

Universität Karlsruhe (TH)  
Fakultät für Informatik  
Institut für Theoretische Informatik

Interactive Systems Labs

**Dynamische Vokabularerweiterung  
für ein grammatikbasiertes Dialogsystem  
durch Online Ressourcen**

**Studienarbeit  
von  
cand. Inform. Landry C. Chouambe**

**betreut von  
M.A. Petra Gieselmann  
Prof. Dr. Alexander Waibel**

**Januar 2006**



# Inhaltsverzeichnis

<b>1</b>	<b>EINLEITUNG</b> .....	<b>4</b>
1.1	MOTIVATION .....	4
1.2	DAS DIALOGSYSTEM TAPAS .....	5
1.3	PROBLEMATIK .....	6
1.4	ÜBERBLICK.....	6
<b>2</b>	<b>GRUNDLAGEN</b> .....	<b>7</b>
2.1	SPRACHDIALOGSYSTEME.....	7
2.1.1	<i>Komponenten</i> .....	7
2.1.2	<i>Typen von Sprachdialogsystemen</i> .....	8
2.2	DIALOG MIT HILFE VON TAPAS.....	10
2.2.1	<i>Ontologie</i> .....	10
2.2.2	<i>Dialog-Ziele</i> .....	10
2.2.3	<i>Dialog-Moves</i> .....	11
2.2.4	<i>Ausführung von Aktionen</i> .....	11
2.2.5	<i>Datenbankkonvertierungsregeln</i> .....	12
2.2.6	<i>Grammatikregel</i> .....	12
<b>3</b>	<b>VERWANDTE ARBEITEN</b> .....	<b>13</b>
3.1	DIALOGSYSTEM MIT DYNAMISCHEM VOKABULAR .....	13
3.2	PLUG & PLAY SPEECH UNDERSTANDING .....	14
3.3	RESTAURANT VOICE PORTAL .....	15
3.4	BEZUG ZU DIESER ARBEIT .....	15
<b>4</b>	<b>DIE REZEPTAPPLIKATION</b> .....	<b>16</b>
4.1	DESIGN .....	16
4.1.1	<i>Herunterladen neuer Rezepte</i> .....	17
4.1.2	<i>Suche nach Rezepten</i> .....	18
4.2	IMPLEMENTIERUNG.....	23
4.2.1	<i>Anwendungsbeschreibung</i> .....	23
4.2.2	<i>Funktionen der Rezeptapplikation</i> .....	24
4.2.3	<i>Datenbank</i> .....	25
4.2.4	<i>Grammatik</i> .....	25
4.2.5	<i>Test-GUI</i> .....	26
4.2.6	<i>Einstellungen</i> .....	26
<b>5</b>	<b>EXPERIMENTE UND EVALUATION</b> .....	<b>27</b>
5.1	DURCHFÜHRUNG .....	27
5.2	ERGEBNISSE.....	28
5.3	DISKUSSION.....	29
<b>6</b>	<b>ZUSAMMENFASSUNG</b> .....	<b>31</b>
6.1	AUSBLICK.....	31
	<b>DANKSAGUNG</b> .....	<b>32</b>
	<b>INDEX</b> .....	<b>33</b>
	<b>LITERATURVERZEICHNIS</b> .....	<b>34</b>

# 1 Einleitung

## 1.1 Motivation

Heutige Dialogsysteme arbeiten mit vordefiniertem Vokabular und manuell erstellten Grammatikregeln. Da dies aufwendig ist und damit nie alle Benutzerformulierungen abgedeckt werden können, geht es in dieser Studienarbeit um die Erweiterung des Dialogsystems zur dynamischen Integration von neuem Vokabular. Dazu werden Informationen aus dem Internet heruntergeladen, aufbereitet und strukturiert und dann in einer Datenbank, auf die der Dialogmanager zur Laufzeit zugreift, gespeichert. Als einen Testfall wurde hier eine Rezeptdatenbank im Internet ausgewählt, die in den Dialogmanager unseres Haushaltsroboters integriert wurde, so dass er jetzt einige tausend Rezepte und Zutaten kennt.

Der Nutzen eines solchen Haushaltsroboters mit einem immensen Wissen über Kochrezepte wird deutlich, wenn man sich die folgenden alltäglichen Situationen vorstellt: Ab und zu wird erst beim Kochen bemerkt, dass man die Garzeit vergessen hat. Manchmal ist die Zutatenliste so lang, dass man immer nachschauen muss. Es kommt auch vor, dass man in der Küche ist, aber nicht weiß, was man mit den vorhandenen Zutaten kochen kann. Wenn man etwas kochen will, von dem nur der Name bekannt ist, oder wenn man in eine der bereits genannten Situationen gerät, dann schaut man in einem Kochbuch nach. Alternativ bieten sich auch Rezeptdatenbanken im Internet dafür an. Wenn man aber schon beim Kochen ist, dann ist das Durchsuchen von Kochbüchern nicht mehr so praktisch, da die Hände voller Teig oder nass sind. Für eine Suche nach den benötigten Informationen im Internet ist dieser Zeitpunkt auch gar nicht geeignet. Wenn man schon etwas in den Ofen oder auf den Herd gestellt hat, dann darf man nicht vergessen, dass die Zeit bei der Suche immer weiter läuft.

Der Haushaltsroboter soll soviel über Rezepte wissen, dass er Fragen wie „*How can I cook Spaghetti Bolognese?*“ (Wie kann ich Spaghetti Bolognese kochen?) oder „*What can I cook with tomatoes and cheese?*“ („Was kann ich mit Tomaten und Käse kochen?“) beantworten kann. Desweiteren soll der Roboter den Benutzer während des Kochens unterstützen, indem er auf Nachfrage die Zutaten des gerade zu kochenden Gerichts sowie dessen einzelne Kochschritte benennen kann.

Dazu wird die vorhandene Dialogapplikation zur Robotersteuerung so erweitert, dass dieser Roboter zum hilfreichen Kochhelfer wird. Es wird eine Datenbank aus dem Internet gelesen und zur Verwendung während des Dialogs aufgebaut.



**Abbildung 1: Haushaltsroboter in der Küche**

## **1.2 Das Dialogsystem TAPAS**

TAPAS (Holzapfel, 2005) ist ein domänen- und sprachunabhängiges Dialogsystem, das auf dem von Matthias Denecke (Denecke 2002) entwickelten Dialogmodell basiert. TAPAS ist damit ein Nachfolger des Dialogsystems ARIADNE, dessen Architektur speziell für die Unterstützung einer schnellen Prototypentwicklung entworfen wurde.

Das System benutzt eine semantische Repräsentierung von Benutzeräußerungen (Sprache oder Gesten). Dafür wird eine kontextfreie Grammatik und ein Domänenmodell (Ontologie) benötigt, in dem alle Konzepte, die der Roboter verstehen kann, definiert sind. Wenn alle benötigten Informationen für einen Dialogziel vorliegen, dann wird der entsprechende Dienst aufgerufen. Bei fehlenden Informationen zur Durchführung eines bestimmten Dialogziels, fragt das System nach.

Für die Spracherkennung wird das Janus Recognition Toolkit (JRtk) (Finke et al 1997) zusammen mit dem Ibis Dekoder (Soltau et al., 2001) verwendet. JRtk wurde von der Universität Karlsruhe (Deutschland) und der Carnegie Mellon University (USA) entwickelt. Der Spracherkenner und der Dialogmanager nutzen die gleichen linguistischen Ressourcen. Andere Spracherkenner, die auch mit kontextfreien Grammatiken arbeiten, können eingesetzt werden.

## 1.3 Problematik

Im Dialogsystem wurden die Rezepte mit deren Zutaten und Kochschritten bisher manuell eingetragen, wofür ein gewisses Verständnis des Dialogaufbaus mit TAPAS erforderlich ist. Dabei müssen auch alle Kombinationen von Zutaten für jedes Rezept, die der Benutzer den Haushaltroboter fragen kann, festgelegt werden. Hier muss man bedenken, dass verschiedene Gerichte mit den gleichen Zutaten gekocht werden können, was insbesondere für die Rezeptsuche nach Zutaten wichtig ist.

Wenn der Benutzer nach einem Rezept mit dem Namen „*Spaghetti Bolognese*“ fragt, dann sucht das Dialogsystem ein Rezept, das genau diesen Namen hat, und gibt die dazugehörenden Zutaten sowie die Kochschritte zurück. Lautet die Benutzeräußerung „*How can I cook Spaghetti*“ („Wie kann ich Spaghetti kochen?“) statt „*How can I cook Spaghetti Bolognese*“ („Wie kann ich Spaghetti Bolognese kochen?“), dann wird nur nach dem Rezept „*Spaghetti*“ gesucht. Bei einer erfolglosen Suche wird der Benutzer dementsprechend informiert. Aber über die Existenz eines ähnlichen – zumindest vom Namen her – Rezepts wird er nicht informiert. Möglicherweise wollte sich der Benutzer nicht die Mühe machen, den ganzen Rezeptnamen auszusprechen, oder nur herausfinden wie viele Sorten von Spaghettirezepten der Roboter/das System kennt.

Für Fragen nach Zutaten wie „*What can I cook with tomatoes and cheese?*“ („Was kann ich mit Tomaten und Käse kochen?“) müssen die Zutaten Tomaten und Käse mit einer der vorher definierten Zutatenkombinationen für irgendein Rezept übereinstimmen, bevor dieses als Vorschlag zurückgeben werden kann. Wenn in dieser Anfrage eine andere Zutat zitiert wird, die aber gleichzeitig mit den anderen Zutaten für kein Rezept verwendet wird, dann ist die Rezeptsuche erfolglos. Dem Benutzer wird dann mitgeteilt, dass man mit seinen Zutaten nichts Sinnvolles kochen kann.

Wenn ein bestimmtes Dialogziel finalisiert ist, dann wird der Diskurs gelöscht. Somit kann man leider nicht mehr wissen, nach welchem Rezept der Benutzer zuletzt gesucht hat. Wenn er schon beim Kochen ist und sagt „*What should I do in step four?*“ („Was muss ich im Schritt vier machen?“), kann nicht mehr auf das entsprechende Dialogziel zugegriffen werden.

