

NLP for NLP (Natural Language Processing for Natural Language Programming)

-

natürlich-sprachliches Programmieren

Kerstin-Anja Enenkiel

Universität Ulm

Proseminar, NLP für NLP, SS'06

Zusammenfassung:

Natürlich-sprachliches Programmieren - ein Begriff, der bei genauerem Hinsehen nicht wirklich in sich stimmig ist. Natürliche Sprache und Programmieren?

Ein Programmcode, auch wenn man manche englische Wörter in ihm entdecken kann, ist vollkommen anders aufgebaut als natürlich gesprochene Sätze. Und es erfordert einiges an Verständnis und Grundwissen der Programmiersprachen, um ebenso fließend programmieren wie sprechen zu können. Gerade für Anfänger ist es ziemlich schwierig, sich in die abstrakte Welt der Programmiersprachen hineindenken und sich mit ihr auseinandersetzen zu können. Somit wäre die Möglichkeit, mit der natürlichen Sprache zu programmieren, ohne die komplexe Syntax von Java, C++ etc. zu lernen, sehr hilfreich, um vor allem die anfänglichen Verständnisbarrieren zu bewältigen.

Dieser Artikel setzt sich mit der Entwicklung eines Systems auseinander, welches die natürlich gesprochene Sprache mittels einer intelligenten Maschine in einen Programmcode übersetzen soll.

5. Juni 2006

Inhaltsverzeichnis:

1. Zusammenfassung	1
2. Inhaltsverzeichnis	2
3. Einleitung	3
4. Hintergrund	3
5. Die Anforderungen der Programmierung an die KI	4
6. Die beschreibende natürlich-sprachliche Programmierung	5
7. Die prozedurale natürlich-sprachliche Programmierung	7
8. Metafor	9
9. Studien, Erkenntnisse und die Zukunft	11
10. Referenzen	13

3. Einleitung:

Künstliche Intelligenz, eine Maschine, die selbstständig denkt und handelt, eine Maschine, mit der der Mensch eine direkte Kommunikation aufbauen kann?

Kann man sich mithilfe eines intelligenten Systems Probleme lösen lassen? Dieser Gedanke besteht mit Sicherheit schon seit dem Beginn der Entwicklung erster Computer.

Bei der hier vorliegenden Ausarbeitung geht es um die Zuhilfenahme einer Maschine, um, wie schon gesagt, die natürliche Sprache in einer Computersprache, eine so genannte Programmiersprache übersetzen zu lassen.

Das Programmieren an sich ist bisher nur für diejenigen möglich, die sich schon länger mit Programmiersprachen auseinandersetzen und erfahren sind mit der Übersetzung von Aufgabenstellungen in Codings.

Das ist eigentlich ein sehr großes Defizit, da es in vielen Firmen und Unternehmen, die sich auch nur ansatzweise mit Technik auseinandersetzen, vorausgesetzt wird, zumindest ein wenig Programmiererfahrung zu haben. Um dieses Defizit auszugleichen und den von nur wenigen Menschen benutzten Bereich auch für Nichtprofis zugänglich zu machen, hat man versucht, ein intelligentes System zu entwickeln welches die natürlich gesprochene Sprache in eine Programmiersprache übersetzt und ein Programmkonstrukt aufbaut.

Ein Computersystem soll die Sprache erkennen, übersetzen, die Problem unterteilen, Wiederholungen und Bedingungen erkennen und daraus einen kompilierfähigen und ausführbaren Code zu entwickeln.

In den folgenden Kapiteln werde ich zunächst ein wenig auf die historische Entwicklung dieses Forschungszweiges eingehen, danach kurz den Bezug zur künstlichen Intelligenz erläutern. Danach werde ich zuerst etwas über die deskriptive und im Folgenden über die prozedurale Programmierung erzählen. Zum Schluss werde ich die Beispielsoftware Metafor kurz vorstellen und meine Arbeit mit ein paar Sätzen über Erkenntnisse, Studien und Zukunft des Forschungszweiges natürlich-sprachliche Programmierung abschließen.

4. Hintergrund:

Schon früher gab es die Idee, eine direkte sprachliche Kommunikation mit einem System herzustellen.

Hier trafen die beiden Forschungsgebiete Informatik und Sprachwissenschaft aufeinander, die sich später zu dem Forschungsgebiet „Computerlinguistik“ zusammenschlossen.

