Universität Regensburg

Philosophische Fakultät III
Sprach- , Literatur- und Kulturwissenschaften
Institut für Information und Medien, Sprache und Kultur (I:IMSK)
Lehrstuhl für Informationswissenschaft

Seminar: Methoden der maschinellen Sprachverarbeitung
Modul: INF-M 31 (M.A.)
WS 2013/14
Leitung: PD Dr. Jürgen Reischer

Möglichkeiten und Grenzen von AIML

Evaluierung der Chatbot-Beschreibungssprache „Artificial Intelligence Markup Language“

Patrick Schneider
Obermünsterstr. 21
93047 Regensburg

Patrick.Schneider@stud.uni-regensburg.de
Informationswissenschaft - 1. Semester M.A.

Abgegeben am 30. April 2014

Inhaltsverzeichnis

1 Motivation und Hintergrund

„Her“ – so lautet der Titel des 2013 erschienenen und Oscar-prämierten Spielfilms des Regisseurs Spike Jonze, in dem sich der US-Schauspieler Joaquín Phoenix in sein Computer-Betriebssystem verliebt. Genauer gesagt, in die weibliche Stimme seines Computers, die nicht nur gesprochene Anweisungen fehlerfrei verstehen und umsetzen kann, sondern auch natürlichsprachige Konversationen führen kann. Das Betriebssystem entwickelt sich dabei ständig weiter und bildet so eine eigene Identität, was unter anderem dazu führt, dass es eigene Emotionen und Gefühle zu seinem Nutzer aufbaut. Dabei geht das Science-Fiction-Drama so weit, dass sich zwischen dem Betriebssystem und seinem Nutzer sogar eine romantische Beziehung entwickelt.[1]

Die Stimme des Betriebssystems mit dem Namen *Samantha* erinnert dabei stark an die von Apple entwickelte Spracherkennungs-Software *Siri*. Der Regisseur des Films, Spike Jonze erklärte in einem Interview, dass er die Idee zu diesem Film schon viele Jahre vor *Siri* hatte. Demnach war Jonze vor über zehn Jahren begeisterter Anwender der Chatbot-Software „*Alicebot*“, die ihn maßgeblich für das Drehbuch des Films inspirierte.[2]

Zugegebenermaßen ist der Film bis dato reine Fiktion, jedoch zeigt er eindrucksvoll, was vielleicht in naher Zukunft auf dem Gebiet der künstlichen Intelligenz möglich sein wird. Forschungseinrichtungen und Unternehmen versuchen immer mehr, Computer zu vermenschlichen. Computer sollen demnach nicht länger reine Rechenmaschinen sein - vielmehr sollen sie auf die individuellen Bedürfnisse ihrer Nutzer möglichst präzise eingehen und selbstständig Lösungsansätze vermitteln können. Dabei spielen sogenannte Chatbot-Systeme eine immer wichtiger werdende Rolle, denn Menschen wollen mit dem Computer aktiv kommunizieren, statt Befehle händisch mittels Tastatur oder Maus einzugeben.[3]

Um derartige Chatbot-Systeme zu realisieren und auf Fragen bzw. Befehle der Nutzer eingehen zu können, benötigt es jedoch eine umfangreiche Wissensbasis. Das 1995 veröffentlichte Chatbot-System *A.L.I.C.E* (auch *Alicebot* genannt) verwendet dabei eine XML-basierte Dokumententypdefinition namens *Artificial Intelligence Markup*

[1] Vgl. Spiegel.de 2014.

[2] Vgl. Latimes.com 2014.

[3] Vgl. Sueddeutsche.de 2010.

Language – kurz *AIML*.[4] Diese Auszeichnungssprache ermöglicht es, eine Wissensbasis abzubilden. Dabei müssen konkrete Frage-Antwort-Paare definiert und miteinander in Beziehung gebracht werden. Neben *AIML* wurden in der Vergangenheit auch andere Chatbot-Beschreibungssprachen, wie etwa *Rivescript* oder *ChatScript* entwickelt.[5] Dabei verfolgen alle Beschreibungssprachen dasselbe Ziel: Eine Software zu entwickeln, die sich so verhält, als kämen die Antworten bzw. Fragen von einem Menschen.

Die seit 1995 entwickelte Beschreibungssprache *AIML* nimmt dabei im Bereich von Chatbot-Systemen eine wichtige Rolle ein, da sie die mitunter am weitesten verbreitete Chatbot-Beschreibungssprache darstellt. Bemerkenswert dabei ist, dass *AIML* auf den ersten Blick nur über einen sehr eingeschränkten Funktionsumfang verfügt. Kurz gesagt: Zu jeder vordefinierten Frage gibt es eine vordefinierte Antwort.[6] Eine Frage, die sich hierdurch stellt ist, inwiefern sich *AIML* wirklich für die Umsetzung eines derart komplexen Systems wie das eines Chatbots eignet und wo dabei eventuelle Schwächen vorliegen.

Ziel dieser Arbeit ist es, die Chatbot-Beschreibungssprache (im Folgenden nur *Chatbotsprache* genannt) *AIML* genauer zu untersuchen und dabei auf mögliche Vor- bzw. Nachteile bei der Umsetzung eines Chatbots mit Hilfe von *AIML* aufzuzeigen. Dazu wird in einem einleitenden Teil zuerst auf die Grundlagen von Chatbots eingegangen. In einem weiteren Punkt werden die drei am weitesten verbreiteten Chatbotsprachen *Rivescript*, *ChatScript* und *AIML* genauer erläutert. Im Hauptteil dieser Arbeit soll dann die Chatbotsprache *AIML* detailliert untersucht werden. Neben dem allgemeinen Aufbau wird auf den Funktionsumfang sowie dessen Tagset näher eingegangen. Darüber hinaus sollen eventuelle Schwächen der Chatbotsprache aufgedeckt und erläutert werden. Im letzten Teil dieser Arbeit werden anschließend Erweiterungen von *AIML* kurz vorgestellt und es wird ein abschließendes Fazit gezogen.

[4] Vgl. Geeb 2007, S. 55.

[5] Vgl. van Rosmalen et al. 2012.

[6] Vgl. Alicebot.org 2014a.

