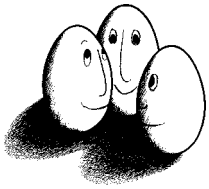
Diplomarbeit

Informationsextraktion aus
Freitext-Einträgen einer Datenbank

Markus Hölscher



Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

2. Dezember 1997

Betreuer:

Prof. Dr. Katharina Morik
Dipl. Inform. Thorsten Joachim

Zusammenfassung

Die vorliegende Arbeit befaßt sich mit Gewinnung von Daten aus natürlichsprachlichen Sätzen, die in einer unvollständigen und teilweise inkorrekten Form vorliegen.
Untersucht wird die Frage, ob durch eine syntaktische Analyse die in den Sätzen enthaltene Information in einen Frame übertragen werden kann, um zu einer strukturierten Darstellungsform zu gelangen.
Diese soll den Anwender in die Lage versetzen, einfach nach Sachverhalten in einer großen Anzahl von kurzen, notizartigen Texten zu suchen.
Durch das Erstellen einer Satzgrammatik werden die Eingabesätze in eine Form umgewandelt, die programmatisch weiterbearbeitet werden kann. Eine graphische Oberfläche wird dem Benutzer eine einfache Möglichkeit geben, Anfragen zu stellen und die Ergebnisse zu untersuchen.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Aufgabenstellung 2
- 1.2 verwandte Arbeiten 2

2 Natürlichsprachliche Systeme **4**

3 System **9**
- 3.1 Preprocessing 10
  - 3.1.1 Satzgrenzenerkennung 10
  - 3.1.2 Rechtschreibüberprüfung 10
  - 3.1.3 Phrasenerkennung 10
- 3.2 Syntaktische Analyse und semantische Representation 11
  - 3.2.1 Baumauswah 11
  - 3.2.2 Die semantische Representation 12
  - 3.2.3 Umsetzen der syntaktischen Beschreibung in die semantische Representation 12
- 3.3 Auswertung 12

4 Das System PCPATR **14**
- 4.1 Zwei-Ebenen-Modell 15
- 4.2 Das Lexikon 17
- 4.3 Wortgrammatik 19
- 4.4 Satzgrammatik 21
  - 4.4.1 Satzformen 22
  - 4.4.2 Verbalphrasen 23
  - 4.4.3 Nominalphrasen 25

5 Die Suchfunktionen und die Benutzeroberfäche **29**
- 5.1 Suche 29
  - 5.1.1 Volltextsuche 29
  - 5.1.2 Erweiterte Suche 30
- 5.2 Die Benutzeroberfläche 32
  - 5.2.0.1 Die Menüleiste 32

5.2.0.2 Der Bedienteil 32
5.2.0.3 Das Eingabefeld 33
5.2.0.4 Der Anzeigenteil 34

6 Implementierung **37**
6.1 Überführung der Strukturbeschreibungen 37
6.1.1 Baumauswahl 38
6.1.2 Die Überführung der einzelnen Satztypen 39
6.2 Auswertung 40
6.2.1 Die Objekte zur Framestruktur 40
6.2.2 Die Objekte der Oberfläche 43
6.2.3 Such- und Statistikobjekte 44

7 Ergebnisse **47**
7.1 Experimente 47

8 Zusammenfassung **52**

Abbildungsverzeichnis

2.1 NLS Architektur 6
2.2 Syntaktische Mehrdeutigkeiten 7

3.1 Die Systemarchitektur 9

4.1 Abbildung eines Wortes auf Morpheme 16
4.2 Abgewandelter Automat 17

5.1 Zukünftige Queryaltenativen 31
5.2 Die Benutzeroberfläche 35
5.3 graphische Framedarstellung 36

6.1 Objekte der Framedarstellung 41
6.2 OrList 42
6.3 Objekte des GUI 44
6.4 TreeView 45
6.5 CardLayout 45

7.1 Parsestatistik I 48

Kapitel 1

Einleitung

Die Verbreitung wie auch die Komplexität von Produkten aus dem Bereich der Informationstechnologien (IT) steigt stetig an, während viele Endbenutzer nur vergleichsweise wenige Kenntnisse von IT-Produkten besitzen. Dadurch gewinnt der Kundenservice zunehmend an Bedeutung, um den Ansprüchen der Anwender zu genügen. Mit dem weiter wachsenden Ausbau der Serviceleistungen für Endbenutzer und dem einhergehenden Einsatz von Softwarewerkzeugen beim Dienstleister, wächst die zu verarbeitende Informationsmenge stetig an. Dazu gehören speziell im Dialog mit dem Kunden erfaßte Daten. Diese müssen oft während einer Beratung erfaßt werden, aber gleichzeitig noch einige Zeit später verständlich sein. Oft können solche Informationen nicht mit speziell entwickelter Software erfaßt, und insbesondere nicht in ein geeignetes Datenformat gebracht werden, sondern nur protokolliert werden. Dazu wird meistens eine Form der (geschriebenen) Alltagsprache benutzt, um die anfallenden Angaben als Texte zu speichern. Aus Zeitmangel ergiebt sich oft eine grammatikalisch falsche bzw. unvollständige Aufzeichnung. Der Zugriff auf diese Informationen ist wegen der unstrukturierten Form nur in eingeschränktem Maß automatisierbar. Daraus folgt, daß momentan neben der (automatischen) Volltextsuche eine weitere Aufarbeitung nur manuell erfolgen kann. Methoden des Informationretrievals scheitern hier, da sie nicht die sprachlichen Strukturinformationen ausnutzen können. Diese sind für ein Erkennen von Sinnzusammenhängen unerläßlich. Bei einem einfachen Suchen nach Stichwörtern, werden viele Texte gefunden, die zwar die nachgefragten Wörter enthalten, aber trotzdem nicht zu der vom Benutzer gewünschten Zielmenge gehören.
Bei einem Aufkommen von mehreren tausend protokollierten Telefonaten im Monat, wie sie in Servicezentralen von großen IT-Anbietern anfallen können, steht eine manuelle Auswertung in keinem aktzeptablen Kosten-/Nutzenverhältnis. Da aber die schnelle Bereitstellung von Informationen immer mehr zu einem entscheidenden Faktor bei der Steigerung von Servicequalität und der Entwicklung von neuen Dienstleistungen wird, kann die computergestützte Auswertung solcher Daten, d.h. ein Informationsgewinn aus vorhandenem Wissen, einen Wett-

bewerbsvorteil bedeuten [1] .

1.1 Aufgabenstellung

Gegenstand der Untersuchung sind Gesprächsnotizen in englischer Sprache, die ein festumrissenes technisches Themengebiet behandeln. Diese sind zu Texteinheiten von wenigen Sätzen zusammengefaßt, von denen mehrere hundert untersucht werden sollen.
Das erste der beiden Ziele dieser Diplomarbeit ist nun, ein System zu entwerfen und zu implementieren, das die in (mehr oder weniger) natürlicher Sprache gespeicherten Sätze. in eine inhaltsbezogene Struktur (Frame) transferiert, in der sie automatisch weiterverarbeitet werden können. Da es sich um vom Standardenglisch abweichende Sätze handelt (stichwortartige Notizen), muß eine Grammatik speziell für die Anwendung erstellt werden.
Das zweite Ziel ist, dem Benutzer ein Werkzeug zur Verfügung zu stellen, mit dem er in der Lage ist, über den transformierten Daten nach selbstdefinierten Mustern zu suchen und dabei die im Frame gespeicherte Strukturinformation auszunutzen. Weiter soll er in die Möglichkeit erhalten, nicht nur in den Frames enthaltene Informationen zu finden, sondern auch dabei unterstützt werden, neues Wissen aus den Daten zu gewinnen.
Dazu soll eine graphische Oberfläche dem Anwender eine einfache Bedienung ermöglichen.
Da beide Aufgaben eigentlich beträchtlich mehr Arbeitsaufwand erfordern als eine Diplomarbeit erlaubt, soll es hier darum gehen, die Machbarkeit durch die Entwicklung eines Prototypen zu untersuchen.

1.2 Verwandte Arbeiten

Die Wissensentdeckung in großen Textsammlungen läßt sich in Arbeiten aufteile, die Texte als Wortvektoren oder als korrelierte Worttupel (n-Gramme), und solche, die die sprachliche Struktur beachten.
Das System SCISOR sucht aus einem Textstrom vom Wirtschaftsdaten die Texte heraus, die sich mit Firmenübernahmen befassen [Jacobs]. Dabei wird mit Hilfe eines Wortmusterabgleiches ein Filtering durchgeführt. Aus als relevant erkannten Texten werden die Informationen extrahiert, die zu einem festgelegten Frame

[1] "Wir müssen lernen, Dinge schneller zu tun. Neue Geschäftsfelder aufzutun und zu reagieren, Programme zu implementieren, neue Kompetenzen zu entwickeln, Entscheidungen zu treffen. (...) Die Geschwindigkeit wird ganz klar eines der Dinge sein, die über unseren Erfolg entscheiden" [Livermoore, 1997].

gehören. Danach wird eine syntaktische Analyse durchgeführt. Informationen, die nicht in einen Slot passen, werden ignoriert.
Die Message Understanding Conference (MUC) [MUC, 1993] Die Unterschiede zu der aufgezeigten Arbeiten liegen zum einen in der Beschaffenheit der Texte. Die Systeme, die für MUC entwickelt wurden, sollten Zeitungartikel über Terrorakte auswerten. Diese unterscheiden sich hinsichtlich Länge und syntaktischer Korrektheit stark von den hier vorliegenden Texten.
Bei beiden Quellen wird Text über Satzgrenzen hinaus in Beziehung gesetzt. Diebesetzung der Slots wird in einem weit größeren Maße von zusätzlichen semantischen Bedingungen abhängig gemacht.
Einen Einblick in das Feld von information Extraction bietet [Cowie, 1996]
Will man die sprachliche Struktur verwenden, so nimmt die Implementation einer Satzgrammatik zur syntaktischen Analyse der Eingabesätze einen großen Teil der Entwicklungszeit ein. Erik Dörnenburg hat in seiner Diplomarbeit untersucht [Dörnenburg, 1997], ob es möglich ist, solch eine Grammatik von einem Programm lernen zu lassen. Leider kam er zu dem Ergebnis, daß es unmöglich ist diese Lernaufgabe, mit dem von ihm untersuchten Algorithmus, in einer vertretbaren Zeit zu erreichen.

Die Arbeit wird nach einem kurzen Einblick in die natürliche Sprachverarbeitung die Systemarchitektur vorstellen und den benutzten Parser samt Lexikon beschreiben. Einen zentralen Teil bildet die Grammatik, mit deren Hilfe die Syntax eines großen Teils des betrachteten Ausschnitts der englischen Sprache erkannt werden kann. Daran schließt sich eine Darstellung der Systemfunktionen und deren Implementation an. Den Schluß bilden die Vorstellung der Meßergebnisse und eine Zusammenfaßung der gewonnenen Erkenntnisse plus Ausblick auf mögliche Erweiterungen und Einsatzmöglichkeiten.
An dieser Stelle möchte ich die Gelegenheit nutzen all denjenigen zu danken, ohne die die Erstellung dieser Arbeit nicht möglich gewesen wäre. Danken möchte ich vor allem meinen beiden Betreuern Prof. Dr. Katharina Morik und Dipl. Inform. Thorsten Joachims, die mir wertvolle Anregungen zur Gestaltung dieser Diplomarbeit gegeben haben. Zu Dank verpflichtet bin ich meinem Freund Gary Totney, der mir manche Feinheit der englischen Sprache erklärt hat, ebenso wie meiner Mutter, ohne deren Unterstützung diese Arbeit nicht geschrieben worden wäre. Alle Mitarbeiter und Studenten des Lehrstuhls VIII der Universität Dortmund haben durch ein freundschaftliches Arbeitsklima, viele Tips und klagloses Korrigieren meiner Schreibversuche zu dem Gelingen dieser Arbeit beigetragen. Mein Dank gilt ebenso der Firma Hewlett Packard, im besonderen Herrn Markus Bär und Frau Feodora Herrmann. Durch ihr Engagement haben sie die Voraussetzungen für einen erfolgreichen Verlauf dieser Arbeit geschaffen.

Kapitel 2

Natürlichsprachliche Systeme

Der Mensch benutzt tagtäglich die Sprache, um mit seinen Mitmenschen zu kommunizieren. Da jeder Mensch, dank seiner angeborenen Sprachfähigkeit, die Sprache (relativ) problemlos lernt, macht man sich selten klar, welch ein aufwendiger Prozeß nötig ist, um ein solch komplexes Gebilde, wie das der natürlichen Sprache zu handhaben. Es ist für den Menschen normal, eben natürlich, mittels der Sprache zu kommunizieren. So erscheint es einsichtig, daß der Einsatz von Systemen, die diese Sprache verarbeiten können, den Umgang mit Computern ungemein vereinfacht.
Exemplarische Anwendungen von natürlichsprachlichen Systemen (engl. Natural Language Systems - NLS) sind z.B.:

- natürlichsprachliche Schnittstellen zu dialogogesteuerten Systemen (DB-Anfragen, sprachliche Systemantworten)
- textverstehende Systeme
- Übersetzungssysteme

Wie allgemein in der KI, gibt es auch auf dem Gebiet der natürlichen Sprachverarbeitung verschiedene Ansätze, unter denen einzelne Systeme entwickelt werden. Dazu zählen kognitionsorientierte Arbeiten, mit denen versucht wird sprachliche Prozesse beim Menschen nachzumodellieren.
Der Beschreibungsansatz will eine operationale Beschreibung von Sprache geben, die aber anders als Sprachverstehen des Menschen funktionieren kann. Dabei werden NLS als Hilfsmittel zur linguistischen Theoriebildung benutzt.
Als Spezialisierung des Ingenieursansatzes, geht es beim 'linguistic engineering' darum, für eine bestimmte Anwendung eine Benutzerschnittstelle zu entwickeln. Dabei muß das NLS meist nur einen vor der Entwicklung definierten Ausschnitt der Sprache verarbeiten können.

Hier soll nun eine kurze Einführung in einige Begriffe gegeben werden, die für ein NLS grundlegend sind. Darauffolgend wird die Architektur eines solchen Systems dargestellt werden.
"**Natürlichsprachliche Systeme** analysieren bzw. generieren natürliche Sprache, wobei sie Wissen über Sprache verwenden." [Morik, 1996]
Die Sprache läßt sich auf verschiedenen aufeinander aufbauenden Ebenen untersuchen, für die in einem NLS entsprechende Operationalisierungen zur Verfügung gestellt werden müssen. Es ist nicht zwingend notwendig, die unten dargestellten Komponenten in einer Implementierung strikt zu trennen. Die nun folgenden Elemente eines Systems zur Verarbeitung natürlicher Sprache stellen die notwendigen Basiskomponenten dar.
Die **Morphologie** erkennt und beschreibt einzelne u.U. zusammengesetzte Wörter anhand von Merkmalen, wie Flektion, Kasus, Genus u.a. Dazu greift sie auf ein Lexikon zurück, in dem die benötigten Daten gespeichert sind. "Die Lehre von den formalen Wortausprägungen und den Wortbildungsprozessen bildet zusammen die Morphologie; man kann sie als die Lehre vom Bau der Wörter bezeichnen". In [Linke, 1991]
Die Lehere von der Struktur der Sätze und ihrer Teile heißt **Syntax**. Hier wird untersucht, ob ein Satz anhand der Regeln einer Grammatik formal korrekt aufgebaut ist. Um zu prüfen, ob die Eingabe der Grammatik genügt wird ein Parser eingesetzt.
Semantik ist die Lehre von der Bedeutung eines Wortes oder eines Satzes (bei einem NLS: die Bedeutung der Eingabe). Sie beschreibt den Zusammenhang zwischen den Zeichen und den damit gemeinten Gegenständen bzw. Sachverhalten. Bei der computerisierten semantischen Interpretation wird vorausgesetzt, daß es eine Reprasentation von Bedeutung gibt. Dabei taucht häufig das Problem auf, daß ein Wort verschiedene Bedeutungen besitzen kann und entschieden werden muß, welche Version für eine sinnvolle Satzkonstruktion ausgewählt werden soll. Doch dies kann nicht immer nur anhand eines vorliegenden Satzes entschieden werden.
Die **Pragmatik** untersucht das Verhältnis von Sprache und dem Kontext in dem sie geäußert wird. Hier wird die kontextabhängige Bedeutung von Wort oder Satz behandelt. Dazu gehören unter anderem Deixis, Präsuppositionen, Ellipsen , Anaphora und Konversationsimplikaturen. Näheres siehe [Levinson, 1983].
Wie die einzelnen Teile zusammenwirken, zeigt folgende idealisierte Architektur eines sprachverarbeitenden Systems (Bild 2.1) . Dabei wird das Zusammenspiel der oben vorgestellten Komponenten und die benutzten Wissensquellen, die sich nicht auf rein sprachliches Wissen beschränken, vorgestellt.

In dem in Bild 2.1 vorgestellten NLS muß der Verarbeitungsfluß nicht notwendigerweise sequentiell ablaufen, sondern kann durchaus Schleifen enthalten, z.B.