„Eliza“, ein 1966 von Joseph Weizenbaum entwickeltes Computersystem, war zu seiner Zeit zum Beispiel ein sehr populäres Beispiel, war aber weniger für den technischen Fortschritt geeignet. Vielmehr sollte es ein exemplarisches Beispiel für die Gutgläubigkeit der Bürger sein, die in dem relativ einfach programmierten Frage-Antwort-System einen wahren Freund meinten gefunden zu haben.

Aber die Idee mit der Problemlösung durch die Frage-Antwort-Nachfrage-Taktik wurde weitergeführt, allerdings in der technischen Richtung.

Es wurden Systeme erstellt, deren Taktik aus gezieltem Hinterfragen der Informationen bestand. Somit wurde der Benutzer von vorneherein gezwungen, die gegebene Information gezielter zu verwerten und einzusetzen.

Ein Beispiel hierfür stellte der „NLC“-Prototyp dar. Er versuchte, gegebene mathematische Aussagen transformieren und verwenden zu können. Aussagen wie ADDDIERE Y1 ZU Y2 wurden zum Beispiel in $Y1 + Y2$ umgewandelt. Eine weitere Anwendung bestand darin, die

gegebenen Informationen des Benutzers in arrays und Matrizen abzuspeichern, mit der Absicht, sie transformieren zu können.

Mittlerweile haben die hierzu geführten Studien eine höhere Bedeutsamkeit und vor allen Dingen ein wesentlich besseres Reservoir an Hintergrund- und Grundwissen erreicht, sodass die derzeitigen Forschungen mit einem weitaus höheren Fortschritt betrieben werden können als noch vor wenigen Jahren.

Den größten Erfolg jedoch erreichten Henry Liebermann und Hugo Liu von der Massachusetts Institute of Technology in Cambridge.

Sie haben sich bisher am meisten mit der Thematik auseinandergesetzt und entwickelten im Jahre 2004 das Programm Metafor. Dieses System übersetzt einen über einen Editor eingegebenen Text Schritt für Schritt in einen Programmcode. Dieser Programmcode ist zwar nicht immer ausführbar, stellt aber zumindest ein Programmskelett dar. Auf dieses Projekt werde ich jedoch in einem folgenden Kapitel jedoch noch näher eingehen.

5. Die Anforderungen an die KI in der Programmierung

Um nachvollziehen zu können, welchen Anforderungen ein künstliches System unter anderem gewachsen sein muss, will ich hier einige Grundprinzipien der Programmierung erläutern.

Das Ziel des zu entwickelnden Systems, welches natürlich-sprachlich programmieren soll, ist es, ein fertiges funktionsfähiges Programm zu erstellen.

Was bedeutet dies nun im detaillierteren Sinne?

Zunächst einmal muss das System erkennen können welche Anforderungen die gegebene Aufgabe stellt. Das Scheitern vieler Entwicklungen insbesondere von unerfahrenen Programmieren liegt oftmals im Missverständnis bzw. der ungeklärten Definition der geforderten Anwendungen. Es ist nicht unbedingt immer einfach, klar zu erkennen, welche Anforderungen in einer gegebenen Aufgabe gestellt werden und wo genau die Unterteilungen der einzelne Aufgabenschritte sein sollen. Eine Hilfe hierzu zum Beispiel das divide-and-conquer-Prinzip. Diese algorithmische Lösungsform teilt ein vorliegendes Problem so lange in Teilprobleme, bis diese jedes für sich auf eine wesentliche einfachere Art zu lösen sind als das große komplexe Gesamtproblem.

Doch allein die Modularisierung stellt den Anfänger oftmals schon vor große Schwierigkeiten. Wo genau liegen die verwertbaren Informationen, wie teile ich sie ein und wie verwende ich sie für eine Lösung? Des Weiteren müssen die einzelnen Programmschritte genau definiert werden: Wo befinden sich Bedingungen, Wiederholungen, welches Objekt ist eine Klasse und welches die Oberklassen. Wo befinden sich weitere Funktionen?

