

Universität des Saarlandes
Fachrichtung 4.7 – Computerlinguistik
Softwareprojekt: Sprechende Roboter mit LEGO Mindstorms
Leitung: Alexander Koller und Geert-Jan Kruijff
Sommersemester 2004

MiCo: Der Barkeeper, der MiniCocktails mixen kann



Sabrina Wilske
wilske@coli.uni-sb.de

1. EINLEITUNG	3
1.1 FUNKTIONALITÄT	3
1.2 AUFBAU	3
1.3 SYSTEMARCHITEKTUR UND TECHNISCHE VORRAUSSETZUNGEN	4
2. DIALOG	5
2.1 EINFÜHRUNG IN DIALOGSYSTEME	5
2.2 DIAMANT	6
2.3 DIALOG-STRUKTUR	7
3. DATENBANK	7
3.1 NEUE COCKTAILS	13
4. SCHLUSS	15
QUELLEN	16

1. Einleitung

Diese Arbeit dokumentiert ein Teilergebnis des Softwareprojekts "Sprechende Roboter mit Lego Mindstorms", das im Sommersemester 2004 an der Universität des Saarlandes als Lehrveranstaltung von Alexander Koller und Geert-Jan Kruijff angeboten wurde.

Der Kurs hatte zum Ziel, mit Hilfe eines Lego Mindstorms¹ Baukastens und diverser Software einen Roboter zu bauen, mit dem man mittels natürlicher Sprache kommunizieren kann. Der erste Teil des Kurses diente der Einführung in die wesentlichen Bereiche des Roboterbaus: Lego-Konstruktion, Programmierung und Dialogsystem. Damit wurden die Grundlagen für den zweiten Teil des Kurses gelegt, der darin bestand, in Teams von je drei Studierenden einen eigenen Roboter von Grund auf zu entwerfen und zu realisieren. Unser Team² wollte einen Roboter kreieren, der die Funktionalität eines Barkeepers erfüllen kann. Damit wollten wir vor allem zwei Ziele erreichen:

- (1) Der Roboter sollte einen praktischen Nutzen haben.
- (2) Der Roboter sollte Spaß machen.

Das größte Risiko bei dieser Unternehmung war die unmittelbare Nähe zu Flüssigkeiten, die den elektrischen Bauteilen des Roboters bei ungeschütztem Kontakt gefährlich werden konnten. Eine trockene Lösung, die Flüssigkeiten mittels kleiner Bausteine nur modellieren sollte, lehnten wir wegen Ziel (1) ab. Wie sich herausstellte, befand sich unser Roboter während der Entwicklungsphase und in Ausübung seiner Tätigkeit nie in ernsthafter Gefahr, einen flüssigkeitsverursachten Schaden zu erleiden. (Was nicht heißen soll, dass nie etwas verschüttet wurde.) Ein viel größeres Problem war vielmehr die zuverlässige Abfüllung der Flüssigkeiten. Die grundlegende und von uns eher unerwartete Schwierigkeit bei der Realisierung des Barkeeper-Roboters lag darin, die Zutaten für die Drinks in angemessenen und genau abgemessenen Mengen abzapfen.

Der Name MiCo ist ein Akronym für Mini-Cocktail und deutet auf die geringe Größe der vom Roboter gemixten Cocktails hin. Diese Einschränkung war hauptsächlich Folge des Abfüllproblems.

Im verbleibenden Teil der Einführung werde ich kurz die Funktionalität unseres Roboters darstellen, dann den konstruktiven Aufbau skizzieren und am Schluss die gesamte Architektur und die technischen Voraussetzungen, die beim Design unseres sprachverstehenden Roboters zum Tragen kamen, erläutern.

1.1 Funktionalität

Ähnlich wie ein Barkeeper in einer richtigen Bar begrüßt MiCo potentielle Kunden und fragt sie nach ihren Wünschen. Die Kundin kann dann entweder ein Getränk aus der Karte wählen oder einen Sonderwunsch äußern. Bei der ersten Option wird MiCo dem Rezept folgend die Zutaten in sein Mischgefäß füllen, das gefüllte Gefäß wahlweise schütteln und dann in ein vorher exakt positioniertes Glas einschenken. Hat der Bargast einen Wunsch, der nicht auf der Karte steht, kann er die Zutaten mit Mengenangabe einzeln abfüllen lassen. Auch hier besteht die Option, die Zutaten durch kräftiges Schütteln mischen zu lassen. Während MiCo die einzelnen Getränke abzapft, unterhält er die Kundin durch mehr oder weniger amüsante Bemerkungen. Sowohl die Rezepte für die Cocktails als auch die Sprüche sind in einer Datenbank gespeichert, auf die MiCo dynamisch zugreifen kann. In der Datenbank ist auch das Wissen über verfügbare Zutaten modelliert, desweiteren zusätzliche Informationen wie Preis und Alkoholgehalt der Cocktails bzw. Zutaten. Aus dem Aufbau ergibt sich eine Beschränkung auf sechs mögliche Grundingredenzien. Bei unseren Demonstrationen waren das Wasser, Zitronensaft, Orangensaft, Ramazzotti, Wodka und Rum. Diese sind grundsätzlich variabel, allerdings muss sich die tatsächliche Befüllung in der Datenbank widerspiegeln. Auch die Spracherkennung muss mit in der Lage sein, die Wörter für die zutreffenden Zutaten zu erkennen.

1.2 Aufbau

MiCo ist ähnlich wie ein richtiger Barkeeper an seine Bar gefesselt, das heißt, er kann sich nicht frei im Raum bewegen. Er steht in der Mitte eines Kreises, der zu zwei Dritteln durch sechs verschiedene Abfüllstationen beschrieben wird. In dem offenen Kreisabschnitt steht das Glas, in das der fertig gemixte

¹ www.legomindstorms.com/

² Thomas Horf, Roland Roller, Sabrina Wilske

Drink gegossen wird. MiCo verfügt über drei verschiedene Freiheitsgrade. Er kann sich einerseits auf horizontaler Ebene kreisförmig bewegen. Der Mittelpunkt dieser Kreisbewegung ist gleichzeitig der Mittelpunkt der kreisförmigen Abfüllkonstruktion, das ermöglicht das Ansteuern der einzelnen Zapfpunkte. Wenn ein Punkt angesteuert wurde, kann MiCo durch eine Aufwärtsbewegung seines oben angebrachten Drückmechanismus den Hebel der zweckentfremdeten Silikonpistole betätigen, welche dann ihrerseits Druck auf die (ebenfalls zweckentfremdete) Wund- und Blasenspritze ausübt. Der Druck auf die Spritze gibt etwas der eingefüllten Flüssigkeit frei, welche in das Mischgefäß läuft. Das Mischgefäß ist am vorstehenden Rumpf von MiCo befestigt und hier befindet sich auch der dritte Bewegungspunkt, der ein kontrolliertes Schütteln der abgefüllten Flüssigkeiten erlaubt. Ein Überschwappen wird einerseits durch die sich oben verengende Form des Gefäßes³ verhindert, andererseits durch die Anpassung des Schüttelimpulses an den geschätzten Füllstand. Die Menge der abzufüllenden Flüssigkeit kann durch die Anzahl der Druckimpulse kontrolliert werden. Pro Druckimpuls werden circa ein bis zwei Milliliter freigegeben.⁴ Abbildung 1 zeigt das Aufbau-Schema.