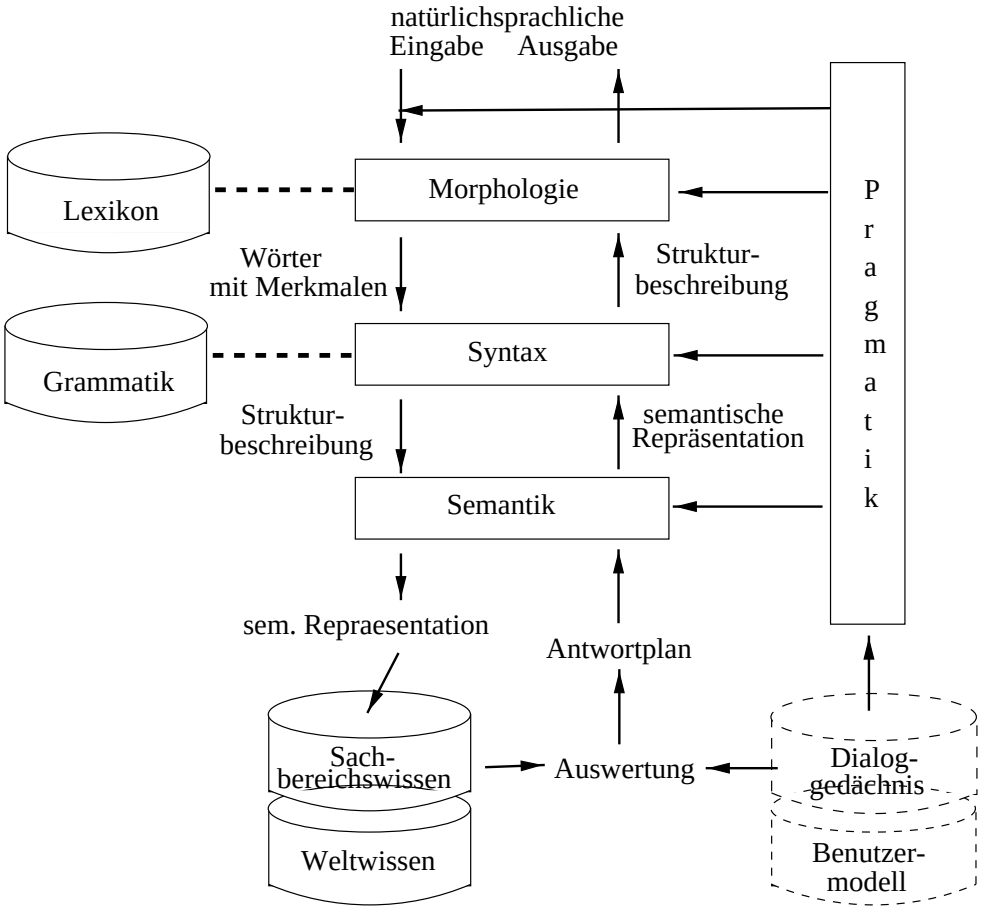


Abbildung 2.1: Die Architektur eines NLS

wenn die Semantik-Komponente eine syntaktisch korrekte aber semantisch falsche Aussage zurückweist. Soll gesproche Sprache benutzt werden, wird vorher zusätzlich noch eine phonologische Komponente benötigt, aus deren Ausgabe von der Morpholgie die einzelnen Wörter identifiziert werden sollen. Hier kann u.U. die Pragmatik unvollständige Eingaben oder Versprecher korrigieren helfen, wenn sie nicht in Phonologie oder Morphologie integriert ist.
Bei der Sprachanalyse prüft die Morphologie die Eingabe unter Nutzung des im Lexikon gespeicherten Wissens über die einzelnen Wörter, und bestimmt für jedes Wort die grammatikalischen Informationen, die am einzelnen Wort erkennbar sind. Die Implementation kann mit Hilfe einer eigenen Wortgrammatik realisiert werden oder komplett als Vollformenlexikon vorliegen (siehe Kapitel 4.2). Dieses Ergebnis wird an die syntaktische Analyse weitergereicht, wo ein Parser, mit Hilfe einer für die benutzte Sprache geeigneten Grammatik, die möglichen syntaktischen Interpretationen des Eingabesatzes generiert. Die Grammatik besteht aus Regeln, die festlegen, wie ein Satz in Kategorien zerlegt werden kann. Die Kategorien ihrerseits können weiter in Subkategorien zerlegt werden, bis zu dem Punkt, an dem es möglich ist, einer (Sub-)kategorie ein einzelnes Wort zuzuord-

nen. Wie hier schon angedeutet wurde, kann es auf jeder Ebene zu Ambiguitäten kommen. In einem solchen Fall werden mehrere Ergebnisse an die nachfolgende Schicht geliefert, die vielleicht eine richtige Auswahl treffen kann.
Die Morphologie könnte für das englische Wort 'views' z.B. einmal das Nomen im Plural 'Sichten' erkennen und andererseits das transitive Verb im Präsens 'anschauen'. In dem Satz 'He views the mail' könnte die Syntaxkomponente (bei entsprechenden Regeln der Grammatik) das Nomen verwerfen und das alternative Verb wählen. Aber es können auch syntaktische Mehrdeutigkeiten auftreten. 'He views the mails with an attachment' kann syntaktisch korrekt erkannt werden als:

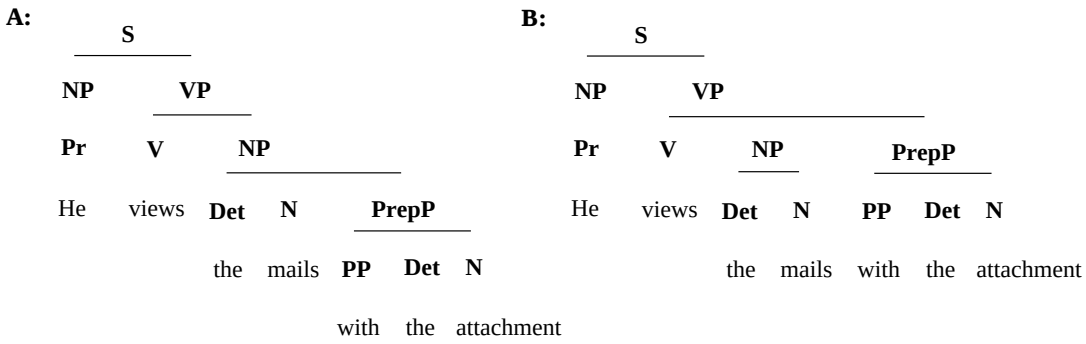


Abbildung 2.2: zwei syntaktisch korrekte Zerlegungen desselben Satzes

Die Auswahl, welche der beiden erkannten syntaktischen Zerlegungen hier gewählt wird, muß nun der semantische Teil des Systems treffen Dazu ist es notwendig, daß eine Wissensbasis existiert, die das für eine solche Entscheidung notwendige Wissen enthält. Nach der Auswahl einer syntaktischen Interpretation, wird daraus eine sprachunabhängige semantische Repräsentation gewonnen. Hier, wie auch bei der Generierung einer Systemantwort kann es ohne eine Prakmatikkomponente teilweise unmöglich werden, die Bedeutung eines Satzes zu erfassen. Als Bsp. für die Notwendigkeit auf Weltwissen zurückgreifen zu können und pragmatische Mehrdeutigkeiten auflösen zu können, kann folgender kleiner Text dienen (leicht abgewandelt, nach Charniak, 1972 in [Levinson, 1983])

> Jill wanted to get Bill a birthday present. Jane too. Jill went and found her piggy-bank; she shook it, but there was no noise; she would have to make Bill a present.

Ohne Wissen über Geschenke, Sparschweine und Geld, wozu u.a. Zweck und Beschaffenheit zählen, wäre es nahezu aussichtslos, diesen Text semantisch zu repräsentieren. Weiter kann ein Mensch problemlos den Sinn der Ellipse 'Jane too' erfassen und etwa zu 'Jane wants to get Bill a present' erweitern. Auch

ist mit dem Pronomen 'she' in diesem Fall offensichtlich Jill gemeint. Für ein Computerprogramm keine leichte Aufgabe.
Soll das NLS mit dem Benutzer einen Dialog führen oder eine Antwort auf eine Anfrage liefern, muß eine Auswertung des erkannten Satzes erfolgen, aus der unter Einbindung eines Dialoggedächnisses ein Antwortplan erzeugt wird. In dem wird festgehalten, wie eine dem Benutzer (ein zusätzliches Benutzermodell vorausgesetzt) bzw. dem Kontext angemessene Antwort aussehen soll Dieser Plan wird von der Semantik in eine semantische Beschreibung überführt, die als Eingabe vom Syntaxteil genutzt wird. Das Ergebnis ist eine Strukturbeschreibung, die von der morphologischen Komponente mit Wörten gefüllt wird und ggf. von der Phonetik in gesprochene Sprache umgewandelt wird. Auch hier ist eine pragmatische Komponente nützlich, um u.a. Ellipsen und Anaphora zu erzeugen. Dadurch wird die Bedeutung nicht geändert, sondern nur der natürlichen Sprechweise angeglichen, um dem Benutzer, wie hier, von monotonen Satzauflistungen zu verschonen.

Kapitel 3

System-Architektur

Das hier entwickelte System kann man grob in drei Teile gliedern, die im folgenden näher vorgestellt werden. Das Ergebnis jedes Teiles ist die Eingabe des folgenden Schrittes, wobei das Gesamtsystem keine Ketten von Prozessen darstellt, die ineinandergreifen, sondern jeweils die gesammte Ausgabemenge bearbeiten. Für die Auswertung ist die gesammte Datenmenge notwendig, während zwischen dem Preprocessing und dem Parsen/Mappen eine Verbindung via Pipes o.ä. vorstellbar ist.

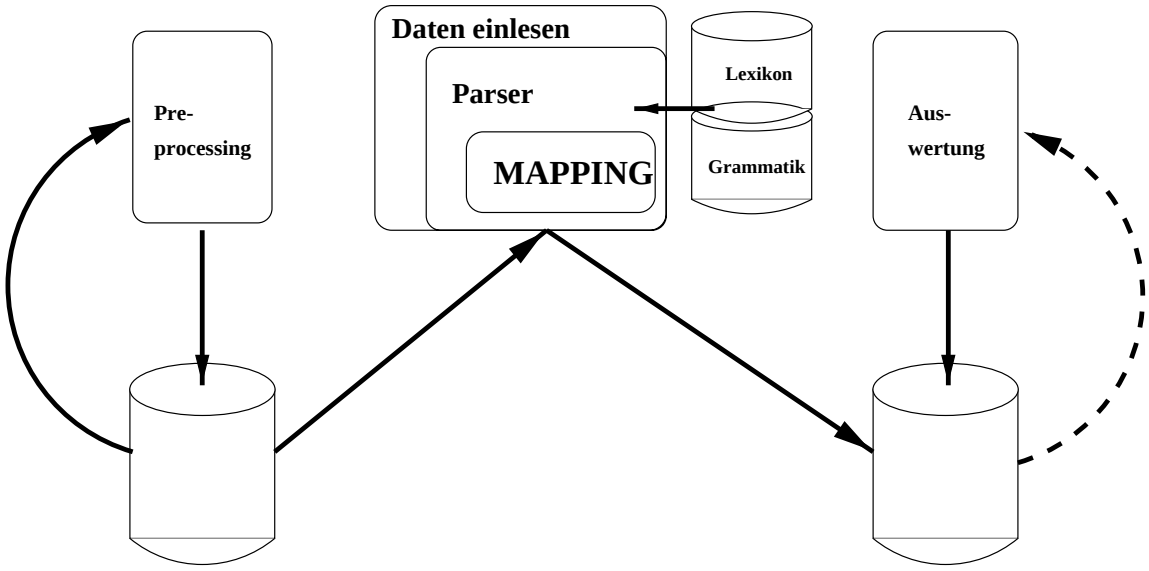


Abbildung 3.1: Systemarchitektur

3.1 Preprocessing

Das Behandeln der Eingabetexte vor der Weitergabe an die Satzgrammatik dient der Verbesserung der Qualität des Parsens und der Zerlegung der Texte in einzelne Sätze, die die Grammatik als Eingabe erwartet.
Das Preprocessing ist in dieser Arbeit nicht implementiert worden, sonder bei einen Einsatz der Software muß der Eingabetext dementsprechend schon in einzelne Sätze gegliedert sein.

3.1.1 Satzgrenzenerkennung

Da die erstellte Grammatik einen Satz als Eingabe erwartet, die auszuwertenden Texte aber nicht satzweise gespeichert sind, ist eine Instanz vonnöten, die in den Texten die einzelnen Sätze identifiziert.
Eine typische Texteinheit könnte folgendermaßen lauten:
file list problem - student can't access files... he need to retieve the specifieded .xyz files.
Man kann sofort erkennen, daß ein Trennen des Textes nach einem Punkt nicht ausreicht. Viele Sätze werden durch andere Zeichen(-folgen) getrennt, während ein Punkt bei Dateinamen erhalten bleiben soll.
Versuche mit einem Part-of-Speech Tagger erwiesen sich als erfolglos, da der verwendete Tagger der Statistical Language Modelling Group von der Carnegie Mellon University für wohlgeformte Texte ausgelegt ist und auch einzelne Sätze der Domäne zum großteil falsch erkannte.

3.1.2 Rechtschreibüberprüfung

Da durch viele Rechtschreibfehler das korrekte Parsen erschwert, bzw. unmöglich gemacht wird, verbessert eine Rechtschreibüberprüfung (Spellcheck) die Parseergebnisse sowie die gewonnen Wortinformationen. So wird der Satz *he needs to* **retieve** *the data* in der falschen Schreibweise nicht geparst, in der richtigen Fassung aber sehr wohl. Diese Problematik tritt speziell bei nicht erkannten Verben auf, denn so entsteht durch die Annahme es handele sich um ein Nomen, oft eine Wortkombination die nicht erkannt werden kann Dies tritt häufig auf bei einer ursprünglichen Infinitivkonstruktion, wie im letzten Beispiel sichtbar wurde. Auch kann für falsch geschriebene Wörter keine Zusatzinformation angegeben werden, die für richtig geschriebene Wörter durch das Lexikon zur Verfügung gestellt wird.

3.1.3 Phrasenerkennung

Oft tauchen in den Texten Begriffe auf, die aus mehreren Wörtern bestehen. Das kann zu unerwünschten Ergebnissen führen, wie z.B. bei *file list problem*. Syntaktisch richtig soll hier eine Nominalphrase erkannt werden. Da aber *list* auch ein

Verb ist, existiert noch die Möglichkeit, einen Aussagesatz zu erkennen. Dabei ist *file* das Subjekt und *problem* das Objekt. Um eine aufwendige (semantische) Lösung des Problems in der Grammatik oder bei der Wahl der besten Parsebäume zu vermeiden, sollte mit einem einfachen Stringmatching die Texte auf solche Phrasen geprüft werden, und durch einen korrespondierendes eindeutiges Wort ersetzt werden. An diesem Wort kann dann auch im Lexikon eine entsprechende zusätzliche Information hinterlegt werden, die aus den einzelnen Phrasenbestandteilen nicht erkennbar ist. Für das letzte Beispiel wird *file list* durch *filelist* eretzt. Daran kann dann im Lexikon weitere Informationen gespeichert werden.

3.2 Syntaktische Analyse und semantische Representation

Der Parser wird von einer Programmschleife umgeben, die einen Satz einliest und dann den Parser aufruft mit den Eingabesatz als Parameter aufruft. Bevor der Satz geparst werden kann, werden die syntaktischen Kategorien der einzelnen Wörter mit Hilfe eines Lexikons ermittelt. (Siehe nächstes Kapitel) Da der Parser ihm unbekannte Wörter nicht verarbeiten kann, wird hier jedes unbekannte Wort (Zeichenkette) als Nomen angesehen und eine entsprechende Datenstruktur generiert, das zusammen mit den übrigen Wörtern geparst wird. Das Ergebnis des Parsers ist dann ein Parameter für die Mappingroutine, die zuerst einen Parsebaum aus den erzeugten Bäumen auswählt, der dann in einen Frame umgesetzt wird.

3.2.1 Baumauswahl

Durch in der Regel mehrdeutigen Parsemöglichkeiten gibt es eine Komponente, die versucht den (einen) korrekten Parsebaum auszuwählen. In diesem System wurde nur eine einfache Heuristik implementiert, die bei der Erzeugung verschiedener Bäume für einen Satz den einen Satztype einem anderen vorzieht. Dabei gibt es keinen alle Satztypen umfassenden Abgleich. Vielmehr werden nur bestimmte Satztypen betrachtet, da dort während der Entwicklung auffällig häufig ein inkorrekter Baum gemappt wurde. Dabei handelt es sich um Sätze, die alleine aus einer Nominalphrase bestehen, und Wenn/dann-Sätze. Erstere werden nur ausgewählt, wenn kein anderer Satztyp aus dem Eingabesatz gewonnen werden konnte. Daraus ergibt sich allerdings ein Problem mit Wenn/dann Sätzen der Form *TRS -> NP VPrel*, wenn die NP aus einem einzelnen Nomen besteht, das auch als Verb gebraucht wird. In den Testtexten erwieß sich dieses Problem jedoch als gravierend, da hiervon nur verbalisierte Nomen betroffen waren, die formal nicht als Verben angesehen werden, aber in der Umgangssprache dazu 'umfunktioniert' worden sind. Ein Beispiel ist das Wort *error*, das u.a. abkürzend aus *System indicates an error* den Ausdruck *System errored* zuläßt.

3.2.2 Die semantische Representation

Ein Frame ist eine Datenstruktur, die ein bestimmtes Konzept representiert. Er besteht aus Slots, in denen bestimmte Merkmalsausprägungen gespeichert werden können. Es soll hier nur die prinzipielle Machbarkeit aufgezeigt werden, einen Frame aufgrund einer syntaktischen Satzzerlegung plus Begriffsinformationen aus dem Lexikon zu füllen. Darum erinnert die Struktur des Frames stark an eine erweiterte Satzstruktur. Neben den syntaktischen Satzkategorien, die sich in dem verwendeten Frame wiederfinden, existieren die Slots PURPOSE und RESULT, in die wieder ein Frame eingetragen wird. Hier wird die Satzstruktur ausgenutzt, um Informationen über die Wortdaten hinaus zu gewinnen. Ausgelassen wurden Slots, die durch die Kombination von syntaktischem Wissen (wer ist Subjekt/Objekt) in Verbindung mit Lexikoninformationen realisiert werden.
Die Struktur des einzigen hier benutzten Frames einspricht der Darstellung der Abb. 6.1. Die Struktur ist rekursiv, so daß beliebig komplexe Sätze dargestellt werden können.

3.2.3 Umsetzen der syntaktischen Beschreibung in die semantische Representation

Der ausgewählte Parsebaum wird zuerst in seine Teilsätze getrennt, wenn er kein Wenn/dann-Satz ist. Mit den TeilSätzen werden dann einzeln e Frames gefüllt. Wenn/dann-Sätze werden nicht aufgesplittet, da sie in einen einzigen Frame gemappt werden. Im nächsten Schritt wird der Satztyp eines Satzes festgestellt. Er wurde in der Grammatik im Merkmal *senttype* gespeichert. Entsprechend dem Ergebnis der Feststellung wird die zu jedem Satztyp existierende Prozedur aufgerufen, die einen initialen Frame erzeugt. Dann werden die einzelnen Slots des Frames gefüllt. Die Prozeduren sind analog zu den Grammatikregeln aufgebaut, wobei sich die Bedeutung von Kategorien je nach Satztyp ändern kann.

3.3 Auswertung

Das Programm zur Auswertung der generierten Framestrukturen ist eine standalone Application. Sie greift lesend auf eine schon erzeugte Framemenge zu, in der nach vom Benutzer spezifizierbaren Mustern gesucht werden kann. Gefundene Treffermengen können als neuer Datensatz wieder abgespeichert werden. Zu Sätzen, die dem Suchmuster entsprechen, werden Werte von Slots gesammelt, die der Benutzer definieren kann. Diese Daten werden kumuliert, so das der Anwender Zusammenhänge zwischen verschiedenen Slots, durch Häufung von Slotwerten erkennen kann. Damit dies effektiv geschehen kann wurden zu Domainrelevanten Wörtern Informationen gespeichert, die eine Begriffshierachie bilden. Dadurch können Synonyme definiert werden und nach Oberkategorien gesucht werden.

