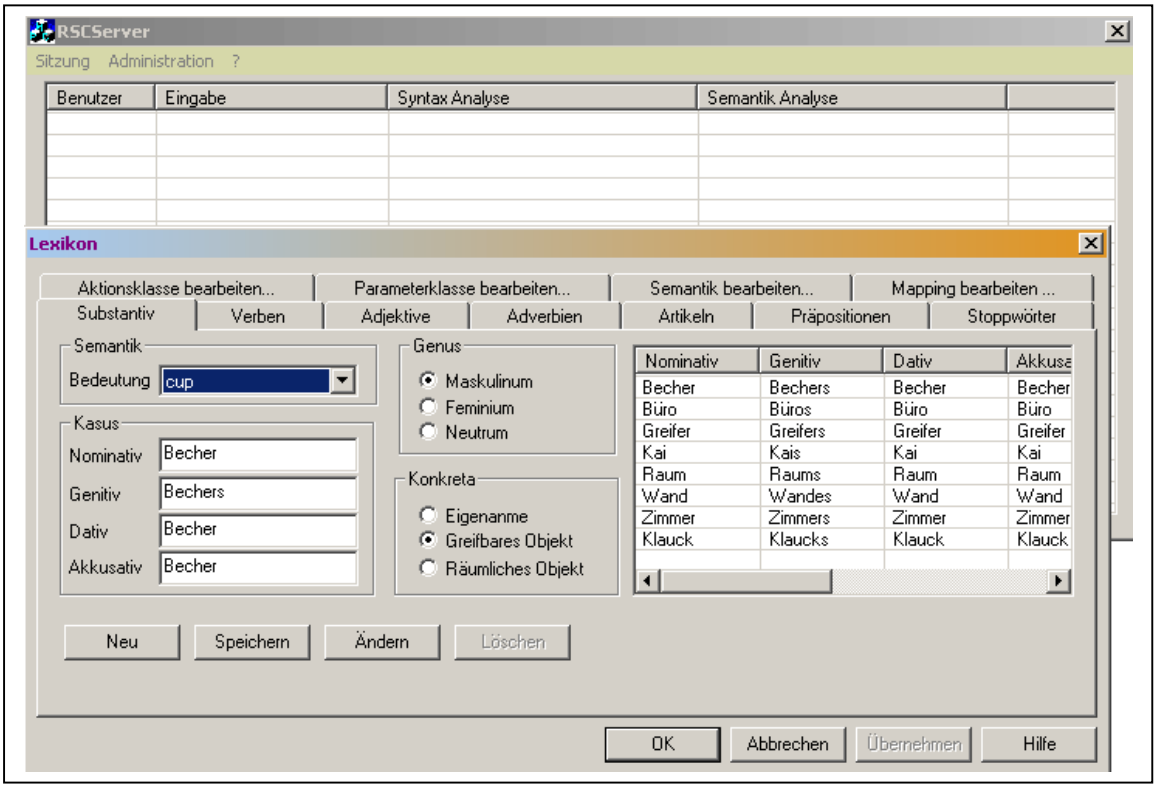


Abb. 24: RSCServer und Lexikon Maske

Abbildung „RSCServer und Lexikon Maske“ zeigt den RSCServer in dem Ausgangzustand und das Lexikon mit der Substantivregisterkarte. Auf der Substantivmaske sind vier Schaltknöpfe zu sehen. Der Schaltknopf „Löschen“ wurde deaktiviert, da diese Funktion noch keine hohe Priorität hat. Zum Speichern eines neuen Substantivs geht man folgendermaßen vor:

- Mit „Neu“ werden die Eingabefelder gelöscht.
- Danach müssen alle Felder „sinnvoll“ mit Daten gefüllt werden.
- Anschließend drückt man auf „Speichern“, um den Datensatz in die Datenbank einzufügen.

Um einen Eintrag zu ändern, wählt man in der Liste das Element, dass geändert werden soll. In den Eingabefeldern wird das markierte Objekt angezeigt. Danach ändert man das Feld und drückt anschließend auf „Ändern“. Der Eintrag ist damit geändert. Die Bedienung der Oberfläche und der Aufbau der anderen Registerfelder (Adverb, Adjektiv, etc.) sehen der Substantivmaske sehr ähnlich. Daher möchte ich auf eine weitere Beschreibung der anderen Registerfelder verzichten.

8.2 RSCClient

Nachdem der Server gestartet wurde, kann der Client gestartet werden. Sollte der Client vor dem Server gestartet sein, kann man die Verbindung zum Server durch Login wiederholen. In das Login Fenster gibt man den Benutzernamen und den Rechnernamen ein, auf den der Server läuft. Wenn die Verbindung zum Server erfolgreich war, erscheint ein Dialog, welches eine Listenansicht, ein Eingabefeld und zwei Schaltknöpfe enthält.