Im Allgemeinen versteht man unter Programmieren selbst, ein Computerprogramm mittels einer Programmiersprache zu erstellen. Die Programme werden in der Regel in einer Programmiersprache ausgedrückt. In dieser Sprache formuliert der Programmierer „Algorithmen“, die genau definierten Handlungsvorschrift zur Lösung des Problems. Zunehmend wird er dabei durch Codegeneratoren unterstützt, die Teile des Programmcodes automatisch erstellen. Die Programmiersprache richtet sich hierbei nach einer vorgegebenen Syntax, welche die grammatischen Grundkonstruktionen der Programmiersprache mittels Regeln definiert. Von der künstlichen Intelligenz wird nun also verlangt, das Problem zu erkennen, zu unterteilen, Lösungswege zu finden und daraus in einer Programmiersprache einen Code zu implementieren.

6. Die beschreibende natürlich-sprachliche Programmierung:

Es gibt verschiedene Arten der Programmierung:

logische, erklärende, prozedurale, funktionale, maschinennahe, objektorientierte, agentenorientierte - um nur einige Beispiele zu nennen. Um Programme zu erstellen, die die reale Welt am besten imitieren, ist am sinnvollsten, entweder die objekt- oder agentenorientierte Programmierung zu benutzen, da sie der Anforderung am ehesten gerecht wird, Beziehungen zwischen Menschen und Dingen (Objekt, Agenten, Klassen) untereinander abstrahiert darzustellen.

Um die deskriptive Programmierung besser zu illustrieren wurde das Programmieren mit der Erzählung eines Märchens beschrieben. Ein Märchen beginnt mit dem Satz „ Es war einmal...“ Der Erzähler geht zu Beginn jedes Märchens zunächst einmal auf den aktuellen Zustand ein, und beschreibt, was existiert (ein Königreich, mit einem Schloss, einem großen Wald...)

Genauso wird zu Beginn eines Programmcodes erst einmal festgelegt, welche Klassen benötigt werden und welche Eigenschaften (public, private), Funktionen, Objekte oder Unterklassen sie beinhalten – was es gibt.

Ist der Erzähler mit der Beschreibung der Umgebung fertig, folgt immer der Satz “Eines Tages“. Hier beginnt der aktive Teil, es passiert etwas, (ein Ritter erscheint, die böse Hexe zaubert). Die Protagonisten beginnen zu agieren und erzeugen mit ihrem Handeln eine Wirkung auf ihre Umwelt. Genauso im Programm: Konstruktoren werden aufgerufen, Variablen übergeben und Objekte werden verändert. Hier würde man dann von der prozeduralen Programmierung sprechen, da sie sich hauptsächlich danach richtet “Was passiert?“ und sich mit dem Prozess der Veränderung auseinandersetzt.

Es gibt mehrere strukturelle Gemeinsamkeiten zwischen der natürlichen Sprache und den beschreibenden Strukturen einer Programmiersprache. Betrachtet man die Grammatiken der heute gesprochenen Sprachen genauer, fällt auf, dass fast alle nach dem gleichen Prinzip aufgebaut sind: Subjekt-Verb-Objekt: Wer tut was?

Natürlich gibt in den Sprachen untereinander wiederum Unterschiede, was den detaillierten Aufbau an sich betrifft. Im Japanischen, Lateinischen oder Türkischen treten Subjekt, Objekt Verb in der oben gezeigten Reihenfolge auf. Auch werden Dialekte vorerst vollkommen ignoriert. Nebenbei bemerkt wird hier derzeit nur von der Sprache Englisch ausgegangen, die folgenden Beispiele sind lediglich zum Verständnis in Deutsch geschrieben,

Ich werde Ihnen nun illustrieren, wie ein Programm mittels deskriptiver Programmierung übersetzt wird. Die jeweiligen Informationen, die für den gegenwärtigen Zustand relevant sind, werden Stück für Stück verarbeitet und in Klassen implementiert. Die Programmiersprache ist Python.

Meine gegebene erste Information ist:

Es gibt eine Bar mit einem Barkeeper, der Cocktails mixt.

Nun wird die Klasse Bar, die einen Barmixer beschäftigt, und der Barkeeper selbst, der die Fähigkeit besitzt, Cocktails zu mixen, implementiert.

```
Class bar:
    Barkeeper = barkeeper ()
Class barkeeper :
    Def drink : pass // „pass“ ist so etwas wie ein leerer Konstruktor
```

Hier zeigt sich, dass schon ein Satz allein mehrere Informationen beinhaltet:

1. Es gibt eine Bar
2. Die Bar beinhaltet einen Teil, der Barkeeper heisst
3. Der Barkeeper kann Cocktails mixen.

Hier muss man außerdem beachten, dass sich das „der“ nach dem Komma auf den Barkeeper und nicht auf die Bar selbst bezieht.