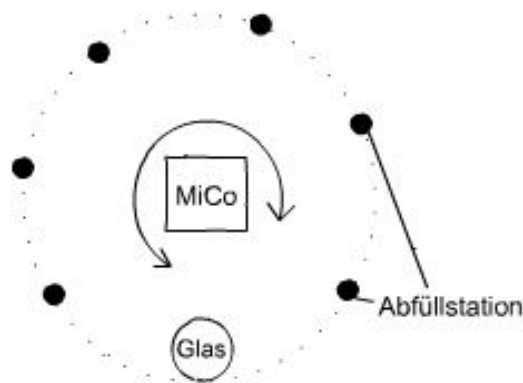


Abbildung 1

1.3 Systemarchitektur und technische Voraussetzungen

In diesem Abschnitt soll die grundlegende Systemarchitektur erklärt werden, wie sie allen im Kurs gebauten Robotern gemein war. Jedes Team bekam als Hardware den RIS (Robotics Invention System) Baukasten (mehr als 700 Teile) und als Ergänzung den UBS-Kasten (Ultimate Builders Set, 316 Teile). Das Herz und Gehirn eines mit Lego Mindstorms gebauten Roboters ist der programmierbare Baustein namens RCX (Robotic Command eXplorer). Dieser ist mit der Außenwelt durch drei Ausgänge für Motoren, drei Eingänge für Sensoren und eine Infrarotschnittstelle verbunden. Über einen Infrarot-Turm, der an einen PC angeschlossen wird, können Daten zwischen PC und RCX ausgetauscht werden. In erster Linie dient dieser Datentransfer dem Laden des RCX mit dem Programm, das er ausführen soll. Für unsere Zwecke war aber auch die Kommunikation des RCX mit dem PC während des Programmablaufs notwendig. Eine dritte, von uns nicht in Anspruch genommene Kommunikationsart ist die zwischen zwei RCX-Steinen. Ein weiterer nützlicher Bestandteil des RCX ist die Flüssigkristallanzeige (LCD), mittels derer sich der interne Zustand des RCX anzeigen lässt. So ist es möglich und für das Fehlerfinden in einem Programm sehr nützlich, Sensorwerte oder Variablen auszugeben. Der RCX ist das Gehirn, aber ohne die Ausgänge für Aktoren, mit denen der Roboter die Welt beeinflussen kann und Eingänge für Sensoren, mit denen er die Welt

³ Landliebe Joghurt

⁴ Das ist zugegebenermaßen nicht sehr genau. Die Ursache für die Ungenauigkeit liegt darin, dass die sechs improvisierten Konstruktionen aus Silikonpistole und Spritze nicht identisch auf denselben Druck reagieren.

wahrnehmen kann, gleiche der Roboter einem gelähmten Menschen ohne Sinnesorgane. Als Aktoren sind bei Lego nur Motoren möglich, an Sensoren gibt es eine reiche Auswahl. Die gängigsten und für unsere Zwecke genügenden waren Lichtsensoren, Tastsensoren und Rotationssensoren. Aber auch mit Gehirn, Sinnesorganen und Muskeln käme ein Mensch nicht weit, fehlten ihm doch die Knochen und Gelenke, die alles zusammenhalten. Hier bietet Lego neben den bekannten Grundbausteinen, Zahnräder, Achsen, Räder und eine Vielzahl anderer Komponenten.

Auf der Softwareseite gibt es drei wesentliche Bausteine.

Zum einen das Programm, welches auf den RCX geladen wird. Es handelt sich dabei um eingeschränkten Java-Code, der mit Hilfe der Programmierumgebung leJOS⁵ kompiliert wird. Aufgrund des beschränkten Speicherplatzes auf dem RCX-Stein ist die Menge der verwendbaren Java-Syntax bzw. vordefinierten Java-Klassen eingeschränkt.

Um die Kommunikation mit natürlicher Sprache zu ermöglichen, ist ein Programm für sprecherinnenunabhängige Spracherkennung und Sprachsynthese nötig. Dabei haben wir das kommerzielle Software-Produkt von Lernout&Hauspie benutzt, das auf der Java Speech API basiert.⁶

Zentrum der auf dem PC ablaufenden Prozesse ist das Dialogsystem Diamant der Firma CLT.⁷ Hier läuft der vorher spezifizierte Dialog ab. Der RCX und die Spracherkennungs- und Synthesesoftware sind hier über Klienten eingebunden. Auch die Datenbank ist über einen Klienten in das Dialogsystem integriert. Beim Start des Dialogs werden die Verbindungen zu den Klienten aufgebaut, die optimalerweise vorher gestartet wurden. Gemäß den Aktionen und Bedingungen, die in Diamant definiert wurden, läuft der Dialog zwischen dem involvierten Menschen und dem System ab. Das grundlegende Schema lässt sich wie dabei folgt zusammenfassen: Der Roboter initiiert die Kommunikation mit einer Frage und führt entsprechend der Antwort Aktionen aus. Während oder nach der Ausführung der Aktionen können weitere Anweisungen vom Menschen erfragt werden. Der Dialog endet, wenn ein Endzustand erreicht wurde. Neben den Parametern, die vom menschlichen Kommunikationspartner gesetzt werden, gibt es Parameter, die von den angeschlossenen Sensoren des Roboters kommen. Diese werden normalerweise direkt in dem Programm behandelt, das auf den RCX geladen wurde; sie könnten aber auch über den RCX in den Dialog hochgereicht und dort berücksichtigt werden. Die dritte Parameterquelle bei MiCo sind die Inhalte der Datenbank. Neben der einfachen Abfrage von Inhalten ist aber auch ihre Veränderung möglich.

Das folgende Kapitel beschäftigt sich mit dem Dialog-Aspekt des Projektes. Ich werde einen Überblick über mögliche und existierende Dialogsysteme geben, die Funktionsweise des von uns verwendeten Systems erklären und die grobe Struktur des Barkeeper-Dialogs umreißen. Im dritten Kapitel werde ich die Inhalte der Datenbank vorstellen und exemplarisch darstellen, wie sie eingebunden wurde.