2 Chatbot-Grundlagen

Die Geschichte von Chatbots reicht bis in die frühen fünfziger Jahre des letzten Jahrhunderts zurück. Damals stellte der britische Mathematiker und Informatiker Alan Turing den nach ihm benannten *Turing-Test* vor.[7] Dieser Test, welcher maßgebend an der Entwicklung des Informatik-Teilbereichs *Künstliche Intelligenz (KI)* beteiligt war, soll feststellen, ob eine Maschine bzw. Computer dasselbe Denkvermögen wie das eines Menschen besitzt. Um diesen Sachverhalt zu überprüfen, führt eine Testperson eine Unterhaltung mit zwei Gesprächspartnern. Dabei handelt es sich bei einem um eine Maschine, bei dem anderen um einen Menschen. Die Testperson interagiert dabei ohne Sicht- und Hörkontakt nur mittels einer Tastatur und einem Bildschirm mit den beiden Gesprächspartnern. Kann die Testperson nach einer gewissen Zeit nicht sagen, bei welchem Gesprächspartner es sich um eine Maschine handelt, ist der Turing-Test bestanden. Damit wäre nach Alan Turing bewiesen, dass die Maschine dasselbe Denkvermögen besitzt wie ein Mensch.[8]

Inspiriert durch Alan Turings Ansätze der künstlichen Intelligenz entwickelte der deutsche Informatiker Joseph Weizenbaum im Jahr 1966 den ersten natürlichsprachigen Chatbot namens *ELIZA*. Hiermit war es erstmals möglich, *echte* Unterhaltungen mit einem Computer zu führen. Dabei funktioniert *ELIZA* nach einem einfachen Prinzip – Aussagen bzw. Fragen eines Nutzers werden in Gegenfragen umformuliert, um so einen kontinuierlichen Gesprächsfluss zwischen Mensch und Maschine zu simulieren. Dabei verhält sich *ELIZA* gegenüber dem Gesprächspartner wie ein Psychotherapeut und verfügt dabei über keinerlei Weltwissen.[9]

Um die Forschung im Bereich der künstlichen Intelligenz weiter anzutreiben, gründete der amerikanische Soziologe Hugh Gene Loebner im Jahr 1991 den *Loebner-Preis*. Der mit bis zu 100.000 Dollar (Stand 2014) dotierte Preis soll an denjenigen Programmierer gehen, der eine (Chatbot-) Software entwickelt, die einen *starken* Turing-

[7] Vgl. Moor 2013, S.2.

[8] Vgl. Saygin et al. 2001.

[9] Vgl. Weizenbaum 1996.

Test erfolgreich besteht.[10] Bisher konnte jedoch noch kein Computerprogramm diesen Test bestehen.

Als Nachfolger von *ELIZA* trat 1995 schließlich der Chatbot *A.L.I.C.E.* (Artificial Linguistic Internet Computer Entity) in Erscheinung.[11] Die Software verwendet dabei zur Abbildung der Wissensbasis die Chatbotsprache *AIML*. Trotz eines relativ großen Entwicklungsteams, das sich aus mehr als 300 Programmierern aus aller Welt zusammensetzt, gelang es bis dato nicht, den Turing-Test erfolgreich zu bestehen. Obwohl die Software über eine umfassende Wissensbasis verfügt, ist es für Menschen nach relativ kurzer Zeit ersichtlich, dass es sich dabei um eine Maschine handelt.

Trotz der relativ langen Geschichte von Chatbots haben sich bis heute nur wenige Chatbots durchsetzen können. Dennoch werden derartige Anwendungen immer wieder v.a. als unterstützende Funktion auf Webseiten eingebunden. So verwendet beispielsweise die Bibliothek Hamburg einen Chatbot namens *Stella*, dessen Schwerpunkt die Vermittlung von Informationen über die Bibliotheksdatenbank ist.[12] Darüberhinaus besteht die Möglichkeit, *Smalltalk* mit Stella zu führen. Auch andere Bibliotheken und Unternehmenswebseiten bieten ihren Nutzern immer häufiger Chatbots als einsteigende Informationsquelle bzw. FAQ-Einheit an. Nicht zuletzt durch geschicktes Marketing machte die Firma Apple im Jahr 2010 die Spracherkennung-Software *Siri* durch die Integration in das iPhone 4s einer breiten Masse zugänglich.[13] Neben einfachen Befehlen reagiert *Siri* auch auf belanglose Fragen wie z.B. „Wie geht es dir?“. Eine weitere Besonderheit besteht darin, dass sich *Siri* an Nutzer anpasst und individuelle Vorlieben und Abneigungen mit in die Ausgabe einbezieht. Die Software lernt somit kontinuierlich dazu und erweitert so stetig ihre Wissensbasis. Je mehr Nutzer also die Software verwenden, desto qualitativ besser wird sie. Nach Apple haben auch andere große Firmen wie z.B. Google oder Microsoft Spracherkennungssoftware in ihre (mobilen) Betriebssysteme integriert.[14] Hierbei ist anzumerken, dass diese Anwendungen chatbot-ähnliche Komponenten besitzen – einen Chatbot, wie ihn

[10] Vgl. Sueddeutsche.de 2011.

[11] Vgl. Shawar, Atwell 2007.

[12] Vgl. Uni-Hamburg.de 2014.

[13] Vgl. Golem.de 2010.

[14] Vgl. Engadget.com 2014.

beispielsweise Joseph Weizenbaum definiert, verkörpern diese jedoch nicht, da man mit ihnen keine echte Unterhaltungen über mehrere Konversationseinheiten führen kann. Vielmehr handelt es sich etwa bei *Siri* um eine Spracherkennungssoftware mit Chatbot-Funktionen.

3 Chatbot-Beschreibungssprachen

Um einen Chatbot zu realisieren und auf Fragen bzw. Anweisungen eines Nutzers reagieren zu können, benötigt es eine umfangreiche Wissensbasis. Idealerweise müssen alle möglichen Eingaben eines Nutzers vordefiniert und mit einer entsprechenden Ausgabe verknüpft sein. Um eine Wissensbasis möglichst präzise und einfach zu erstellen, wurden in der Vergangenheit mehrere Chatbotsprachen entwickelt, mit denen auf unterschiedliche Art und Weise (Welt-)Wissen abgebildet und miteinander in Beziehung gesetzt werden kann. Im Folgenden werden die drei meistverwendeten Chatbotsprachen *Rivescript*, *ChatScript* und *AIML* genauer erläutert.

3.1 Rivescript

Die Chatbotsprache *Rivescript* wurde 2004, ursprünglich als Perl Modul, von Casey Kirsle entwickelt. Im Gegensatz zu *AIML* ist *Rivescript* eine rein klartext-basierte Chatbotsprache, die ohne klassisches Tagset auskommt. Die Chatbotsprache wird dabei Zeile für Zeile vom Interpreter ausgelesen und verarbeitet. Jede Zeile muss dafür mit einem entsprechenden Symbol bzw. Kommando beginnen.[15] Ursprünglich als reine Alternative zu *AIML* entwickelt kamen im Laufe der Zeit viele umfangreiche Funktionen hinzu; es bietet nun nach eigenen Angaben einen sehr viel breiteren Funktionsumfang als *AIML*.[16]

Der Aufbau von *Rivescript* ist relativ einfach gehalten. Ein sogenannter *trigger* bildet eine Nutzereingabe ab und wird mit einem vorangestellten „+“ signalisiert. Ein *response* wird mit einem „-“ gekennzeichnet und symbolisiert die Reaktion auf eine Nutzereingabe. Dabei werden *trigger* im Quelltext fortlaufend kleingeschrieben und

[15] Vgl. Rivescript.com 2014a.