## 1.4 Überblick

In Kapitel 2 werden einige Grundlagen vorgestellt, die für das Verständnis dieser Studienarbeit wichtig sind. Dabei werden Sprachdialogsysteme im Allgemeinen vorgestellt, sowie die Realisierung von Dialogen mit dem Dialogsystem TAPAS. Kapitel 3 stellt ein paar verwandte Arbeiten vor, in denen zu sehen ist, wie ähnliche Probleme gelöst und welche Konzepte dabei angesetzt wurden. Die Konzepte, die im Rahmen dieser Arbeit entwickelt und die Experimente, die durchgeführt wurden, werden in den Kapiteln 4 und 5 beschrieben und erörtert. Im Anschluss wird in Kapitel 6 die Zusammenfassung beschrieben.

## 2 Grundlagen

### 2.1 Sprachdialogsysteme

Sprachdialogsysteme können als eine erweiterte Anwendung von Sprachtechnologien gesehen werden. Sie sind Computersysteme, mit denen Menschen in einer Turn-by-Turn Basis interagieren können, und in denen die natürliche Sprache eine wichtige Rolle bei der Kommunikation spielt. Natürlichsprachliche Dialogsysteme sind deshalb so interessant, weil sie der zwischenmenschlichen Kommunikation am nächsten kommen und damit natürlich als sehr benutzerfreundlich gelten.

Die Hauptaufgabe von Sprachdialogsystemen besteht darin, eine Schnittstelle zwischen einem Mensch und Computer-basierten Anwendungen, wie Datenbanken oder Expertensystemen zu kreieren. Der Kommunikationsmodus geht von minimalen natürlichsprachlichen Eingaben – kleiner Wortmengen wie Zahlen von 0 bis 10 und die Worte *ja* und *nein* – bis zu einer relativ freien Eingabe. Die Eingabe selbst kann gesprochen sein oder durch eine Tastatur eingegeben werden. Die Ausgabe kann in sprachlicher Form, ein auf einem Bildschirm dargestellter Text oder ein Bild bzw. eine Kombination daraus sein.

#### 2.1.1 Komponenten

Nach McTear (McTear 2002) besteht ein solches System aus 6 wesentlichen Komponenten: Spracherkennung, Sprachverständnis, Dialogmanager, Kommunikation mit externen Systemen, Antwortgenerierung und Sprachausgabe.

Die *Spracherkennung* (speech recognition) wandelt das kontinuierliche Sprachsignal der Benutzeräußerung in eine Menge von Wörtern bzw. Phonemen um. Diese Erkennung kann sprecherabhängig oder -unabhängig sein. Charakteristisch für die sprecherunabhängige Spracherkennung ist die Eigenschaft, dass das System ohne eine vorhergehende Trainingsphase sofort mit der Spracherkennung beginnen kann. Heutzutage arbeiten die meisten Spracherkenner mit einer kontinuierlichen Anpassung an einen Sprecher, während dieser Sprecher mit dem System interagiert.

Die *Sprachverständniskomponente*, auch Parser genannt, führt syntaktische und semantische Analysen durch, um eine Repräsentation der Bedeutung der aus der Spracherkennung kommenden Wortsequenz zu erhalten.

Der *Dialogmanager* regelt den Informationsaustausch zwischen dem System und dem Benutzer. Er sorgt dafür, dass alle nötigen Informationen vor der Ausführung einer Aktion oder einer Anfrage an eine externe Applikation vorhanden sind. Mit von der Spracherkennung falsch verstandenen, falsch geparsten oder unvollständigen Eingaben des Benutzers muss hier gerechnet werden.

Dialogsysteme brauchen im allgemein eine *Kommunikation mit externen Systemen* wie Datenbanken oder Expertensystem, da die vom Benutzer angeforderten Daten oder Informationen oft darin liegen.

Bei der *Antwortgenerierung* wird die Information, die an den Benutzer geschickt werden soll, erzeugt. Hier werden der Inhalt, die Struktur und die Form dieser Informationen festgelegt.

Die *Sprachausgabekomponente* wandelt die Ausgabe der Antwortgenerierungskomponente – eine Wortkette – in eine gesprochene Form um. Die Sprachausgabe geschieht im einfachsten Fall durch die Wiedergabe von vorher aufgezeichneten Nachrichten, in denen einzelne Wörter zusätzlich eingefügt werden können. Dies funktioniert aber nur bei bekannter Ausgabe, ansonsten wird eine synthetische Sprachausgabe benutzt.

## 2.1.2 Typen von Sprachdialogsystemen

Basierend auf der Kontrollstrategie des Dialogs, können drei grobe Kategorien von Sprachdialogsystemen unterschieden werden (McTear 2002): Finite state-basierte Systeme, rahmenbasierte Systeme und agentenbasierte Systeme.

### 2.1.2.1 Finite state-basierte Systeme

Ein finite state-basiertes System basiert auf endlichen Automaten. In diesem Dialogsystem wird der Dialog vom System geführt. Der Dialogfluss ist vordefiniert und der Zustandsraum endlich. In jedem Zustand stellt das System eine Frage und wartet auf die Antwort des Benutzers und geht zum nächsten Zustand über. In dieser Antwort soll nur die verlangte Information enthalten sein. Alles andere würde das System stören.

System1: *What is your destination?*  
User1: *London.*  
System2: *Was that London?*  
User2: *Yes.*  
System3: *What day do you want to travel?*  
User3: *Friday.*  
System4: *Was that Sunday?*  
User4: *No.*  
System5: *What day do you want to travel?*

**Dialog 1: Beispieldialog mit einem finite state-basierten System – aus (McTear2002).**

### 2.1.2.2 Rahmenbasierte Systeme

Bei einem rahmenbasierten System gibt es im Gegensatz zu einem finite state-basierten System, Vorlagen für bestimmte semantische Konzepte mit Leerstellen (sog. Slots), die mit den vom Benutzer gegebenen Informationen gefüllt werden. Hier ist der Dialogfluss nicht vordefiniert. Er hängt von der Benutzereingabe und den Informationen ab, die vom System noch gebraucht werden.

System1: *What is your destination?*  
User1: *London on Friday around 10 in the morning.*  
System2: *I have the following connection ...*

**Dialog 2: Beispieldialog mit einem rahmenbasierte System – aus (McTear2002).**



### **2.1.2.3 Agentenbasierte Systeme**

Agentenbasierte Systeme erlauben eine Kommunikation zwischen dem Dialogsystem, dem Benutzer und der darunterliegenden Applikation. Diese Kommunikation wird als eine Interaktion von Agenten, jeder mit seiner eigenen Intelligenz und manchmal auch Wissen über andere Agenten, angesehen.

### **2.1.2.4 Systemtypen im Vergleich**

Finite state-basierte Systeme sind für sehr klar strukturierte Dialoge mit einer eingeschränkten Zahl an einzugebenden Informationen gut geeignet. Der Dialog ist unflexibel und wird bei Eingaben, die über die geforderten Informationen hinausgehen, gestört. Rahmenbasierte Systeme sind dagegen sehr flexibel. Der Benutzer kann in einer natürlichen Art und Weise sprechen. Der Dialogfluss wird von ihm beeinflusst. Wenn es aber um komplexere Dialoge geht, dann sind agentenbasierte Systeme geeignet. Der Benutzer und das System interagieren, um ein Problem zu lösen. Die Anforderung an die verschiedenen Dialogkomponenten ist dabei sehr hoch.

Im kommerziellen Bereich werden finite state-basierte Systeme bzw. Varianten davon wegen ihrer Einfachheit und ihrem kleineren Realisierungsaufwand bevorzugt eingesetzt.

TAPAS kann als eine Erweiterung der rahmenbasierten Dialogsysteme angesehen werden, indem Dialogziele definiert werden, die Aufgaben mit all ihren Parametern angeben, die ein Dialogsystem erfüllen kann. Darüberhinaus ist dieser Ansatz informationsbasiert in dem Sinne, dass Dialogzustände durch die Spezifität der vorhandenen Information beschrieben werden.

## 2.2 Dialog mit Hilfe von TAPAS

Mit TAPAS lassen sich Dialoge schnell realisieren. Dazu müssen in den meisten Fällen lediglich eine Anwendung und eine Grammatik beschrieben werden. Zu der Anwendungsbeschreibung gehören Ontologie, Dialog-Ziele, Dialog-Moves und Datenbankkonvertierungsregeln.

### 2.2.1 Ontologie

Die Ontologie enthält alle Objekte, Aktionen, Zustände und Eigenschaften, die das System wissen muss, um den Benutzer verstehen zu können. Sie ist hierarchisch aufgebaut. Man kann beispielsweise ein Objekt `obj_recipe` mit Rezeptname (`recipename`), Zutaten (`ingredients`) und Kochschritte (`method`) wie folgt definieren:

```
class obj_ingredient inherits generic:objects;

class obj_ingredientslist inherits generic:objects{
    obj_ingredient : ingr1;
    obj_ingredient : ingr2;
    obj_ingredient : ingr3;
};

class obj_recipe inherits generic:objects{
    base:string      : recipename;
    obj_ingredientslist : ingredients;
    base:string      : method;
};
```

#### Beispiel 1: Objekte in der Ontologie

Mit `class obj_ingredient inherits generic:recipe;` wird ein Objekt `obj_ingredient` vom Objekt `objects`, wie in der objektorientierten Programmierung abgeleitet. Alle Eigenschaften vom Objekt `objects` werden dann geerbt. So haben alle im Beispiel 1 definierten Objekte beispielsweise ein Attribut `NAME`, das für `objects` wie folgt definiert wurde: `generic:string : NAME;`

### 2.2.2 Dialog-Ziele

Jedes Dialog-Ziel (*goal*) stellt eine Beziehung zwischen einem Konzept und dem Dienst, der aufgerufen wird, sobald dieses Ziel erreicht ist, dar. Im Beispiel 2 wird ein einfaches Dialog-Ziel definiert, das dazu dient, dem Benutzer zu sagen, ob ein bestimmtes Rezept bekannt ist. Wenn die semantische Repräsentation [ `act_doYouKnowThisRecipe` ] im Diskurs vorkommt, dann wird der Dienst `say` mit dem Parameter `Yes, I know this recipe` (Ja, ich kenne dieses Rezept) aufgerufen. [ `act_doYouKnowThisRecipe` ] kommt hier von der Sprachverstehenskomponente (Parser), deren Aufgabe darin besteht, die Ausgabe vom Spracherkenner in eine bedeutungstragende semantische Repräsentation umzuwandeln. Z.B. `Do you know the recipe of Spaghetti Bolognese?` in `act_doYouKnowThisRecipe` – dabei wird

ein Objekt `obj_recipes` mit seinem Namen und seinen Zutaten, falls diese auch angegeben wurden, mitgeführt.

```
goal DoYouKnowThisRecipe {
  precondition:
    [ act_doYouKnowThisRecipe
      OBJ [ obj_recipes
           generic:NAME [ base:string ]
        ]
    ]
  ->
  bindings:
    internal://dialogue/say "Yes, I know this recipe" ;
};
```

### Beispiel 2: Dialog-Ziel

## 2.2.3 Dialog-Moves

Jedes Mal wenn eine Variable einer Menge von im Dialogmanager vordefinierten Variablen ihren Wert ändert, tritt ein Event auf. Dieses Event sowie der Zustand, in dem sich der Dialogmanager befindet, können innerhalb eines *Moves* abgefragt werden. Bei bestimmter Belegung dieser Variablen bzw. Zustände, kann z.B. geprüft werden, ob die Vorbedingung für ein ausgewähltes Dialog-Ziel erfüllt ist. So kann beispielsweise das Fehlen eines Parameters erkannt und dann ein Subdialog gestartet, ein anderes Dialog-Ziel angestoßen, eine Nachfrage an den Benutzer gestellt werden oder eine einfache Meldung an den Benutzer geschickt werden.