2. Dialog

2.1 Einführung in Dialogsysteme

Das Spektrum existierender Dialogsysteme kann in drei Hauptkategorien eingeteilt werden, dabei orientieren sich die Kriterien für die Einteilung an den Möglichkeiten, wie der Dialog mit dem Benutzer kontrolliert wird. [1]

- (1) Systeme, die auf einem Graphen oder endlichen Automaten basieren
- (2) Frame-basierte Systeme
- (3) Agenten-basierte Systeme

Die Ausführung von zwei der wichtigsten Aufgaben eines Dialogsystems hängen wesentlich von der Kontrollstrategie ab:

- a) die Verarbeitung der Benutzereingaben
- b) die Fehlerbehandlung

⁵ <http://lejos.sourceforge.net>

⁶ Da dieses nur auf der Windows-Plattform läuft, waren wir an dieses Betriebssystem gefesselt. Mit der Verfügbarkeit von leistungsfähigen Spracherkennungs- und Syntheseprogrammen Sphinx und FreeTTS, die auf Java basieren, könnte man in Zukunft aber auch auf Linux/Unix-Plattformen arbeiten.

⁷ www.clt-st.de

Graph- bzw. **automatenbasierte** Systeme sind definiert durch eine Anzahl von Knoten (Zustände) und Kanten (Zustandsübergänge). An einem Zustand kann eine Eingabe angefordert werden (eine Frage gestellt werden) oder eine Eingabe erkannt werden. Entsprechend der erkannten Eingabe werden Aktionen ausgeführt und in einen nächsten Zustand übergegangen. Die möglichen Äußerungen des Benutzers sind relativ eingeschränkt, sie dürfen sich meist nur auf die aktuelle Eingabeaufforderung beziehen und nur aus bestimmten Wörtern oder Phrasen bestehen.

Der Vorteil dieser Beschränkung von Vokabular und Grammatik für jeden Zustand besteht darin, dass die Spracherkennung und das Verstehen sehr einfach sind. Ein Dialog kann relativ schnell und unkompliziert modelliert werden. Für kleine, wohlstrukturierte Domänen ist das ausreichend. [2]

Der Nachteil liegt in der mangelnden Flexibilität des Systems: Die Korrigierung von falsch verstandenen Äußerungen ist für die Benutzerin relativ schwierig, außerdem hat sie keine Möglichkeit, selbst die Initiative zu ergreifen, Fragen zu stellen oder das Thema zu wechseln. Komplexere Domänen abzubilden, kann zu Problemen führen.

Bei **frame-basierten** Systemen ist der Ablauf des Dialogs nicht so starr vorgegeben, hier hängt er davon ab, was das Dialogsystem an Informationen braucht und was der menschliche Partner schon eingegeben hat. Die Lücken im Rahmen können in freier Reihenfolge gefüllt werden, die Benutzerin kann Antworten auf noch ungestellte Fragen geben, die das System dann integrieren kann.

Ganz am Ende der Komplexitätsskala stehen **agenten-basierte** Systeme, die eine komplexe Kommunikation zwischen dem System, dem Benutzer und einer eingebetteten Anwendung erlauben. Hierbei wird der Dialog als Kommunikation zwischen Agenten betrachtet, die Schlüsse über ihre eigenen (und eventuell anderer) Aktionen und Glaubensinhalte ziehen können. Dabei kann der vorausgehende Kontext mit einbezogen werden und das System kann mit Hilfe seiner Erwartungen Benutzereingaben voraussagen und interpretieren. Hier ist es auch möglich, dass die Benutzerin die Initiative ergreift und den Dialog kontrolliert. Das erfordert allerdings eine weit anspruchsvollere Spracherkennungs- und Sprachverstehenskomponente.

2.2 Diamant

Die von uns benutzte Dialogsoftware Diamant modelliert den Dialog als Graph, ist also der ersten Kategorie zuzuordnen. Die Spezifizierung eines Dialogs besteht aus drei Komponenten: 1) dem Graphen, 2) externen Geräten, die im Dialog involviert sind, 3) einer Liste von Variablen, die im Dialog benutzt werden [2]. Beim MiCo-Dialog gibt es drei verschiedene Geräte, die benutzt werden: der RCX-Client, Speech-Client (Spracherkennung und -synthese) und der Datenbank-Klient. Diese werden bei Start des Systems über ihre Ports an Diamant gebunden. Der Graph besteht aus einer (mit zunehmender Reife unseres Barkeeper-Roboters wachsenden) Anzahl von Knoten und verbindenden Kanten. Es gibt verschiedene Arten von Knoten für verschiedene Funktionen: Eingabe, Ausgabe, Bedingung, Variabeltest, Script, Anfang und Ende.⁸ Die beiden letzten eröffnen bzw. beenden den Dialog. Im Ausgabeknoten können Kommandos an die angeschlossenen Geräte gesendet werden. In Eingabeknoten können zusätzlich die von den Geräten erwarteten Eingaben bestimmt werden. Das sind vor allem Eingaben, die von der Spracherkennung kommen, aber auch Ergebnisse der Datenbank-Anfrage. Die Spracherkennung basiert auf der Java Speech API, sie braucht eine im Java Speech Grammar Format⁹ abgefasste Grammatik, gemäß derer sie Äußerungen erkennt. In dieser Grammatik-Datei können Tags definiert werden, die mit geschweiften Klammern an die Regeln der Grammatik gehängt werden und bei erkannter Äußerung gesendet werden. Diese Tags kommen dann über den Eingabeknoten bei Diamant an. Für jedes deklarierte Eingabemuster und für den Fall, dass innerhalb eines Zeitraumes keine Eingabe erfolgt ist, gibt es eine ausgehende Kante. Der Bedingungsknoten hat zwei ausgehende Kanten, je nachdem ob die Bedingung erfüllt ist oder nicht. Im Variabeltestknoten können die Werte einer Variablen getestet werden, (>, <, =), für jede Bedingung gibt es eine ausgehende Kante. Script-Knoten erlauben die Manipulation von Variablen mittels einer Reihe eingebauter Funktionen in Java-Syntax. Von diesen Knoten haben wir beim MiCo-Dialog ausgiebig Gebrauch gemacht, im folgenden Kapitel werde ich einige beispielhaft näher erklären. Mit Hilfe der Variablen werden Werte zwischen den angeschlossenen Geräten transportiert und bei Bedarf in Diamant verändert.

⁸ Es gibt noch mehr Typen, auf die hier aber nicht näher eingegangen wird, weil wir sie nicht benutzt haben.

⁹ <http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/>

2.3 Dialog-Struktur

Es soll hier kurz ein Überblick über die grobe Struktur des Dialogs gegeben werden, Einzelheiten werden in der Arbeit meines Teamkollegen Roland Roller näher besprochen. Ich werde im 3. Kapitel auf die Dialogaspekte eingehen, die sich auf die Interaktion mit der Datenbank beziehen.