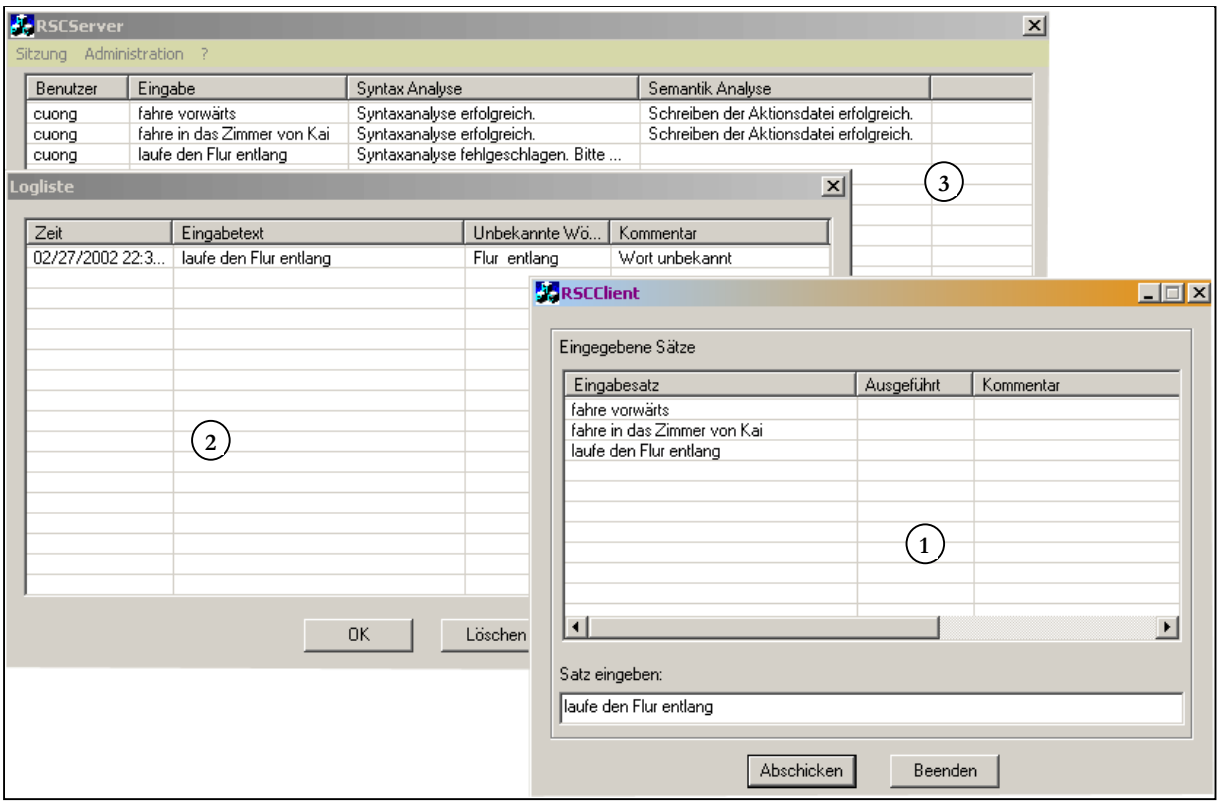


Abb. 25: Client - Server - Log

In Abbildung „Client - Server – Log“ erkennt man den Client (1), die Logliste(2) und den Server(3) im Hintergrund. Im Eingabefeld gibt man die Anfrage ein, welcher dann an den Server gesendet wird. Der Server kann im selben Rechner wie der Client ausgeführt werden oder auf einen beliebigen Netzwerkrechner. Alle eingegebenen Sätze werden im Listenfenster angezeigt. Der Client wurde nicht mit einer Rückmeldungsfunktion ausgestattet, sodass der Client nicht anzeigt,

ob seine Anfrage erfolgreich war oder nicht, da dies den Umfang dieser Diplomarbeit sprengen würde

ZUSAMMENFASSUNG UND AUSBLICK

9 Zusammenfassung

In dieser Arbeit wurde die Grundlage für ein System geschaffen, dass natürliche Sprache in Textform verwendet, um einen Roboter zu steuern. Der Schwerpunkt der Arbeit liegt im Aufbau eines Frameworks, dass sich durch

- Sprachverarbeitung
- Wartung und Pflege des Lexikons und der Syntax
- Client/ Server Architektur

auszeichnet. Kern dieser Arbeit ist die Sprachverarbeitung. Mit einer Prologmaschine und dem DCG Formalismus können einfache Satzkonstrukte abgebildet werden. Durch das Vorverarbeitungsmodul, der DCG-Parser und das Mappingmodul, konnte die Sprachsteuerungsschnittstelle zum Roboter aufgebaut werden.