Gesteuert wird das Programm über eine graphische Benutzeroberfläche, die eine einfache Bedienung, insbesondere der Suchmustereingabe gewährleistet.

Kapitel 4

Das System PCPATR

In dieser Arbeit wird als Grammatikformalismus PATR II [Shieber, 1986] benutzt. Diese Beschreibungssprache ermöglicht die Erstellung von unifikationsbasierten **kontextfreien Phrasen-Struktur-Grammatiken** (CF-PSG), mit deren Hilfe Eingabesätze von einem zugehörigen Parser in eine (oder mehrere) syntaktische Struktur(en) überführt wird.
Eine **unifikationsbasierte Grammatik** ist eine Grammatik, die zur Behandlung syntaktischer Strukturen Merkmale und ihre Werte verwendet, wobei die einzige Operation, die Merkmalen Werte zuweist, die Unifikation ist. [Morik, 1996]
Die Verwendung von Mermalen hat den großen Vorteil, daß viele spezielle Regeln auf eine einzige Regel mit Merkmalsstrukturen hin abgebildet werden können. Dabei können so viele spezielle Regeln eingespart werden, wie das entsprechende Merkmal Werte besitzt.
Als **Beispiel:** können die drei links stehenden Regeln unter Verwendung eines Merkmales Genus mit den Werten feminin, maskulin und neutrum durch die rechte Regel ersetzt werden.

$$\left.\begin{array}{l} NP \rightarrow ART_{fem} NOM_{fem} \\ NP \rightarrow ART_{mas} NOM_{mas} \\ NP \rightarrow ART_{neut} NOM_{neut} \end{array}\right\} \begin{array}{l} NP \rightarrow ART\ NOM \\ ART\ Genus = NOM\ Genus \end{array}$$

Dabei definiert *ART Genus = NOM Genus* eine Bedingung, die sicherstellt, daß nur Artikel und Nomen mit dem gleichen Genus als richtig akzeptiert werden. Dies wird durch die Unifikation gewährleistet. Sobald der Wert des Merkmales für eine Konstituente ermittelt wurde, kann die Regel erfolgreich ausgeführt werden, wenn für die andere Konstituente der gleiche Merkmalswert erkannt wird. Wenn das nicht möglich sein sollte, wird versucht, einen anderen Wert für die erste Konstituente zu gewinnen, mit dem dann noch einmal versucht wird, Gleichheit mit den restlichen Teilen der Regel zu erzielen. Mittels Unifikation ist es zudem möglich Merkmalen Werte zu geben. Erweitert man die rechte Regel um die Zeile *NP Genus = Nom Genus*, so wird dem Merkmal Genus der Nominalphrase der entsprechende Wert von Genus des Nomens zugewiesen, sobald er bekannt ist.

Die benutzte Implementation von PATR II heißt PCPATR vom Summer Institute of Linguistic [SIL] und ist für MS-DOS, MS-Windows, Macintosh sowie für Unix Systeme erhältlich. Weiter nutzt PCPATR einen morphologischen Parser namens PCKIMMO und ein fertiges Lexikon samt vorbereiteter Grammatik und morphologischen Regeln (ebenfalls bei [SIL] erhältlich als ENGLEX). Das Lexikon besteht unter anderem aus ca. 20.000 Einträgen mit Affixen, Wortstämmen und Partikeln, darunter Artikel, Konjunktionen, Präpositionen u.ä.. Es ist kein Vollformenlexikon, sondern PCKIMMO zerlegt mit Hilfe von phonologischen Regeln die Eingabewörter in (eine) Folge(n) einzelner Morpheme [1]. Diese Morphemfolgen werden dann mittels einer in PATR II geschriebenen Wortgrammatik wieder zu zulässigen Worten zusammengesetzt. Zu denen wird jetzt die zum späteren Parsen eines Satzes notwendige Informationen ermittelt. Dazu gehören u.a. die Wortkategorie, (Verb, Nomen, Adverb usw.), Flexionsformen (bei Verben z.B. 3. Person singular Vergangenheit) sowie optional im Lexikon abgelegte Informationen. So ist es möglich, mit einem relativ kleinen Lexikon auf sehr viele Wörter in ihren unterschiedlichen Flexionen zuzugreifen. Die Satzgrammatik wurde ebenfalls in PATR II geschrieben. Durch sie wird die Menge der 'atze definiert die erkannt und bearbeitet werden können. Dazu wird ein Parser benötigt, der versucht zu erkennen, ob eine Eingabe nach den Regeln erkannt werden kann.
"Ein Parser ist ein Mechanismus der prüft, ob eine Eingabe einer zugrundeliegenden Grammatik genügt." [Shapiro, 1087]. Bzw. " (...) ein (berechenbarer) Mechanismus, der eine Grammatik und eine Kette von Wörtern erhält und entweder eine Struktur über der Wortkette generiert, wenn diese nach der Grammatik korrekt ist, oder Nichts". Aus [Gazdar/Mellish,]
Im Folgenden wird ein kurzer Überblick über die benutzten Bestandteile des PCPATR Systems gegeben.

4.1 Zwei-Ebenen-Modell

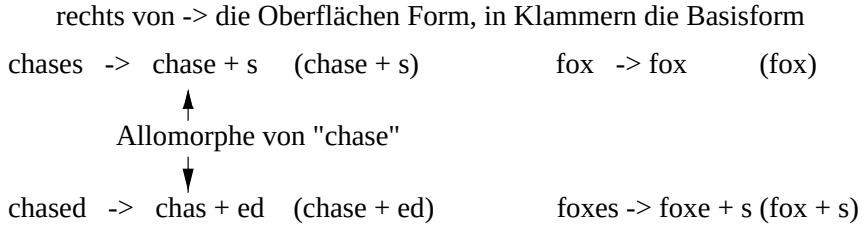
Die phonologische Analyse von PCKIMMO

Es geht dabei darum, ein Eingabewort in einzelne Wortbestandteile zu zerlegen, um so durch weitere Verarbeitungsschritte morphologische Informationen über das Wort zu bekommen. Diese stehen hier nicht direkt im Lexikon, da kein Vollformenlexikon verwendet wurde. Bei einigen Sprachen wäre das auch nur mit einem unvertretbar hohem Aufwand möglich, da u.U. hunderte Flexionen für ein einzelnes Wort existieren können.

Das Problem besteht darin, die korrekten Morpheme zu erkennen. Durch die Kombination von Morphemen kann sich deren Schreibweise ändern. So können

[1] kleinste bedeutungstragende Wortbestandteile

mehrere Schreibweisen (Allomorphe) für ein einziges Morphem existieren. Dabei wird die in der Sprache benutzte Darstellung Oberflächenform genannt, während die daraus resultierende Morphemfolge Basisform genannt wird. Z.B. verliert der englische Wortstamm *chase* in Verbindung mit der Vergangenheitsendung *ed* sein schliessendes *e*. Trotzdem muss in beiden Fällen *chase* erkannt werden. Bei dem Wortstamm *fox* wird jedoch bei Anfügung eines Plural *s* ein zusätzliches *e* eingefügt, so daß *foxes* entsteht.



Diese phonologischen Veränderungen unterliegen gewissen Regeln, die mittels endlicher Automaten repräsentiert werden können [Koskenniemi, 1983].
Nach dem Zwei-Ebenen-Modell, das von Kimmo Koskenniemi entwickelt wurde, werden die Regeln simultan ausgewertet, wobei keine Zwischenergebnisse generiert werden (deswegen zwei Ebenen Modell). Dazu wird mit Paaren von Buchstaben gearbeitet, deren Elemente die Veränderung von der Oberflächenform zur Basisform (und umgekehrt) abbilden. Ist die Anzahl der Buchstaben in den beiden Formen unterschiedlich, so werden Positionen, an denen die eine Form keinen entsprechenden Buchstaben enthält, mit einem Nullsymbol aufgefüllt. Dies repräsentiert je nach Position entweder ein Hinzufügen eines Buchstabens in der Oberflächenform oder das Entfernen . Auch das Trennzeichen, mit dem jedes Morphem beginnt, zählt hierbei als Buchstabe. In Abb. 4.1 wird Repräsentation der Oberflächenform *tying*, durch die Morpheme *tie* und *ing* gezeigt.

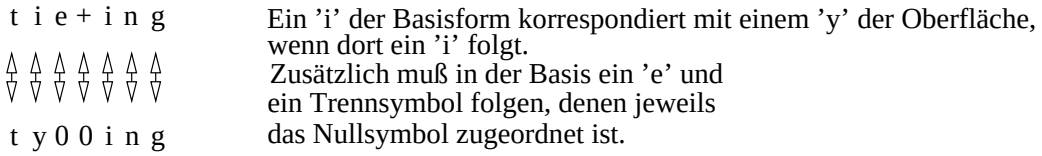


Abbildung 4.1: Unterschied zwischen der Basis- und Oberflächenform

Was zu folgender Regel führt: i:y => _ e:0 +:0 i:i

Wann zum Beispiel einOberflächen-*y* als ein Basis-*ie* erkannt werden soll, wird in Regeln definiert. In einer solchen Regel wird spezifiziert, welche(r) Buchstabe(n) sich wie ändern und in welcher Umgebung dies stattfinden kann. Im Lexikon

wird dann nach der so gefundenen Basisform gesucht. Diese Analyse ist in der Regel nicht eindeutig. Um falsche Ergebnisse auszufiltern, werden diese von einer Wortgrammatik weiterverarbeitet.
Die Regeln von ENGLEX sind bis auf eine Ausnahme vollständig übernommen worden. Sie haben sich als absolut ausreichend für die vorliegende Arbeit erwiesen.
Die Änderung, die vorgenommen wurde, resultiert aus der Erweiterung der Wortgrammatik um Regeln zur Erkennung von E-Mailadressen und Pfadnamen. Dazu war es notwendig, sämtliche Buchstaben in einer eigenständigen Kategorie in das Lexikon aufzunehmen, um beliebige Zeichenketten zu erkennen. In den benutzten Regeln wurden neben den Buchstaben aber zusätzlich die Digraphen ch und sh verwendet. So konnten E-Mailaddressen mit diesen Buchstabenkombinationen in speziellen Buchstabekontexten nicht erkannt werden, da nur die Digraphen erkannt wurden, aber nicht die einzelnen Buchstaben.

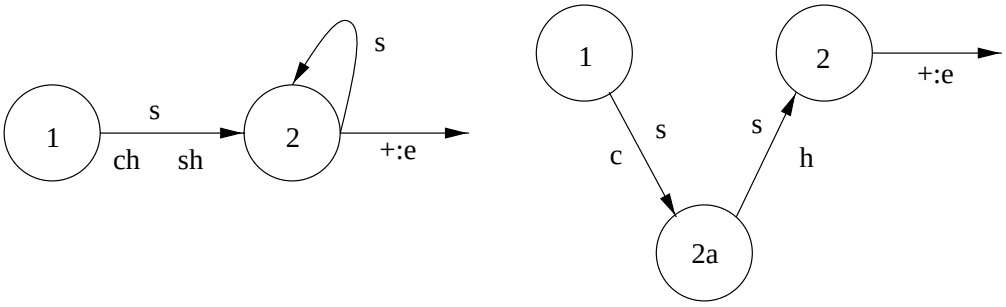


Abbildung 4.2: Einfügen eines Knotens, um die Digraphen 'ch' und 'sh' zu eleminieren

Dazu ist nun in der betreffenden Regel, von der der relevante Teil durch den linken Automaten in Abbildung 4.2 repräsentiert wird, ein zusätzlicher Knoten eingefügt worden. Die Digraphen treten nicht mehr in Erscheinung, sondern die Eingabe wird mit Hilfe des neuen Knotens Buchstabenweise abgearbeitet.

4.2 Das Lexikon

Das Lexikon stellt die Grundlage für das erfolgreiche Trennen eines Wortes in seine Morpheme dar. Es enthält u.a. Informationen, in welcher Reihenfolge es möglich ist, erkannte Morpheme anzuordnen. Diese Positionsanalyse ist recht einfach. Es gibt Wörter, an die keine Affixe vorangestellt bzw. angehängt werden dürfen. Sie werden Partikel genannt. Bei der anderen Möglichkeit muß ein Wort immer aus einem Wortstamm bestehen, dem ein oder mehrere Präfixe voran-, bzw. Suffixe nachgestellt sein können (Sie können auch fehlen). Optional kann eine Flexionsform angehangen werden.

(Prefix)* Stamm (Suffix)* (Flexionsform)
Dazu ist das Lexikon in Sublexika gegliedert, die zu Klassen zusammengefaßt werden, die sich bezüglich der Positionsanalyse gleich verhalten (Alternation). Es gibt ein Initial(sub)lexikon, dessen einzige beiden Einträge die Startpunkte der Worterkennung darstellen. Dort stehen leere Einträge, die mit der Alternationangabe Partikel bzw. Präfix die Suche in den korrekten Startlexika einleiten.

Sublexikon	Alternations
Nomen Verben Adjektive Adverben	Stamm
Prefixe	PREFIX
Suffixe	SUFFIX
Flexionsformen	Flexionsform

Jeder Lexikoneintrag enthält, neben der Angabe zu welchem Sublexikon er gehört, auch eine Alternation, die auf diesen Eintrag folgen kann.
Abgespeichert wird das Lexikon als Wald bestehend aus Buchstabenbäumen der Sublexika, mit den Sublexikonnamen als Wurzeln.

Algorithmus zur Zerlegung eines Wortes in Morpheme
Zum Erkennen eines Eingabewortes wird eine rekursive Erkennungsprozedur mit der Oberflächenform und dem Startlexikon als Parameter aufgerufen. Für jeden Buchstaben der Oberflächenform werden dann die durch die Regeln erlaubten Basisformbuchstaben ermittelt. Dabei werden die Regeln, die ja durch endliche Automaten realisiert wurden, nach jedem korrespondierenden Basisformbuchstaben in den nächsten Zustand gebracht. So können für den nächsten Oberflächenbuchstaben wieder die korrespondierenden Basisformbuchstaben gefunden werden. In den, durch die Alternationangabe (beim ersten Aufruf das Startlexikon) in Frage kommenden Buchstabenbäumen wird nach einem Lexikoneintrag gesucht. Wurde ein Lexikoneintrag gefunden, wird das erkannte Morphem gespeichert und die Prozedur wird mit der verbleibenden Eingabe und der Alternation des gefundenen Eintrags wieder aufgerufen. Dabei wird die Erkennungprozedur für alle möglichen Kombinationen aus Basisformbuchstaben und Sublexika rekursiv aufgerufen.
Ein Nachteil zusammengesetzter Wörter ist, daß sie trotzdem als Vollform ins Lexikon aufgenommen werden müssen, wenn an das betreffende Wort eine Information gehängt werden soll. Dies sieht man an folgendem Beispiel: *user* wird aus dem Verb *use* mit dem Suffix *er* gewonnen. Wenn aber *user* einen bestimmten Merkmalwert besitzen soll, wird ein eigener Lexikoneintrag erforderlich. Denn

ohne eigenen Eintrag werden nur die Informationen gewonnen, die bei *use* im Lexikon hinterlegt sind, und die die Wortgrammatik weitergibt.
Eine andere Möglichkeit ist das Speichern beim Verb von Merkmalen, die eigentlich zum generierten Nomen gehören. Wenn allerdings Merkmale vorgesehen sind, die bei verschiedenen Kategorien erscheinen. So besitzt das Merkmale *akteur* des Verbs *open* einen positiveb Wert, während das das daraus abgeleitete Nomen *opener* einen negativen Wert besitzt.
Dies kann durch verwenden von disjunkten Merkmalsnamen vermieden werden, was aber eine Änderung der Wortgrammatik bedingen würde. Deshalb wurde in dieser Arbeit ein eigener Eintrag für entsprechende Nomen erstellt.
Dadurch wird aber zweimal das selbe Nomen erkannt. Deshalb wird hier noch eine Komponente nötig, die dies erkennt und die zusammengesetzte Variante nicht an den Parser weiterleitet.
Da die vorliegenden Texte häufig mit Abkürzungen durchsetzt sind, erschien es sinnvoll einige oft verwendete Abkürzungen in das Lexikon aufzunehmen. Dies ist besonders für Verben nützlich, da eine Rechtschreibhilfe an *rcvd* als Abkürzung für received scheitern könnte. So während des Preprocessings nicht als Verb erkannt wird die Abkürzung vom System automatisch als Nomen angesehen und kann zu den schon aufgezeigten Problemen führen.

4.3 Wortgrammatik

Hier wird in einem Beispiel demonstriert, wie eine Wortgrammatik die Ergebnisse der morphologischen Analyse verarbeitet. Dabei wird die Arbeitsweise der Unifikation und die Bedeutung der Merkmale deutlich.

Eingabewort: enlargement
wird mit Hilfe der morphologischen Regeln und des Lexikons in einzelne Wortbestandteile zerlegt.

Ergebnisfolge der phonologischen Analyse:

Form	:	en+	large	+ment	+s
Kategorie	:	PREFIX	ROOT	SUFFIX	INFL
Merkmale	:	[fromcat: AJ	[lexcat: AJ]	[fromcat: V	[fromcat: N
		tocat : V]		tocat : N	tocat : N
				number :!SG]	number : PL

Mit einer geeigneten Wortgrammatik können nun die gesuchten Wortinformationen gewonnen werden, die für eine weitere syntaktische Analyse im Rahmen eines Satzes notwendig sind. Die Grammatik soll also im vorliegenden Beispiel erkennen, daß es sich um ein Nomen im Plural handelt.

Ausschnitt einer Wortgrammatik, wobei Word das oberste Kopfliteral(Zielliteral) darstellt.

```
RULE 1
Word → Stem INFL
<Stem lexcat>      = <INFL fromcat   >
<Word lexcat>      = <INFL tocat     >
<Word number>      = <INFL number    >

RULE 2
Stem → Prefix Stem_1
<PREFIX fromcat> = <Stem_1 lexcat    >
<Stem lexcat >     = <PREFIX tocat   >
<Stem number >     = <PREFIX number>

RULE 3
Stem → Stem_1 SUFFIX
<Stem_1 lexcat>    = <SUFFIX fromcat >
<Stem lexcat>      = <SUFFIX tocat   >
<Stem number>      = <SUFFIX number >

RULE 4
Stem → ROOT
<Stem lexcat>      = <ROOT lexcat    >
<Stem number>      = <ROOT number    >
```

Mittels Unifikation wird die Morphemfolge bearbeitet. Dazu wird zuerst das vorderste Element der Folge betrachtet und eine Regel gesucht, deren erstes Literal im Rumpf dazu paßt. Hier kommt nur Regel 2 in Frage.