Die Interaktion mit MiCo lässt sich in folgende Abschnitte unterteilen:

- (1) Anfang (Begrüßung, Aufnahme der Bestellung)
- (2) Mixen eines vorgegebenen Cocktails
- (3) Aufnahme und (4) Mixen eines Sonderwunsches
- (4) Servieren des Drinks
- (5) Verabschiedung

In (1) nimmt MiCo das Gespräch mit einer Begrüßung auf und fragt, ob er einen Cocktail anbieten kann. Jetzt hat die Kundin die Möglichkeit, ja oder nein zu sagen, oder direkt eine Bestellung aufzugeben. Wird ein „ja“ oder kein gültiger Cocktailname (das heißt, einer von der Grammatikdatei spezifizierter) erkannt, spricht MiCo eine Empfehlung aus und fragt erneut nach einem Wunsch. Bei einem „nein“ geht der Dialog direkt in den Verabschiedungsabschnitt, bei einem erkannten Cocktail geht es zu (2).

Kann die Bestellung der Kundin nicht erkannt werden, wird die Möglichkeit angeboten, ein Rezept genau nach Anweisung auszuführen. Dabei ist man beschränkt auf die zur Verfügung stehenden Einzelzutaten. Nach Wahl kann der Name der Eigenkreation registriert werden, um ihn später in der Datenbank abzuspeichern (3). Dann wird Schritt für Schritt die gewünschte Menge jeweils einer Zutat abgefüllt (4). Ist das Rezept fertig abgearbeitet, fragt MiCo, ob der Cocktail geschüttelt werden soll. Bei positiver Antwort und bei den Standardrezepten wird der Abfüllbehälter geschüttelt und das Getränk in das bereitgestellte Glas gegossen. MiCo verlangt den Preis - bei Standardrezepten gemäß Karte, bei Eigenkreationen errechnet aus Mengen und Preisen für die Einzelzutaten. Nach Wunsch erteilt er auch Auskunft über den Alkoholgehalt der Bestellung (5). MiCo verabschiedet sich, fragt nach der nächsten Bestellung und begrüßt gegebenenfalls den nächsten Gast. Meldet sich niemand, wird der Dialog beendet.

3. Datenbank

Die Einbindung einer Datenbank zur Darstellung von Rezepten bietet gegenüber einer festen Einkodierung in das Barkeeper-Programm eine höhere Flexibilität. Die Datenbank wurde mit Hilfe von MySQL-Front¹⁰ erstellt, einem Programm, das es auf einfache Art und Weise ermöglicht, Datenbanken im frei verfügbaren MySQL-Format anzulegen und zu verwalten. Abbildung 2 zeigt die Bedienoberfläche dieses Programms.

¹⁰ www.mysqlfront.de/

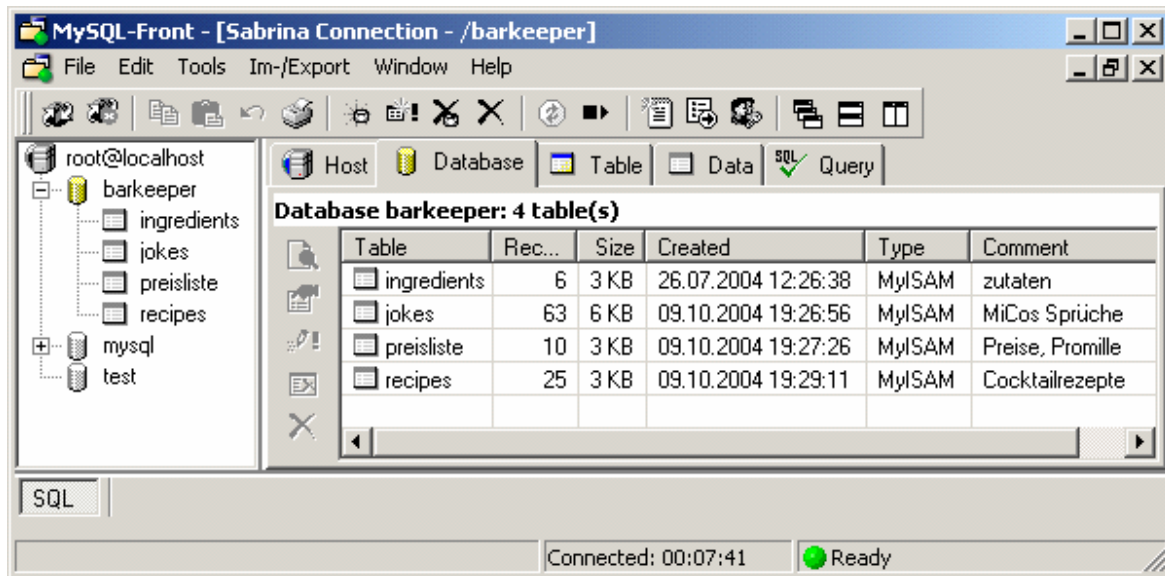


Abbildung 2

Die Barkeeper-Datenbank besteht aus insgesamt vier Tabellen: `ingredients`, `jokes`, `preisliste` und `recipes`.

In der `ingredients`-Tabelle werden die verfügbaren Einzelzutaten, also die Befüllungen der sechs Spritzen gespeichert. Zu jeder Zutat gibt es eine Position (`pos`), die die Spritze identifiziert. Diese Angabe braucht MiCo um die jeweilige Station anzusteuern. Außerdem werden der Preis (`preis`) und der Alkoholgehalt in Promille (`alcohol`) gespeichert. Diese Angaben werden gebraucht, um bei Bedarf später den Alkoholgehalt und Preis eines selbst kreierten Cocktails berechnen zu können. Tabelle 1 zeigt den Inhalt der Tabelle für unsere erste Präsentation:

id	name	alcohol	pos	preis
0	zitronensaft	0	2	0.3
3	ramazzotti	30	3	1.5
4	vodka	50	5	1
5	rum	40	6	2
6	orangensaft	1	4	0.4
7	wasser	0	1	0.1

Tabelle 1

In der Tabelle `preisliste` (Tab. 2) sind die Namen aller verfügbaren Cocktails vermerkt (`ctname`), dazu ihr Preis und ihr Alkoholgehalt. Eine weitere wichtige Information ist die Lautform. Diese ist notwendig, weil wir eine deutsche Spracherkennung und Synthese benutzen, viele Cocktails aber Namen aus dem englischen Sprachraum haben. Damit die Namen richtig ausgesprochen werden, muss dem Syntheseprogramm eine den deutschen Graphem-Phonem-Regeln entsprechende Form übergeben werden.

ctname	preis	alcohol	lautform
rum fizz	4	6	rumfiss
rum sour	4	7	rumssaua
screwdriver	4	5	skrudreiwier
brittany	5	5	brittannie
joe collins	6.6	4	dscho kollins
eskimoflip	1.5	0	eskimoflip
ozi	3.2	0	ozi
labamba	4.2	4	labamba
waikiki	5.1	5	waikiki
rammora	4	6	rammora