Das folgende Beispiel stellt den Move „StatementSearchRecipe“ für die Suche nach Rezepten dar, der den Dienst „say“ mit dem Parameter „*I don't understand the name of the recipe you are looking for. Please say it again*“ („Ich habe den Namen des Rezepts, wonach Sie suchen nicht verstanden. Bitte sagen Sie es noch einmal.“) aufruft, wenn das Dialog-Ziel SearchRecipe determiniert ist und der Name des Rezepts, das gesucht werden soll undefiniert ist. Es kann sein, dass dieser Name falsch erkannt oder gar nicht eingegeben wurde.

```
move StatementSearchRecipe {
  goal:( SearchRecipe = determined),
  path:($sem.[generic:ARG|generic:NAME] is undefined ) ->
  bindings:
    internal://dialogue/say " I don't understand the name of the recipe you are
                           looking for. Please say it again ";
};
```

### Beispiel 3: Dialog-Move

## 2.2.4 Ausführung von Aktionen

Im „bindings“-Bereich eines Goals oder Moves, erlaubt das System den Aufruf von Methoden bzw. Diensten innerhalb eines bestimmten Protokolls. Das Protokoll *internal* ist eines davon. Es dient dazu, interne Methoden des Dialogsystems aufzurufen (z.B: *say*). Ein weiteres Protokoll heißt *jpkg* und dient dazu, externe Java-Applikationen anzubinden. Die Methodenaufrufe

erfolgen synchron und werden als direkte Methodenaufrufe über die JavaBeans-API ausgeführt. Durch diese Schnittstelle kann die Java-Applikation angebunden, und eine Methode daraus aufgerufen werden.

### **2.2.5 Datenbankkonvertierungsregeln**

Die Datenbankkonvertierungsregeln teilen dem Dialogsystem mit, wie es partielle Informationen über ein Konzept, das im Diskurs vorliegt, mit zusätzlichen Informationen aus einer gegebenen Tabelle einer bestimmten Datenbank ergänzen kann.

### **2.2.6 Grammatikregel**

Die Grammatikregeln sind kontextfrei und sehen wie folgt aus:

```
public <act_hello,V,_> = 'hello';
```

```
public <act_tellRecipeNumber,VP,_> = 'how many recipes do you know';
```

Dabei müssen beispielsweise `act_hello` und `act_tellRecipeNumber` in der Ontologie definiert sein. Mit jeder hier definierten Benutzeräußerung ist ein Dialog-Ziel verbunden und es kann nur das erkannt werden, was hier definiert ist.

## 3 Verwandte Arbeiten

### 3.1 Dialogsystem mit dynamischem Vokabular

In (Chung et al 2004) wird ein Sprachdialogsystem vorgestellt, bei dem das Vokabular dynamisch erstellt wird. Ein wichtiges Ziel bei der Entwicklung dieses Sprachdialogsystems bestand darin, ein System zu entwickeln, das es während des Betriebs erlaubt, neue Wörter – vom Benutzer oder von dynamischen Informationsquellen über das Internet – dem Dialog hinzuzufügen.

Es wird ein Restaurantsauskunftssystem aufgezeigt, das diese Vision teilweise implementiert. Dabei wird die Klasse von Restaurantnamen ständig in Abhängigkeit von dem Kontext aktualisiert. Sogar wenn der Datenbank neue Restaurantnamen hinzugefügt werden, dann müssen die Haupt *finite-state transducers* (FSTs) im Spracherkenner oder Parser nicht neu kompiliert werden.

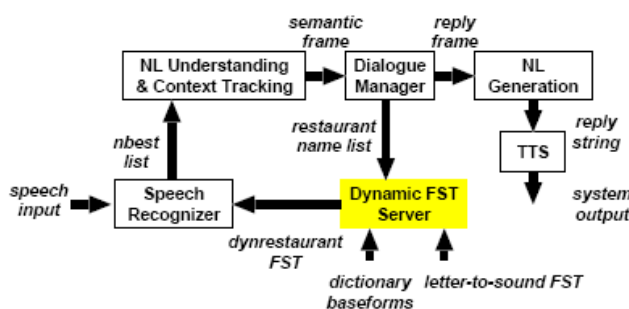


Abbildung 2: Interaktion von Komponenten – aus (Chung et al 2004)

Das System ist mit einem dynamischen FST Server ausgestattet. Gegeben eine Liste von Restaurantnamen, sucht es nach existierenden Aussprachen in einem Wörterbuch. Für unbekannte Worte benutzt es ein *letter-to-sound* Modul.

Bei einem Dialog, in dem der Benutzer nach einem Restaurant sucht und Informationen gibt wie Stadt, Typ des Restaurant etc., führt der Dialogmanager einen Datenbankzugriff aus und gibt eine Antwort zurück. Wenn höchstens 6 Restaurants gefunden wurden, dann werden diese Namen mit ihren Aliases an den dynamischen FST Server gesendet. Aus diesen erzeugt er die dynamische Erkennungsgrammatik. Wenn die Ergebnismenge dagegen zu groß ist, dann wird der Benutzer nach weiteren Informationen über das gesuchte Restaurant gefragt.

Es kann auch vorkommen, dass der Benutzer nach einem Restaurant fragt, das sich nicht im aktuellen Diskurs, also nicht in der dynamischen Grammatik für die Restaurantnamen befindet. In diesem Fall findet eine erneute Datenbankabfrage statt mit allen vom Benutzer bisher eingegebenen Informationen. Ist dieser Datenbankzugriff erfolgreich, fordert der Dialogmanager den FST Server an, eine neue Grammatik mit den gefundenen Namen zu erzeugen. Damit wird die dynamische Klasse aktualisiert und der Spracherkenner erneut gestartet, so dass der (vorher) unbekannte Restaurantname in eine der existierenden dynamischen Klassen aufgelöst werden kann.

Dieses Dialogsystem wurde intensiv und mit Erfolg getestet, wobei die Testdaten nicht von Benutzern stammen, sondern von einem Simulationsserver.

## 3.2 Plug & Play Speech Understanding

In (M. Rayner et al. 2001) wird die Architektur eines Dialogsystems beschrieben, das ein Netzwerk von Plug & Play Objekten in seiner Domäne unterstützt. Jedes Element dieses Netzwerkes trägt die linguistischen Informationen und die Dialogmanagementinformationen, die für seinen Betrieb und seine dynamische Einbindung in das Dialogsystem nötig sind.

Als konkretes Szenario wird die Steuerung von Geräten in einem Haus ausgewählt und vorgestellt. Der *Demonstrator*, der dieses implementiert, kennt drei Haupttypen von Geräten: schaltbare, dimmbare und Sensoren. Schaltbare Geräte haben nur zwei Zustände: ein oder aus. Dieser Zustand kann gesetzt oder abgefragt werden. Bei dimmbaren Geräten variiert der Zustand in einem Wertebereich. Hier kann der Zustand gesetzt, geändert oder abgefragt werden. Bei Sensoren dagegen kann der Zustand nur abgefragt werden. Befehle und Abfragen werden mit Hilfe von Slot-Werte Strukturen kodiert und müssen von der Grammatik der Geräte generiert werden.

Die Interpretation der Dialoge erfolgt in drei Schritten. Zuerst werden die Konjunktionen, die sich in den Eingaben befinden, in einer Menge von Slots-Strukturen (insgesamt 7) umgesetzt. Dann wird eine Art elliptischer Resolution gemacht, in der fehlende Slots einfach mit ihren letzten Werten gefüllt werden. Und zum Schluss wird versucht Geräte-Identifikatoren für Pronomen und beschriebene Geräte zu finden.

Die Grammatik ist verteilt. Jedes Gerät liefert die Informationen, die der Spracherkennung braucht, um die für dieses Gerät relevanten Äußerungen zu erkennen und in eine semantische Repräsentation umzuwandeln. Dies erfolgt, indem jede Art von Gerät die Kern-Grammatik, die aus linguistisch orientierten geräteunabhängigen Regeln besteht, in drei möglichen Richtungen erweitert. Sie kann um neue lexikalische Einträge erweitert werden. Dabei geht es um gerätespezifische Namen oder Sätze – ein Gerät muss z.B. mindestens einen Namen liefern, mit dem man es identifizieren kann. Sie kann um neue Grammatikregeln erweitert werden – eine dimmbare Lampe muss z.B. Regeln wie „nochmals 10 Prozent“ oder „etwas heller“ einbringen. Die Kern-Grammatik kann schließlich um neue Merkmale erweitert werden.

Die Kern-Grammatik wird vererbt und weiter verfeinert. Wenn neue Geräte, die die schon vorhandenen Typen weiter verfeinern, eingefügt werden, dann wird die Grammatik des Systems vom Compiler angepasst. Das Gerät steht dann zur Verfügung.

### 3.3 Restaurant Voice Portal

Das *Restaurant Voice Portal* ist ein Dialogsystem für Restaurants, das von A. Gruenstein (Gruenstein 2002) entwickelt wurde. Es generiert seine Grammatik automatisch, basierend auf seiner back-end Datenbank.