Regel 2 beginnen	einzige Regel, deren erstes Literal im Rumpf ein PREFIX ist,
Stem : en+ Stem_1	die Regel ist noch nicht abgearbeitet, deswegen
	sind noch keine Wertzuweisungen zu den Merkmalen erfolgt

Da das letzte Rumpfliteral das Literal Stem ist, das als Kopf einer Regel erscheint, muß, wird jetzt versucht, mit dem folgenden Eingabeelement eine Regel zu erfüllen, deren Kopfliteral ebenfalls Stem lautet. Da das nächste Morphem die Kategorie ROOT hat (nicht zu verwechseln mit einer lexikalische Kategorie), muß nun Regel 4 abgearbeitet werden, denn sie besitzt als einzige ROOT als erstes Rumpfliteral.

Regel 4 anwenden

Stem_1 : large
lexcat: AJ — lexcat wird von ROOTs lexcat übernommen
number: — number bleibt leer, da in diesem ROOT nicht vorhanden

Jetzt ist ein Literal Stem erfolgreich gewonnen worden, dessen lexcat (AJ) mit tocat von PREFIX übereinstimmt, so daß Regel 2 abgeschlossen werden kann.

Regel 2 beenden
Stem : enlarge — Wert von fromcat von en+ erhält Wert von lexcat von Stem_1
lexcat: V — wird von PREFIXs tocat übernommen
number:

Da das nächste Element von der Kategorie SUFFIX ist und die nächste anzuwendende Vorschrift Stem als führendes Rumpfliteral haben muß (um die Eingabereihenfolge einzuhalten), wird nun mit der Regel 3 weitergearbeitet.

Regel 3 anwenden
Stem : enlargement
lexcat: N — wird von SUFFIXs tocat übernommen
number: PL — wird von SUFFIXs number übernommen (überschreibbar)

Auch hier sind die Werte vom lexcat bei Stem und fromcat bei SUFFIX die gleichen, so daß die Ausführung der Regel erlaubt ist. Zusätzlich wird das Merkmal number des Kopfes mit dem entsprechenden Wert des SUFFIXes gefüllt. Wieder ist das Ergebnis der jetzt abgearbeiteten Regel vom Typ Stem. Das letzte Morphem ist eine Pluralendung vom Typ INFL. Dieser Typ kommt nur in der ersten Regel vor, die jetzt auch abgearbeitet werden kann, so daß das Zielliteral erfolgreich ausgewertet werden kann.

Regel 1 anwenden
Word : enlargements
lexcat: N — wird von INFLs tocat übernohmen
number: PL — wird von INFLs number überschrieben

Auch hier wurde, wie bei den phonologischen Regeln ,die von ENGLEX gelieferte Wordgrammatik benutzt. Sie wurde erweitert, um neudefinierte Merkmale an die Satzgrammatik weiterzureichen. Zudem wurden Regeln hinzugefügt, mit denen E-Mailaddressen, Laufwerksbezeichnungen und Dateibezeichner erkannt werden können.

4.4 Satzgrammatik

Die Satzgrammatik, die jetzt erläutert werden wird, ist wie die Wortgrammatik in dem Grammatikformalismus PATR II geschrieben. Sie erhält die Informationen eines erkannten Wortes durch die zugehörigen Merkmal/Wert Paaren von

der Wortgrammatik. Die Grammatik hat einen Umfang von über 400 Regeln. Sie ist auf den vorliegenden Korpus zugeschnitten und soll nur zum Erkennen von Sätzen eingesetzt werden. Ein Generieren von Sätzen ist nicht vorgesehen. Dementsprechend konnte die Vermeidung der Übergenerierung vernachläßigt werden. Da sowohl syntaktisch korrekte, wie auch unvollständige Sätze erkannt werden sollen, würden auch viele falsche Konstruktionen geparst. Da aber die Quelle der Eingabetext ein Mensch ist, wird davon ausgegangen, daß sinnlose Sätze nicht auftreten. Desweiteren wird nur der Sprachausschnitt behandelt, der in den vorliegenden Texten erscheint. So werden beispielsweise nur Fragen modelliert, die mit einem Fragewort beginnen, aber keine Fragen, die mit einem (Hilfs-) Verb behinnen. Dieser Fragetyp ist in der Domäne bis auf vereinzelte Ausnahmen nicht vorhanden. Das folgt daraus, daß diese Frageform dazu genutzt wird mit jemandem in einen Dialog zu treten, was hier nicht der Fall ist. Es wird in der folgenden Beschreibung nicht auf alle geschriebenen Regeln und eingeführten Merkmale eingegangen. Durch den vorgestellten Ausschnitt der Grammatik wird der Aufbau der Grammatik und die Wirkungsweise der Merkmale erläutert.
Die Kategorie TTop kann an die Kategorie Top Zeitinformation anhängen, bzw. voranstellen, wie *yestreday, 10am ....* Diese Informationen werden in dieser Programmversion nicht ausgewertet, sondern beim Mapping ignoriert. Top verbindet an sich eigenständige Sätze miteinander. Dazu zählen normale Aussagesätze, Sätze, die allein aus Nominalphrasen bestehen und Sätze nach dem Schema Wenn ... dann. Hierzu wurden zwei Kategorien von Konjunktionen gebildet. In der einen stehen die Verbindungswörter und Wortkombinationen, die zwei Sätze nicht in eine Ursache/Wirkungsbeziehung bringen, wie die andere Kategorie. Diese Zuordnug ist leider nicht immer eindeutig. So verbindet das Wort *and* zwei Sätze einmal im Sinne einer Ursache/Wirkungsbeziehung und einmal nicht. Der letzter Fall überwiegt in den vorliegenden Texten, weshalb *and* hier zu der einfachen Konjunktionenklasse gezählt wird. Wichtig ist hier das Merkmal *withCJ*, das anzeigt, ob es sich um einen zusammengesetzten Satz handelt. Dies wird von der Mappingsoftware verlangt, die daraufhin die (Teil-)sätze trennt. Es wurden nur Regeln implementiert, die bis zu drei Teilsätze verknüpfen, um das mappen einfach zu gestallten. In den nicht geparsten Sätzen bestehen nur 9 % aus drei oder mehr Teilsätzen. Dieser Wert kann in Kauf genommen werden und erleichtert zudem auch das Parsen. Denn Regeln auf solch hohem Abstraktionsniveau führen leicht zu falschen Parses und die Beschränkung durch syntaktische Merkmale ist sehr aufwebdig.

4.4.1 Satzformen

Die bei weitem häufigste Form ist der einfache Aussagesatz. Der Aussagesatz besteht aus Subjekt und Verbalphrase, zu der auch ein eventuell vorhandenes Objekt gehören kann. Da in der vorliegenden Domäne das Subjekt oft fehlt, ist es hier optional. Fragen stellen einen eigenen Satztyp dar Es werden nur Fra-

gen, die mit Fragewörtern beginnen, abgedeckt. Andere Frageformen tauchen im Korpus nur sehr selten auf. Die Wenn/dann-Sätze sind geteilt in Umstand und Resultat. Dem Umstand steht meistens eine Phrase voran, die die Abhängigkeiten anzeigt, wie *when, if because of, due to*. Beide Kategorien werden wiederum aus ganzen Sätzen oder Subkategorien gebildet, wobei durch die Bildung weiterer Unterklassen für Umstand und Resultat nur erwünschte Kombinationen zugelassen werden, um Ambiguitäten zu vermeiden. Bei dieser Form von Unterklassen wurden nicht neue Bezeichner gewählt, sondern ein Postfix angehängt. Dies erleichtert dem Mapping die Verarbeitung von Wenn/dann-Sätzen, da nur nach den Begriffen *Umstand* und *Resultat* gesucht werden braucht. Wie dann weitergeparst wird, hängt von dem Satztypen ab, der wie bei den verschiedenen Aussagesätzen auch hier in dem Merkmal *senttype* gespeichert wird. Eine weitere Satzform ist eine alleinstehende Nominalphrase ohne anschliessende Verbalphrase (im folgenden als Snp bezeichnet). Darunter fällt auch ein einzelnes Verb in der Verlaufsform, z.B. *booting*. Diese Form taucht insbesondere bei den Wenn/dann-Sätzen auf. Dies ist im Deutschen vergleichbar mit *Das Öffnen von ....*

4.4.2 Verbalphrasen

Die Verbalphrasen sind das zentrale Element eines Satzes. Sie bestimmen den Satztyp, der in dem Merkmal *senttype* gespeichert wird. Dadurch kann das Mapping erkennen, mit welchen Anordnungen von Kategorien in einem geparsten Satz es zu rechnen hat.
In dieser Grammatik existieren zehn verschiedene Typen von Verbalphrasen, die im folgenden kurz vorgestellt werden sollen. Die häufigste Form ist die 'Standartform', die Präsens- Imperfekt- und Perfektphrasen abdeckt. Um die Zahl von High Level Regeln klein zu halten und nicht für jede der drei Formen eine eigene Regel zu schreiben, bzw. eine einzige Regel mit Alternativen zu überfrachten, wurden sie in einer Unterkategorie zusamengefast.

```
RULE
NormV → Vnorm
RULE
NormV → Vpast
RULE
NormV → Vpastp
RULE
Vnorm → V
<V head vform>      = NORM
RULE
Vpast → V
<V head vform>      = ED
RULE
Vpastp → AUX V
<AUX gloss>         = have
<V head vform>      = EN
```

Es reicht natürlich nicht aus, nur einzelne Verben zu erkennen. Adverben können die Verbformen ergänzen. Im Vergleich zu den Nominalphrasen verdoppeln sich hier die Varianten, da Adverben auch hinter den Verben stehen können, z.B. *a window pops up suddently.*
Zusätzlich kann ein Verb auch in verneinter Form erscheinen. Hier sind zwei Varianten in der Grammatik verankert. Einmal in der Form *a window doesn't pop up* oder *a window pops not up.* Das führt zu erweiterten Regeln (hier stellvertretend nur für eine Form). Die erste Möglichkeit mit dem *do* kann auch nicht-negiert zur Betonung verwendet werden *a window does pop up.* Daraus resultiert, daß in der unten abgebildeten Regel nur die *do* Form und nicht zusätzlich eine Negation gefordert wird.

```
RULE
Vnorm → (AV) (AUX) V (AV)
<AUX gloss>         = do
RULE
Vnorm → (AV) V NOT (AV)
```

Da die Information, ob ein Verb negiert ist, für die Auswertung interessant ist, muß das Mapping auch die Möglichkeit haben, darauf zuzugreifen. Es existieren zwei prinzipielle Möglichkeiten. Die Erste überläßt es dem Mapping den Parsebaum nach negierten Hilfsverben oder der Kategorie *NOT* zu durchsuchen. Die Andere besteht darin, die Information, die ja schon in den einzelnen Worten vorliegt, über ein Merkmal hochzureichen, das während des Mappen einfach ausgelesen werden kann. Hier wurde die letztere Methode gewählt, da sie einfacher zu implementieren ist. Weiter kann in einer Verbalphrase mehr als ein Verb existieren, wie folgendes Beispiel zeigt. *The system counts and maps the calls*

RULE
Vnorm → (AV) (AUX) V (AV) (CJ Vnorm)
<AUX gloss> = do
<Vnorm negform> = <AUX head neg>

Die komplette Verbalphrase unterscheidet dann nur noch zwischen transitiven und intransitiven Verben. Bei Ersteren wird das Vorhandensein eines Objektes erzwungen, während es bei intransitiven Verben verhindert wird. Wegen der geringen Relevanz von Sätzen mit ditransitiven Verben in der Domäne, wurde auf deren Realisierung verzichtet.

RULE
VPnorm → NormV
<NormV transtype> = I
RULE
VPnorm → NormV Obj
<NormV transtype> = T

Die nächsten Kategorien behandeln Sätze in der Verlaufs- und Futurform. Sie sind genauso strukturiert, wie die eben vorgestellte Satzkategorie. Nur die Verbform ist entsprechend den Zeiten abgeändert.
Ein anderer Satztyp bezieht sich nicht so sehr auf eine sprachliche grammatikalische Paralelle, sondern verwendet einen Worttypen, der hier neu eingeführt wurde. Es handelt sich hierbei um Verben, die ausdrücken, daß der Handelnde des Satzes etwas tun will (*want, need, try*).
Ähnlich gestaltet sich eine weitere oft vertretene Kategorie, die der gerade beschriebenen sehr ähnelt. Das charakteristische dieser Sätze ist, daß sie Frageworte enthalten. Sie protokollieren die Frage eines Dritten und sollen als Frage gemappt werden. Durch eine eigene Kategorie wird dem Mappingprogramm mitgeteil, daß Teile des Satzes ignoriert werden können. Bei *he needs to know how to start a program* wird so *needs to know* beim Mapping nicht berücksichtigt.

4.4.3 Nominalphrasen

Im Mittelpunkt steht das Nomen. Aber es gibt viele Konstellationen von Wörtern, die um ein (oder mehrere) Nomen gruppiert sein können und auch eine Nominalphrase bilden. Eine Erweiterung eines einzelnen Nomens zu einer Nominalphrase entsteht durch der Kombination mehrerer Nomen. Was im Deutschen durch Aneinanderfügen zu einem Wort geschieht, wird auf englisch meistens durch getrennte Nomen ausgedrückt. Zusätzlich kann eine Adjektivkonstruktion vorhanden sein. Hierzu zählen sowohl Past Particle , wie auch Verben in der Verlaufsform, neben den eigentlichen Adjektiven. Beispiele hierfür sind : *the completed program, the running program, the successfull program.* Die Verbformen können

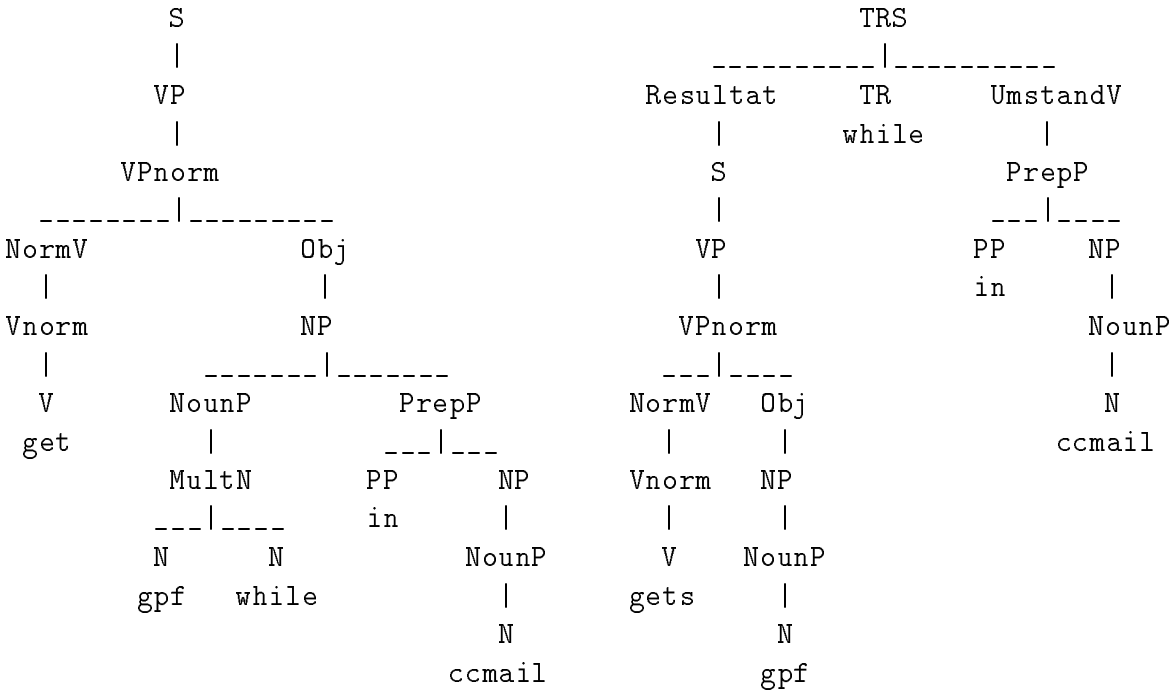
selbstverständlich auch noch durch Adverbien näher charakterisiert sein. Um es nicht zu einfach werden zu lassen, können sowohl Adjektive als auch Nomen durch Konjunktionen, prinzipiell beliebig, erweitert werden. *the successfully completed and stable program or a often not tested megahard software product.* Die nächste Erweiterung sind die Artikel. Im letzten Beispiel ist ein bestimmter und ein unbestimmter Artikel aufgetaucht. Dazu kommen noch Zahlangaben, wie *2 user* oder *more than three computers.*

```
RULE
NounP → (DT) (AdjP) {N / MultN} (PlusN)
RULE
MultN → N (N(N(N)))
RULE
PlusN → CJ NounP

RULE
AdjP → (AV) V (CJ AdjP)
<V head vform>      = ING
RULE
AdjP → (AV) PastParticiple (CJ AdjP)
RULE
AdjP → AJ (CJ AdjP)

RULE
PastParticiple → V
<V head vform>      = ED
```

Es existieren einige Wörter, für die mit Hilfe eines Merkmales ein Artikel erzwungen wird, um Zweideutigkeiten zu verhindern, wie bei *while*. Ohne den Zwang einen Artikel zu besitzen, erzeugt das Erkennen von *while* viele falsche Parses, da die Konjunktion *while* häufig im vorliegenden Korpus erscheint.