Tabelle 2

Die Tabelle `recipes` (Tab.3) ist das Herzstück der Datenbank. Hier sind alle Rezepte vermerkt. Dabei entspricht ein Tupel der Relation einem Abfüllvorgang. Das Tripel `<rum fizz, rum, 4>` besagt, dass für den Cocktail Rum Fizz vier Einheiten Rum gezapft werden müssen. Für jede andere Zutat von Rum Fizz gibt es ein anderes Tupel der Form `<rum fizz, ingredient, amount>`. Um das vollständige Rezept für einen bestimmten Cocktail zu extrahieren, wird folgende Datenbankabfrage gestellt:

```
"SELECT ingredient, amount, pos FROM recipes, ingredients WHERE ctname=Cocktail
AND recipes.ingredient=ingredients.name ORDER BY pos DESC"
```

Dabei referenziert die Variable `Cocktail` einen String, der einem der verfügbaren Cocktailnamen entsprechen sollte. Dass es nur solche Abfragen gibt, für die es auch Inhalte in der Datenbank gibt, ist durch den manuellen Abgleich mit der Grammatikdatei sichergestellt. Es sollten nur solche Strings als Namen für Cocktails erkannt und an die entsprechende Variable gebunden werden, die auch in der Datenbank existieren. Diese Invariante zu garantieren, ist bei diesem relativ kleinen System noch recht einfach. Allerdings wäre es bei größeren und komplexeren Anwendungen vielleicht sinnvoll, einen Mechanismus zu implementieren, der die Synchronisierung zwischen Erkennungsgematik und Datenbankinhalten verwaltet. Wenn ein Cocktail angefragt wird, für den es keine Rezepte in der Datenbank gibt, ist die Rückgabe an den Klienten eine leere Liste. Das würde bei MiCo dazu führen, dass er gar nichts abfüllen würde, sondern direkt in den Zustand nach Abarbeitung eines Rezeptes überginge. Eine sinnvolle Erweiterung wäre hier, dass angemessen auf diesen Ausnahmefall reagiert wird, anstatt so zu tun, als wäre nichts gewesen und dem Kunden das Nichts ins Glas zu gießen.

Noch einmal zurück zur oben zitierten Datenbankabfrage: Neben der `recipes`-Tabelle, in der die Mengen der einzelnen Zutaten für den Cocktail abgespeichert sind, wird hier gleichzeitig auch die `ingredients`-Tabelle benutzt, um die Positionen der Zutaten zu erfahren. Die Kopplung der beiden Tabellen erfolgt über die Gleichung `recipes.ingredient=ingredients.name`. Damit MiCo möglichst effizient alle nötigen Flaschen anfährt, werden die Rückgabewerte nach der Positionsangabe sortiert (`ORDER BY pos DESC`). Damit wird verhindert, dass der Roboter unnötige Wege zwischen den Abfüllstationen zurücklegt.

id	ctname	ingredient	amount
23	brittany	ramazotti	3
24	brittany	orangensaft	4
25	brittany	zitronensaft	2
11	eskimoflip	wasser	8
12	eskimoflip	zitronensaft	2
9	joe collins	vodka	4
10	joe collins	zitronensaft	2
16	labamba	rum	3
17	labamba	zitronensaft	4
18	labamba	orangensaft	7
13	ozi	zitronensaft	5
15	ozi	orangensaft	6
7	rammora	ramazzotti	4
8	rammora	orangensaft	8
1	rum fizz	rum	4
2	rum fizz	zitronensaft	3
14	rum fizz	wasser	3
3	rum sour	rum	5
4	rum sour	zitronensaft	3
5	screwdriver	vodka	3
6	screwdriver	orangensaft	8
19	waikiki	zitronensaft	3
20	waikiki	rum	3
21	waikiki	orangensaft	3
22	waikiki	wasser	4

Tabelle 3

Die letzte Tabelle `jokes` enthält eine Anzahl von Sprüchen, die MiCo während der Bedienung eines Kunden äußert. Sie sollen den Roboter als Gesprächspartner interessant und unterhaltsam machen. Jeder Spruch ist entweder einer bestimmten Zutat oder einem Cocktail zugeordnet. Während MiCo einen Cocktail zubereitet und dabei die nötigen Zutaten abfüllt, macht er jeweils einen entsprechenden Witz. Um eine gewisse Varianz zu sichern bzw. den Schein von Indeterminanz zu wahren, wird aus der Menge der Sprüche per Zufall ausgesucht. Ebenfalls per Zufall wird einer von sieben Begrüßungssprüchen ausgesucht, auch hier soll Monotonie vermieden werden. Im Prinzip wäre jeder Teil des Dialogs variabler zu gestalten durch die Angabe einer Menge von möglichen funktionsgleichen Äußerungen, aus denen dann eine blind gezogen wird. Der Nachteil erwächst aus der benötigten Zeit für die Datenbankabfrage und die Zufallsauswahl. Tabelle 4 bietet einen Auszug aus den verwendeten Sprüchen.

id	joke	thema
19	hallo	hallo
20	Guten Abend	hallo
21	grüss gott	hallo
22	hei	hallo
23	Guten Tag	hallo
24	halli hallo	hallo
63	meun meun	hallo
3	Hoffentlich können sie mit dem ganzen Koffein heute nacht noch schlafen!	cola
12	der ist frisch gepresst.	orangensaft
14	mit grüßen von Onkel Dittmeier.	orangensaft
35	aaah, unser Vitamin O.	orangensaft
4	unser guter kubanischer Rum. Waren sie schon mal auf Kuba?	rum
5	ach, der Rachenputzer.	rum
16	der hat ganz schön viele Umdrehungen. hoffentlich vertragen sie den.	screwdriver
49	aah. jemand, der etwas von Cocktails versteht.	screwdriver
26	das ist ein echer Vitamin C schocker	ozi

Tabelle 4

Nachdem ich die Datensätze vorgestellt habe, auf die MiCo während einer typischen Interaktion zugreifen muss, möchte ich auf einige weitere Abfragen eingehen, die dabei vom Klienten gestellt werden. Um Sprüche zu einem bestimmten Thema auszuwählen, wird im Dialogsystem folgende Anfrage generiert:

```
"SELECT joke FROM jokes "WHERE thema=Thema"
```

Thema ist eine Variable, die den Wert einer Cocktailbezeichnung, einer Zutatenbezeichnung oder 'hallo' annehmen kann, wobei letzteres für das Thema Begrüßung steht. Als Rückgabe erhält der Klient eine Liste von Records, die im Dialogsystem an die Variable `jokesList` gebunden werden. Ein einfaches Record hat normalerweise die Form:

```
{ joke = "halli hallo" }
```

Eine Rückgabeliste wäre dann zum Beispiel:

```
[
  { joke = "halli hallo" },
  { joke = "Guten Tag" },
  { joke = "meun meun" },
]
```

Um von der Menge der passenden Sprüche einen auszusuchen, wird ein Script-Knoten eingefügt. Das Script erfragt die Länge der Antwortliste, benutzt die Zufallsfunktion `random`, um den Index `i` eines Elements auszuwählen, greift mit der `get`-Funktion auf das `i`-te Element zu, und extrahiert mit einer weiteren `get`-Funktion den String, der den Spruch referenziert. Dieser wird dann an die Variable `joke` gebunden. Für den (Un-)Fall, dass die Antwortliste leer ist, wird `joke` an einen Default-Wert gebunden.

```
if (!empty(jokesList))
{
  int l = length(jokesList);
  int z = random(0,l-1);
  struct rs = get(jokesList,z);
  joke = get(rs,"joke");
}
else joke = "hallo und guten Tag";
```

Alle benutzten Funktionen und syntaktischen Konstrukte sind in Diamant frei verfügbar und orientieren sich an Java-Syntax.