Kommen wir zur nächsten Information:

Die Bar hat eine Karte, auf der einige Getränke stehen:

Eine alkoholfreie Apfel-Schorle, ein Caipirinha, und ein Tequila-Sunrise:

Nun sieht die Implementierung folgendermaßen aus:

```
class bar:
    der_barkeeper = barkeeper();
    die_karte = karte();
class barkeeper:
    def mixen(getraenke):
pass class karte:
    getraenke = [apfelschorle, caipirinha, tequila sunrise]
class getraenke: pass
class apfelschorle(getraenk):
    properties = [alkoholfrei]
class caipirinha(getraenke): pass
class tequila-sunrise(getraenke): pass
```

Aufgrund des Schlüsselworts „einige“ wird das Nomen Getränke sofort als Liste identifiziert, weil das Wort ja signalisiert, dass es mehrer Objekte einer Art gibt. Jedes dieser Objekte beinhaltet die Oberklasse „getraenke“. Das Wort „alkoholfrei“ steht als Adjektiv direkt vor dem Nomen und wird aufgrund der Satzbau-Identifizierung als Eigenschaft erkannt.

So sieht man also, wie eine vorgegebene Information, in deskriptiver Weise umgesetzt wird.

7. Die prozedurale natürlich-sprachliche Programmierung:

Kommen wir zu einer Alternative der deskriptiven Programmierung, der prozeduralen Programmierung. Bei ihr wird der zu programmierende Text anhand der aktiven Eigenschaften interpretiert.

Um einen gegebenen Text in einen Programmcode zu übersetzen, müssen die vorhandenen Informationen zunächst einmal herausgefiltert werden. Das Programm wird nach vorhandenen Informationen abgesucht. Die für uns nun wesentlichen Informationen lassen sich in drei Teile gliedern

1. Schritte: Sie unterteilen das Programm in mehrere Teilschritte, welche wiederum nach Informationen abgesucht bzw. gefiltert werden.
2. Schleifen: durch sie werden die auszuführenden Anweisungen an bestimmte Bedingungen geknüpft
3. Kommentare: Sie ergänzen den Programmcode um weitere Informationen aus dem gegebenen Text.

Das System arbeitet nun quasi wie ein Sieb: es geht durch den Text durch und filtert nach und nach die nötigen Informationen heraus.

Da es wie oben genannt 3 Grundbausteine gibt, gibt es auch 3 „Siebmethoden“:

1. Step-Finder
2. Loop Finder
3. Comment Identification Component

Zuerst wird der vorhandene Text durch den Step-Finder in Programmschritte gegliedert. In der prozeduralen natürlich-sprachlichen Programmierung besteht ein Programmcode normalerweise aus einer Reihe von auszuführenden Anweisungen. Somit wird der Text Stück für Stück durchgegangen und sequentiell in einen Code übersetzt. Das Prinzip erinnert an divide-and-conquer, die Teilerlegung eines großen Problems in mehrere kleine Probleme, deren Lösung wesentlich einfach umzusetzen ist.

Als nächstes wird die Comment Identification Component eingesetzt. Obwohl Kommentare nicht unbedingt einen wichtigen Teil eines Computerprogramms einnehmen, geben sie doch hilfreiche Informationen, welche das Verständnis erleichtern. Es wird nach Sätzen gesucht, die eine beschreibende Rolle des aktuellen Schrittes übernehmen könnten.

Hierbei wird nach Merkmalen innerhalb des Satzes wie „Zum Beispiel“ gesucht die Zweitrangigkeit signalisieren. Ein weiterer Indikator sind die passive oder konjunktive Verbformen. Bei jedem Programmschritt wird geprüft, ob er eventuell auch eine beschreibende Rolle übernehmen könnte.

In der Ausgabe der Comment Identification Component werden die Programmschritte je nach Status (ob nun „kommentarwürdig“ oder nicht) mit einem Flag markiert. Man muss hierbei bemerken, dass jeder Schritt, auch wenn er schon als Programmschritt identifiziert wurde, die Rolle eines Kommentars übernehmen kann. Umgekehrt jedoch ist das nicht mehr möglich, da Text schon „deaktiviert“ wurde.