[16] Vgl. Rivescript.com 2014a.

ohne Satzzeichen verwendet. Ein einfaches Frage-Antwort-Beispiel sieht demnach wie folgt aus:

```
+ how are you
- I'm great, how are you?
```

Gibt der Nutzer die Frage „how are you“ ein, antwortet der Chatbot mit dem Satz „I’m great, how are you?". Um den Chatbot jedoch etwas menschlicher wirken zu lassen, besteht die Möglichkeit, verschiedene Antworten auf dieselbe Nutzereingabe ausgeben zu lassen.

```
+ how are you
- I'm great, how are you?
- I'm good, you?
- Good :) you?
```

Gibt der Nutzer nun dieselbe Eingabe wie oben ein, wählt der Chatbot eine der drei Antwortmöglichkeiten zufällig aus und gibt diese anschließend aus. Durch Gewichtung der einzelnen Antworten kann Einfluss auf die Häufigkeit der gegebenen Antworten genommen werden. Im folgenden Beispiel wird die erste Antwort bei 20 von insgesamt 100 Aufrufen ausgegeben. Die anderen Antworten jeweils 50 und 30 Mal.

```
+ how are you
- I'm great, how are you? {weight=20}
- I'm good, you? {weight=50}
- Good :) you? {weight=30}
```

Jeder Chatbot unter *Rivescript* benötigt eine Datei namens *begin.rs*, welche verschiedene Konfigurationen beinhaltet. Ein wichtiger Bestandteil sind hier die Substitutionen; d.h. es wird definiert, welche Abkürzungen wie interpretiert werden sollen. Nutzereingaben wie z.B. „i’m fine“ und „i am fine“ sind zwar inhaltlich identisch, werden aber unterschiedlich geschrieben. Damit der Interpreter derartige Aussagen als dieselbe verarbeitet, müssen diese vorher in der Datei *begin.rs* definiert werden (vgl. folgendes Beispiel).

```
! version = 2.0
// Bot variables
! var name = Tutorial
! var age  = 5
// Substitutions
! sub i'm  = i am
! sub i'd  = i would
! sub i've = i have
! sub i'll = i will
```

Nutzereingaben wie „i'll“ und „i will“ werden demnach gleich verstanden. Somit ist sichergestellt, dass alle Schreibweisen vom Interpreter erkannt werden.

Damit der Chatbot nicht nur auf komplette *trigger*, sondern auch auf einzelne Satzteile davon reagieren kann, gibt es sogenannte *Wildcards*. Mit diesen Elementen ist es möglich, einzelne Wörter oder Satzteile der Nutzereingabe mit in die Antwort einfließen zu lassen. Folgende Beispiele sollen die Verwendung von *Wildcards* in *Rivescript* verdeutlichen.

```
+ my name is *
- Nice to meet you, <star1>!

+ * told me to say *
- Why would <star1> tell you to say "<star2>"?
- Did you say "<star2>" after <star1> told you to?

+ i am * years old
- A lot of people are <star1> years old.
```

Mit dem Stern-Zeichen „*“ können Nutzereingaben offen gelassen werden. Gibt der Nutzer beispielsweise „my name is patrick“ ein, antwortet der Chatbot „Nice to meet you, Patrick!“. Die durchnummerierten Tags *<star1>*, *<star2>* etc. beziehen sich dabei jeweils auf die einzelnen Wildcards bzw. „*“. Somit kann der Chatbot individuell auf Nutzereingaben eingehen und diese mit in die Antwort einfließen lassen.

Eine weitere Form von *Wildcards* sind sog. *Optionals* und *Alternatives*. Diese werden dazu verwendet, um *Wildcards* auf einige wenige (oder keine - dann *Optionals*) Wörter zu beschränken, d.h. der Chatbot reagiert nur auf bestimmte, vorher definierte Wörter (vgl. dazu folgendes Beispiel).

```
+ how [are] you
- I'm great, you?

+ what is your (home|office|cell) [phone] number
- You can reach me at: 1 (800) 555-1234.
```

Wörter, die in eckige Klammern gesetzt sind, müssen nicht zwingend in der Eingabe vorkommen und sind somit optional. Wörter, die in runden Klammern stehen, sind *Alternatives*, und es muss mindestens eines davon in der Nutzereingabe bzw. im *trigger* vorkommen, damit der Chatbot darauf entsprechend reagiert. Diese Funktion erlaubt es, Satzzerlegungen relativ einfach umzusetzen.

Damit Nutzereingaben und Chatbot-Ausgaben nicht isoliert voneinander stehen, gibt es *Short discussions*. Damit besteht die Möglichkeit, auf vorherige Nutzereingaben einzugehen und so einen kleinen Gesprächsfluss zu erzeugen.

```
+ knock knock
- Who is there?

+ *
% who is there
- <star> who?
```

Lautet die Nutzereingabe „knock knock“ antwortet der Chatbot mit „Who is there?“. Die Antwort des Nutzers auf die Frage „who is there“ kann mit Hilfe des %-Symbols referenziert werden und in das darauffolgende Statement mittels *<star>* eingebunden werden.

Diese hier aufgezeigten Funktionen von *Rivescript* gehören zu den Grundfunktionen. Darüber hinaus gibt es noch zahlreiche Weitere, welche im Rahmen dieser Arbeit jedoch nicht weiter betrachtet werden können. Eine ausführliche Anleitung zum gesamten Funktionsumfang sowie weiterführende Themen hat Rivescript-Entwickler Casey Kirsle auf der offiziellen Website unter *www.rivescript.com* veröffentlicht.

3.2 ChatScript

Die Chatbotsprache *ChatScript* wurde vom ehemaligen Computerspiele-Entwickler Bruce Wilcox erfunden und im Jahr 2011 als Open-Source Projekt veröffentlicht.[17] Bereits vor der Veröffentlichung wurde *ChatScript* für die Entwicklung der Chatbots *Suzette* (2008) und *Rosette* (2011) verwendet.[18] Für beide Chatbots erhielt Wilcox den begehrten *Loebner-Preis*, jedoch nicht für einen *starken* Turing-Test.

ChatScript zeichnet sich durch einen regelbasierten Ansatz zur natürlichsprachigen Verarbeitung aus und verwendet eigene Ontologien zur Analyse von Satzbausteinen. Regeln werden in *topics* gegliedert und top-down abgearbeitet. Dabei kann eine Regel standardmäßig nur einmal für eine Konversation genutzt werden.[19] Ähnlich wie *Rivescript* kommt *ChatScript* ohne jegliches Tagset aus. Dabei werden unabhängige Reaktionen mit „*t:*“ gekennzeichnet. Eingaben, die vom Chatbot abgeglichen und auf die dementsprechend reagiert werden sollen, werden mit „*u:*“ markiert. Folgendes Beispiel zeigt den Aufbau eines einfachen *ChatScripts*:

```
topic: ~introductions keep repeat []

t: HI () [Hello] [Hi] [Hey], [talk] [speak] [say
something] to me!

u: WHAT (what are you) I am a bot.

u: (tell me about yourself) ^reuse(WHAT) ^reuse(HI)
```

Die erste Zeile gliedert die Konversation in ein *topic*. Dabei signalisiert *keep repeat []*, dass die folgenden Regeln nach der Konversation wiederverwendet werden sollen. Die zweite Zeile zeigt eine nutzerunabhängige Ausgabe mit verschiedenen Alternativen. Der Chatbot sagt beispielsweise „Hello, say something to me!“. Dabei wird diese Ausgabe mit dem Label „HI“ zur späteren Wiederverwendung gekennzeichnet. Lautet die Nutzereingabe nun „What are you?“, antwortet der Chatbot mit dem Satz „I am a bot.“ –

[17] Vgl. Brilligunderstanding.com 2014.