Für die Bearbeitung von Rezepten werden es etwas mehr Angaben gebraucht.
Zur Erinnerung hier noch einmal die Abfrage:

```
"SELECT ingredient, amount, pos FROM recipes, ingredients WHERE ctname=Cocktail  
AND recipes.ingredient=ingredients.name ORDER BY pos DESC"
```

Auch hier wird eine Liste von Records zurückgeliefert, nur haben die Records mehr Felder:

```
{ingredient = "wasser", amount=3, pos=1}
```

Die Liste wird an die Variable `rezept` gebunden und wiederum in einem Script verarbeitet. Hier müssen nun gleich drei Werte registriert werden, sie werden jeweils an Variablen gebunden, die dann für den Aufruf von Funktionen des RCX benutzt werden. Dabei wird schrittweise vorgegangen, die einzelnen Zutaten werden in einer Schleife abgearbeitet, dabei verkürzt das Script jeweils die Liste um das aktuelle Element. Die Schleife ist nicht im Script implementiert, sondern als eine Folge von Knoten. Ist die Liste leer, wird die Knotenschleife verlassen und in einen anderen Abschnitt des Dialogs übergegangen.

Hier der Code für das Script:

```
if (!empty(rezept))  
{  
    struct rs = head(rezept);  
    posi = str(get(rs, "pos"));  
    mass = str(get(rs, "amount"));  
    liquid = get(rs, "ingredient");  
    rezept = tail(rezept);  
    listempty = false;  
}  
else listempty = true;
```

Die boolesche Variable `listempty` wird in einem Testknoten abgefragt, der je nach Wert darüber entscheidet, ob die Schleife verlassen wird.

Kann der Bargast keinen Cocktail nach seinem Geschmack auf der Karte finden oder wird sein Wunsch nicht erkannt, hat er die Möglichkeit, einzelne Zutaten zu bestellen. Dazu ist außer dem Namen auch noch die Menge der Zutat wichtig. Verläuft der Erkennungsprozeß erfolgreich, so gibt es zwei neue Variablenbindungen. `liquid` bezeichnet den Namen der Einzelzutat, `mass` die Menge. (Wobei die Menge in Anzahl der Druckbewegungen auf die Spritze ausgedrückt wird.) MiCo muss nun Informationen über die Position der Zutat erhalten, um sie abfüllen zu können, außerdem braucht er Angaben über Preis und Alkoholgehalt. Dies wird durch folgende Anfrage realisiert:

```
"SELECT pos, preis, alcohol FROM ingredients WHERE name=liquid"
```

Die ein-elementige Ergebnisliste wird an die Variable `zutatInfoList` gebunden. Wiederum wird ein Scriptknoten benutzt, um alle Werte des Records zu extrahieren und an Variablen zu binden, die weiter benutzt werden sollen. Neben dem Abfüllen der geforderten Menge der Zutat, muss über Preis und Alkoholgehalt Buch geführt werden. Dies alles geschieht in einem Scriptknoten.

```
if (!empty(zutatInfoList))  
{  
    struct zutatInfo = head(zutatInfoList);  
    int imass = parseInt(mass);  
    preis = preis + imass*get(zutatInfo, "preis");  
    alcohol = alcohol + imass*get(zutatInfo, "alcohol");  
    posi = str(get(zutatInfo, "pos"));  
}
```

3.1 Neue Cocktails

Als zusätzliche Funktion soll sich der Barkeeper das neue Rezept auch merken können. Das heißt, er muss die entsprechenden Daten in die Datenbank schreiben. Dazu braucht er einen Namen für den neuen Cocktail und es muss ein String generiert werden, der den Schreib-Befehl darstellt. Zwei Lösungen sind denkbar: Die Datenbank könnte entweder nach jedem Abfüllvorgang aktualisiert werden oder ganz am Ende, wenn der Drink fertig gemischt ist. Wir haben uns für letzteres entschieden. Dafür braucht es eine String-Variable, die mit jeder neuen Zutat inkrementiert wird. Wie das funktioniert, sollte gleich klar werden. Der etwas aufwändige Teil ist es, den Namen für das neue Getränk zu erhalten. Dieser sollte völlig frei wählbar sein und keinen Beschränkungen unterworfen. Leider konnten wir aber nur eine sehr beschränkte Grammatik spezifizieren, die nur wenige Worte erkennen kann. Die Lösung war, mit einer endlichen Anzahl von Wörtern, eine praktisch unendliche Menge von Kombinationen zu erlauben. Es war naheliegend, einfach die 26 Buchstaben des Alphabets zu nehmen und die Benutzerin den gewünschten Namen buchstabieren zu lassen. Der Nachteil an dieser Methode ist die Zeit, die dabei vergeht, jeden Buchstaben abzufragen, zu erkennen und sich bestätigen zu lassen. Dieser Prozess kann schnell ermüden. Um die Erkennung zu vereinfachen, sollen einzelne Buchstaben mit Hilfe der Buchstabiertafel geäußert werden. Also zum Beispiel "B wie Berta" und "D wie Dora". So kann eine Verwechslung zwischen B und D vermieden werden.

In der Grammatik-Datei für den Spracherkenner sieht das wie folgt aus:

```
<buchstabe> = [a wie] anton {letter:anton} | [b wie] berta {letter:berta} ...
```

theoretisch würde also auch allein die Äußerung "anton" als a erkannt werden.

Abbildung 3 zeigt den Graphausschnitt, in dem der Name des neuen Cocktails aufgenommen wird. Ich beschreibe kurz, was in den einzelnen Knoten passiert.