Mit diesem System können Restaurantinhaber einfach und schnell ihre Menüs aktualisieren, und dies ohne Wissen über VXML oder Grammatiken. Dazu müssen Sie nur in der Lage sein einen Web-Browser zu benutzen, denn das System verfügt über eine Web-basierte Schnittstelle, die es erlaubt Restaurantmenüs zu erstellen und zu bearbeiten. Man kann auch Gruppen wie *Drinks*, *Pasta*, *Entrees*, etc. anlegen sowie eine Beschreibung dazu. Diese Schnittstelle sind einfache HTML-Formulare, die mit PERL generiert werden.

Über ein Telefon kann der Benutzer mit dem System kommunizieren. Er kann durch das Menü navigieren und eine Bestellung durchführen. Ihm wird nicht eine lange Menüliste auf einmal dargeboten, sondern immer in dreier Listen zunächst von verschiedenen Kategorien, dann von konkreten Menüs. Die Anzahl der zurückgegebenen Elemente ist auf drei Chunks festgelegt, damit sich der Benutzer noch erinnert, wie die übergeordnete Kategorien heißen. Für die Auswahl eines Menüantrags mit dem Namen „*Hobee's gourmet granola*“ kann beispielsweise der Name „*granola*“ genügen, wenn man in dem entsprechenden Kontext ist. Diejenigen, die schon wissen, was Sie wollen, müssen nicht durch den ganzen Dialog gehen, sondern können auch direkt den (gekürzten) Namen des gewünschten Gerichts sagen.

Überall im System wird automatisch generierte Grammatik benutzt, besonders bei den folgenden Bereichen:

1. Hauptmenü: für die Erkennung aller Menüanträge
2. Kategorie: für die Erkennung einer der drei gerade erwähnten Kategorien
3. Kategorieanträge: für die Erkennung einer der drei gerade erwähnten Kategorieanträge (Rezepte etc.)
4. Checkout: für Entfernung von Elementen aus der Auswahlliste.

Auf der einen Seite ist die automatisch generierte Grammatik ein Vorteil für das System, da das Restaurantmenü vom Restaurantinhaber ohne spezielle Kenntnisse erstellt und bearbeitet werden kann. Auf der anderen Seite werden sehr viele Sätze automatisch generiert, die der Benutzer höchstwahrscheinlich nicht sagen wird. Das System funktioniert viel besser als ein anderes mit einer statischen Grammatik, der Dialog ist für den Benutzer viel angenehmer.

### 3.4 Bezug zu dieser Arbeit

Ein der wichtigsten Ziele in den drei oben vorgestellten Arbeiten ist es, ein Dialogsystem zu entwickeln, das mit dynamischen Informationen arbeiten kann; ein Dialogsystem, bei dem die darunterliegende Datenbank während dem Betrieb aktualisiert werden kann. Dabei muss nicht das Dialogsystem neu kompiliert werden. Die Grammatik wird automatisch generiert. Man muss sich nur um die neuen Informationen kümmern (neue Geräte hinzufügen, neue Einträge in die Datenbank schreiben usw.). Ähnlich soll es auch in unserem System möglich sein, neue Rezepte, zur Laufzeit aufnehmen oder entfernen zu können. Das Einfügen neuer Rezepte soll einfach und ohne Bezug auf die vom Dialogsystem verwendete semantische Repräsentation sein. Die Grammatik soll dynamisch sein.

## 4 Die Rezeptapplikation

Im Rahmen dieser Studienarbeit wurde eine Applikation entwickelt, die auf der einen Seite Rezepte aus dem Internet herunterlädt und sie in einer lokalen Datenbank ablegt und auf der anderen Seite Rezeptsuche nach verschiedenen Kriterien in dieser Datenbank durchführt. Das Herunterladen von Rezepten kann automatisch oder auf Wunsch des Benutzers gestartet werden. Dank der dynamischen Vokabularerweiterung kann der Roboter sein Wissen über Rezepte ständig aktualisieren und erweitern. Mit einem Namen, einem Teil dieses Namens oder mit einer Liste von Zutaten kann er eine Rezeptsuche durchführen. Werden ein Rezept und eine Liste von Zutaten genannt, kann er auch sagen, welche bestimmte Zutaten noch zu besorgen sind.

Die online-basierte Datenbank, auf die zugegriffen wird, heißt *Food Down Under* (<http://www.fooddownunder.com>). Sie wurde aufgrund der detaillierten Informationen, die sie über Rezepte enthält, ausgewählt. Dazu zählen Zutaten mit Mengenangabe und sinnvoll aufgeteilte Kochschritte. Für die lokale Datenbank wurde MySQL aufgrund seiner Plattformunabhängigkeit ausgewählt.

Diese Applikation wird in das Dialogsystem als externe Applikation eingebunden. Wenn eine Benutzeräußerung richtig erkannt wurde, hat der Dialogmanager dafür zu sorgen, dass eine bestimmte Funktion dieser Applikation aufgerufen wird. Bei dem Aufruf werden Informationen, die aus dieser Benutzeräußerung extrahiert wurden, als Parameter übergeben.

### 4.1 Design

Alle Rezepte werden in einer lokalen Datenbank gespeichert. Diese besteht aus insgesamt drei Tabellen. Die erste enthält die Rezepte. Die zweite enthält die einzelnen Worte aus denen die Rezeptnamen gebaut sind. Die dritte Tabelle enthält alle Zutaten, die vom System erkennbar sind. Eine unsere zwei Hauptaufgaben besteht darin, Rezepte im Internet zu finden und in diese Datenbank einzufügen. Die andere ist die eigentliche Suche nach den Rezepten. Die Suche wird in der Datenbank und nicht direkt im Internet durchgeführt. Sie würde sonst zu langsam sein und einen Betrieb ohne Internetverbindung unmöglich machen.



## 4.1.1 Herunterladen neuer Rezepte

Jederzeit kann mit einer Äußerung wie „*Download some recipes*“ („Lade Rezepte herunter“) dem Roboter mitgeteilt werden, dass er Rezepte aus einer vorher festgelegten Rezeptdatenbank holen soll. Der Rezeptparser, der innerhalb der externen Applikation zusammen mit den anderen Funktionen implementiert ist, wird dann gestartet. Der Parser kann auch so eingestellt werden, dass er automatisch nach dem Programmstart mit dem Download beginnt.

Beim Starten, erzeugt dieser Parser die Datenbanktabellen, die er braucht, wenn sie noch nicht existieren und anschließend füllt er eine dieser Tabellen, *ingredients*, mit allen bekannten Zutaten aus. Sie kommen aus einer manuell erstellten Liste aller möglichen Zutaten. Danach wird eine Anfrage mit einer Nummer als Parameter an den Webserver gesendet, der die Rezepte enthält. Als Antwort liefert er das unter dieser Nummer gespeicherte Rezept in Form einer HTML-Seite zurück. Wenn unter dieser Nummer kein Rezept zugeordnet ist, dann wird eine Fehlerseite zurückgegeben.

Die vom Webserver gelieferte HTML-Seite wird geparkt. Es gibt bestimmte Sequenzen von HTML-Tags und Worte, die helfen, Informationen wie z.B. Rezeptname oder Zutaten in dieser Seite zu finden. Wenn das Ende der Seite erreicht ist, dann wird überprüft, ob alle Informationen, die ein Rezept ausmachen – mindestens ein Rezeptname, eine Zutat und ein Kochschritt – gefunden wurden. Falls nicht werden alle auf dieser Seite gefunden Informationen gelöscht. Wenn ja, dann wird das neu gewonnene Rezept in die Datenbank aufgenommen, unter der Voraussetzung, dass es noch kein Rezept gibt, das den gleichen Namen hat. Sonst wird es weggeworfen. Das Parsen erfolgt mit einer Geschwindigkeit von ungefähr 1,5 Rezepte pro Sekunde bei einer Internetverbindung von 11Mbit/s.

### 4.1.1.1 Aufnahme in die Datenbank

Ein neues Rezept in der Datenbank besteht aus dem Einfügen eines neuen Datensatzes in die Tabelle *recipes*. Danach werden auch die einzelnen Worte, aus denen der Name dieses Rezepts gebaut ist, in einer anderen Tabelle, *recipenamewords*, gespeichert. Vorher werden Funktionswörter – wie Präpositionen, Artikel etc. – gelöscht. Es werden nur die Worte gespeichert, die sich noch nicht in dieser Tabelle befinden. Sie muss alle Worte, aus denen Rezeptnamen bestehen, enthalten. Diese Worte werden später dazu dienen, die Benutzereingaben bei der Rezeptsuche nach Namen zu erleichtern. Näheres dazu im Abschnitt 4.1.2.1.

Für ein Rezept namens „*Southern Fried Fish and Hush Puppies*“ werden also die Worte *Fried*, *Fish*, *Hush*, und *Puppies* gespeichert. *Southern* und *and* werden wegelassen (siehe Beispiel 4).

Rezeptnamen	Zu speichernde Worte
Spaghetti Napoli	Spaghetti Napoli
Spaghetti Bolognese	Bolognese
Kung Pao Tofu	Kung Pao
Southern Fried Fish and Hush Puppies	Tofu Fried Fish Hush Puppies

**Beispiel 4: Worte aus den Rezeptnamen**

#### 4.1.1.2 Verfügbarkeit während des Parsens

Während der Rezeptparser läuft, kann sich der Benutzer nach der Anzahl der schon gefundenen Rezepten erkundigen und diesen auch vorzeitig beenden, wenn er beispielsweise denkt, dass die Rezeptanzahl für ihn ausreichend ist. Die anderen Funktionen des Systems sind immer verfügbar und können auch während dem Download aufgerufen werden. Dem Benutzer sollte nur klar sein, dass diese Operation in Abhängigkeit von der Datenrate seiner Internetverbindung viel Zeit in Anspruch nehmen kann.