Mit Hilfe der Prolog/C++ Schnittstelle wurde ein Framework gebaut, in der die Sprachverarbeitung und die Pflege der Syntax- und Lexikondatei integriert sind. Dies hat für die Pflege des Lexikons und der Syntax enormen Vorteil. Müsste in der Syntaxdatei gewichtige Änderungen vorgenommen werden, so haben diese unter Umständen gravierende Änderungen der Lexikondatei zufolge. Man denke dabei an einer Veränderung der Agreements. Die Konsequenz daraus wäre eine Änderung in fast jedem Lexikoneintrag. Um dieses vorzubeugen, wurde das Lexikon in einer einfachen Relationalen Datenbank abgebildet. Diese einfache Relationale Datenbank lässt sich auch auf einer Oracle oder SQL Server Datenbank migrieren. Die Datenbank stellt somit den aktuellsten Datenbestand des Lexikons dar. Durch die Menüauswahl „Generiere -> Lexikon“ kann die DCG Lexikondatei aus den Informationen der Datenbank erzeugt werden.

Um aus einigen Räumen im elften Stockwerk der Hochschule für Angewandte Wissenschaft den Roboter steuern zu können, bedarf es einer Software Architektur, die in einem Netzwerk funktioniert. Das Ergebnis ist eine rudimentäre aber funktionsfähige Client/Server Architektur. Es erlaubt die Serveranwendung auf einen Rechner auszuführen, dass nicht die Client Anwendung enthält. Dadurch ist ein Grundstein für eine Software gelegt, welche mehreren Benutzern erlaubt

einen oder viele Roboter zu steuern. Und das von einem beliebigen Raum im 11. Stockwerk. Natürlich unter der Voraussetzung, dass die Karte von dieser Etage den Roboter bekannt ist.

Der Prototyp ist in der Sprache C++ geschrieben, um die Verwandtschaft mit den anderen Roboterprogrammprojekten zu gewährleisten. Dadurch hat man einen einfachen Übergang von einem Roboterprojekt zum anderen. Zumal die Saphira Schnittstelle ebenfalls in der Sprache C ist und somit eine eventuelle Anpassung des Codes zum `Saphira` Server erleichtert wird.

10 Ausblick

Die Arbeit hat gezeigt, dass Sprachverarbeitung kein triviales informatiktechnisches Gebiet ist. Es erfordert doch eine Menge linguistisches Wissen, um die Sprachverarbeitung in den Griff zu bekommen. Die Arbeit hat eine einfache Sprachverarbeitungsanwendung hervorgebracht, die nur auf Roboterkommandos beschränkt ist. Trotzdem ist sie meiner Erfahrung nach, keine Anwendung in der man Daten in eine Datenbank schaufelt und sie wie aus der Datenbank liest und anzeigt. Die Probleme liegen im Verständnis der Grammatik und die daraus resultierenden Syntaxdefinitionen für den DCG Parser. Viele der Sprachproblematiken, wie Komposition, Derivation, Homonyme oder Ambiguitäten aller Art (akustische, syntaktische, semantische Ambiguität) [ECL1], [ECL1] sind hier nicht behandelt worden. Ob nun diese Probleme mit einer Strategie der Tiefenanalyse, oder besser einer Breitenanalyse in den Griff zu bekommen ist, vermag ich nicht zu sagen. Die Tiefenanalyse ist vielleicht in solch einer kleinen Anwendungsdomäne von Vorteil, da die Komplexität der Anwendung noch eine hohe Korrektheit und hohe Robustheit besitzt. Der DCG Parser von Prolog und die Prolog/C++ Schnittstelle erlauben in Kombination eine gute Entwicklungsplattform für eine Sprachverarbeitung. Auf der C++ Seite hat man einen schnellen Compiler, verbunden mit vielen Schnittstellen (Datenbank, grafische Steuerelemente) und auf der anderen Seite hat man ein Regelwerk, dass bestens für syntaktische Regeln ausgelegt ist (Unifikation).

Die Implementierung des Prototyps wurde auf rudimentäre Funktionalität beschränkt. Insbesondere wurde nicht betrachtet in wie weit die Multi-User und Multi-Roboter Steuerung tragfähig ist. Der Prototyp ist im Hinblick auf Multi-User und Multi-Roboter entwickelt aber noch nicht vollständig dafür ausgelegt. Ferner sind im System noch einige kleine Verbesserungen notwendig, die das System robuster und benutzerfreundlicher machen.

Diese wären:

Reduzierung von Tippfehlern:

Solange in der Anwendung noch keine Spracherkennungssoftware integriert ist, wäre zu überlegen, ob nicht ein Algorithmus in Frage käme, welche die lästigen „Vertipper“ bei der Anfrageneingabe reduziert. Dabei bietet sich der Damerau-Levenstein-Metrik Algorithmus[LEV] an. Speziell zur Tippfehlerkorrektur wurde die Damerau-Levenstein-Metrik als Ähnlichkeitsmaß für Zeichenketten entwickelt. Diese Metrik soll beim Vergleich eines Textwortes mit einem Wort aus dem Wörterbuch die Zahl der Tippfehler annähern, um dann das ähnlichste Wort auszuwählen. Es werden vier mögliche Arten von Fehlern angenommen. Einfügung, Löschung, Substitution, Transposition(Vertauschung zweier benachbarter Symbole). Für zwei Zeichenketten `s` und `t` wird dann die minimale Anzahl von Fehlern berechnet, mit der diese ineinander überführt werden können. Hierdurch erreicht man eine höhere Robustheit des Systems, da die Tippfehler reduziert werden können.