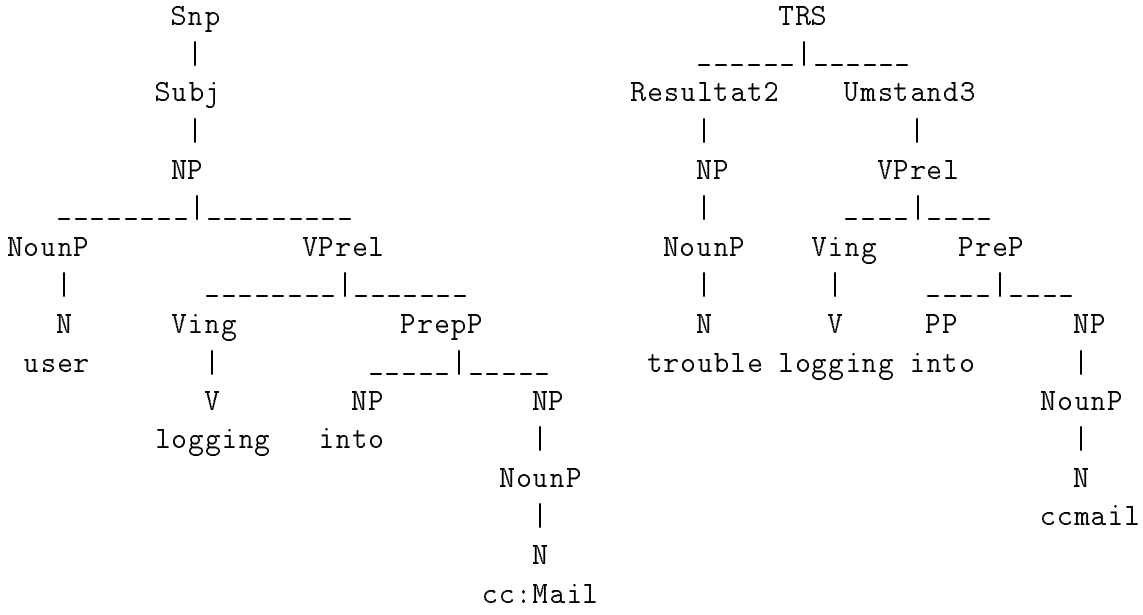
Diese Information ist im Lexikon abgelegt und den betroffenen Wörtern nach dem Erkennen von Fehlinterpretationen des Parsers hinzugefügt worden. Dadurch wird die NounP Regel aufgesplittet in:

RULE
NounP → (DT) (AdjP) {N / MultN} (PlusN)
<N dt> = -
RULE
NounP → DT (AdjP) {N / MultN} (PlusN)
<N dt> = +

Als zusätzliche Erweiterung können Relativsätze mit und ohne Pronomen auftreten. Dabei sind Nominalphrasen mit Relativprnomen einfach abzubilden durch das anhängen von dem Pronomen und einer Verblaphrase. Eine weitere wichtige Kategorie sind Verbalphrasen, wie sie normalerweise in Relativsätzen ohne Relativpronomen verwendet werden. Z.B. *the computer booting from the harddisk*, im folgenden VPREL gennant. Daher werden die Regeln für die Nominalphrasen noch um die Möglichkeit erweitert, Relativsätze ohne Relativpronomen zu erkennen. Zusätzlich wird dieser Phrasentyp auch häufig als Kurzform der Verlaufsform verwendet. Der Unterschied liegt nicht nur in dem fehlenden *is*, sonder auch darin, daß VPrels oft ohne Objekt benutzt werden. Dementsprechend wurde hier keine strenge Unterscheidung in intransitive und transitive Verben implementiert.

RULE
VPrel → V
<V head vform> = ING

Ein weiteres interessantes Merkmal verwendet eine semantische Information. Der folgende linke Parsebaum wird als Nominalphrase mit Relativsatz behandelt, während der linke Baum eine wenn/dann Struktur darstellt. Obwohl die Kategorien der Eingabewörter identisch sind, kann der Parser anhand des Merkmales *akteur* beide Sätze sinngemäß richtig bestimmen. Dabei wird hier erzwungen, daß bei einer wenn/dann Beziehung eine einzelne Nominlaphrase ein Resultat darstellt, wenn sie keinen Akteur darstellt, der Umstand in Form einer VPrel aber einen Akteur fordert. Beim linken Parsebaum dagegen sind nur alle anderen Wertekombinationen von Akteur bei Nominal- und Verbalphrase erlaubt.



Mit dieser Grammatik kann ein großer Teil der Sätze der Domaine richtig erkannt werden (siehe Abb. 7.1). Um den Anteil von nicht geparsten Sätze zu verringern ist aber eine Verbesserung der Grammatik nötig. Besonders die Erkennung von Zitaten wird hilfreich sein. Diese treten in Form von Systemnachrichten in den Texten auf.

Kapitel 5

Die Suchfunktionen und die Benutzeroberfäche

5.1 Suche

Es werden zwei unterschiedliche Methoden der Suche in den aufbereiteten Texten zur Verfügung gestellt. Dazu gibt der Benutzer an, welche Datei mit den Texten durchsucht werden soll und wählt eine Suchfunktion aus. Beide Sucharten nutzen die gleiche Datei. Die Suche ist satzorientiert, d.h. es wird bei beiden Verfahren nur innerhalb von einzelnen Sätzen gesucht. In der Ergebnisliste werden die eindeutigen Indentifikationen der Texte angezeigt, zu denen ein Satz gehört, der die vorgegebenen Suchkriterien erfüllt. Durch doppelklicken auf eine Auswahl der angezeigten Indentifikationen, werden die entsprechenden Texte mit allen zugehörigen Sätzen im nebenstehenden Fenster angezeigt. Es besteht die Möglichkeit, eine neue Suchanfrage auf so erzielten Zwischenergebnissen zu starten, oder für einen späteren Gebrauch abzuspeichern. Die Ergebnisse werden in getrennten, sich ausschließenden Anzeigen ausgegeben, können aber zur Gegenüberstellung auch in einer gesonderten Anzeige dargestellt werden.

5.1.1 Volltextsuche

Zu Vergleichszwecken kann eine boolsche Volltextsuche eingesetzt werden. Bei dieser Suchart kann der Benutzer eine Folge von konjugierten Disjunktionen aus Wörtern in ein Eingabefenster eingeben, nach denen in den einzelnen Sätzen gesucht wird. Optional kann angegeben werden, ob die Stichworte in der eingegebenen Reihenfolge im Text erscheinen müssen, um zu einem Sucherfolg zu führen. Ohne diese Option ist die Reihenfolge der gesuchten Strings in den durchsuchten Sätzen nicht relevat. Zusätzlich ist es möglich, Wildcards (Platzhalter für ein beliebiges Wort oder Buchstaben) zu gebrauchen, so daß auch Begriffskontexte gefunden werden können. In der vorliegenden Programmversion müssen für

die Suche mit variabel vielen Wildcards, alle bis zur gewünschten Obergrenze vorkommenden Kombinationen angegeben werden.

5.1.2 Erweiterte Suche

Hierbei handelt es sich um eine um syntaktische und semantische Elemente angereicherte Suche. Es kann in Frameslots nach Instanzen der verschiedenen Wortarten, wie Nomen, Verben, Adjektiven, bzw. Adverbien gesucht werden. Zusätzlich kann zu Nomen und Verben eine semantische Information, hier eine Kategorienangabe, zur Verfügung stehen.
Die Eingabe eines Suchmusters erfolgt wahlweise mit Hilfe eine Satzschemas, in das die zu suchenden Einzelteile eingetragen werden, oder mit Schemata für unterschiedliche Subslots, die in verschiedenen vorkommen. Im benutzten Frame sind dies die syntaktischen Kategorien, wie Nominal- und Verbalphrasen. Dazu wird das Frame-/Phrasengerüst graphisch als Baum dargestellt (Abb. 5.3). Ein erzeugter Zweig wird nur dann in der aktuellen Verzweigung zur Suche benutzt, wenn mindestens ein Blatt mit einem Suchbegriff, der nicht ausschließlich aus einem Wildcard bestehen darf, gefüllt ist. Sonst wird bei der Suche der entsprechende Teilbaum eines zu testenden Frames als nicht relevant für das Suchergebnis erachtet. Eine Ausnahme bilden hier die Slots PURPOSE und RESULT. Wird für sie die nächsttiefere Frameebene angezeigt, aber nicht mit Daten gefüllt, so ist alleine das Vorhandensein dieser Struktur bei einem Vergleichssatz ein Erfolgskriterium. In Abb. 5.3 liefert die linke Framedarstellung alle Frames mit einem gefüllten Slot RESULT. Die ebenfalls in ihrer Untergliederung angezeigten Slots SUBJECT und ACTION sind für das Suchergebnis nicht relevant, da keine terminalen Slots mit Daten gefüllt sind.
Ein Satz wird in die Treffermenge aufgenommen, wenn alle Einträge im aktuell betrachteten Satz mit den spezifizierten Einträgen des Suchmusters übereinstimmen. Eine denkbare Erweiterung zur Steigerung der Ausdrucksfähigkeit der Suchanfrage besteht in der Angabe von Teilbäumen als Alternativen. Ansatzweise wurde dies im vorliegenden Programm realisiert. So kann der Benutzer an terminalen Knoten eine Auswahl an Ausdrücken angeben, von denen nur einer im Vergleichssatz enthalten sein muß. Der Suchaufwand steigt aber dadurch an und die Darstellung einer Alternative ist schwierig. Verkompliziert wird die Darstellung, wenn Alternativen auf beliebigen Ebenen eines frames erlaubt sind. In Abb. 5.1 wird der Unterschied der Disjunktion auf verschiedenen Ebenen dargestellt.

Die gepunktete Linie zeigt die Ebene der Alternative an. Das linke Muster wird von Sätzen erfüllt, bei denen die Unterbäume (A ODER C) UND (B ODER D) das Suchmuster matchen. Auf der rechten Seite muß ein Eingabesatz die Unterbäume (A UND B) ODER (B UND D) dem Muster entsprechend gefüllt haben.

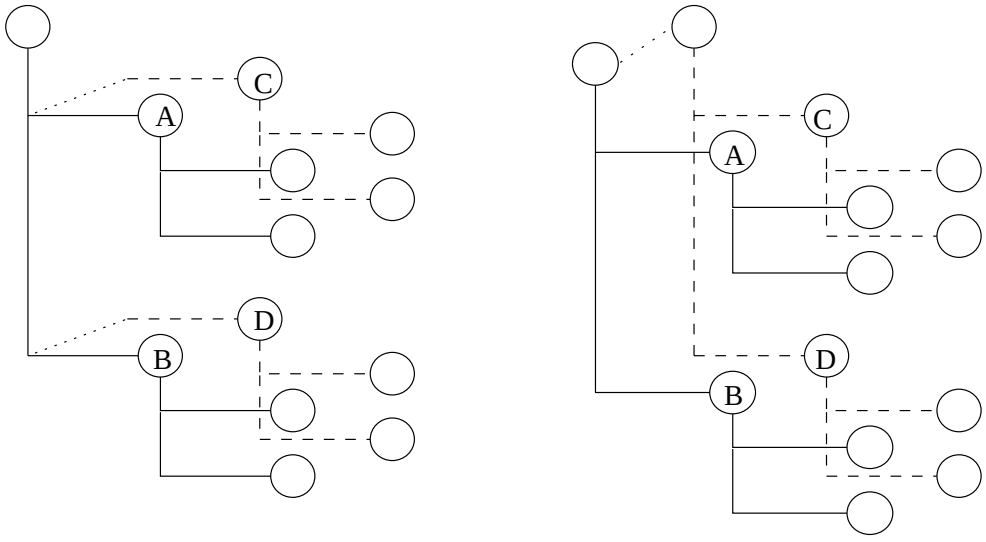


Abbildung 5.1: Erweiterte Möglichkeiten eine Anfrage zu stellen durch die Verwendung von Alternativen

Diese Informationen können Gegenstand des Benutzerinteresses sein, wenn der Anwender noch nicht in der Lage ist zum Zeitpunkt der Suche sein Intersse genau zu spezifizieren, bzw. alle durch die Rolle eines Slots definierten, relevanten Texte einsehen will.

Von Interessse ist bei der Suche nach bestimmten Mustern auch die Belegung von den in der Anfrage nicht spezifizierten Slots. So kann mit dem benutzten Frame Häufungen von Ursachen eines Resultattypes erkannt werden, der vom Benutzer durch eine Anfrage ausgewählt wurde. Durch Auswahl des Subframes für den diese Daten gewonnen werden sollen, kann der Anwender den Informationsursprung auswählen. So kann er die Einbeziehung seiner Meinung nach nicht relevanter Subframes verhindern.

Ber Benutzer kann weiter wählen, von welchen Slots er die Daten sammeln will. Implementiert ist die Auswahl zwischen SUBJECT und/oder OBJECT. Zusätzlich werden sowie automatisch die ACTION-Daten gezählt. Dabei werden nur Nomen, verben und die Lexikoninformation eines Wortes betrachtet.

Dazu nutzt das System die bestehende Kategorienhierachie aus und kumuliert für die erschienen Lexikoninformationen auch alle Superkategorien.

Man gewinnt, bedingt durch den sehr allgemein gehaltenen Frame, nur eine sehr breitgefächerte Datensammlung. Reichert man den Frame an, sind wesentlich aussagekräftigere Informationen erhältlich.

Die Anzeige, welche terminalen Slots ausgewertet werden, kann der Benutzer in dieser Version nicht wählen. Eine Verbesserung bei der Auswahl, der zu betrachtenden Slots, besteht in der Verwendung einer Baumdarstellung, wie sie auch für die Suchmustereingabe verwendet wird.

5.2 Die Benutzeroberfläche

Es hat sich gezeigt, daß zur einfachen Bedienung des Programmes eine graphische Oberfläche unverzichtbar ist. Gerade die Eingabe eines Satzmusters, nach dem gesucht werden soll, ist ohne graphische Darstellung eine sehr unübersichtliche Angelegenheit. Nachträgliche Korrekturen der Eingabe würden unmöglich gemacht und eine Wiederverwendung des Satzmusters nicht unterstützt. Durch die stark gegliederte Framestruktur ist eine intuitive Suchmustereingabe kaum möglich, während mit der gewählten graphischen Form auf einen BLick den Sinn einer Query erkannt werden kann.
Die Bedieneroberfläche ist untergliedert in ein grafisches Eingabefeld auf der linken Seite, einen Anzeigenteil mit zwei grossen Textfeldern und zwei Listboxen und einem Bedienteil im oberen Fensterabschnitt siehe Abb 5.2.

5.2.0.1 Die Menüleiste

In der Menüleiste existieren drei Menüpunkte. Unter dem Punkt *File*, kann der Nutzer die Eingabe datei mit Hilfe eines Verzeichnisbrowsers eingeben, der bei der Wahl *open file* angezeigt wird. Mit der Wahl von *save file* kann man ein Zwischenergebnis abspeichern. Auch dazu wird der Dateibrowser dargestellt. Durch *exit* wird das Programm beendet. Der Menuepunkt *Statistic* erlaubt den Aufruf eines fensters, in dem entweder die spezifizierten Kontextinformationen der ermittelten Treffermenge (*search result*) angezeigt werden oder nur eien Liste aller in der Eingabedatei enthaltenen Nomen, Verben und Lexikoninformationen (Punkt*global*). Bei der mit *global* präsentierten WörterListen besteht die Möglichkeit, eine Suche nach alle Texten in denen dieses Wort vorkommt zu starten, indem das Wort angeclickt wird und der *search*-Knopf im Darstellungsfenster gedrückt wird. Jenach Wortkategorie wird dann eine Suche nach Nominal- bzw. Verbalphrasen initiert. Wird eine Lexikoninformation nachgefragt, so wird in Nominal- und Verbalphrasen gesucht. Die gefundenen Texte werden im Hauptfenster ausgegeben. Mittels *help* wird ein Fenster mit allen benutzerdefinierten Kategorien des Lexikons angezeigt.

5.2.0.2 Der Bedienteil

Im oberen der beiden Anzeigenfelder wird der Name der Datei angezeigt, die gemappten Textdaten enthalten muß und in der gesucht werden soll. In das Feld kann der Nutzer den Dateinamen von Hand eintragen oder ändern. Das darunter liegende Feld zeigt die Query einer Volltextsuche an. Dieses Feld kann nicht editiert werden.
Darunter liegen fünf Knöpfe, die die Suche starten, das Eingabefenster für die Volltextsuche erstellen und schon erledigte Queries wieder löschen. Die beiden

linken Knöpfe gehören zur syntaxunterstützten Suche. Mit *RESET* wird ein, im Anzeigenteil erstelltes Suchmuster gelöscht und mit SEARCH wird die Suche mit dem aktuell angezeigten Muster in der angezeigten Datei gestartet. Ist noch keine Datei spezifiziert worden wird automatisch der Dateibrowser dargestellt. Die drei rechten Knöpfe betreffen die Volltextsuche. Auch hier löscht *RESET* eine bestehende Query, die zuvor in ein Eingabefenster eingegeben wurde, das per Knopfdruck auf INPUT gestartet wurde. Mit *FULL* wird schließlich die Suche mit der darüber angezeigten Query gestartet.
Die Schalter der linken Spalte rechts daneben beeinflussen die Art der Suche. Mit dem oberen wird der syntaxunterstützten Suche der Zugriff auf die semantischen Daten erlaubt oder verwehrt. Diese Funktion wurde zu Testzwecken implementiert, um einfach zu Ergebnissen mit und ohne sematischer Information zu gelangen , die auf der selben Query basieren.
Der untere Schalter steuert das Eingabefeld. In der Position *SENTENCE* kann ein Satzmuster im Eingabefeld erstellt werden. Steht der Schalter auf *PHRASE*, kann der Nutzer einzelne Satzfragmente definieren, nach denen gesucht werden soll. Dazu ändert sich die Darstellung im Eingabefeld (siehe unten). Der dritte Schalter bestimmt für die Volltextsuche, ob die Suchbegriffe in der eingegebenen Reihenfolge in einem Satz stehen müssen oder einfach nur in einer beliebigen Permutation dort vorhanden sein sollen.
Die rechte Schalterreihe dient zum Umschalten des Anzeigenteils auf die Darstellung der (verschiedenen) Ergebnisse der beiden Suchmethoden mit den oberen beiden Schaltern und der Anzeige beider Ergebnisse mit Hilfe des Schalters *BOTH*. In dieser Reihe kann immer nur ein Schalter angewählt werden. Dabei ändert sich die daruter liegende Anzeige. Zwar werden Listboxen und Textfelder beibehalten, aber der Inhalt unterscheidet sich, und Bedienelemente, die nur für eine Suchart gedacht sind, werden auch nur bei dieser angezeigt.

5.2.0.3 Das Eingabefeld

In dieser Anzeige kann mit Hilfe eines Treeviews ein Frameschema eingegeben werden, nach dem dann in den gemappten Textdaten gesucht wird. Die Elemente des Frameschemas sind vorgegeben, in das die Sätze gemappt wurden. Der Nutzer kann jetzt angeben, welche Daten die Sätze enthalten müssen, damit sie zur Zielmenge gehören.

Dazu müssen die Kategorien angeklickt werden, in die ein Datum eingegeben werden soll. Wenn noch kein Blattknoten erreicht wurde, in den allein ein Datum geschrieben werden kann, so werden automatisch die Elemente der folgenden Ebene angezeigt. Z.B. sind Trigger, Subjekt und Objekt in *Nounphrases* gegliedert, unter denen erst die Blattknoten wie Nomen, Adjektive oder die semantischen Informationen stehen. Da die Framestruktur rekursiv aufgebaut ist, kann

das Schema beliebig wachsen. Für praktische Anwendungen reicht mit dem benutzten Frameschema Muster mit höchstens eine Rekursionsstufe völlig aus. Dies deshalb, weil die allermeisten Sätze kurz sind und bei langen Sätzen die Gefahr grösser ist, daß sie falsch geparst worden sind. Bei erweiterten Frames ist aber die Benutzung tiefer geschachtelter Anfragen vorstellbar, z.B. wenn Baugruppen im Frame repräsentiert werden und nach sehr spezifischen Teile gefragt wird.