[18] Vgl. Chatbots.org 2014.

[19] Vgl. Wilcox 2013.

das Label für diese Aussage ist „WHAT“. Gibt der Nutzer nun den Satz „Tell me about yourself“ ein, verweist das *ChatScript* auf die beiden Labels „WHAT“ und „HI“ und gibt die dementsprechenden Aussagen zurück.

Um auf unterschiedliche Nutzereingaben identisch zu reagieren, müssen die Schlagwörter in runde und eckige Klammern nacheinander aufgelistet werden. Auf jede Nutzereingabe, die nun die Wörter „what“ oder „where“ beinhaltet, reagiert der Chatbot mit der Antwort „good question.“.

```
u: ([what where]) good question.
```

ChatScript verfügt, wie auch *Rivescript* und *AIML*, über einen breiten Funktionsumfang. Neben Standardfunktionen wie die Wiederverwendung von Nutzereingaben, Zufallsausgaben oder Verweise, ist auch ein sogenanntes Langzeitgedächtnis implementiert, das Nutzereingaben langfristig und über mehrere Konversationseinheiten hinweg speichern und bei Bedarf abrufen kann.

Da es sich bei *ChatScript* um eine noch relativ junge Chatbotsprache handelt, findet diese bis dato noch kaum Anwendung. Zudem ist sie im Vergleich zu *Rivescript* oder *AIML* um einiges komplexer, bietet jedoch viele Funktionen und wird laufend von dessen Erfinder Wilcox weiterentwickelt.[20]

3.3 AIML

Die Chatbotsprache *AIML* wurde im Jahr 1995 vom Informatiker Dr. Richard Wallace als Open-Source-Projekt ins Leben gerufen. Bis zum Jahr 2000 arbeiteten über 500 Entwickler aus aller Welt an der Chatbotsprache. Wallace schuf damit die Grundlage für den von ihm entwickelten Chatbot namens A.L.I.C.E., welcher seit der Veröffentlichung im Jahr 1995 bereits dreimal den *Loebner-Preis* gewann.[21] 2002 ging Wallace eine Kooperation mit dem Unternehmen *Franz, Inc.* ein, um einen *AIML*-Server und -Interpreter, geschrieben in Common Lisp, online zur Verfügung zu stellen. Später ging daraus das Unternehmen *Pandorabots Inc.* hervor - ein kostenloser Online-Service zur Erstellung von

[20] Vgl. Chatscript.sourceforge.net 2014.

[21] Vgl. Alicebot.org 2014b.

Chatbots mit *AIML*.[22] Die Entwicklung von *AIML* wird stetig vorangetrieben und dessen Funktionsumfang laufend erweitert. So ist seit Anfang 2013 die Version 2.0 online verfügbar.

Eine genaue Erläuterung von *AIML* und dessen Funktionsumfang wird im folgenden Kapitel 4 beschrieben.

4 Evaluierung von AIML

Das folgende Kapitel untersucht die Chatbotsprache *AIML* und geht dabei auf deren Aufbau und Funktionsumfang näher ein. Außerdem sollen mögliche Schwächen aufgedeckt und erläutert werden.

4.1 Aufbau und Syntax

AIML ist eine XML-basierte Chatbotsprache, welche ein vordefiniertes Tagset verwendet. Einzelne Konversationseinheiten werden durch festgelegte Elemente in spitzen Klammern differenziert.

Jede Konversationseinheit wird in Kategorien aufgeteilt. Dabei besteht jede Kategorie (*category*) aus genau einem *pattern*- und einem *template*-Element. Ein *pattern* beinhaltet die Aussage bzw. Frage, ein *template* die Reaktion auf ein *pattern*.[23] Folgendes Beispiel verdeutlicht den grundlegenden Aufbau eines *AIML*-Scripts:

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0">

<category>
     <pattern>HALLO CHATBOT</pattern>
     <template>Hallo Mensch!</template>
</category>

</aiml>
```

Die ersten beiden und die letzte Zeile definieren den *AIML*-Bereich für den Interpreter. Das Element <*category*> definiert eine Konversationseinheit, welche aus dem Frageteil (<*pattern*>) und Ausgabebereich (<*template*>) besteht. Gibt der Nutzer nun die Aussage

[22] Vgl. Pandorabots.com 2014a.

[23] Vgl. Alicebot.org 2014a.

„Hallo Chatbot!“ ein, antwortet dieser mit „Hallo Mensch!“. Groß- und Kleinschreibung sowie Satzzeichen werden vom Interpreter ignoriert. Zu beachten ist, dass der Chatbot nur auf exakt diese Nutzereingabe reagiert – Unterschiede in der Rechtschreibung oder im Satzbau werden vom Chatbot in dieser Form nicht toleriert.

Um eine Wissensbasis aufzubauen, müssen alle Fragen und Antworten jeweils in einzelne *category*-Elemente nacheinander aufgelistet werden. Durch bestimmte Funktionen (siehe Kapitel 4.2) kann auf andere Kategorien verwiesen werden. Grundsätzlich ist zu sagen, dass der Aufbau einer umfangreichen Wissensbasis in *AIML* einen enormen Schreibaufwand mit viel Redundanz darstellt. Um einzelne Themen sinnvoll zu gruppieren, besteht jedoch die Möglichkeit, eine Wissensbasis in mehrere *AIML*-Dateien aufzugliedern.

4.2 Funktionen und Tagset

4.2.1 Random-Element

Um zufällige Antworten auszugeben, gibt es das *random*-Element. Damit besteht die Möglichkeit, aus einer beliebigen Anzahl von Antworten eine zufällig ausgeben zu lassen. Anders als bei *Rivescript* können Zufallsantworten jedoch nicht gewichtet werden (vgl. Kapitel 3.1).[24] Beim folgenden Beispiel wird auf die Eingabe „Hallo“ eine der drei nachstehenden Antworten per Zufall ausgegeben.