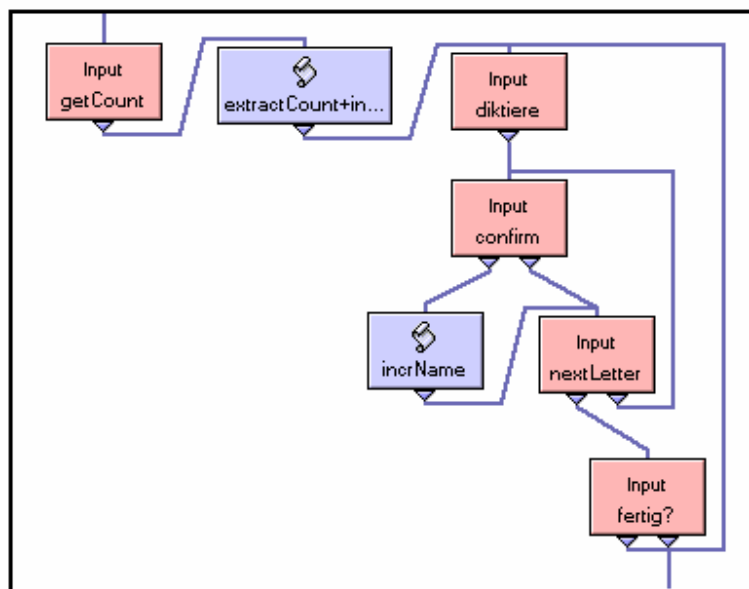


Abbildung 3

Im ersten Knoten `getCount` wird eine Datenbankabfrage losgeschickt, die herausfinden soll, wieviele Einträge es in der Rezepte-Tabelle schon gibt. Diese Zahl ist notwendig zu wissen, um neue Einträge zu generieren, die eine eindeutige ID haben müssen. Alle Einträge werden aufsteigend nummeriert.

Der Datenbank-Befehl lautet: `"SELECT count(*) FROM recipes"`

Eine Anfrage an die Datenbank wird übertragen, in dem eine Instanz des Musters: { query = "SELECT x FROM y WHERE ..." } in das Ausgabe-Feld im Eingabe-Knoten geschrieben wird.

Im zweiten Knoten wird der Wert für Anzahl der vorhandenen Einträge extrahiert. Außerdem wird das Kommando für die Datenbankaktualisierung initialisiert. Abbildung 4 zeigt eine Aufnahme des Knotenfensters, in dem das Script spezifiziert wird.

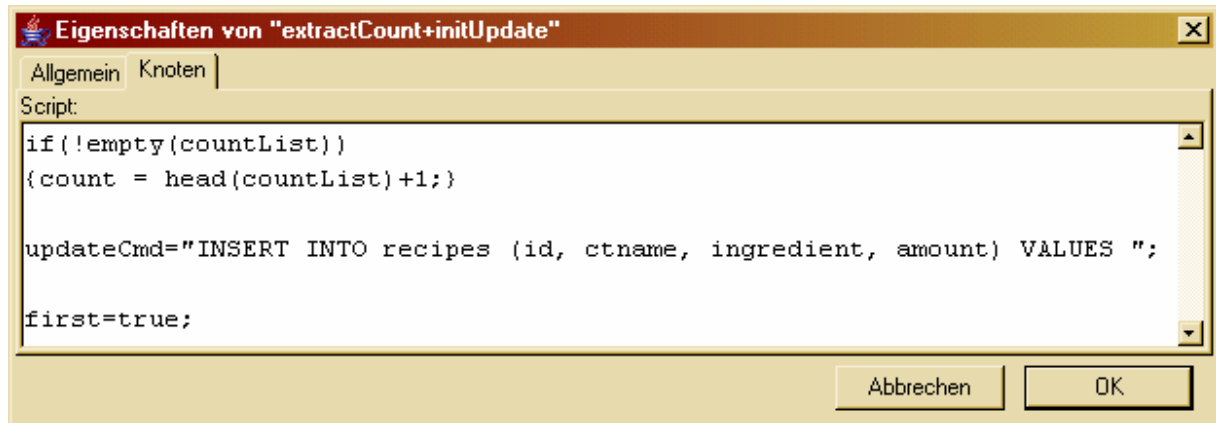


Abbildung 4

Die Variable `countList` enthält den Rückgabewert des Datenbank-Klienten für die oben genannte Anfrage. Die Liste sollte im Normalfall genau ein Element enthalten, nämlich den Wert für die Anzahl aller vorhandenen Einträge. Das Element ist in diesem Fall kein Record wie in den vorher genannten Beispielen, sondern ein atomarer Wert.¹¹ Deshalb kann man direkt über einen Aufruf der eingebauten `head`-Funktion auf den Wert zugreifen.

Der zweite Teil des Scripts initialisiert die Variable `updateCmd`. Hier wird der Anfang des Kommandos angegeben, mit dem später alle Rezeptposten eingetragen werden. Die Variable `first` wird auf `true` gesetzt, weil wir den Anfang des Befehls behandeln. Sobald wir das Kommando um die tatsächlichen Werte erweitern, wird die Variable auf `false` gesetzt. Die verschiedenen Werte im Kommando müssen durch Kommata getrennt werden, aber das erste Komma kommt erst nach dem ersten Wert. Die Variable wird dazu gebraucht, zu entscheiden, ob ein Komma angefügt werden muss. Hier ist das Script für die Inkrementierung des Datenbank-Kommandos.

```
if (!first)
    {updateCmd = updateCmd + " , " ;}
else
    {first = false;}

updateCmd = updateCmd + " ("+(count++)+", \""+newName+"\", \""+liquid+"\",
"+mass+") " ;
```

Im Kommando werden die Variablen `count`, `newName`, `liquid` und `mass` gebraucht. `count` gibt die ID des Eintrages an und wird für jeden neuen Eintrag, der einer Zeile in der Datentabelle entspricht, um eins hochgezählt. `newName` ist der String, der den Namen des unbekanntes Cocktails angibt, `liquid` referenziert die aktuelle Zutat und `mass` die Menge.

Der weitere Verlauf im Subdialog für die Namenserkennung ist wie folgt:

Der Trinkwillige wird aufgefordert, den Namen zu buchstabieren:

¹¹ Bei der Involvierung des Datenbank-Klienten gibt es die Option, sich statt der Records direkt Werte zurückgeben zu lassen, dies ist aber nur möglich, wenn die Rückgabe der Anfrage nur aus einer Spalte besteht. Die genaue Syntax für die Anfrage an den Datenbank-Klienten ist so:

```
{ query = "SELECT x FROM y WHERE ..." }
{ query = "SELECT x FROM y WHERE ...", returnRecords = false }
```

"Mit welchem Buchstaben fängt Ihr Cocktail an? Bitte diktieren Sie entsprechend der Buchstabiertafel. A wie Anton zum Beispiel."

Der erkannte Buchstabe wird an die Variable `letter` gebunden. Dann wird nach einer Bestätigung gefragt:

"Ich habe " +charAt(letter, 0) + " wie "+ letter+ " verstanden. Wenn das richtig ist, bestätigen Sie bitte mit okee, wenn nicht, sagen sie bitte nein!"