5.2.0.4 Der Anzeigenteil

Hier werden die Suchergebnisse dargestellt. Dabei erscheinen die Textindentifikationsnummern der Texteinheiten, in denen wenigstens ein der Anfrage entsprechender Satz enthalten ist, in der oberen Listbox. Per Mausklick kann eine Auswahl von Calls getroffen werden, die dann komplett mit allen Sätzen im nebenstehende Textfeld nach einem Doppelklick angezeigt werden. Mit dem unter der Listbox liegendem Knopf ALL können auf einfache Weise alle gefundenen CallIDs ausgewählt werden, bzw. mit dem Knopf NONE alle angewählten IDs wieder entlassen werden. Das kleine Feld zwischen diesen Knöpfen zeigt die Anzahl der gefundenen Calls an.
Die darunter liegenden Anzeigen sind dazu gedacht mit dem Result einer Suche einen neuen Suchvorgang zu starten. Mit dem Knopf *TAKE*, zwischen den beiden Listboxen, kann eine Auswahl an CallIDs von der oberen in die untere Box übernommen werden. Der Schalter SOURCE zwischen den beiden Textanzeigen ermöglicht die Wahl der Datenquelle für die nächste Suche. Wird FILE ausgewählt, so wird in der Datei gesucht, die in der oberen Anzeige des Bedienfeldes steht. Bei der Auswahl SELECTION wird mit dem, in die untere Listbox übernohmenen Untermenge der Texte gesucht. Technisch wird in der angegebenen Datei gesucht, aber die nicht in der Auswahl enthaltenen Calls werden übersprungen.
In der Anzeige der Syntax unterstützten Suche sind zwei Auswahlmenues vorhanden, durch die eine statistische Erhebung von Verben, Nomen und semantischen Informationen gesteuert wird. Durch Auswahl eines Menuepunktes unter dem Menuenamen *Statistics* wird ein neues Fenster erzeugt, in dem die Anzahl von Nomen, Verben und sem. Imformationen angezeigt wird. Der obere Menuepunkt *global* zeigt die Werte für alle Sätze der ausgewählten Datei an, während der untere Menuepunkt *search result* nur in den Treffersätzen einer Anfrage die Kategorien zählt.

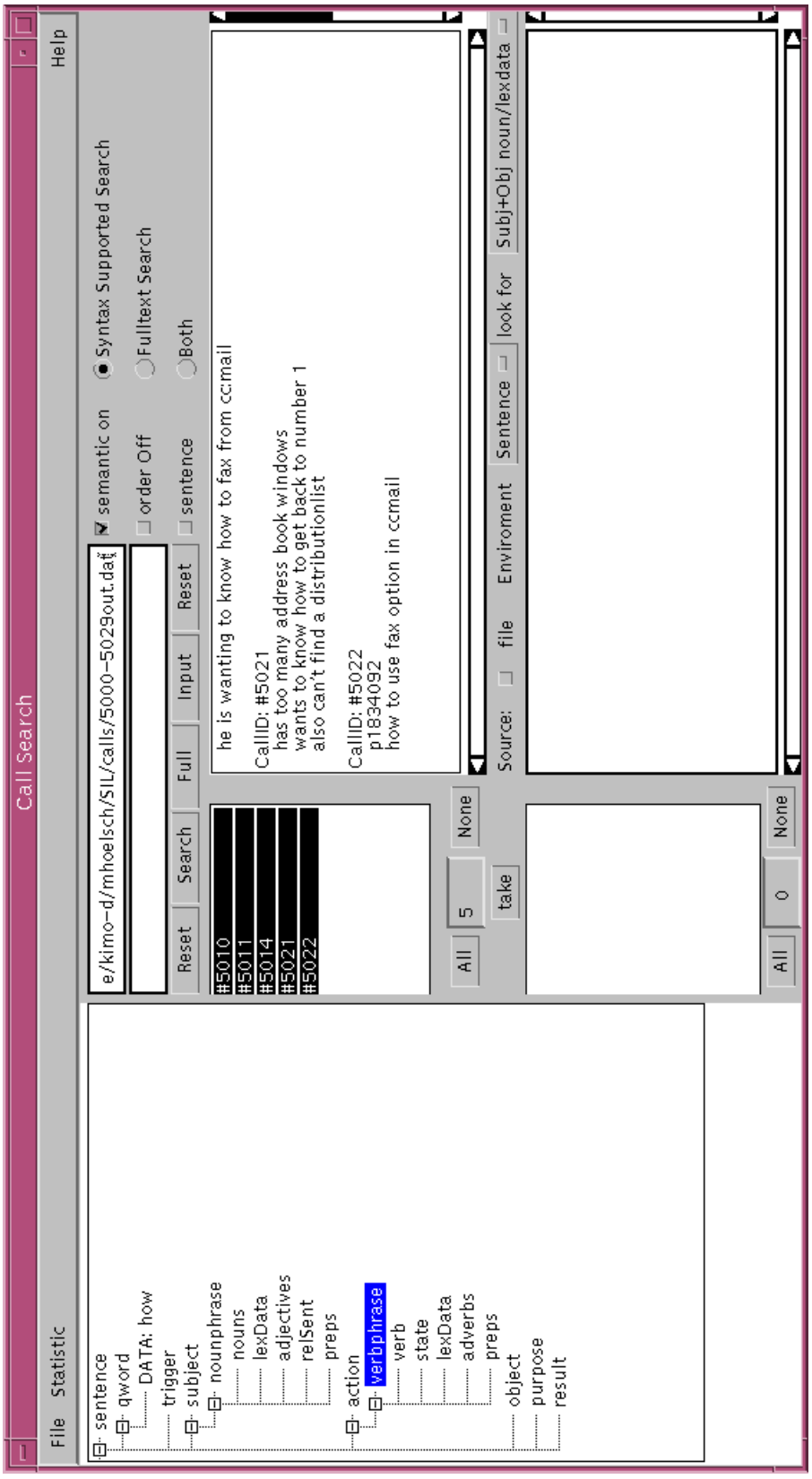


Abbildung 5.2: GUI

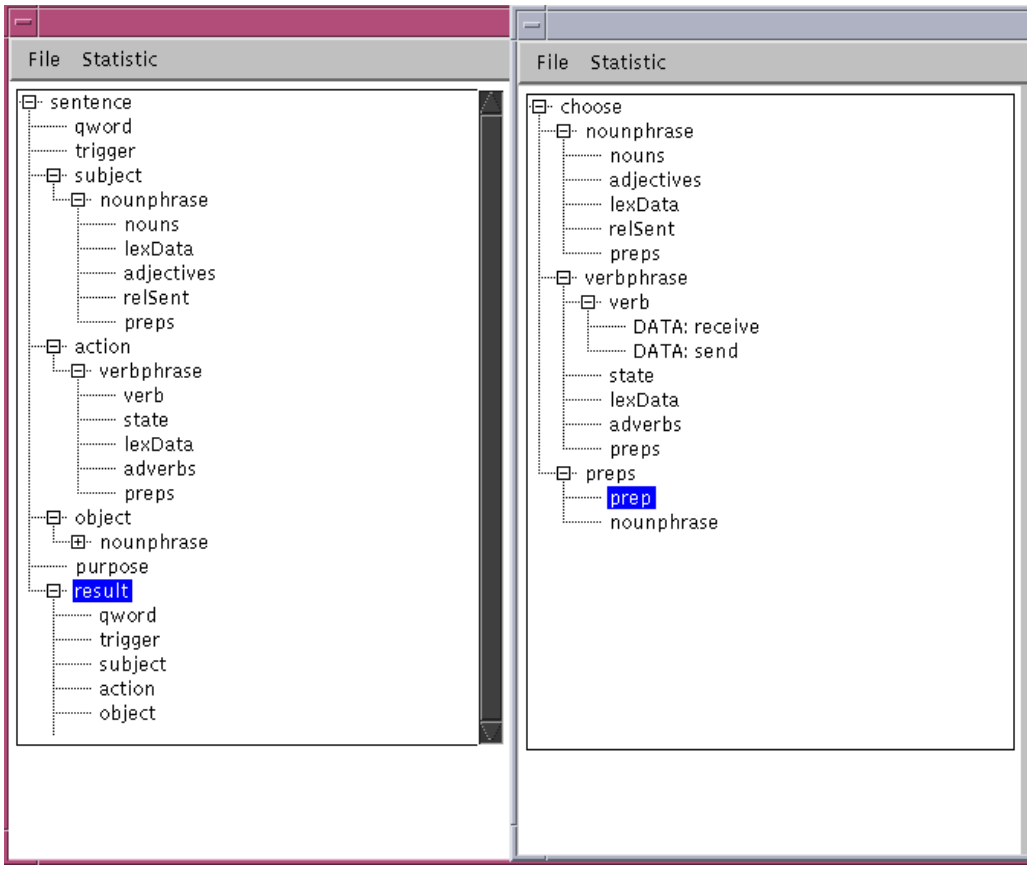


Abbildung 5.3: graphische Framedarstellung

Kapitel 6

Implementierung

Die bei der Implementation zu beachtenden Aspekte waren zum einen die Codeeinbettung von bzw. in PCPATR. Zum anderen sollte schnell ein ansprechende graphische Oberfläche erzeugt werden, die wie das PCPATR Programmpaket auf verschiedenen Plattformen lauffähig sein soll.
Die vorliegende Arbeit wurde auf einer SUN Workstation mit dem Betriebssystem Solaris entwickelt. Programmiert wurde mit dem GNU C Compiler und das JAVA in der Version 1.1.1.

6.1 Überführung der Strukturbeschreibungen

Dieser Systemteil ist wie PCPATR komplett in C geschrieben, in das es integriert werden mußte. Dies war nötig, da kein API existiert, das benutzt werden könnte. Dazu wurde PCPATR so ergänzt, daß es in seiner Funktion nicht eingeschränkt ist. Durch den Aufruf mit bestimmten Kommadozeilenparametern verzweigt PCPATR zu den eingefügten Erweiterungen.
Dabei stehen zwei Modi zur Verfügung. Ein Testmodus der mit dem Parameter *test* gestartet wird und der Modus zum automatischen Bearbeiten von Datenmengen aus einer Datei. Der Aufruf mit dem Parameter *file* und dem Dateinamen startet die automatische Verarbeitung aller in der angegebnen Datei enthaltenen Texte. Der Dateiname muß mit ".dat" enden. Diese Endung muß beim Programmaufruf weggelassen werden. Die Ergebnisse werden in verschiedenen Dateien gespeichert. In einer Datei werden die erzeugten Frames gespeichert. Zu Kontrollzwecken werden in jeweils getrennten Dateien nicht geparste, nicht gemappte Sätze und in einer weiteren Datei die ausgewählten Parsetrees gespeichert. Der Aufruf erfolgt für den Testmodus mit:

```
/home/kimo-d/mhoelsch/SIL/pc-parse/pcpatr/pcpatr test
```

Der Aufruf für die automatische Dateiverarbeitung mit:

```
/home/kimo-d/mhoelsch/SIL/pc-parse/pcpatr/pcpatr
```

mit dem Parametern:

```
file <Pfadname>/<Dateiname ohne '.dat'>
```

Die Testversion erlaubt die manuelle Eingabe von Sätzen. Diese werden geparst und gemappt und anschließend wird der erzeugte Frame am Bildschirm angezeigt.

6.1.1 Baumauswahl

Die Wahl des Parsebaumes unterscheidet nur nach vier Satztypen. In die Sätze, die nur aus einer Nominalphrase bestehen, in Sätzen vom Typ HowTo, in Wenn/dann-Sätzen (TRS-Sätzen) und in den gesamten Rest. Die Auswahl erfolgt nach dem folgendem Schema. Eine einzelne Nominalphrase wird nur ausgewählt, wenn kein anderer Satztyp als Alternative existiert. Wenn ein HowTo-Satz vorkommt, wähle diesen aus. Sonst nimm einen guten TRS-Satz, wenn einer vorhanden ist. Ein TRS-Satz wird als gut angesehen, wenn er keine einzelne Nominalphrase als Ursache oder Resultat besitzt. Existiert keiner nimm einen anderen Satz. Wenn nur TRS-Sätze existieren, wähle den mit den wenigsten (einer) Nominalphrasen als Teilsatz aus.

```
HowTo   = NULL     /*result vars*/
goodTRS = NULL
TRS     = NULL
other   = NULL

loop over each parsetree
   if      (current tree is of type Snp)
      get next tree
   else if (current tree is of type HowTo)
      HowTo = current tree
      stop loop
   else if (current tree is of type TRS && goodTRS == NULL)
      TRS = better (TRS, current tree)
      if (current tree a good TRS)
         goodTRS = current tree
      get next tree
   else
     other = current tree
do
if (result == NULL)
   if (goodTRS == NULL)
      if (other == NULL)
```

```
         if (TRS == NULL)
            result = current tree   /*choose a Snp*/
         else
            result = TRS            /*choose a TRS*/
      else
         result = other             /*choose another sentence*/
   else
      result = goodTRS              /*choose a good TRS*/
else
   result = HowTo                   /*choose a HowTo*/
return result
```

6.1.2 Die Überführung der einzelnen Satztypen

Für jeden Satztyp der Grammatik existiert eine Prozedur, die einen Frame erzeugt und mit Daten füllt. Da sich die Regeln eines Satztyps häufig gleichen, werden möglichen Abweichungen vom, allen Satztypregeln gemeinsamen, Grundmuster mit if Abfragen abgeprüft und die gefundenen Konstituenten im Frame an die richtige Stelle eingefügt. Dabei ist es vom Satztyp abhängig, welchen Frameslot eine syntaktische Kategorie füllt. Beispielsweise wird die Infinitivform *to do* in dem Satz *He wants to know how to do s.th.*, in den ACTION Slot des Basisframes gefüllt. Während die Infinitivform in *He starts the computer to print s.th.* die ACTION-Werte des Slots RESULT liefert.
Da viele Satztypen nicht flach strukuriert sind, sondern stark unterteilt sind, gibt es verschiedene Routinen, die im Parsebaum nach angegebenen Kategorien suchen. Es existieren Prozeduren zur Tiefen- und Breitensuche. Die Suche kann auf die nächste Ebene beschränkt werden. Weiter kann die Kategorie an den Enden einer Ebene eines Teilbaumes abgefragt werden. Bei all diesen Funktionen hat man die Wahl eine bestimmte Kategorie zu suchen oder eine zusammenfassende Kategorie. So gibt es in der Grammatik die Kategorie *VP* zu der alle verschiedenen Verbalphrasen zusammengefaßt werden. Jede Phrase hat einen eigenen Namen, der aber bei allen mit *VP...* beginnt. Realisiert ist dies einfach über das Matching der Eingabe mit dem vollständigen Namen einer Kategorie oder nur mit einem Präfix des Namens.
Die Rollen der Nominalphrasen sind oft schon durch die Grammatik benannt. Dort wird zwischen Subjekt und Objekt unterschieden. So weis das Mappingprogramm, welche Slots bei einem Vorkommen der beiden Kategorien gefüllt werden müssen.
Das Besetzen der Subslots einer Nominalphrase wird von einer Unterprozedur übernommen. Es wird nach den möglichen Elementen gesucht, und wenn sie existieren wird Speicher für die betreffende Datenstruktur angefordert und die geforderten Merkmalswerte mit einer weiteren Unterprozedur direkt aus der vom Parser erzeugten Datenstruktur ausgelesen. So kann für jedes Erscheinen einer No-

minalphrase,sei es in Subjekt, Objekt oder Präpositionen, immer dieselbe Mappingprozedur verwendet werden.

```
                                                NOUN:       programm
np = the fast programm        fillNounP(n)      LEXDATA:    software
                                                ADJECTIVE:  fast
```

6.2 Auswertung

Die Auswertung mit zugehöriger graphischer Oberfläche ist in JAVA programmiert. So war es möglich schnell einen lauffähigen Prototypen zu erstellen und eine mögliche Portierung auf andere Hardwareplattformen einfach zu halten. Der Nachteil eines langsameren Programmablaufes wurde in Kauf genommen, weil im Vordergrund die Realisierung stand und die angekündigte JAVA-Machine eine Geschwindigkeitssteigerung um Größenordnungen erwarten läßt. Die Objektorientierung erlaubt eine Trennung in Darstellungs- und Datenelemente. Bei letzteren sind die Methoden zur Auswertung angelegt.

6.2.1 Die Objekte zur Framestruktur

Ein zentraler Begriff in dem Gesamtsystem ist der Frame. Er ist die Entität, um die sich alles dreht. Ein solches Frameobjekt hat zwei Aufgaben: Die Repräsentation eines vollständigen Frames nachdem es aus der Datenbasis eingelesen wurde und die Darstellung der Framemuster, nach denen gesucht werden soll. Der gesammte Frame ist modular aufgebaut, d.h. daß zwar die toplevel Slots vorgegeben sind, aber die weitere Auffächerung der Datenstruktur mit Hilfe von eigenen Objekten geschieht (siehe Abb. 6.1). Im vorliegenden Frame sind dies Verb- bzw. NounList-Objekte und eingebettete Frameobjekte, sowie eine StringList (siehe Abb. 6.1). Die Verb-/NounList-Objekte bestehen wiederum aus einzelnen Verb-/NounKnoten, in denen der nächste Strukturlevel des Frames realisiert wird. Die Slots bestehen aus StringList-Objekten, die eine verkette Liste von Strings repräsentieren. In StringList-Objekten werden die erkannten/gesuchten Daten gespeichert. Durch die Speicherung eines Wortes als seperates Listenelement besteht jederzeit ein einfacher Zugriff auf mehrwortige Ausdrücke.
Durch die Verwendung von Listen wird es möglich, beliebig viele Einträge unter einem Slot zu speichern. Im Prinzip können so z.B. beliebig lange konjugierte Phrasen, die aus mehreren Elementen bestehen, gespeichert werden.