Transformationsgrammatikkomponente:

Um die Robustheit weiter zu erhöhen, könnte eine Transformationsgrammatikkomponente integriert werden. Dabei werden anders formulierte Imperativsätze wie zum Beispiel der Fragesatz; „*würdest du bitte nach vorne fahren?*“ zu einem Satz „*fahre nach vorne* “ transformiert. Durch Wortmustersuche könnte man solche Satzbauformen erkennen und diese in einfache imperative Satzformen umwandeln. Dadurch ließe sich eine Syntaxvergrößerung vermeiden.

Verbesserung der Benutzerfreundlichkeit:

In der Serveranwendung sind viele Dialogmasken zu sehen. Insbesondere ist die Wartung von Lexikon und Logliste erschwert, wenn beide Fenster nicht gleichzeitig zu sehen sind. In der Logliste Maske erkennt man die Wörter, die nicht im Lexikon enthalten sind, oder ein Satz wurde so eingegeben, das die Wörter zwar im Lexikon bekannt sind aber die Syntax den Satz nicht akzeptiert. Für solche Fälle muss man das Lexikon oder die Syntaxdatei nachpflegen. Dabei ist es einfacher für den Benutzer, dass er beide Fenster zu sehen bekommt. Die MFC Bibliothek kennt modale (Standardfenster wie in der Serveranwendung) und nichtmodale Fenster. Die nichtmodalen Fenster erlauben zwischen den Dialogfenstern zu klicken, ohne dass das Kindfenster vorher geschlossen werden muss.

In den Masken sind Funktionen nicht deaktiviert, die als Folge der vorigen Eingabe hätten deaktiviert werden müssen. So zum Beispiel darf in der Lexikonmaske, wenn auf die Schaltfläche „Neu“ geklickt wurde, der Schaltknopf „Ändern“ nicht aktiviert werden.

Socket Implementierung:

Bei diesem Prototyp wurde beim Client aus Zeitgründen die Feedbackfunktion nicht implementiert. Es wäre wünschenswert, dass der Client benachrichtigt wird, damit er erfährt, ob seine Eingabe ausgeführt wurde und wenn nicht, wieso sie nicht ausgeführt wurde. Die Clientanwendung muss also auch ein Horchsocket bekommen, damit sie die Serveranwendung benachrichtigen kann. Ein weiteres Manko ist die Tatsache, dass beim Server die Socketverbindung nicht auf ‚standby' gesetzt werden kann. Der Server kennt momentan nur den Verbindungszustand horchen oder die Verbindung abbrechen bzw. beenden.

Ausbau der Grammatik:

In der Diplomarbeit wurden hauptsächlich Imperativsätze verarbeitet. Es wäre interessant zu erfahren, wie das System sich bei größeren Grammatiken und Lexikoneinträgen verhält. Zum Beispiel wäre es sinnvoll über die Sprachverarbeitung bestimmte Statusvariablen des Roboters zu setzen bzw. abzufragen. Solche Variablen können sein, Max-, Min-, Standardgeschwindigkeit oder auch Umgebungskoordinaten des Roboters. Insbesondere der letzte Punkt ist besonders sinnvoll, da dem Roboter durch seine ungenauen Sensoren gelegentlich die Koordinatendaten verfälschen. Diese beiden Gruppen von Sätzen sind also Aussage- und Fragesätze, die dem System ergänzt werden könnten.

GLOSSAR

Adverb Mit Adverbien (Beiwörtern) werden ganz allgemein nähere Umstände angegeben. Im engeren Sinne wird es auch als Beiwort zum Verb verstanden. Zu den Adverbien gehören Wörter wie: links, rechts, vorn, hinten etc.. [DUDGRA]

Ambiguität Mehrdeutigkeit. Mehrdeutigkeiten können auftreten bei Syntax-(das Wort in: Präposition in Dativ und Akkusativ oder als Angabe von Zeitpunkt oder Zeitraum), Semantik-(Geldbank oder Sitzbank), Phonologieanalyse (setze oder Sätze).

DCG Definite Clause Grammar**.** Ein Formalismus zur Beschreibung einer Grammatik in Prolog.

Deklination Die systematische Veränderung des Nomens, Artikels, Adjektivs nach Numerus, Genus und Kasus.

Derivation Derivation bezeichnet die Verbindung eines Wortes mit einem Affix (z.B.. –lich oder un-) in ein neues Wort. Sie beinhaltet sowohl die Suffigierung (z.B.. klein/lich) als auch die Präfigierung (z.B. un/schön) [GCRHAUSSER S.266]

Flexion Flexion bezeichnet die systematische Variation, mit der ein Wort sich an verschiedene syntaktische Umgebungen anpasst, beziehungsweise syntaktische und semantische Funktionen ausübt. [GCRHAUSSER S.266]

Homonyme Mehrere Bedeutungen zu einem Begriff. Zum Beispiel Bank. Entweder als Sitzmöglichkeit oder als Geldinstitut zu verstehen.