Hier wird die eingebaute Funktion `charAt` benutzt, um auf die erste Stelle der Langform zuzugreifen. Bestätigt die Durstige mit "Okay", wird die Variable `newName` in einem nächsten Script-Knoten um den Buchstaben erweitert.

```
newName = newName+charAt(letter, 0);
```

Nach dieser Aktualisierung oder wenn der erkannte Buchstabe nicht bestätigt wird, wird nach dem nächsten Buchstaben gefragt:

"Der nächste Buchstabe bitte oder sind wir fertig?"

Entsprechend der Antwort startet ein erneuter Diktierdurchgang oder die Aufnahme des neuen Namens ist beendet.

Dies ist eine relativ langwierige Prozedur, aber mit den verfügbaren Mitteln meines Erachtens ohne Alternative. Natürlich kann die Kundin auch einen Drink nach eigenem Rezept bestellen, ohne einen Namen dafür einzugeben. Dann kann das Rezept nicht abgespeichert werden. Das Kommando für die Aktualisierung der Datenbank wird in jedem Fall generiert, bei fehlendem Namen mit einem leeren String. Ob das Kommando auch ausgeführt und das neue Rezept eingefügt wird, entscheidet sich dann abhängig vom Wert der `newName`-Variable. Dafür wird ein Konditional-Knoten benutzt, der nur im Falle einer nichtleeren Variablen zum Knoten überleitet, in dem das Einfügen passiert. Ansonsten wird dieser Knoten übersprungen. Der Datenbank-Klient erhält zwei Anweisungen:

```
{update = updateCmd}
```

```
{update = "INSERT INTO preisliste (ctname, preis, alcohol, lautform) VALUES  
(\ "+newName+"\", "+preis+", "+alcohol+", \ "+newName+"\" )" }
```

Mit der ersten wird die `recipes`-Tabelle um eine Anzahl von Zeilen erweitert, mit der zweiten wird die Preisliste um den neuen Cocktail samt Preis und Alkoholgehalt aufgestockt. Der Name und die Lautform sind dabei gleich, weil es keine Möglichkeit gibt, englische Namen zu erkennen und automatisch in eine deutsche Umschrift zu konvertieren. Preis und Promille wurden mittels anderer Script-Knoten errechnet, auf die ich hier nicht weiter eingehen will.

4. Schluss

Ich habe in meiner Arbeit versucht, die Funktionsweise des von meinem Team gebauten Roboters zu erläutern. Dabei habe ich mich vor allem auf die Einbindung der Datenbank in den Dialog konzentriert. Ich habe auch einige technische Details zum Dialogsystem beschrieben. Auf den Dialog als Ganzes und auf die Programmierung des Roboters ist mein Teamkollege Roland Roller eingegangen. Die Konstruktion und technische Umsetzung beschreibt Thomas Horf in seiner Arbeit.

Nach sieben Wochen Arbeit an MiCo lässt sich folgende Bilanz ziehen: Wir haben unser Ziel, einen Roboter, der Cocktails mixen kann, erreicht. Das größte Problem war dabei die Konstruktion des Abfüllmechanismus. Auf vielen Zwischenschritten funktionierte es zwar im prinzipiell, aber es mangelte sehr an Zuverlässigkeit. Je nach Position, Füllstand der Spritze und Ladung der Batterien, gelang es mal, aber misslang das andere Mal.

Bis zur abschließenden Präsentation gelang es uns schließlich mittels unterstützender Federn und Gummis den Widerstand der Spritzpistolen so zu verringern, dass die schwachen Motoren genug Druck ausüben konnten, um jede Abfüllstation zu bedienen. Nichtsdestotrotz macht diese Bastelei keinen sehr

zuverlässigen Eindruck. Eine Optimierung in diesem Bereich könnte darauf hinzielen, den Mechanismus so zu verbessern, dass wir Entwickler ohne Bangen und Zittern, sondern ganz entspannt eine Präsentation durchführen können. Es würde sich vielleicht lohnen, noch mal auf unsere anfängliche Idee eines einfachen Ventils zurückzukommen, das ohne viel Kraftaufwand gedrückt werden könnte. Das erfordert aber einerseits funktionierende Ventile, andererseits einen fundamentalen Umbau des Roboters. Zur Herstellung der Ventile fehlte uns leider das Knowhow und Marktrecherchen nach vorhandenen Lösungen blieben ohne Erfolg.

Erweiterungen auf der Softwareseite betreffen vor allem die Vervollständigung des Lernvorgangs für neue Cocktails. Bis jetzt kann MiCo zwar neue Rezepte lernen und diese abspeichern, es ist aber nicht möglich, zu einem späteren Zeitpunkt diesen neuen Cocktail zu bestellen. Das liegt daran, dass die Grammatik für die erkennbare Sprache vor Dialogstart festgelegt ist und nicht zwischendurch dynamisch verändert werden kann. Das heißt, dass die neuen Namen von der Spracherkennung nicht erkannt werden. Dazu müsste ein Programm geschrieben werden, das die neuen Grammatiken generiert. Diese müssten dann automatisch im Spracherkenner geladen werden.

Eine weitere Erweiterung wäre, dass der Barkeeper noch in einer anderen Sprache als deutsch kommunizieren kann. Dazu müsste es dann für jede Sprache eine eigene Grammatik-Datei geben. Im Dialog müsste am Anfang abgefragt werden, welches die Kommunikationssprache sein soll. Schöner, aber viel schwerer wäre es, die Sprache automatisch zu erkennen.

Ein weiteres Feature von MiCo könnte sein, dass er zu verschiedenen Stimmungen in der Lage wäre und je nach Laune eher fröhliche oder alberne oder schlechtgelaunte oder strenge Äußerungen machen würde. Bedeutendste Einschränkung der Kommunikation stellt sicher das Dialog-Modell des endlichen Automaten dar. Sie macht den offenen, indeterministischen Dialog praktisch unmöglich. Weiteres Hindernis dafür ist aber auch die beschränkte Größe der Erkennungsgrammatik. Hätten wir die Möglichkeit, eine möglichst große und umfassende Untermenge der deutschen Sprache zu erkennen, wäre es vielleicht auch möglich, Eliza einzubinden. Das Programm von Joseph Weizenbaum (1966), das dem Benutzer die Illusion eines intelligenten Gesprächspartners verschafft, könnte die Starrheit des Dialogs aufbrechen, indem es für eine vorher festgelegte Zeitspanne in einem Zustand des Dialogs ausgeführt wird.

Quellen

[1] Michael F McTear (2002) Spoken dialogue technology: enabling the conversational interface. ACM Computing Surveys, Volume 34 , Issue 1 (March 2002), pp. 90 - 169.)

[2] Dimitra Tsovaltzi, Stephan Walter and Aljoscha Burchardt (2003) Dialogue. Teaching material within the MilCA project.