Für die Suche nach Texten, die einem vorgegebenen Satzmuster entsprechen, ist der Vergleich zweier Frames von zentraler Bedeutung. Durch die Untergliederung wird zudem der Vergleich von zwei Frames wesentlich vereinfacht. Anstatt eine

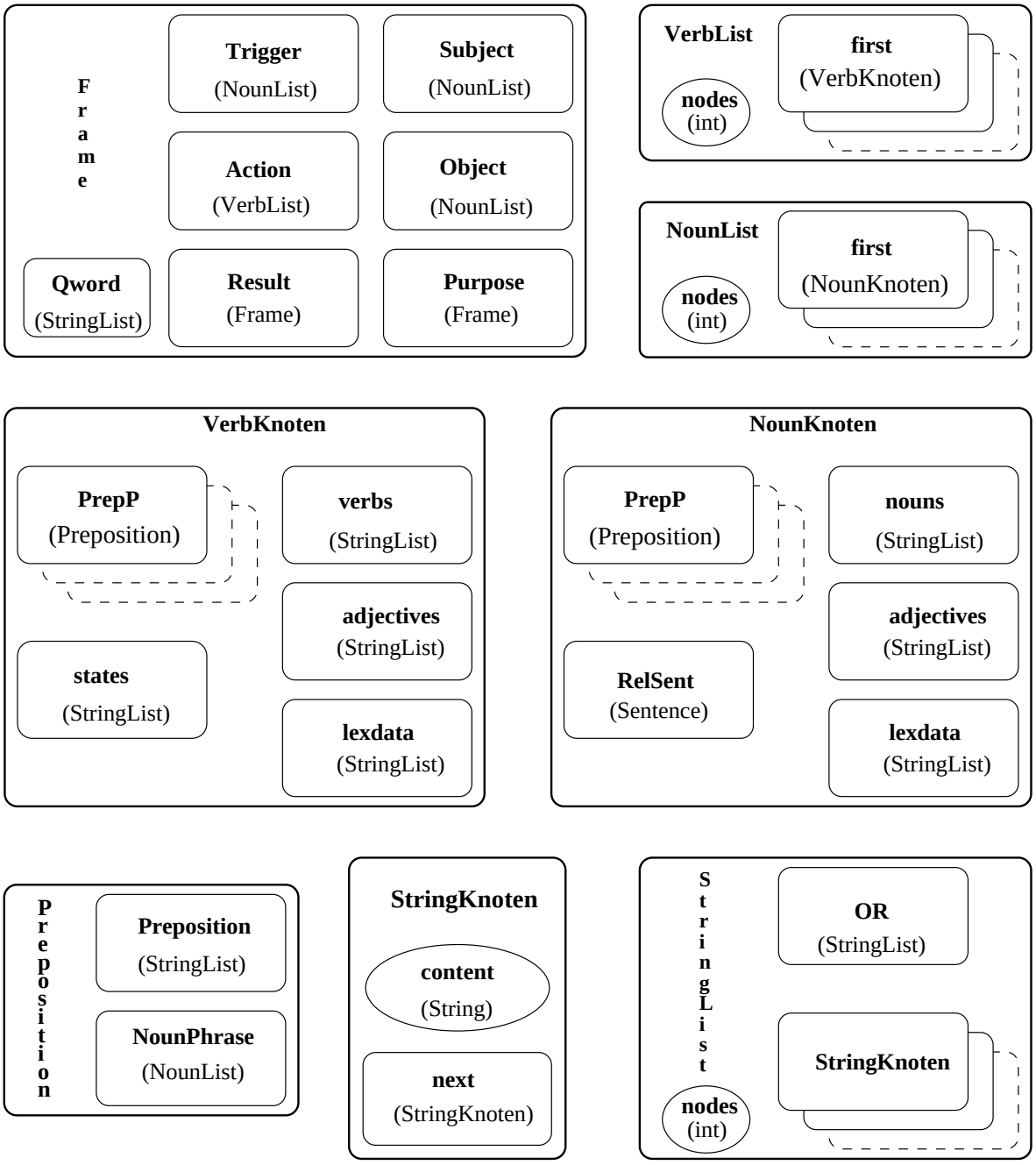


Abbildung 6.1: Frameobjekte

große Vergleichsmethode zu schreiben, kann man hier den Aufwand auf die einzelnen Objekte verteilen. Objekte die in der Framehierachie höher stehen erteilen an ihre Subobjekte einen Vergleichsauftrag und werten dann nur noch die Ergebnisse aus. Ein Frame-Objekt muß einfach die Vergleichsergebnisse von den Listen, aus denen es besteht, und den korrespondierenden Listen des zu vergleichenden Frames auswerten, um zu bestimmen, ob die Frames vergleichbar sind. Dabei heißt

vergleichbar im vorliegenden Fall, daß der aufrufende Frame (das Framemuster) an den Stellen, an denen er definiert ist, mit dem zweiten Frame übereinstimmt. Entsprechend sind die Methoden der Listenelemente realisiert.

```
if (this.qword.compareWith  (2ndFrame.qword)   == true  &&
    this.trigger.compareWith(2ndFrame.trigger) == true  &&
    this.subject.compareWith(2ndFrame.subject) == true  &&
    this.action.compareWith (2ndFrame.action)  == true  &&
    this.objext.compareWith (2ndFrame.object)  == true  &&
    (this.purpose == null ||
     this.purpose.compareWith(2ndFrame.prupose) == true)&&
    (this.purpose == null ||
     this.purpose.compareWith(2ndFrame.prupose) == true)  )

   return (true);
else
   return (false);
```

Die List-Objekte vergleichen die Ergebnisse der einzelnen Listenelemente. Das StringList-Objekt besitzt einen zusätzlichen Slot für eine alternative Liste. In der werden die Disjunktionen verwaltet, die der Benutzer bei terminalen Slots angeben kann. Am Ende dieser Kette von Vergleichsaufrufen steht ein simpler Stringvergleich. Für die Suche nach Frames, in denen entweder *user error* oder *system error* vorkommt, wird ein StringList-Objekt wie folgt gebildet:

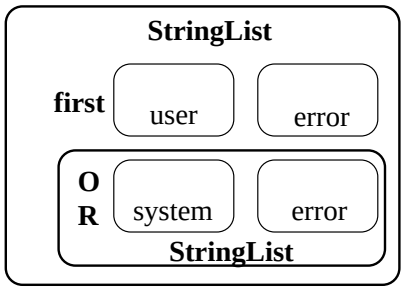


Abbildung 6.2: OrList

```
boolean ok = false

while there is a listelement && ok == false do
      if (this.element.compareWith(2ndList.element) == true)
         ok = true
      get next element
```

```
// die folgende if Abfrage ist nur
// in dem StringList-Objekt vorhanden
////////////////////////////////////////////////////////////

if (ok == false && this.or != null)
    ok = this.or.compare(2ndList)

////////////////////////////////////////////////////////////

return (ok)
```

Um die Volltextsuche ausdrucksstärker zu gestalten, wurde der Stringvergleich noch um Wildcards zum Konstruieren von Ausdrücken erweitert. Unterstützt wird ein '?', an dessen Stelle ein beliebiger Buchstabe stehen darf, und ein '*', das eine beliebige Zeichenkette repräsentiert.

6.2.2 Die Objekte der Oberfläche

Die graphische Bedienoberfläche besitzt verschiedene Fenster, von denen das Hauptfenster ständig sichtbar ist. Die anderen Fenster werden zur Dateneingabe, bzw. zur Datenansicht erzeugt oder sichtbar gemacht, wenn die betreffende(n) Information(en) eingegeben oder angesehen werden sollen (siehe Abb. 6.3). Für die verschiedenen Fensterarten existieren jeweils einzelne Objekte, die die Aufgabe besitzen, sich darzustellen und die Benutzerkommandos in entsprechende Methodenaufrufe der beteiligten Objekte umzusetzen und die Ergebnisse anzuzeigen.
In den Fenster-Objekten sind die von JAVA zur Verfügung gestellten Darstellungprimitive, wie Textanzeigen, Knöpfe und Listen, zu Einheiten zusammengefaßt (siehe Abb. 6.5, die in einem Container-Objekt (Panel) verwaltet werden. Dadurch ist es nicht nötig, aufwendig und unübersichtlich alle Primitive gleichzeitig anzuordnen. Einige zusammengehörige Elemente können einfach gruppiert werden und werden auf dem nächsthöheren Level als Einheit behandelt. So kann der Aufwand auf jeder Ebene überschaubar gehalten werden.
Die zentralen Datenstrukturen sind wie die anderen Fenster zugehörige Objekte des Hauptfensters. Diese Subfenster füllen die Datenstrukturen und können teilweise Methoden dieser Objekte aufrufen. Die Ergebnisse von Suchanfragen werden direkt im Hauptfenster angezeigt. Dazu wird das entsprechende Darstellungsobjekt der datenliefernden Methode als Parameter übergeben.

Wichtig war die Möglichkeit zur graphischen Darstellung eines Frames, da eine reine Texteingabe zu unübersichtlich ist, insbesondere unter dem Gesichtspunkt,

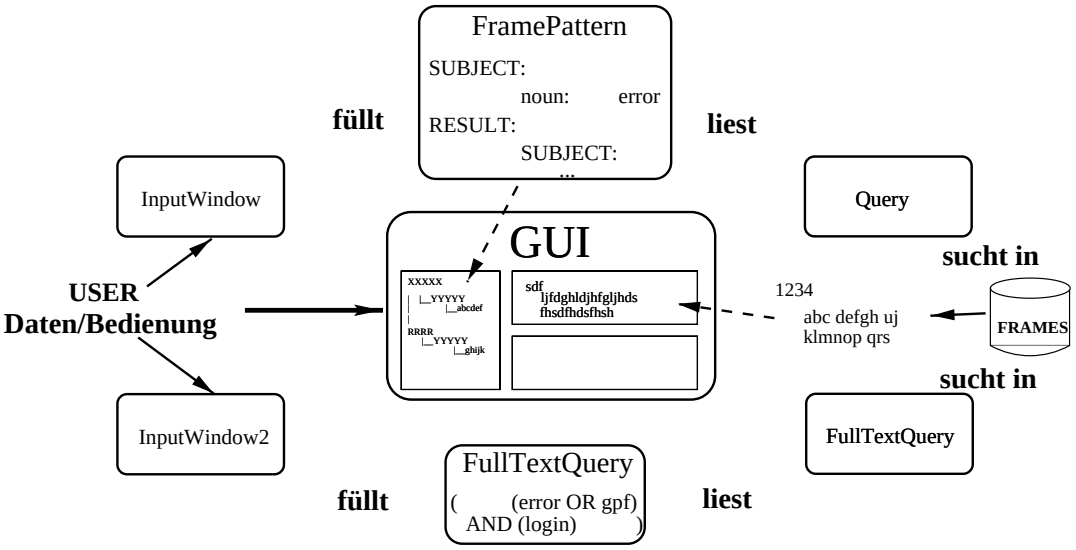


Abbildung 6.3: GUIobjekte

daß eine beliebige Schachtelungstiefe erreicht werden kann. Durch den hierachischen Aufbau der Frames bot sich eine Darstellung in Form eines Baumes an, wie er auch zur Darstellung von Dateisystemen verwendet wird. Dabei wird jedes Frameelement durch einen eigenen Knoten (TreeNode) repräsentiert (siehe Abb. 6.4). Ein TreeNode besitzt ein Datenslot für beliebige Objekte. Eine TreeNode-Instanz erhält als Namen den Begriff des Frameelementes, das er darstellen soll. Dieser Name wird auf dem Bildschirm angezeigt. Der Datenslot wird mit dem Element belegt. Dies wird bis auf die StringList-Ebene durchgeführt. Der Inhalt einer StringList wird als Name eines eigenen TreeNodes gewählt, dem eine Kennzeichnung voransteht, damit deutlich wird, daß es sich um ein Datum handelt. Wenn eine Alternative zu dem in einer StringList gespeicherten Datum existiert (in Or Slot), werden alle Alternativen als Sohnknoten dargestellt.

Um eine übersichtliche Darstellung zu gewährleisten, werden nicht ständig alle verfügbaren Daten angezeigt, sondern benutzergesteuert nur diejenigen Daten, die gerade von Interesse sind. Dabei hat sich die JAVA-eigene Klasse CardLayout-Manager besonders bewährt. Eine Instanz dieser Klasse beinhaltet verschiedene Layouts, die an derselben Stelle in einem Fenster dargestellt werden sollen.

6.2.3 Such- und Statistikobjekte

Die Kontextinformationen werden während der Ausführung der Suchanfrage gesammelt. Zum Aufruf der Query wird der Methode per Parameter mitgeteilt,

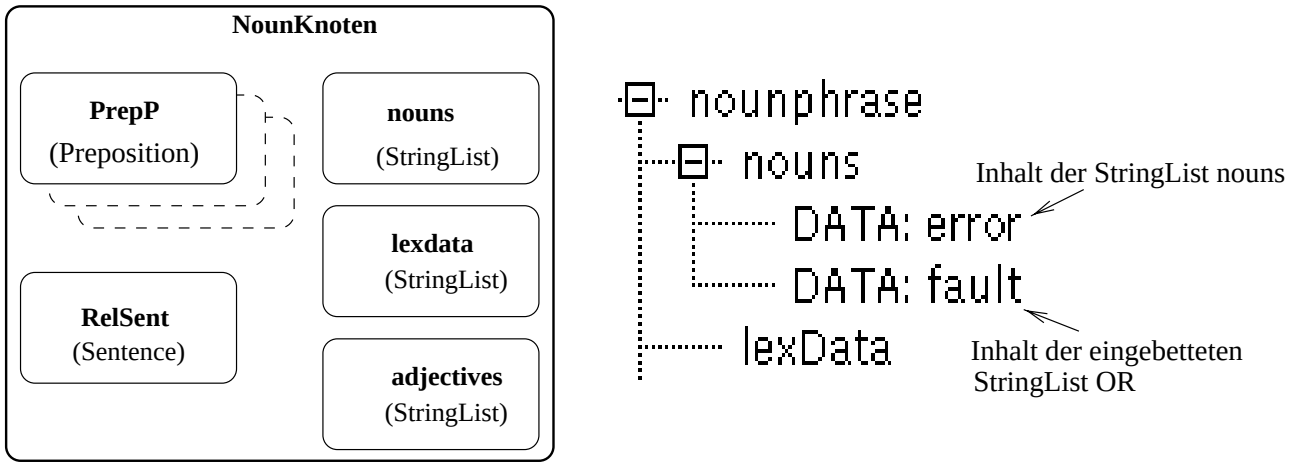


Abbildung 6.4: TreeView

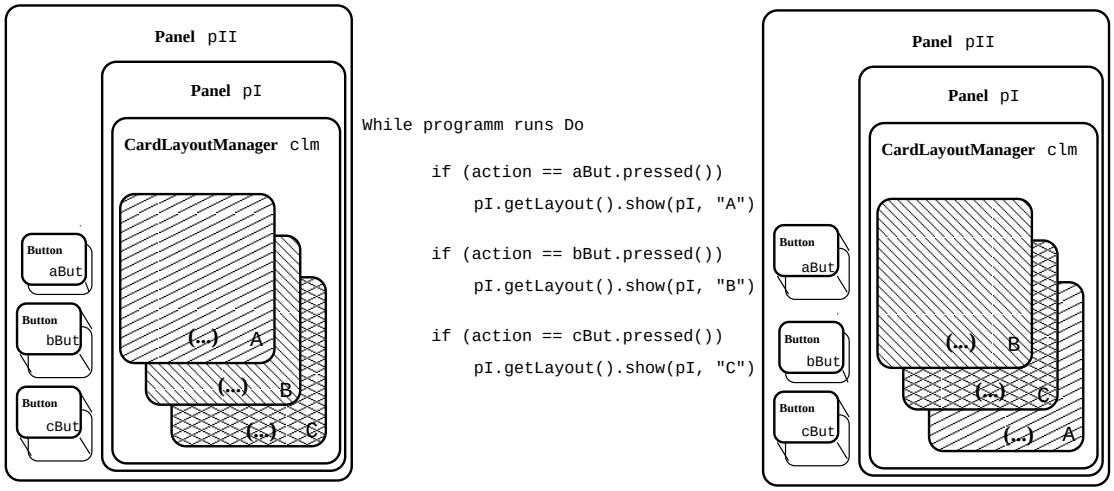


Abbildung 6.5: GUIobjekte

welche Daten, in welchem Subframe gesucht werden sollen. Nachdem ein Satz als Treffer erkannt wurde, wird der gewünschte Unterframe nach den gewählten Informationen durchsucht. Dies führt eine Methode des Frame-Objektes aus. Die ermittelten Informationen werden in Listen gespeichert, deren Knotenobjekte neben dem gefundenen Ausdruck die Häufigkeit festhalten, mit der der Ausdruck vorkommt. Dazu besitzt die Liste Methoden, um zu überprüfen, ob ein Begriff schon in der Liste enthalten ist, sowie eine erweiterte Einfügemethode. Diese legt keine Dublikate eines schon vorhandenen Eintrags an, sondern erhöht nur dessen Zähler. Die für jeden Treffersatz erzeugten Listen werden sukzessive zu einer einzigen Liste zusammengefaßt. Die Ausgabe erfolgt in die, für die verschiedenen Informationen vorgesehenen Ausgabefelder eines seperaten Fensters. Dies existiert zu diesem Zeitpunkt bereits, wird aber erst nach Anwahl eines entsprechenden Menuepunktes im Hauptfenster dargestellt.

Die Erkennung der Oberbegriffe erfolgt durch das Durchsuchen einer Datei, in der für jede vorkommende Kategorie alle existierenden Superkategorien aufgelistet sind.

Kapitel 7

Ergebnisse

Bei der Morphologie konnten keine Fehler festgestellt werden. Wörter, die nicht oder falsch erkannt wurden, fehlten im Lexikon.

7.1 Experimente

Um das System zu testen, sind verschiedene Testläufe mit zufällig ausgewählten Texten durchgeführt worden. Der zur Verfügung stehende Textkorpus wurde manuell in einzelne Sätze aufgeteilt. Die Sätze wurden in einer Datei gespeichert, pro Zeile ein einzelner Satz. Zusammengehörige Sätze wurden durch ein spezielles Symbol getrennt und mit einem Identifikationsstring versehen. Es wurde keine Rechtschreibkorrektur vorgenommen und nur zwei Phrasen, die jeweils aus zwei Wörtern bestehen, durch einen Begriff ersetzt. Weiter sind alle Satzzeichen gelöscht worden, bis auf Punkte und Slashes in Datei- bzw. Pfadangaben.
Die Texte sind mit derselben Grammatik getestet worden.
Die Auswertung der Parsequalität erfolgte per Hand. Der erzeugte Parsebaum wurde auf syntaktische und semantische Korrektheit überprüft. Trat ein falscher Baum auf, wurden noch einmal alle für den betreffenden Satz möglichen Parses erzeugt und nach dem korrekten Baum durchsucht. So konnte festgestellt werden, ob die Routine zur Parsebaumwahl einen falschen Baum ausgewählt hat. Durch die manuelle Auswertung bedingt ist nur eine relativ kleine Textmenge überprüft worden. So sind die hier gewonnen Ergebnisse aus ca. 850 Sätze, die in 400 Texteinheiten gegliedert sind, ermittelt worden. Die Sätze enthalten im Schnitt sieben Wörter und es dauert im Schnitt es fünf Sekunden, um einen Satz aus einer Datei auszulesen, zu parsen und zu mappen. Das sind eine Stunde und gut 23 Minuten für 1000 Sätze. Trotz der langsamen Verarbeitung ist das System einsetzbar. Dieser Prozeß muß nur einmalig durchgeführt werden und arbeitet vollautomatisch. Eine Geschwindigkeitssteigerung ist durch Verwendung eines effizienteren Parsers und die Einlagerung der Texte in den Hauptspeicher zu erreichen.
Der Grenzwert, ab dem ein NLS System als brauchbar angesehen wird, liegt bei

ca. 70% korrekt erkannter Sätze. Die Rate dieses Systems im Parsen von Sätzen zeigt das Tortendiagramm 7.1. Die Rate von ca. 15% nicht geparster Sätze kann noch verkleinert werden. Es gibt mehrer Ursachen, warum ein Satz nicht geparst werden kann. Zum einen existieren noch immer einige Wörter, die nicht im Lexikon enthalten sind. Einige Sätze enthalten Rechtschreibfehler, die zu nicht erkennbaren Konstruktionen führen (siehe S. 10) oder sind unvollständig. Diese beiden Fehlerarten liegen bei 4 % bzw. 8 % der geprüften nicht geparsten Sätze vor. Die restliche Fehlerrate von 88 % resultiert aus der noch zu verbessernden Grammatik. Auch diese Werte wurden durch manuelle Kontrolle der nicht geparsten Sätze gesammelt. Oft handelt es sich um zusammengesetzte Sätze, in denen ein Füllwort oder eine Zeitangabe enthalten ist, die von der Grammatik nicht abgedeckt wird. Dies wird untermauert, wenn man in einer groben Näherung die Wortanzahl als Anhalt für die Anzahl an Teilsätzen nimmt. Im Gesamtkorpus enthält ein Satz im Durchschitt sieben Worte. Bei den nicht geparsten Sätze beträgt die Wortanzahl dagegen elf Worte. Dabei ist anzumerken, daß hier auch Stichworte, Referenznummern o.ä. Sätze darstellen, die immer erkannt werden, weil unbekannte Zeichenfolgen als Nomen angesehen werden und fast alle Stichwörter Nomen sind. Die, um aus einzelnen Nomen bestehenden Sätze, bereinigte Zahl für den Gesammtkorpus beträgt dann acht Wörter.