Komposition Komposition bezeichnet die Verknüpfung zweier oder mehrerer Wörter zu einem Wort (z.b.: Haus/schuh) [GCRHAUSSER S.266]

Kongruenz Übereinstimmung von Genus, Kasus und Numerus.

Konjugation Die Veränderung der Form beim Verb nennt man Konjugation. [DUDGRA]

Konstituente Eine Phrase oder ein Teilausdruck aus einem Satz.

Linguistik Linguistik ist die Wissenschaft der Sprache. Die wichtigsten Teildisziplinen sind Phonetik, Syntax, Semantik und Pragmatik.

Morpheme Morpheme sind die kleinsten bedeutungstragenden Einheiten eines Wortes.

Morphologie Die Morphologie beschäftigt sich mit Wortformen und deren Zusammenhänge (Flexion, Komposition, Derivation). Ihre Bedeutung liegt in der Erkennung von Wortformen und Zurückführung des Wortes auf seinen Wortstamm (geh/-st,-en,-t).

Parser Parser sind automatische Analyseprogramme, die für einen bestimmten Typ von Regeln funktionieren. Diese Regeln können Satzregeln oder auch Wortregeln (morphologische Parser) sein.

Partikel Die Partikel erfüllen - vorzugsweise in der gesprochenen Alltagssprache – vielfältige Aufgaben. Sie geben u.a. den Grad oder die Intensität an; dienen der Hervorhebung; drücken die innere Einstellung des Sprechers aus, und spielen eine wichtige Rolle in der Gesprächsführung. (...) Im Gegensatz zu den Wörtern der drei Hauptwortarten und den meisten Adverbien haben sie keine eigentliche ([nenn]lexikalische) Bedeutung oder sind jedenfalls bedeutungsarm. [DUDGRA]

Phonetik Phonetik ist eine Teildisziplin der Linguistik und beschäftigt sich mit akustischen Lauten.

Pragmatik Die Pragmatik untersucht den Gebrauch von sprachlichen Ausdrücken (Wörtern, Phrasen, Sätzen) und ihre tatsächliche Bedeutung in Abhängigkeit zum Kontext, indem sie gebraucht werden (im Gegensatz zur Semantik, die die wörtliche Bedeutung von Ausdrücken untersucht).

Prosodie Prosodie bezeichnet die Betonung oder die Melodie eines Satzes. Durch Sprechpausen oder Tonhöhenverlauf können Satz- oder Phrasengrenzen erkannt werden.

PSG Phrasenstrukturgrammatik. Grammatik die auf Teilausdrücke basiert.

SPIDER Ein Sprachverarbeitungsprojekt der Universität Hamburg Fachbereich Informatik, bei dem ebenfalls einen Roboter in der Spahira Umgebung durch natürliche Sprache gesteuert wird.

Synonyme Mehrere Wörter zu einer Bedeutung. Zum Beispiel „fahre“ und „gehe“ meinen eine Fortbewegung.

VerbMobil Ein Universitätsübergreifendes Forschungsprojekt, in der auch einige Industrieunternehmen beteiligt sind um Sprachverarbeitung zu erforschen.

Anhang

Anhang A: Sprachkorpus

Kategorie 1: Vor- und Rückwärtsfahren

Vorwärts:

1. Fahre/Gehe/Laufe nach vorn.
2. Fahre/Gehe/Laufe schnell nach vorn.
3. Fahre/gehe/laufe vorwärts.
4. Fahre/Gehe/Laufe gerade aus.
5. Bewege dich nach vorn.
6. Bewege dich vorwärts.

Rückwärts:

7. Fahre/Gehe/Laufe nach hinten.
8. Fahre/gehe/laufe rückwärts.
9. Fahre/Gehe/Laufe eins eins null Zentimeter nach hinten.
10. Fahre/Gehe/Laufe schnell zwei null Zentimeter rückwärts.
11. Bewege dich nach hinten.
12. Bewege dich rückwärts.

Kategorie 2: Fahren mit Ziel Angabe

13. Fahre/Laufe/Gehe bis zu der Wand.
14. Fahre/Laufe/Gehe nach vorne bis zu der Wand.
15. Fahre/Laufe/Gehe zu der Wand.
16. Fahre/Laufe/Gehe die Wand entlang.
17. Laufe/Gehe/Fahre in das Zimmer/Büro von Kai.
18. Laufe/Gehe/Fahre in das Zimmer/Büro eins eins acht null.
19. Laufe/Gehe/Fahre eins zwei null Zentimeter nach vorne.
20. Bewege dich langsam zwei null Zentimeter nach vorn.
21. Laufe/Gehe/Fahre in den Raum von Kai.
22. Fahre/Gehe/Laufe ein bisschen/wenig nach vorne.
23. Fahre/Gehe/Laufe ein bisschen/wenig nach hinten.
24. Fahre/Gehe/Laufe ein bisschen/wenig rückwärts.
25. Fahre/Gehe/Laufe ein bisschen/wenig vorwärts.