User1: *Please download some new recipes*  
System1: *I am going to get some recipes from the internet.  
You can stop this download every time you want.  
You can also ask the number of recipes I have already found on the internet.*

User2: *Stop the download.*  
System2: *I am stopping the download.  
Do you want to know what I found on the internet?*

User3: *Yes*  
System3: *I have found 365 recipes.*

**Dialog 3: Neue Rezepte suchen**

### 4.1.2 Suche nach Rezepten

#### 4.1.2.1 Suche mit Namen

Das System erkennt eine Frage wie: „*How can I cook Spaghetti Napoli?*“ („Wie kann ich Spaghetti Napoli kochen?“). Dabei gibt der Benutzer einen vollständigen Rezeptnamen oder Teile eines Rezeptnamens ein. Wenn das System also die Rezepte *Spaghetti Napoli*, *Spaghetti Bolognese*, *Kung Pao Tofu* und *Southern Fried Fish and Hush* kennt, dann kann der Benutzer nach *Bolognese* statt *Spaghetti Bolognese* oder nach *Fried Fish* statt *Southern Fried Fish and Hush* etc. fragen (siehe Abschnitt 4.1.1.1).

how can I cook Spaghetti Napoli?  
how can I cook Spaghetti?  
how can I cook Napoli?  
how can I cook Spaghetti Bolognese?  
how can I cook Bolognese?  
how can I cook Kung Pao Tofu?  
how can I cook Kung?  
how can I cook Kung Pao?  
how can I cook Kung Tofu?  
how can I cook Pao Tofu?  
...

#### Beispiel 5: Suche mit Namen

Die Suche erfolgt in zwei Schritten. Zunächst wird geprüft, ob es ein Rezept gibt, das genau den eingegebenen Namen enthält. Wenn ja, dann wird er genommen und die Suche beendet. Wenn dieser Schritt aber kein Ergebnis geliefert hat, dann werden in einem zweiten Schritt alle Rezepte genommen, die die Mehrheit der eingegebenen Worte – aus dem Rezeptnamen – enthalten. In diesem Schritt werden noch mal alle Rezepte durchgegangen und Punkte vergeben. Für jedes eingegebene Wort, wird ein Punkt an jedes Rezept vergeben, das es enthält. Am Ende werden die Rezepte mit der höchsten Punktzahl genommen. Es werden maximal 10 Rezepte genommen, wobei jedes mindestens 1 Punkt haben muss. Wenn keines der Rezepte einen Punkt bekommen hat, dann gilt die Suche als erfolglos.

#### 4.1.2.2 Suche mit Zutaten

Wenn nur die Zutaten bzw. nur ein Teil von ihnen bekannt ist, dann kann man eine Frage stellen wie: „*What can I cook with tomatoes and cheese?*“ („Was kann ich mit Tomaten und Käse kochen?“). Alle Rezepte die *tomatoes* und *cheese* in ihrer Zutatenliste enthalten, werden damit gefunden. Genommen werden die Rezepte, die die meisten dieser Zutaten enthalten. Die Suche verläuft ähnlich wie die Suche bei bekanntem Namen (siehe 4.1.2.1). Dabei wird für jede enthaltene Zutat ein Punkt vergeben.

Für das Rezept *Raw Apple Relish* hat man beispielsweise die folgenden Zutaten:

- 2 lrg red apples
- juice of 1 lemon
- 1 med onion, chopped
- ¼ cup sliced sweet gherkins
- 1 tbl capers
- ¼ cup sliced ripe olives

Als Mensch ist es sehr einfach zu verstehen, dass es um *lemon* geht, wenn *juice of 1 lemon* geschrieben ist. Die Zutaten tauchen fast immer mit Mengenangaben zusammen und in verschiedenen Formen auf, so dass die gemeinte Zutat daraus zunächst extrahiert werden muss. Um dieses Problem zu lösen, wird eine Liste aller bekannten Zutaten verwendet (siehe Abschnitt 4.1.1). Zur Laufzeit wird für jede Zutat eines Rezepts geprüft, ob sie eines der Wörter aus dieser Liste enthält. Wenn ja, dann wird dieses Wort als eine der Zutaten dieses Rezepts genommen. So kann eine vereinfachte Version der Zutaten für jedes Rezept erstellt werden, um den Benutzer über ein Rezept kurz zu informieren. Die Originalversion der Zutaten wird aber immer gebraucht, da sie auch Informationen über die Menge einer Zutat enthält.

Für eine Suche mit Zutaten, gibt der Benutzer ausschließlich die Zutaten, die aus der vordefinierten Zutatenliste kommen. Man muss also dafür sorgen, dass diese Liste wirklich alle möglichen Zutaten enthält. Dabei wird nicht nur das eingegebene Wort in der Rezeptdatenbank gesucht, sondern auch sein Singular bzw. sein Plural. D.h. unabhängig davon ob der Zutatename „*tomato*“ oder „*tomatoes*“ ist, dann wird immer nach beiden Alternativen gesucht.

#### 4.1.2.3 Ergebnis einer Suche

Wenn eine Suche erfolgreich abgeschlossen wurde, dann werden zunächst die Rezeptnamen zurückgeben mit Nummern davor, die dazu dienen, auf ein Rezept zu referieren. Die gefundenen Rezepte – zunächst nur die Namen – werden in einer Dreier-Liste vorgestellt, so dass die Anzahl von Informationen, die der Benutzer auf einmal bekommt, auf drei Chunks begrenzt ist, um die Kapazität des Kurzzeitgedächtnisses des Benutzers nicht zu überlasten. Er kann dann diese Liste vor- und rückwärts blättern, um sich alle Rezeptnamen angeben zu lassen. Um eins der Rezepte auszuwählen, muss der Benutzer die Nummer oder den Namen dieses Rezepts sagen. Er kann beispielsweise sagen: „*I would like the recipe number one*“ („Ich möchte das Rezept Nummer 1“) oder „*Please select the first one for me*“ („Bitte wähle das erste für mich aus !“), wenn er das erste Rezept will. Wenn ein Rezept namens „Spaghetti Napoli“ in der aktuellen Rezeptliste vorhanden ist, dann kann er auch einfach diesen Namen sagen (oder nur einen Teil davon), um dieses Rezept auszuwählen. Z.B: *Spaghetti*. Enthält die Liste mehr als ein Rezept mit dem (Teil-) Namen „Spaghetti“, dann wird nur das erste ausgewählt.

Auswählbar sind auch die, die nicht unter den drei aktuell vorgestellten Rezepten sind. Wenn der Benutzer vergessen hat, wie sich eines dieser Rezepte nennt, dann kann er auch nach dessen Namen fragen. Ferner kann er eine weitere Gruppierung der gefundenen Rezepte vornehmen. Dazu nennt er ein paar Zutaten und die Rezepte, die diese Zutaten nicht enthalten, werden von der Ergebnismenge gelöscht.

Nachdem ein Rezept ausgewählt wurde, können weitere Informationen darüber angefordert werden. Wenn er nach den Zutaten fragt, dann werden die Namen der Zutaten des gewählten Rezepts zurückgeben. Die gesamte Zutatenliste, mit Angaben über Mengen und Form, kann auch per Wunsch gegeben werden. Wenn der Benutzer bei der Suche eine unvollständige Zutatenliste gegeben hat, dann kann er nach den anderen Zutaten fragen, die in seiner Liste fehlen. Er kann sich diese übrigen besorgen und das Rezept tatsächlich kochen. Er kann sich die verschiedenen Kochschritte nach und nach oder alle auf einmal geben lassen. Ein Rezept bleibt solange ausgewählt, bis der Benutzer ein anderes Rezept auswählt. Dies bedeutet, dass der Benutzer eine neue Rezeptsuche starten kann, oder dem System eine andere Aufgabe geben kann, und trotzdem jederzeit nach den Zutaten oder der Kochmethode des zuletzt ausgewählten Rezepts fragen kann. Wenn er selbst vergessen hat, um welches Rezept es sich handelt, dann kann er nach dem Namen nachfragen. Der Benutzer kann zum Beispiel sagen: *Which ingredient do I need to cook that?* („Welche Zutaten brauche ich, um das zu kochen?“) oder *Which recipe have you selected for me?* („Welches Rezept hast du für mich ausgewählt?“).

User1: *Hi Robbi*  
System1: *Hello! My name is Robbi. I am a little robot who can help you in the kitchen. What do you want me to do?*

User2: *Please do you know how to cook spaghetti bolognaise?*  
System2: *I can give you the recipe of: spaghetti bolognaise.  
Do you want me to list you now all the ingredients or to start with the cooking method?*

User3: *I want to know the ingredients*  
System3: *You need the following ingredients: spaghetti, beef, oil, tomato and oregano.  
I can also give the complete ingredients list, if you are interested.*

User4: *Tell me how many steps i need for this recipe please*  
System4: *You need 4 steps to cook spaghetti bolognaise.  
Do you want to start with the cooking method?*

User5: *Yes*  
System5: *You can cook it in 4 steps.  
Step 1: Peel and finely chop the onion.  
Heat the oil in a frying pan, and fry the onion until it is transparent, then the meat, crumbling it into the pan with your fingers.  
Add the tomatoes and the puree, salt, pepper and oregano.  
Mix together well, and simmer over a low heat.*

User6: *Show me the step number two.*  
System6: *Step 2: Bring to the boil a large saucepan of salted water.  
When the water is boiling, put in the spaghetti without breaking it.  
It is cooked when al dente, i.e. still firm when bitten.*

User7: *...*  
System7: *...*

**Dialog 4: Der Benutzer sucht ein Rezept, dessen Name er kennt.**

User1: *Hi*  
System1: *Good morning! How can I help you?*

User2: *What can i cook with tomato?*  
System2: *I know 10 recipes, which can interest you:  
number 1: spagetti sauce, number 2: spagetti sauce - bulgar vegan, and number 3: spagetti sauce italiano.  
You can say: next, or: previous to get the other recipes.  
If you want me to reduce this list of recipes, please tell me some more ingredients.  
To select one of first recipe, say for example: select the recipe number one.*

User3: *Next recipes*  
System3: *number 4: spagetti sauce with meatballs, number 5: spaghetti, and number 6: spaghetti al pomodoro e basilico.*

User4: *What is the recipe number eight?*  
System4: *The name of the recipe number 8 is spaghetti ala suzy de bus.*

User5: *I would like the recipe number two please*  
System5: *Okay! Do you want me to list you now all the ingredients or to start with the cooking method?*

User6: *Which other ingredients do i need?*  
System6: *For spagetti sauce - bulgar vegan you also need garlic, parsley, oregano and pepper.*

User7: *Tell me the next recipes please*  
System7: *number 3: spagetti sauce italiano, number 4: spagetti sauce with meatballs, and number 5: spaghetti.*

User8: *Show me the previous recipes*  
System8: *number 1: spagetti sauce, number 2: spagetti sauce - bulgar vegan, and number 3: spagetti sauce italiano.*

User9: *Please tell me the complete ingredients list*  
System9: *You need the following ingredients:  
Water for saute.  
1 cup: chopped onion.  
2 : cloves garlic, minced.  
1 cans: chopped peeled tomatoes, with liquid.  
1 cans: tomato sauce.  
1 tablespoon: parsley.  
1 : bay leaf.  
1/2 teaspoon: oregano.  
1/4 teaspoon: sugar.  
1/4 teaspoon: salt.  
1/8 teaspoon: pepper.  
1 cup: cooked bulgur.*

User10: *...*  
System10: *...*

**Dialog 5: Der Benutzer hat ein paar Zutaten und sucht ein passendes Rezept.**

## 4.2 Implementierung

Die Implementierung wurde in der Programmiersprache Java gemacht. Die programmierte Applikation wird durch eine JavaBean-Klasse (siehe <http://java.sun.com/products/javabeans/>) repräsentiert und angesprochen. Durch entsprechende Einstellungen wurde sie in das bestehende Dialogsystem eingebunden. In der bestehenden Grammatikdatei wurden neue Regeln eingefügt. Die Ontologie wurde um neue Dialog-Ziele und Moves erweitert.