Als wichtigster Teil eines Programms gelten wohl die Schleifen.

Sie sind für jede Ausführung verantwortlich, bestimmen sie doch erstens die Bedingung und zweitens die Anzahl der Ausführungen.

Die Rolle des Loop-Finders beinhaltet, die natürlich-sprachlichen Strukturen zu finden, die auf eine Wiederholung hinweisen. Auch er geht durch den Text durch und überprüft jeden Schritt. Zuerst wird nach expliziten Merkmalen wie bestimmte Schlüsselwörter einer Wiederholung gesucht. Nehmen wir zum Beispiel den Begriff „jede“. Das Wort setzt voraus,

dass „mehrere“ existieren und jedes einzelne angesprochen wird. So muss es also zu einer Wiederholung kommen. Steht fest, dass eine Schleife existiert, sucht der Loop-Finder nun nach der so genannten „Schleifenvariablen“. Sie ist die Bedingung der Schleife, von der die Ausführung der Anweisungen abhängt.

Falls bei der Prüfung keine Wiederholungsmerkmale dieser Art gefunden wurden, wird als nächstes nach Pluralformen eines Nomens gesucht. Das hat einen ähnlichen Grund wie das Wort „jede“: es existieren mehrer Dinge einer Art, also muss es zu einer Wiederholung kommen. Da auch hier wiederum die Tätigkeit begrenzt sein muss, geht man nun daran herauszufinden, an welche Bedingungen die Ausführung gekoppelt ist bzw. wie oft sie durchgeführt wird. Einfach ist es, wenn Wörter, die leicht zu identifizieren sind, sich innerhalb des Satzes befinden, wie zum Beispiel „Zahl“. Findet man dann auch noch wirklich eine Zahl, kann man schon eine Schleife zusammenschließen. Die gefundenen Wörter, die eine gewisse Rolle übernehmen, werden nun in die aktuell benutzte Programmiersprache eingesetzt.

Beispiel: *Erzeugen Sie 1000 Zufallszahlen und zählen sie ihre Summe zusammen*

```
public class zufallszahl{
    Public static void (String [] args){
        For ( int i = 0; i < 1000, i++){
            int Summe = 0;
            int zahl = random.zahl;
            Summe = +zahl;
        }
    }
}
```

Schleifenvariable und Schleifenzahl befinden sich hier innerhalb eines Satzes und können problemlos identifiziert werden. Eine weitere wichtige Rolle des Loop-Finders ist es, herauszufinden ob sich mehrer Wiederholungsanweisungen unter einer Schleife vereinen lassen. Bei dem obigen Beispiel wären es beispielsweise „erzeugen“ und „Summe zusammenzählen“.

Allgemein ist die prozedurale Programmierung ein wesentlich wichtigerer Teil der natürlich-sprachlichen Programmierung, da erst sie wirklich das Programm laufen lässt und ein Ergebnis erzeugt. Sie prüft „Siebung“ für „Siebung“, welche Informationen wie verwertbar sind, und versucht daraus ein Programmskelett zu erstellen.

Beispiel:

Wenn ein sich ein Kunde ein Getränk bestellt, mixt der Barkeeper ihn. Wenn der Barkeeper beauftragt wird, ein Getränk zu mixen, macht er es und gibt es dem Kunden, aber nur wenn es das Getränk auf der Karte gibt, ansonsten sagt der Barkeeper: „ Entschuldigen Sie, aber ich weiss nicht wie dieses Getränk gemixt wird.“

```
class barkeeper:
    def make(getraenke):
        if (drink in .getraenke):
            barkeeper.mixen(getraenke)
            barkeeper.geben(getraenke, kunde)
```

Die If-Schleife wird aufgestellt:

Falls der Kunde ein Getränk bestellt, wird es der Barkeeper für den Kunden mixen.

```

else:
    barkeeper.say( \
"Entschuldigen Sie, aber ich weiss nicht wie dieses Getränk
gemixt wird.", kunde)

```

Ansonsten sagt der Barkeeper: „Entschuldigen Sie, aber ich weiss nicht, wie dieses Getränk gemixt wird.“

```

def geben(getraenk, zu_kunde): pass
def say(quote, zu_kunde): pass

```

Zum Schluss wird nun die Funktion definiert, das Getränk zu mixen, vorausgesetzt dass es bestellt wurde.

```

class customer:
    def bestellen(getraenk):
        barkeeper.mixen(getraenk)