Kategorie 3: Richtungsänderungen

26. Drehe/Wende dich nach links/rechts.
27. Drehe/Wende dich X Grad nach links/rechts.
28. X Grad nach links/rechts drehen/wenden bitte.
29. Drehe dich etwas nach links.

Kategorie 4: Geschwindigkeitsänderung

30. Schneller!
31. Langsamer!
32. Halt!
33. Stopp!
34. Anhalten!
35. Sofort anhalten!
36. Sofort stoppen!

Kategorie 5: Objekt holen

37. Hole den Becher aus dem Zimmer/Raum/Büro von Kai.
38. Bringe mir den roten Becher von Kai
39. Hole mir den Becher aus Kais Zimmer/Raum/Büro.

Kategorie 6: Greifer steuern (nachträgliche Änderungen)

40. Öffne/schließe den Greifer.
41. Greifer öffnen/schließen.
42. Senke/hebe den Greifer.
43. Fahre den Greifer nach oben/unten.
44. Beweg den Greifer rauf/ab.

Anhang B: Korpus mit Konstituentenstruktur

Vor- und Rückwärtsfahren

Vorwärts

Fahre/Gehe/Laufe nach vorn.	VP(V) APP(PP ADV)
Fahre/Gehe/Laufe schnell nach vorn.	VP(V ADJ) APP(PP ADV)
Fahre/gehe/laufe vorwärts.	VP(V) ADP(ADV)
Fahre/Gehe/Laufe gerade_aus.	VP(V) ADP(ADV ADV)
Bewege dich nach vorn.	VP(V PRO) APP(PP ADV)
Bewege dich vorwärts.	VP(V PRO) ADP(ADV)

Rückwärts:

Fahre/Gehe/Laufe nach hinten.	VP(V) APP(PP ADV)
Fahre/gehe/laufe rückwärts.	VP(V) ADP(ADV)
Fahre/Gehe/Laufe eins eins null Zentimeter nach hinten.	VP(V) GP(ZAHL EINH) APP(PP ADV)
Fahre/Gehe/Laufe schnell zwei null Zentimeter rückwärts.	VP(V ADJ) GP (ZAHL EINH) APP(PP ADV)
Bewege dich nach hinten.	VP(V PRO) APP(PP ADV)
Bewege dich rückwärts.	VP(V PRO) ADP(ADV)

Fahren mit Ziel Angabe:

Fahre/Laufe/Gehe bis_zu der Wand.	VP(V) OPP(PP NP)
Fahre/Laufe/Gehe nach vorne bis_zu der Wand.	VP(V) APP(PP ADV) OPP(PP NP)
Fahre/Laufe/Gehe zu der Wand.	VP(V) OPP(PP NP)
Fahre/Laufe/Gehe die Wand entlang.	VP(V) NP ADP(ADV)
Laufe/Gehe/Fahre in das Zimmer/Büro von Kai.	VP(V) OPP(PP NP(Det N PP NPERS))
Laufe/Gehe/Fahre in das Zimmer/Büro eins eins acht null.	VP(V) OPP(PP NP(Det N ZAHL))
Laufe/Gehe/Fahre eins zwei null Zentimeter nach vorne.	VP(V) GP(ZAHL EINH) APP
Bewege dich langsam zwei null Zentimeter nach vorn.	VP(V ADJ) GP (ZAHL EINH) APP(PP ADV)
Laufe/Gehe/Fahre in den Raum von Kai.	VP(V) OPP(PP NP(Det N PP NPERS))
Fahre/Gehe/Laufe etwas nach vorne.	VP(V) APP(PRO PP ADV)
Fahre/Gehe/Laufe etwas nach hinten.	VP(V) APP(PRO PP ADV)
Fahre/Gehe/Laufe etwas rückwärts.	VP(V) ADP(ADV ADV)
Fahre/Gehe/Laufe etwas vorwärts.	VP(V) ADP(ADV ADV)

Richtungsänderungen:

Drehe/Wende dich nach links/rechts.	VP(V PRO) APP(PP ADV)
Drehe/Wende dich X Grad nach links/rechts.	VP(V PRO) GP(ZAHL EINH) APP(PP ADV)
X Grad nach links/rechts drehen/wenden.	GP(ZAHL EINH) APP(PP ADV) VP(V)

Geschwindigkeitsänderung:

Schneller!	SR(S)
Langsamer!	SR(S)
Halt!	SR(S)
Stopp!	SR(S)
Anhalten!	SR(S)
Sofort anhalten!	SR(S)
Sofort stoppen!	SR(S)

Objekt transferieren

Hole den Becher aus dem Zimmer/Raum/Büro von Kai.	VP(V) NP(DET N) OPP(PP NP(Det N PP NPERS))

Bringe mir den Becher von Kai	VP(V) NP(DET N) OPP(PP NP(Det N PP NPERS))
Bringe mir den roten/blauen/gelben Becher aus dem Zimmer von Kai	VP(V) NP(DET N) OPP(PP NP(Det N PP NPERS))

Greifer steuern (nachträgliche Änderungen):

Öffne/schließe den Greifer.	VP(V) NP(Det N)
Greifer öffnen/schließen.	NP(N) VP(V)
Den Greifer öffnen/schließen.	NP(Det N) VP(V)
Den Greifer senken/heben.	NP(Det N) VP(V)
Senke/Hebe den Greifer.	VP(V) NP(Det N)
Bewege den Greifer nach oben/unten.	VP(V) NP APP(PP ADV)
Bewege den Greifer rauf/ab.	VP(V) NP ADP(ADV)

Die Abkürzungen NP, VP, PP etc. stehen im Abkürzungsverzeichnis.