```
<category>
      <pattern>HALLO</pattern>
      <template>
            <random>
                  <li>Hi! Wie geht’s?</li>
                  <li>Hallo Mensch</li>
                  <li>Hallo, wie kann ich dir helfen?</li>
            </random>
      </template>
</category>
```

[24] Vgl. Rivescript.com 2014b.

4.2.2 Wildcards

Analog zu *Rivescript* und *ChatScript* unterstützt auch *AIML* sogenannte *Wildcards*. Damit besteht die Möglichkeit, auf beliebige Nutzereingaben zu reagieren. Folgende Beispiele veranschaulichen die verschiedenen Arten von *Wildcards* in *AIML*.[25]

```
<category>
      <pattern>_ MUTTER</pattern>
      <template>Kennst du meine Mutter?</template>
</category>

<category>
      <pattern>MUTTER *</pattern>
      <template>Kennst du meine Mutter?</template>
</category>

<category>
      <pattern>_ MUTTER *</pattern>
      <template>Kennst du meine Mutter?</template>
</category>
```

Beim ersten *category*-Element wird eine *Wildcard* vor dem Schlagwort „Mutter“ verwendet. Gibt der Nutzer also eine beliebige Eingabe ein, bei der am Satzende „Mutter“ steht, reagiert der Chatbot darauf mit „Kennst du meine Mutter?“. Beim zweiten Beispiel reagiert der Chatbot nur auf Eingaben, bei denen das Schlagwort am Anfang des Satzes steht. Beim dritten Beispiel genügt es, wenn das Wort „Mutter“ an einer beliebigen Stelle im Satz vorkommt. Dabei ersetzen die Symbole „*“ und „_“ beliebig viele Wörter.

4.2.3 Srai-Element

Das *srai*-Element ist ein rekursiver Operator in *AIML* und kann für mehrere Fälle verwendet werden.[26]

Die wohl gängigste Anwendung eines *srai*-Elements ist die Verknüpfung von ähnlichen bzw. gleichen Kategorien (*categories*). Inhaltsgleiche Nutzereingaben können dadurch auf dieselbe Ausgabe verwiesen werden, ohne diese explizit im Quelltext

[25] Vgl. Pandorabots.com 2014b.

[26] Vgl. Alicebot.org 2014c.

wiederholt eingeben zu müssen. Das in Kapitel 4.2.2 aufgeführte Beispiel kann mit Hilfe eines *srai*-Elements vereinfacht werden:

```
<category>
      <pattern>MUTTER</pattern>
      <template>Kennst du meine Mutter?</template>
</category>

<category>
      <pattern>_ MUTTER</pattern>
      <template><srai>MUTTER</srai></template>
</category>


<category>
      <pattern>MUTTER *</pattern>
      <template><srai>MUTTER</srai></template>
</category>


<category>
      <pattern>_ MUTTER *</pattern>
      <template><srai>MUTTER</srai></template>
</category>
```

Das erste *category*-Element beinhaltet die Antwort bzw. Ausgabe auf das Schlagwort „Mutter“. Die drei darauffolgenden Kategorien mit *Wildcards* verweisen jeweils auf die erste Kategorie. Somit können alle Fälle mit derselben Antwort abgefangen werden, ohne diese jedes Mal erneut aufzuführen. Das *pattern* „MUTTER“ ist damit ein Label, auf das mit `<srai>MUTTER</srai>` verwiesen wird. Mit Hilfe des *srai*-Elements kann somit ein Keyword-Matching innerhalb von *AIML* realisiert werden.

Eine weitere Anwendung des *srai*-Elements ist die Verarbeitung von Synonymen.[27] Unterschiedliche Schreibweisen von Nutzereingaben können somit abgefangen werden. Folgendes Beispiel verdeutlicht die Verarbeitung von synonymen Nutzereingaben:

[27] Vgl. Alicebot.org 2014c.

```
<category>
      <pattern>HALLO</pattern>
      <template>Hallo, wie geht's?</template>
</category>

<category>
      <pattern>HI</pattern>
      <template><srai>HALLO</srai></template>
</category>
```

Gibt der Nutzer „Hallo“ oder nur „Hi“ ein, antwortet der Chatbot mit der Ausgabe „Hallo, wie geht's?“, da die zweite Kategorie auf die erste verweist wird.

Eine weitere nützliche Anwendung von *srai*-Elementen ist die Korrektur von Nutzereingaben bzw. die Erkennung von alternativen Schreibweisen oder Abkürzungen (vgl. dazu folgendes Beispiel):

```
<category>
    <pattern>YOU ARE A *</pattern>
    <template>Why do you think that?</template>
</category>

<category>
    <pattern>YOUR A *</pattern>
    <template>
           I presume you mean "you are a ..."
           <srai>YOU ARE A <star/></srai>
     </template>
</category>
```

Das obenstehende Beispiel gibt bei der Nutzereingabe „**You are** a Chatbot.“ die Antwort „Why do you think that?“ zurück. Gibt der Nutzer aber die Frage in einer anderen Schreibweise ein, also „**Your** a Chatbot.“, reagiert der Chatbot mit der Antwort „I presume you mean „you are a ...“ Why do you think that?“. Das *template* der zweiten Kategorie verweist mit Hilfe eines *srai*-Elements auf die erste Kategorie. Hierbei ist anzumerken, dass *Wildcards* im *template*-Teil immer mit <*star*/> symbolisiert werden.

4.2.4 That-Element

Das *that*-Element ermöglicht in *AIML*, sich rückwirkend auf eine vorherige Antwort des Nutzers zu beziehen. Damit besteht die Möglichkeit, einen Gesprächsfluss zwischen Chatbot und Nutzer aufzubauen.[28] Häufig wird diese Methode für Ja-Nein-Fragen verwendet. Folgendes Beispiel zeigt eine kontextabhängige Frage in *AIML* mit Hilfe des *that*-Elements:

```
<category>
  <pattern>ICH MAG TIERE</pattern>
  <template>Hast Du einen Hund?</template>
</category>

<category>
  <pattern>JA</pattern>
  <that>Hast Du einen Hund?</that>
  <template>Wie heißt Dein Hund?</template>
</category>
```

Auf die Eingabe „Ich mag Tiere“ antwortet der Chatbot mit der Frage „Hast Du einen Hund?“. Wird diese Frage vom Nutzer mit „Ja“ beantwortet, gibt der Chatbot die Frage „Wie heißt Dein Hund?“ aus. Das *that*-Element in der zweiten Kategorie verweist dabei auf das *template* der ersten Kategorie. Wird auf dieses *template* (Label: „Hast Du einen Hund?“) mit „Ja“ geantwortet, gibt der Chatbot die Frage „Wie heißt Dein Hund?“ aus.

4.2.5 Topic-Element

Um inhaltsgleiche oder ähnliche Fragen und Antworten sinnvoll zusammenzufassen, gibt es in *AIML* das sogenannte *topic*-Element. Mit diesem Element können Themenkreise unter einem *Topic* kategorisiert werden. Zum einen verhilft das *topic*-Element zu einer besseren Übersicht innerhalb des *AIML*-Codes, zum anderen ermöglicht es, Konversationen besser auf ein bestimmtes Thema zu fokussieren. Idealerweise beinhaltet ein *topic* mehrere *pattern-template*-Paare. Dabei kann es durchaus der Fall sein, dass identische *patterns* innerhalb einer *AIML* Datei vorkommen. Der *AIML*-Interpreter geht beim Matching einer Nutzereingabe dabei grundlegend nach folgender Reihenfolge vor: <*pattern*> → <*that*> → <*topic*>. Eine Nutzereingabe wird top-

[28] Vgl. Alicebot.org 2014d.

down mit allen vorhanden *patterns* verglichen. Gibt es mehrere Kategorien, dessen *patterns* mit der Nutzereingabe übereinstimmen, wird diese mit eventuell vorhandenen *that*-Elementen verglichen. Gibt es auch hier mehrere identische *that*-Elemente, die mit der Nutzereingabe übereinstimmen, wird mit dem *topic* verglichen.[29] Folgendes Beispiel soll dieses Prinzip erläutern:

```
<category>
  <pattern>_ AUTOS</pattern>
  <template>Was sind Autos?</template>
</category>

<category>
  <pattern>_ AUTOS</pattern>
  <template>Du willst also über
     <set name="topic">AUTOS</set> reden
  </template>
</category>

<topic name="AUTOS">
  <category>
    <pattern>*</pattern>
    <template>Hast du ein Auto?</template>
  </category>
</topic>
```

Gibt der Nutzer z.B. „Ich mag Autos“ ein, vergleicht der Interpreter die Eingabe zuerst mit allen vorkommenden *patterns*. Hier würden nun zwei Kategorien mit der Eingabe übereinstimmen. Da *topics* aber bevorzugt werden, bekommt der Nutzer die Ausgabe „Du willst also über Autos reden“ zurück. Gleichzeitig wird in das *topic* „AUTOS“ gesprungen. Egal welche Eingabe der Nutzer nun tätigt, der Chatbot gibt die Frage „Hast du ein Auto?“ aus. Darauffolgende Nutzereingaben werden nun zuerst mit *patterns* verglichen, die unter diesem *topic* zusammengefasst sind.[30]

Mit Hilfe von *topics* können Konversationen also thematisch im Kontext gehalten werden. Dadurch können inhaltlich zusammenhängende Konversationen erzeugt werden.

[29] Vgl. Pandorabots.com 2014c.

[30] Vgl. Alicebot.org 2014e.

4.3 Grenzen von AIML

In den vorangegangenen Kapiteln 4.1 und 4.2 wurden der grundlegende Aufbau sowie alle wichtigen Funktionen von *AIML* erklärt. In diesem Kapitel soll die Chatbotsprache auf weitere Fähigkeiten, insbesondere im Bereich der maschinellen Sprachverarbeitung, untersucht werden.

4.3.1 Pattern-Matching

Ein wichtiger Bestandteil bei Chatbots ist das sogenannte *Pattern-Matching*, also der Vergleich einer Nutzereingabe mit der vorhandenen Wissensbasis. Da niemals alle möglichen Nutzereingaben exakt in der Wissensbasis vordefiniert werden können, müssen andere Möglichkeiten herangezogen werden. Um eine Nutzereingabe mit eventuell übereinstimmenden *patterns* zu vergleichen, werden in *AIML* nur ganze Wörter oder Satzteile mit der Nutzereingabe auf Gleichheit hin überprüft. Dabei wird die Groß- und Kleinschreibung von Wörtern sowie Interpunktion nicht beachtet und beim *Preprocessing* entfernt.

Eine Möglichkeit, Nutzereingaben auf Übereinstimmungen hin zu prüfen, bieten *Wildcards* (vgl. dazu Kapitel 4.2.2). Dabei können für die *Wildcards* „_“ und „*“ ein oder mehrere Wörter stehen. Um eine einfache Aussage wie z.B. „(...) I love you (...)“ in allen möglichen Varianten zu vergleichen, sind in *AIML* insgesamt vier Regeln bzw. Kategorien notwendig.

```
<category>
      <pattern>I LOVE YOU</pattern>
      <template>Thank you</template>
</category>

<category>
      <pattern>_ I LOVE YOU</pattern>
      <template>Thank you</template>
</category>

<category>
      <pattern>I LOVE YOU *</pattern>
      <template>Thank you</template>
</category>
```

```
<category>
      <pattern>_ I LOVE YOU *</pattern>
      <template>Thank you</template>
</category>
```

Das obenstehende Beispiel deckt jedoch nicht die Aussage „Chatbot, I really love you!“ ab. Für diese Aussage müssten nochmals separate Regeln definiert werden, was zu einer enormen Redundanz und Schreibarbeit führt. Würden *Wildcards* in AIML statt *ein oder mehrere* Wörter zugleich auch *null* Wörter unterstützen, wäre das obenstehende Beispiel mit einem einzigen Befehl abgedeckt.[31] Wären zudem *Wildcards* zwischen zwei Satzteilen erlaubt, könnten mit folgendem fiktiven Befehl auch Nutzereingaben wie z.B. „Chatbot, I really love you!“ abgedeckt werden:

```
<pattern>_ I * LOVE YOU *</pattern>
```

Eine weitere und durchaus effizientere Möglichkeit des Pattern-Matchings in *AIML* bietet die rekursive Substitution mit Hilfe des *srai*-Elements (vgl. Kapitel 4.2.3). Durch rekursive Verweise können längere Sätze in mehrere Satzteile aufgegliedert und anschließend einzeln verglichen werden. So kann die Nutzereingabe „Can you please tell me what LINUX is right now“ durch sukzessive Aufteilung in die Frage „What is LINUX?“ gekürzt werden. Folgender *AIML*-Code zeigt diese rekursive Substitution:

[31] Vgl. Wilcox 2011, S. 2.

```
<category>
      <pattern>* RIGHT NOW</pattern>
      <template><srai><star/></srai></template>
</category>

<category>
      <pattern>CAN YOU PLEASE *</pattern>
      <template><srai>PLEASE<star/></srai></template>
</category>

<category>
      <pattern>PLEASE TELL ME WHAT *</pattern>
      <template><srai>TELL ME WHAT<star/></srai></template>
</category>


<category>
      <pattern>TELL ME WHAT * IS</pattern>
      <template><srai>WHAT IS<star/></srai></template>
</category>

<category>
      <pattern>WHAT IS LINUX</pattern>
      <template>LINUX is an operating system.</template>
</category>
```

Folgende Zerlegung erfährt dabei obenstehender Satz:

1. Can you please tell me what LINUX is right now
2. Can you please tell me what LINUX is
3. Please tell me what LINUX is
4. Tell me what LINUX is
5. What is LINUX

→ LINUX is an operating system

Insgesamt lässt sich sagen, dass *AIML* über ein sehr primitives Pattern-Matching verfügt, das zudem meist sehr aufwendig zu implementieren ist. Wünschenswert wären hier beispielsweise erweitere *Wildcard*-Funktionalitäten, um Redundanzen zu minimieren.

Diese Redundanzen lassen auch erahnen, wieso der Chatbot A.L.I.C.E. über 120.000 Regeln bzw. Kategorien aufweist.[32]

4.3.2 Wortzerlegung

Ein weiterer wichtiger Bestandteil bei der maschinellen Sprachverarbeitung stellt die Wortzerlegung (engl. *Stemming*) dar. Das auch als Stamm- oder Grundformenreduktion bezeichnete Verfahren soll verschiedene Varianten von Wörtern auf ihre Wortstämme reduzieren.[33] Angefangen von der einfachen Umwandlung von Plural in Singular (Chatbots → Chatbot) bis hin zur Grundformreduktion von konjugierten Verben (sah → sehen) gibt es insbesondere im Fachgebiet des *Information Retrievals* diverse Methoden bzw. Algorithmen zur Wortzerlegung.

Leider bietet *AIML* keine konkreten Lösungsansätze für einfache Wortzerlegungen. Es besteht also nicht die Möglichkeit, einzelne Wörter aufzugliedern. Soll der Chatbot beispielsweise auf das Signalwort „Programm“ eine bestimmte Antwort liefern, müssen alle möglichen Varianten dieser Grundform vorher definiert werden. Wörter wie „programmieren“, „Programmierer“, „Programme“ oder „programmiert“ müssen also alle vorher durch eine eigene Kategorie vordefiniert werden. Zwar können die einzelnen Kategorien mittel *srai*-Element miteinander verknüpft werden, es stellt jedoch trotzdem eine erhebliche Redundanz dar.