### 4.2.1 Anwendungsbeschreibung

Die Hierarchie von Objekten und Aktionen, die in der Ontologie bereits verwendet wurde, wurde hier weiterverwendet und erweitert. Im Beispiel 6 ist ein Auszug daraus zu sehen.

Eine Liste von vier Zutaten wurde definiert. Bei einer Suche nach Zutaten, enthält diese Liste alle vom Benutzer eingegebenen Zutaten. Sollte es für den Benutzer möglich sein, mehr als vier Zutaten auf einmal pro Äußerung geben zu können, dann muss die Größe dieser Liste dementsprechend erhöht werden. Bei einer Suche nach Namen, kann der Benutzer anstatt eines vollständigen Rezeptnamen, bis zu drei bedeutungstragende Worte aus allen möglichen Rezeptnamen eingeben. Diese Worte werden zu einer Liste zusammengefasst, nach dem gleichen Prinzip wie die Zutatenliste.

```
class ingredient inherits generic:object;
class ingredientlist inherits generic:object {
    ingredient : INGREDIENT1;
    ingredient : INGREDIENT2;
    ingredient : INGREDIENT3;
    ingredient : INGREDIENT4;
};
class recipeNameWord inherits generic:object;
class recipeNameWordList inherits generic:object{
    recipeNameWord : WORD1;
    recipeNameWord : WORD2;
    recipeNameWord : WORD3;
};
```

#### Beispiel 6: Auszug aus der Ontologie

Für jede Aktion gibt es üblicherweise ein Dialog-Ziel mit seinen Parametern und den konkret auszuführendem Dienst. Die Überprüfung von Parametern und die Verknüpfung mit der Java-Applikation wurden innerhalb von Moves realisiert. Beispiel 8 zeigt einen der Moves, der für die Suche nach Rezepten bei bekanntem Namen verwendet wird. Das Dialog-Ziel *RecipeByName* wird bei einer bestimmten Benutzeraussage vom Dialogmanager ausgewählt und ausgeführt. Wenn der Zustand dieses Dialog-Ziels dabei auf *finalized* im Dialogmanager übergeht, dann wird geprüft, ob ein Rezeptname gegeben wurde. Wenn dies der Fall ist, dann wird die Funktion *searchByRecipeName* mit diesem Namen aufgerufen. Alle Rezepte, die diesen Namen haben, werden gesucht. Das Ergebnis der Suche – die Namen der ersten drei gefundenen Rezepten, wenn die Suche erfolgreich war, oder eine Meldung, dass kein Rezept gefunden werden konnte – wird in einer Datei vorläufig gespeichert. Mit Hilfe eines Perl-Scripts wird dieses Ergebnis gelesen und an den Dienst *say* zum Vorlesen weitergegeben. Anschließend wird der Inhalt dieser Datei gelöscht.

Eine Suche in dem Dialogmanager selbst wäre sehr schwer zu realisieren. Die Operationen, die er zur Verfügung stellt, sind für die Implementierung der hier vorgestellten Suchverfahren nicht ausreichend. Desweiteren können Aktionen mit dem Dialogmanager nicht parallel geführt. Z.B. das Herunterladen von Rezepten und die gleichzeitige Suche von Rezepten. Ein einziger Zugriff auf der im Dialogmanager integrierten Datenbank würde bei 30800 mehr als 30 Sekunden dauern, wobei für eine Rezeptsuche 1 bis 3 Datenbankzugriffe gebraucht werden. Anschließend müssen die gefundenen Rezepte noch bewertet werden, so dass der Benutzer nur die Besten bekommt. Für die gesamte Suche braucht die Rezeptapplikation bei der gleichen Rezeptanzahl maximal 4 Sekunden.

```
goal RecipeByName {
  precondition:
    [ act_seachRecipeByName ]
    ->
  bindings:
};
```

### Beispiel 7: Goal RecipeByName

```
move TMRecipes on variable Intention changed to finalized{
  goal:(RecipeByName = finalized),
  path:($sem.[RECIPENAME|generic:NAME] is defined) ->
  bindings:
    jpgk://localhost:5454/searchByRecipeName $objs.[RECIPENAME|generic:NAME];
    internal://dialogue/say °{
      from recimod.resultLoader import TextResult
      textToSay = TextResult.get()
      TextResult.clear()
      result = textToSay
    }°;
    internal://dialogue/addToTransformation "speech", [], robbi:act_navNextRecipes -> [], (act_showNextRecipesNames);
    internal://dialogue/addSymbolToSubGrammar "speech", [], <robbi:act_navNextRecipes,VP,_>;
    internal://dialogue/createSubGrammar "speech", "answer.xml";
};
```

### Beispiel 8: Move für die Suche nach Rezepten bei bekanntem Namen

## 4.2.2 Funktionen der Rezeptapplikation

Die Applikation selbst steht in Form eines Java-Archivs zur Verfügung und muss zum Klassenpfad (*classpath*) der Java-Virtuellen Maschine hinzugefügt werden. Der Name der JavaBean-Klasse, durch die diese Applikation repräsentiert wird, muss im Property-File von TAPAS angegeben werden. Der Dialogmanager ruft zur Laufzeit Funktionen daraus auf und gibt dabei die von der Benutzeräußerung extrahierten Informationen als Parameter.

Wenn beispielsweise das Herunterladen von Rezepten gestartet werden soll, dann muss die Funktion *startParsing()* aufgerufen werden. Mit der Funktion *stopParsing()* kann es wieder beendet werden. Für die Suche müssen die Funktionen *searchByIngredients(String strIngredients)* oder *searchByRecipeName(String strName)* aufgerufen werden. Diese Funktionen und alle Anderen, die das Dialogsystem aufruft, legen nach ihrer Ausführung die Ergebnisse in einer temporären Datei ab. Diese müssen dann vom Dialogmanager selbst abgeholt werden.



### 4.2.3 Datenbank

Die Implementierung wurde mit einer objekt-orientierten Datenbank angefangen. Diese wird als eine einfache Text-Datei gespeichert. Aufgrund einer großen Rezeptanzahl, waren die Datendankzugriffe bei komplexen Anfragen zu langsam, so dass stattdessen eine MySQL-Datenbank verwendet wurde (siehe <http://www.mysql.com>).

### 4.2.4 Grammatik

Die Grammatikregeln, die im Beispiel 9 zu sehen sind, erlauben es dynamische Regeln zu erzeugen, wie beispielsweise:

```
public <recipeNameWordList,NP,_> = spaghetti alla puttanesca bianca
    : spaghetti bake
    : buttermilk panna cotta with macerated strawberries
    : spaghetti
    : spaghetti alla
    : spaghetti puttanesca
    : spaghetti bianca
    : spaghetti cotta
    : spaghetti panna
    : spaghetti macerated
    ....
    : buttermilk
    : buttermilk panna
    : buttermilk cotta
    : buttermilk macerated
    : buttermilk strawberries
    : buttermilk puttanesca
    ...
    : banke;
```

Die vollständigen Rezeptnamen kommen aus der Datenbank von der Tabelle und die einzelnen Wörter von der Tabelle *recipenamewords*. Ähnlich wird auch die Eingabe von Zutaten ermöglicht. Dabei werden die Wörter ausschließlich aus der Tabelle *ingredients* genommen.

```
public <recipeNameWordList,NP,_> <recipeNameWord1>
    : <recipeNameWord1> <recipeNameWord2>
    : <recipeNameWord1> <recipeNameWord2> <recipeNameWord3>;

<recipeNameWord1> = <CompleteRecipeName>
    : <recipeNameWord,N,_> {WORD1 recipeNameWord};

<recipeNameWord2> = <recipeNameWord,N,_> {WORD2 recipeNameWord};

<recipeNameWord3> = <recipeNameWord,N,_> {WORD3 recipeNameWord};

public <recipeNameWord,N,_> = import jpgk://localhost:5454/Env?jpgk Recipenames name
    {generic:NAME import};

<CompleteRecipeName> = import jpgk://localhost:5454/Env?jpgk Recipe name {generic:NAME
    import};
```

#### Beispiel 9: Auszug der Grammatikregeln

## 4.2.5 Test-GUI

Das Testen der Funktionalitäten der Java-Applikation kann etwas mühsam sein, wenn man sie nur durch das Dialogsystem benutzen muss. Aus diesem Grund wurde eine kleine Benutzeroberfläche erstellt, um die Funktionalität der Java-Applikation außerhalb des Dialogsystems testen zu können. Sonst müsste man immer Grammatikregeln schreiben, neue Konzepte in der Ontologie und ein Dialog-Ziel definieren, um etwas testen zu können.

Abbildung 3 zeigt ein Screen-Shot der Test-GUI. Bis auf einige Ausnahmen, steckt hinter jedem der Buttons auf dieser Oberfläche eine Funktion, die vom Dialogmanager aufgerufen wird.