Anhang C: Interface Datei zum Aktionsmodul

Unter Absprache mit [Diekmann/Gossen] haben wir die Schnittstelle für das Aktionsmodul folgendermaßen definiert:

Die Datei soll in einem festdefinierten Verzeichnis gespeichert werden, um vom Aktionsmodul gelesen werden zu können. In ihr werden alle Information, d.h. alle Aktionen und Parametern, gespeichert. Die Parameter, welche nicht verwendet werden, sind mit „`nil`“ markiert. Alle übrigen Parameter werden mit passenden Attributen belegt. Die Schnittstellendatei wird eingeleitet mit commandclass. Die Datei hat z. B. für „*fahre schnell nach vorne*“ folgende Attribut-Belegungen:

```
commandclass:motion
action:move
speed:fast
quantity:nil
direction:forward
distance:nil
target:nil
degree:nil
object:nil
do:nil
color:nil
room:nil
```

Anhang D: Mappingstruktur

Zur Zeit verfügt die Mappingtabelle über folgende Informationen. Sie kann gegebenenfalls erweitert werden.

Mapping : Tabelle

Semantik	Parameterklas	Aktionsklasse	Kommandokla	Optional
backward	direction	MOVE	MOTION	☐
blue	color	GET	MOTION	☑
close	do	GRIPPER	MOTION	☑
cm	distance	MOVE	MOTION	☑
cup	object	GET	MOTION	☐
default	speed	MOVE	MOTION	☑
deg	degree	TURN	MOTION	☑
down	direction	GRIPPER	MOTION	☑
fast	speed	MOVE	MOTION	☑
forward	direction	MOVE	MOTION	☐
left	direction	TURN	MOTION	☐
less	quantity	MOVE	MOTION	☑
more	quantity	MOVE	MOTION	☑
nil	nil	STOP	MOTION	☑
number	degree	TURN	MOTION	☑
number	distance	MOVE	MOTION	☑
number	target	GET	MOTION	☑
number	target	MOVE	MOTION	☑
open	do	GRIPPER	MOTION	☑
red	color	GET	MOTION	☑
right	direction	TURN	MOTION	☐
room	target	GET	MOTION	☑
room	target	MOVE	MOTION	☑
slow	speed	MOVE	MOTION	☑
up	direction	GRIPPER	MOTION	☑
yellow	color	GET	MOTION	☑
*				☐

Datensatz: 1 von 26

Wie man sieht, enthält die Tabelle nur eine Kommandoklasse MOTION. Es können jedoch, wenn die Grammatik und das Lexikon erweitert werden, mehrere Kommandoklassen hinzukommen. SET könnte eine Kommandoklasse zum setzen von bestimmten Statusvariablen des Roboters sein. Diese könnte man untergliedern in Setzen von Koordinaten oder Standardwerte für das Fahren. Ferner könnte, wenn die Grammatik ergänzt wird, auch Fragesätze verarbeiten.

LITERATURVERZEICHNIS

[ABNEY1] Abney Steven P., „Partial Parsing via Finite-State Cascades“, http://www.research.att.com/~abney/publications.html

[ABNEY2] Abney Steven P., „Parsing By Chunks“, http://www.research.att.com/~abney/publications.html

[ASBSSH] Sander, Stucky, Herschel, „Automaten und Formale Sprachen“, Teubner Stuttgart Verlag, 1995

[BADE] Bade Richard, „Der Chunkparser in Verbmobil“, http://www-ai.cs.uni-magdeburg.de/lehre/sommer-01/sprachtechnologie/

[Diekmann/Gossen] Arne Diekmann und Andrej Gossen, „Realisierung von Sprachsteuerung für Pioneer 1 Roboter“, Studienarbeit 2002 TI

[DRAGON] Dragon NaturallySpeaking SDK C++, http://developer.dragonsys.com/

[DUDGRAM] DUDEN, „ Die Grammatik“, Dudenverlag 1998

[ECL1] Prof. Dr. Michael Hess, „Einführung in de Computerlinguitsik 1“, 2001, http://www.ifi.unizh.ch/CL/hess/classes/ecl1/ecl1.l.0.pdf

[ECL2] Prof. Dr. Michael Hess, „Einführung in de Computerlinguitsik 2“, 2001, http://www.ifi.unizh.ch/CL/hess/classes/ecl1/ecl2.l.0.pdf

[EM96] Gamma, Helm, Johnson, Vlissides, „Entwurfsmuster“, Addison-Wesley, 1996

[FRIEND] Uni Bremen, Institut für Automatisierungstechnik (IAT), http://www.iat.uni-bremen.de/