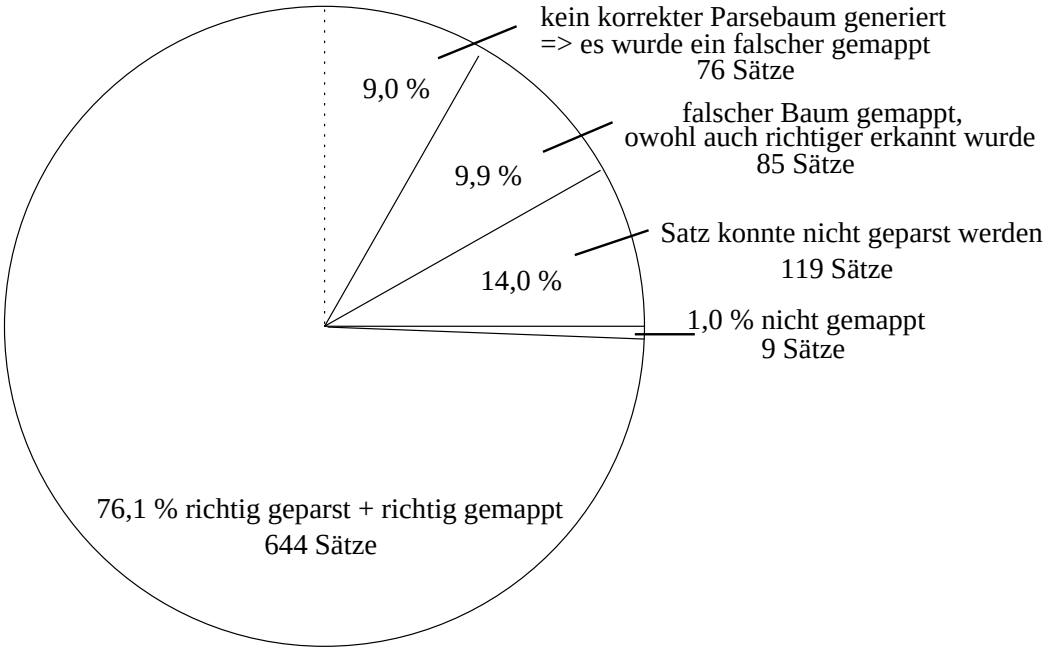


Abbildung 7.1: Parseergebnisse von 853 Sätzen

Dabei existiert eine Rate von durchschnittlich 17,2 Parsebäumen pro Satz. Dieser hohe Durchschnitt relativiert sich aber aus zwei Gründen. Zum einen ist die Grammatik in Teilen redundant, wodurch häufig mehrmals die gleichen Bäume für den selben Satz erkannt werden. Dazu tragen auch doppelt von PCKIMMO

erkannte Wörter bei:
Zum einen handelt es sich um Wörter, die ursprünglich nicht als eigener Lexikoneintrag vorhanden waren, sondern durch die Wortgrammatik gebildet wurden. Wenn ein solches Wort einen eigenen Eintrag in das Lexikon erhält (siehe S. 18), soll das zusammengesetzte Wort nicht an die Grammatik weitergegeben werden. Dieser Systemteil ist aber nicht implementiert. Auch werden Verben, die in unterschiedlichen Zeiten die gleiche Form besitzen mehrfach an die Satzgrammatik weitergegeben. Da in manchen Grammatikregeln keine Einschränkung hinsichtlich der Zeitform eines Verbs gemacht wird, erkennt der Parser so einen Satz mehrfach. Zum anderen sinkt der Durchschnitt auf 5,4 Prozent, wenn die 6 Prozent der geparsten Sätze, die zu mehr als 48 Parsebäumen führen, nicht mitgerechnet werden.
Die Erkennungsrate ist bei diesem System auch nicht so entscheidend, da ein Produktionssystem über mehreren tausend Texteinheiten mit zehntausenden Sätzen arbeiten kann. Wenn keine Korrelationen zwischen Syntax und Inhalt existieren, ist dann auch die Verarbeitung einer Teilmenge sinnvoll, da die Stichprobe groß genug sein wird.
Durch die Darstellung u.a. von Information, die in der sprachlichen Struktur der Sätze liegt, können Anfragen gestellt werden, die mit einer reinen wortbasierten Suche nicht möglich sind. Hervorzuheben ist im vorliegenden Fall die Suche nach bestimmten Ereignissen, die das Resultat einer Handlung sind bzw. umgekehrt. Allein diese Anfragen sind mit einer Volltextsuche nicht realisierbar. Eine Beispielanfrage verdeutlicht dies:
Gesucht werden soll nach allen Texten die etwas mit Druckern zu tun haben. Die Volltextsuche nach *printer* liefert vier Treffertexte. Nach der Anfrage nach Texten mit *print** ergeben sich 20 relevante Texte. Demgegenüber erreicht die Suche nach der Kategorie *printer* im Slot LEXDATA in allen Nominal- und Verbalphrasen 15 Treffer. Die Differenz von fünf Texten folgt aus zwei nicht geparsten relevanten Sätzen und drei nicht vollständig gefüllten Frames. In dieser allgemein gehaltenen anfrage erweist sich die Volltextsuche als überlegen. Spezialisiert man aber die Suche z.B. nach den Texten mit einem Druckversuch. Dazu wird eine Suche über alle Verbalphrasen gestartet. Die Zielsätze müssen im Slot LEXDATA wieder den Wert *printer* aufweisen und zusätzlich im Slot STATE den Wert *attempt*. Diese Anfrage liefert vier Treffer. Ein Versuch diese Anfrage mittels folgender Volltextquery (try*) AND (print*), liefert nur drei erkannte Texte. Davon wurde einer falsch geparst, so daß er nicht in der oberen Treffermenge enthalten ist, während zwei Texte eine andere ausdrucksweise aufweisen (*wants to print ...*) und von der Volltextsuche nicht erkannt werden konnten. Eine weitere semantische Anfrage nach allen Texten, die auf die Aktion drucken mit einem Resultat reagieren, liefert von der Volltextsuche (when OR while) (print*) einen Text mehr, der aber nicht zur Treffermenge gehört. Die beiden anderen Texte sind dieselbe korrekten Texte, die unter Ausnutzung der Framestruktur erziehlt wurden. Durch einen nicht geparsten und einen falsch gemappte Satz, existiert aber ein

Verbesserungspotential, das für die Volltextsuche nicht mehr vorhanden ist. Das Dilemma der Volltextsuche ist, daß sie eine möglichst spezielle Auflistung von relevanten Wörtern benötigt. Ist die Liste zu klein findet sie nicht alle Flexionen eines Wortes. Wird aber ein zu großzügiger Gebrauch der Wildcards gemacht, enthält die Treffermenge zu viele falsche Texte.
Um festzustellen, ob diese Handlungen Gemeinsamkeiten aufweisen, wird durch das Suchprogramm gezählt, wieviele Informationen der Ursache in den gefundenen Sätzen übereinstimmen. Einen wesentlichen Beitrag dazu stellt die im Lexikon hinterlegte semantische Information und die darüber gebildete Hierachie dar. Allein die Möglichkeit nach Synonymen für unterschiedliche Wörter zu suchen stellt einen großen Vorteil gegenüber der einfache Volltextsuche dar. Damit der Anwender bei der Volltextsuche eine, der Suche über den Frames vergleichbare Treffermenge erziehlt, muß er mit einer Disjunktion alle Begriffe auflisten, die auf eine in Lexikon gespeicherte Kategorie abgebildet werden. Dann kann die Volltextsuche auch überlegene Ergebnisse liefern, da hier alle Sätze in die Suche miteinbezogen werden. Wenn allerdings Suchausdrücke mit Wildcards verwendet werden, um z.B. alle Formen eines Verbes in die Suche mit einzubeziehen, können Texte gefunden werden, die nicht zur gesuchten Menge zählen. Ohne die Abbildung verschiedener Ausdrücke auf einen Begriff, bleibt die Aussagekraft einer Summierung von Kontextinformationen allerdings schwach, was in der sprachliche Ausdrucksvielfalt begründet liegt. Zusätzlich wird dieses Feature durch die Suche nach Superkategorien erweitert.
Durch die Nutzung der im Lexikon abgelegten Daten werden alle Sätze, die die richtig geparst wurden, auch bei einer Anfrage angezeigt. Dies kann man als eine Art Precision Wert ansehen. An diesem Punkt hat die Volltextsuche den Vorteil, daß sie über allen vorhandenen Sätzen arbeitet. Damit hat sie die Möglichkeit alle relevanten Texte zu erkennen.
Sogar falsch geparste Sätze besitzen noch einen Teil ihres Informationsgehaltes, wenn nur nach Kategorien gefragt wird. Nutzt man dagegen einen erweiterten Frame kann eine in einen flaschen Slot eingetragene Information zu einem verfälschtem Suchergebnis führen. Deswegen erscheint das Verringern der falsch geparsten Sätze vordringlicher zu sein als eine Steigerung der Zahl der erkannten Sätze.
Die Art der Ermittlung der Treffertexteinheiten kann zu einer unvollständigen Liste der gesammelten Kontexinformationen führen. Die gesamte Einheit wird als Treffer angezeigt, wenn ein Satz dem Suchmuster entspricht. Andere Treffersätze in der Texteinheit werden nicht mehr für die Sammlung der Kontextinformationen genutzt. Dies erscheint aber vertretbar, da es sich in der Regel um kurze Texte handelt (in Schnitt 2.1 Sätze pro Texteinheit).
Ein Nachteil des implementierten Systems ist, daß es keine Anaphern auflösen kann. Eine Anapher stellt eine Referenz auf ein schon benanntes Objekt dar. In dem Satz *The student prints his thesis to show it his supervisor* stellt *it* eine Anapher dar, die sich auf *thesis* bezieht. Da sie häufig benutzt werden, stellen sie

einen großen Anteil der Kontextinformation. Durch ihre Auflösung gewinnen die gesammelten Informationen zusätzlich an Aussagekraft.

Kapitel 8

Zusammenfassung

Das Ziel dieser Arbeit war die Generierung von Frames zur Speicherung von Informationen, die in natürlichsprachlichen Texten enthalten sind. Für die einfache Anwendung sollte eine intuitive zu benutzende Oberfläche erstellt werden, die den Benutzer bei der Suche nach neuem Wissen unterstützt.
Das implementierte System umfaßt drei Stufen. Die syntaktische Analyse wandelt einen Eingabestring um in eine Baumstruktur. Diese Daten werden in der nächsten Stufe in einem Frame umgesetzt. Über dieser Repräsentationsform führt das Programm in der dritten Stufe vom Benutzer definierte Suchaufträge aus und sammelt gleichzeitig Informationen über die Inhalte der gefundenen Sätze, wobei es Informationen aus dem Lexikon zur Verfügung hat. Über diesen Informationen ist eine Hierachie definiert, die durch die Generalisierungsmöglichkeit eine große Ausdrucksmächtigkeit besitzt.
Damit vereint das System Elemente der Wissensrepresentation mit Information Retrievalelementen und Informationextraktion, unter Verwendung von NLP Techniken. Die in der Aufgabenstellung beschriebenen Ziele sind erreicht worden. Es können die überwiegende Anzahl von Sätzen einer Textsamlung automatisch in eine semantische Repräsentation überführt werden.
Dem Benutzer wird mit einer graphischen Oberfläche eine einfach zu bedienenden Form geboten, Framemuster zu definieren. Die Suche nach diesen Mustern entsprechenden Textrepräsentationen arbeitet zuverlässig. Die Sammelung von Kontextdaten funktioniert ebenfalls, hat aber noch den Nachteil nicht aufgelöster Anaphers (siehe unten).
Eine weitere in dieser Hinsicht interessante Möglichkeit, Informationen zu extrahieren, liegt in der Anwendung von Lernverfahren über den erzeugten Frames. So können u.U. Informationszusammenhänge gefunden werden, die ein Mensch aus einer sehr großen Datenmenge mangels Überblick nicht entdecken kann.
Die Resultate haben gezeigt, daß schon der benutzte einfache Frame eine Gewinnung von Informationen ermöglicht, die mit reiner Volltextsuche nicht zu leisten sind. Für die effektive Nutzung der Kontextinformationen ist ein wichtiger Punkt die Auflösung von Anaphern. Die Auswertung der Satzkontexte einer Treffermen-

ge wird dadurch aussagekräftiger. Wenn ein erweiterter Frame mit vordefinierten Rollen benutzt wird gilt dasselbe, da von den nicht aufgelösten Pronomen keine verwertbaren Informationen aus dem Lexikon zu erhalten sind.
Ein spezielles Interesse liegt in der Erweiterung der Framestruktur. Für einen festumrissenen Anwendungszweck liegt es nahe, verschiedene Konzepte zu definieren , um so zu anwendungsrelevanteren Strukturen zu gelangen. Dazu werden zusätzliche semantische Informationen nötig sein, um zu entscheiden womit welcher Slot gefüllt werden kann.
Die improvisierte Implementation der Kategorienhierachie kann durch ein KL-1 System ersetzt werden, um einfacher zu einer konsistenten Modellierung bei steigender Anzahl von Kategorien zu gelangen. Für eine erfolgreiche Informationsgewinnung ist die Moddellierung der Domäne durch einen Experten sinnvoll, der die Anwenderinteressen kennt und bei der Kategorienbildung berücksichtigen kann. Dabei muß auf eine genügend feine Abstufung in der Hierachie geachtet werden, um beachtenswerte Teilmengen einer Kategorie ansprechen zu können.
Um die Erkennungsrate der Sätze zu erhöhen, sollten Versuche mit dem Verwenden von geparsten Teilbäumen unternommen werden, um einen Teil der Informationen aus einem ansonsten nicht erkannten Satz zu extrahieren.
Zur besseren Auswahl von mehrdeutigen Parsetrees kann die Einführung von Wahrscheinlichkeiten führen. Dies kann durch die Erweiterung der bestehenden Auswahl um weitere Satztypen versucht werden. Das könnte dazu führen für bestimmte Mengen von Satztypen eine Präferenz festzulegen, die u.U. von den Lexikoninformationen bestimmter Kategorien abhängig gemacht werden kann/muß, z.B. abhängig davon, ob das Subjekt einen Akteur darstellt.
Im vorliegenden System wurde die Datenhaltung mittels Dateien realisiert. Für Produktionssysteme sollte es problemlos möglich sein eine Datenbankschnittstelle zu realisieren. Da der Frame dann als Datenbankobjekt vorliegt, wird es sogar einfacher sein, als das Lesen aus einer Datei.

Literaturverzeichnis

[Charniak, 1993] Eugene Charniak
Statistical language learning
1993 MIT Press Cambridge, Massachusetts

[Cowie, 1996] Jim Cowie, Wendy Lehnert
Information Extraction
1996 in Communications of the ACM, Vol. 39

[Dörnenburg, 1997] Dörnenburg Erik
Extension of the EMILE algorithm for inductive learning of context-free grammars for natural languages
1997 Diplomarbeit an der Universität Dortmund

[Flanagan, 1996] David Flanagan
JAVA in a Nutshell
1996 O'Reilly

[Fuhr, 1996] Norbert Fuhr
Informationssysteme
Skript zur Vorlesung an der Universität Dortmund, 1996

[Gazdar/Mellish,] Gerald Gazdar, Chris Mellish
Natural Language Processing in PROLOG
Workingham, England: Addison-Wesley

[Jacobs] Paul S. Jacobs, Lisa F. Rau
SCISOR: Extracting Information from On-line News
1990 in Communications of the ACM, Vol. 33, No. 11

[Koskenniemi, 1983] Kimmo Koskenniemi
Two-level morphology: a general computational model for word-form recognition and production
1983 Publication No.11 University of Helsinki Department of General Linguistics

[Levinson, 1983] Stephen C. Levinson
Pragmatics
1983 Cambridge university press

[Linke, 1991] Angelika Linke, Markus Nussbaumer, Paul R. Portmann
Studienbuch Linguistik
1991 Niemeyer, Tübingen

[Livermoore, 1997] Ann Livermore, General Manager, HP Worldwide Customer Support Organisation
in: Hewlett Packard, Das Herzblatt Ausgabe 6 - 04/97,
Aktuelle Mitarbeiter-Information Unternehmensbereich Dienstleistungen

[Morik, 1997] Katharina Morik
Einführung in die künstliche Intelligenz
1997 Skript zur Vorlesung an der Universität Dortmund

[Morik, 1996] Katharina Morik
Natürlichsprachliche Systeme
1996 Skript zur Vorlesung an der Universität Dortmund

[MUC, 1993] Morgan Kaufmann
Proceedings of the third Message Understanding Conference
1991 San Diego, California

[Reimer, 1991] Reimer, U.
Einführung in die Wissensrepräsentation
1991 Teubner Stuttgart

[Shapiro, 1087] Stuart C. Shapiro
Encyclopedia of artificial Intelligence

1987 John Wiley & Sons, Inc

[Shieber, 1986] Stuart M. Shieber
An introduction to unification-based approaches to grammar
1986 CSLI Lecture Notes No.4. Stanford, CA

[SIL] Summer Institute of Linguistics
http://www.sil.org