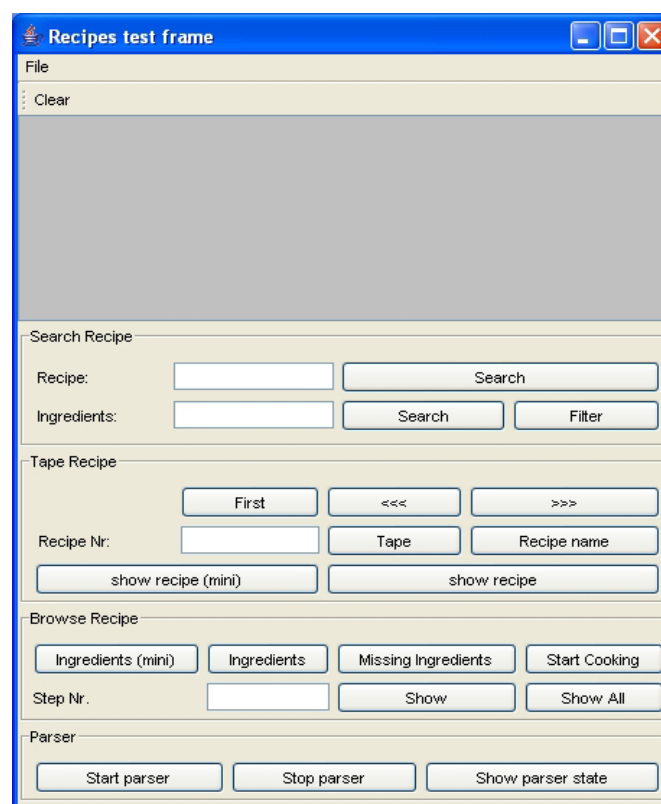


Abbildung 3: Test-GUI

## 4.2.6 Einstellungen

In der Property-File der Rezeptapplikation können verschiedene Einstellungen gemacht werden. Besonders wichtig sind Einstellungen für die Internetverbindung, die URL der Webserver für die Rezeptdatenbank und der Bereich, der beim Parsen durchsucht werden soll. Dabei geht es um Zahlen, die als Kennung für Rezepte verwendet werden. Wenn dieser Bereich z.B. von 20 bis 50 geht, dann werden 31 HTTP-Anfrage an den Webserver gesendet, mit jeweils einer Zahl als Parameter.

## 5 Experimente und Evaluation

Ein Benutzertest mit dem Gesamtsystem wurde durchgeführt. Die Testumgebung bestand aus:

- Rechner mit Intel(R) Pentium(R) CPU 3.00GHz, 1024MB Arbeitsspeicher
- Betriebssystem SUSE Linux 9.3, Kernelversion 2.6.11.4-20a-smp
- TAPAS Version 1.3.0 (SFB-Implementation)
- Rezeptapplikation (mit 30879 Rezepten)
- Spracherkennung Janus V5.0
- Sondereinstellung: PCM = 61, MIC = 81, CAPTURE = 87

Insgesamt haben 6 Benutzer daran teilgenommen, davon 33,33% weiblich und 66,67% männlich. Sie arbeiten alle sehr oft mit einem Rechner. Die Hälfte hat schon einen Spracherkennung benutzt, und ca. 16% ein Dialogsystem. Alle sind Studenten verschiedener Fachrichtungen.

### 5.1 Durchführung

Die Aufgabenbeschreibung war auf Deutsch, aber die Benutzer mussten sie auf Englisch lösen, weil die Rezeptapplikation nur für die englische Sprache entwickelt ist. Der Test an sich bestand aus den folgenden Aufgaben:

1. Der Roboter weiß viel über Rezepte. Er kann zahlreiche Fragen darüber beantworten. Begrüßen Sie ihn. Danach fragen Sie ihn, was man mit bestimmten Zutaten kochen kann oder stellen andere Fragen, die Ihnen einfallen.
2. Stellen Sie sich vor, Sie wollen *spaghetti carbonara* / *ravioli* / *fish filet* / *pancakes* kochen und haben aber nicht das Rezept dazu. Was würden Sie den Roboter fragen?
3. Sie haben Hunger. In Ihrer Küche befinden sich *mushrooms*, *tomatoes* und *onions*. Leider wissen Sie nicht, was Sie damit kochen können, aber der Roboter vielleicht schon. Welche Fragen würden Sie ihm stellen?
4. Fragen Sie ihn nach folgenden Rezepten: *pork cheese spaghetti* / *island fish filets* / *potatoes gratin*. Anschließend versuchen Sie die Zutaten für dieses Rezept herauszufinden.
5. Fragen Sie ihn nach folgenden Rezepten: *pork cheese spaghetti* / *island fish filets* / *potatoes gratin*. Anschließend versuchen Sie die verschiedenen Kochschritte dieser Rezepte herauszufinden.
6. Fordern Sie ihn auf, neue Rezepte aus dem Internet herunterzuladen. Danach lassen Sie ihn das Herunterladen stoppen. Fragen Sie ihn nach der Anzahl der gefundenen Rezepte.

Die erste Aufgabe diente dazu, die Benutzer mit dem System vertraut zu machen und zu sehen, wie sie damit umgehen. Die folgenden Aufgaben sollten die Benutzer dazu bringen, einige der entwickelten Funktionalitäten des Systems in Anspruch zu nehmen. Es sollte eine Rezeptsuche gestartet und dessen Ergebnis beurteilt werden. Von Interesse waren hier die Erkennungsrate vom Spracherkennung, die Tatsache ob die Aufgabe gelöst werden konnte und wie lange, der

Benutzer dafür gebraucht hat. Es wurde auch untersucht, ob der Benutzer Interesse hatte, den Dialog weiterzuführen oder eher abubrechen.

Für die Bewertung, mussten die folgenden Fragen mit Begründung beantwortet werden:

1. Wie finden Sie generell die Idee, sich mit einem Roboter in natürlicher Sprache zu unterhalten?
2. Was hätten Sie noch vom Roboter erwartet?
3. Wie fanden Sie die Antworten des Roboters?
4. Kann der Roboter tatsächlich beim Kochen helfen?
5. Würden Sie ihn als Alternative zu einem Kochbuch benutzen?

## 5.2 Ergebnisse

Bis auf einen Benutzer, der nach mehreren hintereinander folgenden falschen Erkennungen immer nervöser wurde und deswegen die Hälfte der Aufgabe abgebrochen hat, wurde der Test komplett von allen durchgeführt. Die Benutzer haben ihre Fragen und zum Teil Dialogausschnitte wiederholt, um wahrscheinlich zu sehen, ob sich die Antworten vom System ändern oder nicht. Ab und zu haben manche Benutzer nach ein paar falsch erkannten bzw. verstandenen Sätzen, einen der zuvor richtig erkannten Sätze (ohne Rücksicht auf dem Kontext) einfach wiederholt. Sie wollten ständig eine korrekte Rückmeldung vor einer Fortsetzung des Dialogs. Nach Erhalt einer richtigen Antwort, haben manche sich sogar gefreut und gelacht (was vom System als Äußerung interpretiert wurde...).

Die Ergebnisse des Tests sind aus der folgenden Tabelle zu entnehmen.

	<b>B1</b>	<b>B2</b>	<b>B3</b>	<b>B4</b>	<b>B5</b>	<b>B6</b>	<b>Gesamt</b>
Richtig verstandene Aussagen	45,21%	38,03%	30,77%	36,59%	36,08%	20,00%	34,44%
Falsch verstandene Aussagen	54,79%	61,97%	69,23%	63,41%	63,92%	80,00%	65,56%
Erfüllung der Aufgaben	100,00%	100,00%	100,00%	100,00%	100,00%	50,00%	91,67%
Dialogdauer[turn]	73	71	91	82	158	105	96,67

**Abbildung 4: Ergebnisse des Benutzertests. Dabei stehen B1 bis B6 für Benutzer 1 bis 6.**

Im Allgemeinen finden es alle Benutzer ganz interessant, sich mit einem Roboter auf natürliche Weise zu unterhalten. Die Gründe dafür sind für die einen verschiedene Einsatzmöglichkeiten in der Zukunft. „Mit ihm zu sprechen erleichtert die Kommunikation und lässt die Hände frei. Man kann während der Unterhaltung und möglicher Informationssammlung auch andere Dinge erledigen. Möglicherweise fühlt man sich auch weniger alleine“, so ein Benutzer. Die anderen sind eher skeptisch, was die sprachliche Kommunikation mit einem Roboter angeht. Sie lassen sich gern erklären, was der Roboter machen kann, aber wenn sie vor ihm stehen, dann wollen sie schnell herausfinden, was er nicht kann. Sie erwarten, dass alles was sie sagen, verstanden wird, deshalb wiederholen sie eine Äußerung eher als ihre Frage umzuformulieren.

Die Benutzer hätten noch eine stärkere Interaktion mit dem Roboter erwartet. Außerdem beschwerten sie sich, dass sie oft nicht gleich verstanden wurden. Sie hätten noch erwartet, dass

der Roboter die Denkpausen während einer Äußerung als solche versteht und nicht als Satzende. Es wurde auch erwartet, dass Interjektionen, Wiederholungen von Wörtern und andere Besonderheiten der natürlichen Sprache dementsprechend behandelt werden. Die Antworten des Roboters fanden 83,33% der Benutzer zum Teil gut und zum Teil befriedigend. Der Roboter spreche zwar deutlich und ausführlich, aber man könnte ihn leider nicht unterbrechen. Die Übrigen 16% waren mit diesen Antworten nicht zufrieden.

Auf die Frage, ob der Roboter tatsächlich beim Kochen helfen kann, haben 16,67% der Benutzer mit „ja“ beantwortet. Sie sagen, dass die Informationen über die Zutaten und die Kochschritte eines Rezeptes sehr ausführlich sind. 66,67% der Benutzer antworten mit „vielleicht“. Für einige davon sollte er zuerst besser Verstehen lernen und langsamer sprechen. Die Unterhaltung sei für 16,67% der Benutzer nicht fließend genug. „Er braucht viel Zeit, um zu verstehen, was gesagt wurde. Man muss viel Geduld haben!“, sagte ein Benutzer. 50% der Benutzer sagen, dass sie ihn vielleicht als Alternative zu einem Kochbuch benutzen. Manche sagen, dass man sich von einem Buch inspirieren lassen kann. Er würde vielleicht viel Platz in der Küche wegnehmen. Für die anderen, erspart er nachblättern und Flecken auf dem Buch. 33% würden ihn als Alternative zu einem Kochbuch benutzen, „weil man Rezepte nach Zutaten suchen kann“. Die Übrigen 16% würden es nicht machen. Sie finden, dass es viel Zeit verlangt, sich vom Roboter verstehen zu lassen und es meistens nur nach vielen Wiederholungen klappt.