```

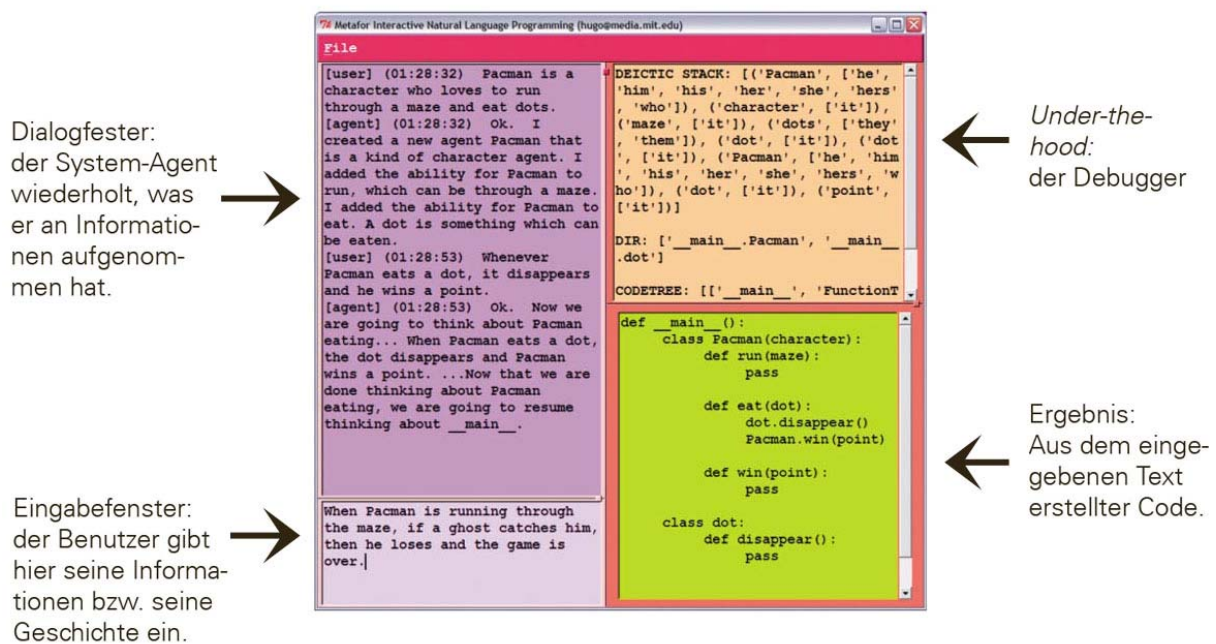
Hier wird nun die Klasse Kunde mit den Funktionen “bestellen” (ein Getränk zu bestellen, welches der Barkeeper dann mixt) implementiert.

Existiert das Wort “wenn” in einem Satz, so wird die folgende Äußerung von vorneherein eingegrenzt. Hier wird also sofort das Wort „wenn“ als Schlüsselwort für eine Bedingung erkannt, so dass dieser Satz mit dem jeweiligen Nomen als Schleifenvariable dient. Im folgenden Hauptsatz werden dann Pronomen wie „es“, „er“ und „sie“ als Bedingungsvariablen weiterhin aufgefasst und verarbeitet. Das Hauptverb wird im Normalfall sofort als Funktion umgesetzt.

Die prozedurale Programmierung stellt eine Alternative zur deskriptiven dar, man kann die eingegeben Informationen so oder so auffassen.

8. Metafor

Hier präsentiere ich Ihnen nun eine Software, die die natürlich-sprachliche Programmierung umsetzt. Das „Metafor“ genannte Programm ist ein von Hugo Liu und Henry Liebermann entwickeltes Verfahren, dass zumindest annähernd gesprochene Sprache in einen Code, bzw. in ein Programmskelett umsetzt. Es besteht aus vier Fenstern, die wie hier dargestellt, unterschiedliche Funktionen übernehmen. Das erste violette Fenster ist das Dialogfenster. Hier werden die gegebenen Informationen noch einmal hinterfragt und wiedergegeben. Im Eingabefenster darunter (hellviolett) gibt der Benutzer die jeweiligen Informationen oder Aufgabenstellungen schriftlich ein. Nun aktualisiert das Programm sich immer wieder und baut somit mit den jeweils in den Editor neu eingegeben Informationen Stück für Stück ein Programmskelett auf. Auf dem Debugger auf der rechten oberen Seite kann man die aktuelle Aktion beobachten, im grünen Fenster unten links ist das derzeitige Ergebnis, das Programmskelett, zu sehen.