[GCRHAUSSER] Hausser Roland, „Grundlager der Computerlinguistik- Mensch-Maschine-Kommunikation in natürlicher Sprache “, Springer Verlag, 20000

[GL] Gewehr Wolf, Klein Klaus-Peter, „Grundprobleme der Linguistik“, Pädagogischer Verlag, 1982

[GPL] König E., Seiffert R., „Grundkurs Prolog für Linguisten“, UTB Verlag, 1989

[HKI] Görz, Rollinger, Schneeberger, „Handbuch der Künstlichen Intelligenz“, Oldenburg Verlag, 2000

[IBM] IBM ViaVoice SDK, http://www-4.ibm.com/software/speech/dev/

[KANTRA] KANTRA: „Human-Machine Interaction for Intelligent Robots Using Natural Language”, http://www.dfki.uni-sb.de/vitra/papers/ro-man94/ro-man94.html

[KübHin] Kübler S. und Hinrichs E.W.; „Robustes Chunkparsing mit variabler Analysetiefe“, Seminar für Sprachwissenschaft. Ebehard-Karls-Universität, http://www8.informatik.uni-erlangen.de/IMMD8/Research/vm_papers/konvens00.pdf

[LEV] Damerau-Levenstein-Metrik Algorithmus, www.math.fu-berlin.de/teach/lect/ir/kap7.2up.ps.gz

[LuckLP] von Luck, Kai, „Logische Programmierung“, FH-Hamburg, Softwaretechnik, http://www.informatik.fh-hamburg.de/~luck/lectur.html

[MINERVA] Tourguide Roboter, Carnegie Mellon University , http://www-2.cs.cmu.edu/~minerva/

[MorikNS] Morik Katharina, „Natürlichsprachliche Systeme“, Uni-Dortmund, Fachbereich Informatik, Oktober 1995, http://www-ai.cs.uni-dortmund.de/LEHRE/VORLESUNGEN/NLS/nls.html

[MSDN] Microsoft Developtment Network, ist eine Referenz für Entwickler, die Microsoft-Entwicklungstools verwenden. Siehe auch im Internet unter: http://www.microsoft.com/msdn.

[MÜLLERS1] Müller Stefan, http://www.dfki.de/~stefan/Vorlesungen/2001/cl.html

[MÜLLERS2] Müller Stefan, „*Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*“, Tübingen: Max Niemeyer Verlag, 1999

[NLP] Naumann Sven, Langer Hagen, „Parsing – Eine Einführung in die maschinelle Analyse natürlicher Sprachen“, Teubner Stuttgart Verlag, 1994

[NURSEBOT] http://www.cs.cmu.edu/~nursebot/web/press.html

[PHILLPS] Phillips Speech SDK, http://www.speech.be.philips.com/pc/pages/pc_home.htm

[Pollard and Sag 87] Pollard Carl and Sag Ivan: *Information-based Syntax and Semantics. Volume 1: Fundamentals.* Stanford: University of Chicago Press. 1987

[Pollard and Sag 94] Pollard Carl and Sag Ivan: *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press. 1994

[PP] Marcus C., „Prolog Programming- Application for Database Systems, Expert Sytems and Natural Languages“, Addison Wesley Verlag, 1986

[SNS] Günther Görz „Strukturanalyse natürlicher Sprachen“, Addison-Wesley, 1988

[SPIDER] Allgemeine Info über SPIDER: http://nats-www.informatik.uni-hamburg.de/~spider

[SPIDER1] Projekt Dokumentation in Postscript: http://nats-www.informatik.uni-hamburg.de/~spider/doku/manual.ps

[SSAHLT1] „Survey of the State of the Art in Human Laguage Technology“, Kapitel 1, http://cslu.cse.ogi.edu/HLTsurvey/download.html, November 1995

[SSAHLT3] „Survey of the State of the Art in Human Laguage Technology“, Kapitel 3, http://cslu.cse.ogi.edu/HLTsurvey/download.html, November 1995

[SWI] „SWI Prolog“, http://www.swi.psy.uva.nl/

[SWICPP] „A C++ interface to SWI-Prolog“, Wielemaker Jan, 01.11.2000 Installationsverzeichnis von Prolog\doc\packages\pl2cpp.html

[VERBMOBIL] Maschinelle Übersetzung, http://verbmobil.dfki.de

[VOLKM] Volk Martin, Skript: „Vorlesung: Formale Grammatiken und Syntaxanalyse“, http://www.ifi.unizh.ch/~volk/

[WissVer] Jochen Heinsohn, Rolf Socher-Ambrosius., „Wissensverarbeitung“, Spektrum Verlag, 1999

[WAHLSTERVM] Prof. Dr. Wahlster W.(Projektleiter Verbmobil), „Erkennung, Analyse, Transfer, Generierung und Synthese von Spontansprachen“, Juni 1997, http://verbmobil.dfki.de/ww.html

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §21(7) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum Unterschrift