### 5.3 Diskussion

Die Erkennungsrate bzw. die Anzahl der richtig verstandenen Aussagen ist bei vielen Benutzern sehr niedrig gewesen, zum Teil wegen dem großen Vokabular, mit dem der Test ausgeführt wurde, wegen nicht genug verfeinerter Grammatikregeln aber auch wegen der schlechten Aussprache einiger Benutzer – insbesondere bei B6. Negativ trugen auch die langen Denkpausen sowie die Wiederholungen von Wörtern während dem Dialog bei. Meistens wurden die Rezeptnamen falsch erkannt.

Das Herunterladen von Rezepten lief an sich gut. Da fast alle gefundenen Rezepte schon in der Datenbank vorhanden waren, und da keine Duplikate zugelassen sind, wurde nichts in die Datenbank aufgenommen. Die Benutzerfragen über die Anzahl gefundener Rezepte wurden dann immer mit „*I can not find any recipe on the internet*“ („Ich konnte kein Rezept im Internet finden“) beantwortet, was einige Benutzer irritiert hat. Diese Antwort gibt leider keine Information über Rezepte, die gefunden und aber nicht gespeichert wurden, weil sie z.B. in der Datenbank schon vorhanden waren. Sie gibt nur die Anzahl der Rezepte, die beim Herunterladen wirklich in die Datenbank hinzugefügt wurden. Man könnte z.B. den Benutzer über die anderen Aktionen bzw. Aktivitäten, die währenddessen im System durchgeführt wurden, informieren.

Es gibt so viele Rezepte, dass dem Benutzer immer ein interessanter Vorschlag für fast jede Zutat gegeben werden kann. Alle Rezepte sind auch in verschiedenen Varianten vorhanden. Diese große Anzahl führt auch zu einem großen Vokabular, was die Erkennungsrate vom Spracherkennung leider negativ beeinflusst. Die Benutzer haben oft kurze Sätze formuliert, die zum Teil in der Grammatik nicht vorgesehen waren und zum Teil wegen einer schlechten Erkennungsrate nicht erkannt wurden. Für Fragen wie „*How can I cook spaghetti napoli?*“ („Wie kann ich Spaghetti Napoli kochen?“), ist der Name möglicherweise einer der 30800 Rezeptnamen – aus der Datenbanktabelle *recipes*. Wenn es aber kein Rezept mit diesem Namen gibt, dann sind die einzelnen Worte *spaghetti* und *napoli* Worte aus 1 oder 2 Rezeptnamen – entnommen von der Datenbanktabelle *recipenamewords*. Die Grammatikregel dafür ist so konstruiert, dass anstatt eines Rezeptnamens, 1 bis 3 Worte aus diesem Namen eingegeben werden können. Diese Worte wurden beim Herunterladen von Rezepten aus den Rezeptnamen extrahiert und in der Datenbanktabelle *recipenamewords* abgelegt (siehe 4.1.1.1). Diese Tabelle

enthält 8453 Worte. Mit allen diesen Möglichkeiten ist die Wahrscheinlichkeit, dass ein gesprochener Rezeptname tatsächlich richtig erkannt wird nicht groß genug. Viel besser war die Suche nach Zutaten, weil die Menge viel kleiner ist – insgesamt 455 (siehe 4.1.2.2). Der Benutzer konnte für diese höchstens 3 Zutaten pro Frage eingeben.

## 6 Zusammenfassung

Mit Hilfe zusätzlicher Wissensquellen aus dem Internet ist es möglich, die linguistische Wissensbasis des Dialogmanagers zu erweitern. Auf diese Weise kann dynamisch neues Vokabular hinzugefügt werden, ohne neue Grammatikregeln oder Ontologieeinträge implementieren zu müssen. Dieser Ansatz wurde am Beispiel einer Rezeptdatenbank aus dem Internet evaluiert. Benutzertests haben gezeigt, dass die Benutzer zwar damit recht zufrieden sind, dass aber die Erkennungsrate sich durchaus noch verbessern lässt, worauf ich im folgenden Kapitel noch eingehen werde.

### 6.1 Ausblick

Der Spracherkenner braucht für jedes Wort eine Aussprache, wofür die sog. letter-to-sound-Rules verwendet werden. Mit dem Download von Rezepten, werden auch neue Wörter gefunden und zu dem Vokabular hinzugefügt. Diese können nicht ohne weiteres erkannt werden. Zuerst muss eine Aussprache für sie vorliegen. Die Generierung von letter-to-sound-Rules müsste also in den kompletten Workflow noch integriert werden.

Für die Verbesserung der Erkennungsrate, könnte die Größe der Datenbank auf eine wesentlich kleinere und sinnvollere Anzahl von Rezepten reduziert werden. Man könnte auch die Rezepte in verschiedene Kategorien unterteilen und den Dialog so aufbauen, dass der Benutzer Rezepte nach Kategorien sucht und erst danach vielleicht nach Namen.

Zur Erleichterung der Interaktion mit dem Roboter könnten für die zeitaufwendige Operationen – sei es vom Spracherkenner oder von der Sprachverstehenskomponente – bestimmte Mimiken gezeigt oder Klänge abgespielt werden, so dass der Benutzer immer sofort eine Rückmeldung zu seinen Äußerungen bekommt.

## **Danksagung**

Mein Dank geht an meiner Betreuerin Petra Gieselmann für ihre Beratung und Unterstützung während dieser Arbeit. Ich möchte mich bei Hartwig Holzapfel bedanken für seine Bereitschaft und Hilfe bzgl. TAPAS. Ein ganz besonderer Dank geht an meiner Frau Syriane Chouambe für ihre Unterstützung und Verständnis während der arbeitsintensiven Tage. Für die Möglichkeit, diese Arbeit an seinem Institut durchführen zu können, bedanke ich mich bei Herrn Prof. Dr. Alex Waibel. Bei allen, die den Benutzertest ermöglicht und bei allen, die daran teilgenommen haben, möchte ich mich herzlich bedanken.



# Index

- Agent-basierte Systeme* 9
- Aktionen* 10, 23
- Aliases* 13
- Antwortgenerierung* 7
- Antwortgenerierungskomponente* 8
- Anwendungsbeschreibung* 10
- aufgezeichnete Nachricht* 8
- Aufruf von Methoden* 11
- automatisch* 15
- Benutzertest* 27
- Bewertung* 28
- bindings* 11
- blättern* 20
- Chunks* 15, 20
- Computersysteme* 7
- Core Grammatik* 14
- Datenbank* 7, 25
- Datenbankkonvertierungsregeln* 12
- Definition eines Goals* 11
- Definition eines Moves* 11
- Definition eines Objekts* 10
- Dialogmanager* 7
- Dialogsystem* 5, 7
- Dialogziel* 5, 10
- Dienste* 11
- Diskurskontext* 6
- dynamisch* 13
- dynamische Einbindung* 14
- dynamische Erkennungsgrammatik* 13
- dynamische Klasse* 13
- dynamischen Informationsquellen* 13
- Eigenschaften* 10
- endliche Automaten* 8
- Ergebnisse* 28
- Expertensystem* 7
- externe Applikation* 7
- externe Systeme* 7
- finite state-basiertes System* 8
- finite-state transducers* 13
- generierte Grammatik* 15
- Grammatikregeln* 12, 25
- grobe Kategorien* 8
- Gruppierung* 20
- Ibis* 5
- Interaktion von Agenten* 9
- Janus* 5
- Kochmethode* 20
- Komponenten* 7
- Kontrollstrategie* 8
- letter-to-sound* 13
- lokale Datenbank* 16
- Menüliste* 15
- Nachfrage* 11
- neues Rezept* 17
- Objekte* 10, 23
- Ontologie* 5, 10, 23
- parsen* 17
- Parser* 7
- Plug & Play Objekte* 14
- Protokoll internal* 11
- rahmenbasiertes System* 8
- Restaurantauskunftssystem* 13
- Restaurantmenüs* 15
- Rezeptparser* 17
- Rezeptsuche* 18
- Schnittstelle* 7
- semantische Repräsentation* 10
- Slots* 8
- Sprachausgabe* 8
- Sprachausgabekomponente* 8
- Sprachdialogsystem* 7, 13
- Spracherkennung* 5, 7
- Sprachverständniskomponente* 7
- statische Grammatik* 15
- Subdialog* 11
- Suchergebnis* 20
- Tabelle ingredients* 17
- Tabelle recipenamewords* 17
- Tabelle recipes* 17
- TAPAS* 5, 10
- Testen* 26
- Test-GUI* 26
- Trainingsphase* 7
- Turn-by-Turn* 7
- Zustände* 10

## Literaturverzeichnis

- G. Chung, S. Seneff, C. Wang, and L. Hetherington, 2004.  
„A Dynamic Vocabulary Spoken Dialogue Interface.“ *Proc. Interspeech*, 327-330, Jeju, South Korea
- M. Denecke., 2002.  
„Rapid prototyping for spoken dialogue systems.“ *Proceedings of the 19th International Conference on Computational Linguistics*, Taiwan.
- P. Gieselman und Alex Waibel, 2005.  
„What makes Human-Robot Dialogues struggle?“ *Proceedings of the Ninth Workshop on the Semantics and Pragmatics of Dialogue (DIALOR)*, Nancy.
- A. Gruenstein, 2002.  
„Automatic Grammar Construction.“
- H. Holzapfel, 2005.  
„Towards Development of Multilingual Spoken Dialogue Systems.“ *Proceedings of the 2nd Language and Technology Conference*.
- M. F. McTear, 2002.  
„Spoken Dialogue Technology: Enabling the Conversational User Interface.“
- M. Rayner, I. Lewin, G. Gorrell und J. Boye, 2001.  
„Plug and Play Speech Understanding.“ *Proceedings of 2nd SIGdial Workshop on Discourse and Dialogue*, Aalborg, Denmark.
- M. Finke, P. Geutner, H. Hild, T. Kemp, K. Ries and M. Westphal, 1997.  
„The Karlsruhe-Verbmobil Speech Recognition Engine.“ *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, (ICASSP-97)*.
- Soltau, H., F. Metze, C. Fuegen, and A. Waibel, 2001.  
„A one pass- decoder based on polymorphic linguistic context assignment.“ In *Proceedings of the Automatic Speech Recognition and Understanding Workshop, ASRU-2001*. Madonna di Campiglio, Trento, Italy.