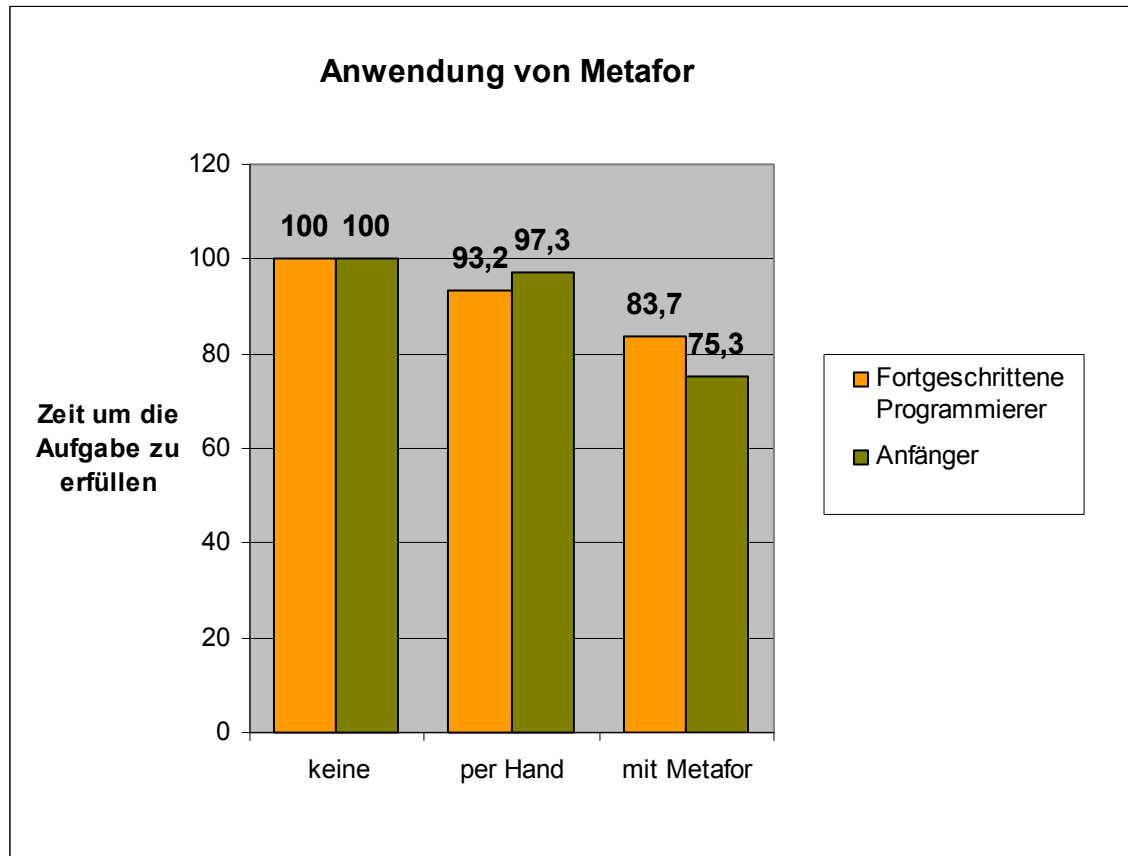
Natürlich kann der Analysator des Programmes nicht jede grammatische Redewendung der Eingabe interpretieren, dennoch erreicht er dank in den vorherigen Kapiteln beschriebenen Techniken wie Loop-Finder, Step-Finder etc. Erfolge, was den groben Programmaufbau betrifft. Die ursprüngliche Idee ging jedoch davon aus, das der Benutzer zumindest schon teilweise etwas von Programmiersprachen versteht und Metafor für ihn eher ein Anschauungsobjekt darstellen soll, wie die jeweiligen Informationen in das Programm hineingearbeitet werden. Somit ist das Programm immer noch vielmehr eine Hilfestellung für eine „teilweise“ Programmierung als ein wirkliches Übersetzungsprogramm.