```
<category>
      <pattern>_ AUTO *</pattern>
      <template>Besitzt du ein Auto?</template>
</category>

<category>
      <pattern>_ AUTOS *</pattern>
      <template><srai>AUTO</srai></template>
</category>


<category>
      <pattern>_ AUTOMOBIL *</pattern>
      <template><srai>AUTO</srai></template>
</category>
```

[32] Vgl. Wilcox 2011, S. 5.

[33] Vgl. Hermans 2008, S. 37

Das obenstehende Beispiel mit der Grundform „Auto“ verdeutlicht, wie man *Stemming* in *AIML* umsetzen kann. Eine Nutzereingabe vor dem Matching aufzugliedern ist also mittels *AIML* nicht möglich. Es können immer nur ganze Wörter verarbeitet werden, nicht aber einzelne Teile von Wörtern.

4.3.3 Reguläre Ausdrücke

Unter dem Begriff *Reguläre Ausdrücke* versteht man speziell kodierte Zeichenketten, die als Muster zum Finden von Strings genutzt werden.[34] Dabei stehen verschiedene Operatoren zur Verfügung, um einzelne Ausdrücke miteinander zu verbinden. Ein einfaches Beispiel wäre der reguläre Ausdruck „`a|r`“. Dieser Ausdruck sucht nach Zeichenketten, welche den Buchstaben *a* oder *r* beinhalten. Das Wort *Hallo* würde demnach von diesem Ausdruck gefunden werden. Der reguläre Ausdruck „`M[ae][iy]e?r`“ würde beispielsweise alle acht Varianten des Nachnamens *Meier* finden.

Eine derartige Mustererkennung lässt sich mit Hilfe von *AIML* nicht umsetzen.[35] Wie in Kapitel 4.3.2 erläutert, können einzelne Wörter nicht aufgegliedert werden - somit kann auch nicht nach bestimmten Mustern innerhalb von Wörtern gesucht werden. Die einzigen regulären Ausdrücke, welche *AIML* beherrscht, sind die in Kapitel 4.2.2 erklärten *Wildcards*. Wobei mit *Wildcards* lediglich nach ein oder mehreren Wörtern [1,n] gesucht werden kann. Die Menge [0,n] wird nicht unterstützt. Ebenso kann keine exakte Angabe über die Anzahl der vorkommenden Wörter gemacht werden. Es kann also nicht angegeben werden, dass beispielsweise nach dem Wort „Auto“ maximal drei weitere Wörter folgen dürfen.

Der reguläre Ausdruck „`M[ae][iy]e?r`“ kann in *AIML* nur durch explizite Auflistung aller möglichen Schreibweisen von „Meier“ implementiert werden. Es müssten also acht Kategorien angelegt werden. Um das Wort „Meier“ im Kontext einer Frage bzw. Nutzereingabe zu bearbeiten, müssten weitere Kategorien angelegt werden. Dass reguläre Ausdrücke von AIML nicht oder nur in einem sehr begrenzten Rahmen umgesetzt werden können, wird oftmals stark kritisiert.[36]

---

[34] Vgl. Fitzgerald 2012, S. 1.

[35] Vgl. Forum.alicebot.org 2014.

[36] Vgl. Forum.alicebot.org 2014.

5 Fazit

Die in dieser Arbeit vorgestellten Chatbotsprachen *Rivescript*, *ChatScript* und *AIML* sind die Grundlage für die Implementierung von Chatbots. Es hat sich gezeigt, dass alle drei Sprachen im Grunde dieselben Anforderungen erfüllen: Eine Nutzereingabe mit der Wissensbasis vergleichen und ggf. eine passende Antwort ausgeben. Besonders die hier untersuchte Chatbotsprache *AIML* wurde schnell an ihre *Grenzen* gebracht. Zwar verfügt *AIML* über eine sehr einfache und verständliche XML-Syntax, jedoch ist sie v.a. im erweiterten Funktionsumfang stark beschränkt. Das eingeschränkte Pattern-Matching und das Fehlen regulärer Ausdrücke bedürfen einer intensiven Überarbeitung, um den heutigen Erwartungen an ein Chatbot-System gerecht zu werden. Die enorme Leistungsfähigkeit moderner Computer wird mit *AIML* bei weitem nicht ausgereizt.

Um den heutigen Anforderungen besser gerecht zu werden, wurde deswegen Anfang 2013 ein Konzept für *AIML 2.0* vorgestellt.[37] Zu den neuen Funktion gehören demnach u.a. die Unterstützung von [0,n]-*Wildcards*, Matching-Priorisierung, Ansätze von Grundformenreduktion sowie Erweiterungen für den Einsatz auf mobilen Endgeräten.[38] Ein erster in Java implementierter *AIML-2.0*-Interpreter ist bereits auf der Website des Projekts frei verfügbar.[39] Weitere Untersuchungen und Evaluierungen dieser zweiten Version von *AIML* wären hier wünschenswert.

Insgesamt lässt sich sagen, dass der Einsatz von Chatbot-Systemen bzw. Spracherkennungs-Systemen in naher Zukunft weiter zunehmen wird - nicht zuletzt für den Einsatz bei mobilen Endgeräten und in der Automobilbranche.

Das natürlichsprachige *Reden* bzw. *Unterhalten* mit einem Computer muss also, wie im Spielfilm „Her“, nicht weiterhin nur Zukunftsmusik bleiben.

[37] Vgl. Alicebot.org 2013.

[38] Vgl. Docs.google.com 2014.

[39] Vgl. Code.google.com 2014.

Quellenangaben

Alicebot.org (2014a). AIML: Artificial Intelligence Markup Language. Retrieved from: http://www.alicebot.org/aiml.html [22.04.2014]

Alicebot.org (2014b). Dr. Richard S. Wallace. Retrieved from: http://www.alicebot.org/bios/richardwallace.html [22.04.2014]

Alicebot.org (2014c). Symbolic Reductions in AIML. Retrieved from: http://www.alicebot.org/documentation/srai.html [22.04.2014]

Alicebot.org (2014d). AIML Reference Manual – That-Element. Retrieved from: http://www.alicebot.org/documentation/aiml-reference.html#that [22.04.2014]

Alicebot.org (2014e). AIML Reference Manual – Topic-Element. Retrieved from: http://www.alicebot.org/documentation/aiml-reference.html#topic [22.04.2014]

Alicebot.org (2013). AIML 2.0 draft specification released. Retrieved from: http://alicebot.blogspot.de/2013/01/aiml-20-draft-specification-released.html [22.04.2014]

Brilligunderstanding.com (2014). Vita: Bruce Wilcox, Technology Creator. Retrieved from: http://brilligunderstanding.com/wilcoxresume.html [22.04.2014]

Chatbots.org (2014). Chatbots By Bruce Wilcox. Retrieved from: http://www.chatbots.org/developer/bruce_wilcox/ [22.04.2014]

Chatscript.sourceforge.net (2014). ChatScript - Project Information. Retrieved from: http://chatscript.sourceforge.net [22.04.2014]

Code.google.com (2014). Program-ab - Reference AIML 2.0 Interpreter. Retrieved from: https://code.google.com/p/program-ab/ [22.04.2014]

Docs.google.com (2014). AIML 2.0 Working Draft. Retrieved from: https://docs.google.com/document/d/1wNT25hJRyupcG51aO89UcQEiG-HkXRXusukADpFnDs4/pub [22.04.2014]

Engadget.com (2014). Microsoft zeigt Cortana, die Antwort auf Siri und Google Now. Retrieved from: http://de.engadget.com/2014/04/02/microsoft-zeigt-cortana-die-antwort-auf-siri-und-google-now/ [22.04.2014]

Fitzgerald, Michael (2013). Einstieg in Reguläre Ausdrücke. O'Reilly Germany Auflage 1.

Forum.alicebot.org (2014). ALICE A.I. Foundation Promoting the development and adoption of ALICE and AIML free software - Support regex nodes? Retrieved from: http://forum.alicebot.org/viewtopic.php?t=95 [22.04.2014]

Geeb, Franziskus (2007). Chatbots in der praktischen Fachlexikographie und Terminologie. LDV-Forum Band 22(1), S. 51-70. Online verfügbar unter: http://media.dwds.de /jlcl/2007_Heft1/Franziskus_Geeb.pdf [22.04.2014]

Golem.de (2010). Apple kauft Anbieter von Spracherkennungs-App Siri. Software aus der Militärforschung läuft auf iPhone und iPod touch. Retrieved from: http://www.golem.de/1004/74817.html [22.04.2014]

Hermans, Jan (2008). Ontologiebasiertes Information Retrieval für das Wissensmanagement. Logos Verlag Berlin GmbH, Auflage 1.

Latimes.com (2014). NYFF 2013: With voice-centric 'Her,' Spike Jonze makes a statement. Retrieved from: http://www.latimes.com/entertainment/movies/moviesnow/la-et-mn-spike-jonze-her-joaquin-phoenix-20131013,0,4625932.story#axzz2yUh6KBXx [22.04.2014]

Moor, James (2003). The Turing Test: The Elusive Standard of Artificial Intelligence. Springer, Auflage 1.

Pandorabots.com (2014a). Build a Chatbot For advertising, customer service, virtual assistance, education, entertainment, internet-connected devices, and more. Retrieved from: http://www.pandorabots.com [22.04.2014]

Pandorabots.com (2014b). AIML Overview By Dr. Richard S. Wallace. Retrieved from: http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html [22.04.2014]

Pandorabots.com (2014c). A Tutorial For Adding Knowledge to Your Robot What about <topic>?. Retrieved from: http://www.pandorabots.com/botmaster/en/tutorial?ch=4 [22.04.2014]

Rivescript.com (2014a). What is RiveScript?. Retrieved from: http://www.rivescript.com/rivescript [22.04.2014]

Rivescript.com (2014b). RiveScript vs. AIML Functionality Comparison. Retrieved from: http://www.rivescript.com/aiml [22.04.2014]

van Rosmalen, Peter / Eikelboom, Johann / Bloemers, Erik / van Winzum, Kees / Spronck, Pieter (2012). Towards a Game-Chatbot: Extending the Interaction in Serious Games. In: 6th European Conference on Games Based Learning, Cork, Ireland. Online verfügbar unter: http://dspace.ou.nl/bitstream/1820/4308/1/ECGBL2012-chatbot-project-v11062012.pdf [22.04.2014]

Saygin, Ayse Pinar / Cicekli, Ilyas / Akman, Varol (2001). Turing Test: 50 Years Later. Department of Cognitive Science, University of California, San Diego. Online verfügbar unter: http://crl.ucsd.edu/~saygin/papers/MMTT.pdf [22.04.2014]

Shawar, Bayan Abu / Atwell, Eric (2007). Chatbots: Are they Really Useful?. Online verfügbar unter: http://media.dwds.de/jlcl/2007_Heft1/Bayan_Abu-Shawar_and_Eric_Atwell.pdf [22.04.2014]

Spiegel.de (2011). Sci-Fi-Romanze "Her" mit Joaquin Phoenix: Computer sind die besseren Liebhaber. Retrieved from: http://www.spiegel.de/kultur/kino/her-von-spike-jonze-mit-joaquin-phoenix-scarlett-johansson-a-960028.html [22.04.2014]

Sueddeutsche.de (2010). Steve Ballmer-Interview "Anwender wollen mit ihrem Computer sprechen". Retrieved from: http://www.sueddeutsche.de/digital/steve-ballmer-interview-anwender-wollen-mit-ihrem-computer-sprechen-1.835149 [22.04.2014]

Sueddeutsche.de (2011). Netz-Depeschen Geschnatter um Authentizität. Retrieved from: http://www.sueddeutsche.de/kultur/netz-depeschen-geschnatter-um-authentizitaet-1.1131504 [22.04.2014]

Uni-Hamburg.de (2014). Fakten zu „Stella“. Retrieved from: http://www.sub.uni-hamburg.de/fileadmin/redaktion/Bibliotheken/factsheet_stella.pdf [22.04.2014]

Wilcox, Bruce (2011). Beyond Façade: Pattern Matching for Natural Language Applications. Online verfügbar unter: http://chatscript.sourceforge.net/Documentation/Pattern_Matching_for_Natural_Language_Applications.pdf [22.04.2014]

Wilcox, Bruce (2014). Writing a Chatbot For: UC Berkeley's Center for New Media ChatScript Hackathon Oct./2013. Online verfügbar unter: http://brilligunderstanding.com/writingachatbot.pdf [22.04.2014]

Weizenbaum, Joseph (1996). ELIZA - A Computer Program For the Study of Natural Language Communication Between Man and Machine. In: Communications of the ACM, Volume 9, S. 36-45. Online verfügbar unter: http://www.academia.edu/download/31085335/ElizaScript.pdf [22.04.2014]