9. Studien, Erkenntnisse und die Zukunft:

Es wurden mehrere Studien durchgeführt, inwiefern effektiv die Benutzung von natürlich-sprachlichen Sprachverarbeitungsprogrammen im Vergleich zum selbstständigen Programmieren ist. Ungeübte Programmierer bzw. Programmieranfänger sowie sehr geübte Programmierer und Entwickler wurden getestet. Ihnen wurden verschiedene Programmieraufgabenstellungen gegeben, die sie einmal mit und einmal ohne die Zuhilfenahme von Metafor erfüllen sollte.

Hierbei zeigte sich, dass die Benutzung für den Anfänger wesentliche Ergänzungsmöglichkeiten darbot. Sie konnten die Aufgaben schneller und mit einer größeren Präzision und Richtigkeit lösen. Bei den fortgeschrittenen Programmieren jedoch verlagerte sich die Kurve, und ihr Tempo, die Aufgaben zu lösen war teilweise sogar schneller und effektiver als die von Metafor. Dies zeigt, dass das Programm seine Grenzen erreicht, sobald der Benutzer einen gewissen Erfahrungsgrad im Programmieren erreicht hat. Natürlich bietet es große Vorteile für Programmieranfänger, die Programmieren lernen möchten und somit für jede Hilfestellung dankbar sind.

Allerdings sind die Möglichkeiten der natürlich-sprachlichen Programmierung bald erschöpft, da es beim derzeitigen Entwicklungsstand über die Erkennung von Bedingungsschleifen, Programmschritten, Funktionen und Programmschritten nicht hinausreicht. Bei dieser Graphik kann man die zeitliche Erfolgsverteilung bei der Benutzung von Metafor für erfahrene und unerfahrene Programmierern verfolgen.



Würden die Bedingungen schon mehr als 2 x verschachtelt werden oder simple Schlüsse aus dem Kontext zu ziehen sein, sind die Kapazitäten dieses Systems endlich, da es sich ausschließlich nach regelmäßigen Gemeinsamkeiten der natürlichen Sprache richtet. Die gesprochene Sprache ist jedoch bei weitem nicht regelmäßig, sondern beinhaltet Mehrdeutigkeiten und Kontextbezüge, wie auch in der realen Welt. Eine über längere Zeit erfolgte Kommunikation bezieht sich neben aktuellen Aussagen und Ausformulierungen vor allem auf den Kontext, und hier wird es schwierig den Kontext mit einem Programm, das sich ausschließlich mit der Auflösung der Syntax befasst, zu erkennen. Dies ist jedoch, nebenbei bemerkt auch zwischenmenschlich nicht immer gleich erkennbar. Nehmen wir zum Beispiel den Satz

Peter sah Maria mit dem Fernglas.

Sah nun Peter Maria, weil er das Fernglas benutzte oder sah er Maria, die ein Fernglas in der Hand hielt...?

Simple Kommunikationsschwierigkeiten, die eben auch genauso in der realen Welt existieren, erschweren die Erstellung eines Spracherkennungssystem enorm. Es wird vielleicht möglich sein, kleinere Aufgabenstellungen oder Teilprobleme deren Informationsgehalt endlich ist, zu

übersetzen. Sobald es aber auf größere Projekte, wie zum Beispiel die Erstellung eines komplexeren Programms zugeht, welches ganz unterschiedlich Aufgaben, die ein detailliertes Lösungskonzept erfordern, zu bewältigen hat, wird man weiterhin die aktive Mitarbeit und das detaillierte Nachfragen einer realen Intelligenz benötigen.

10. Referenzen:

1. RADA MIHALCEA, HUGO LIU, AND HENRY LIEBERMANN, 2006:
NLP (Natural Language Processing) for NLP (Natural Language Programming)
2. HUGO LIU AND HENRY LIEBERMANN, MIT MEDIA LABORATORY:
<http://ll4.csail.mit.edu/slides/metafor.pdf>
english, the lightest weight programming language of them all
3. HUGO LIU AND HENRY LIEBERMANN April 2005:
Programmatic Semantics for Natural Language Interfaces
4. HUGO LIU AND HENRY LIEBERMANN Januar 2005:
Metafor: Visualizing Stories as Code
5. <http://www.primidi.com/2005/03/24.html>
6. <http://de.wikipedia.org/wiki/Computerlinguistik>
7. http://en.wikipedia.org/wiki/Natural_language_processing