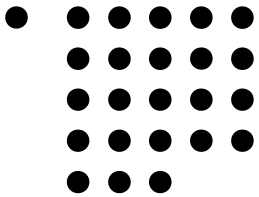

Fachhochschule Köln
University of Applied Sciences Cologne

07 Fakultät für Informations-, Medien- und Elektrotechnik, Studiengang "Master of Information Engineering"

Institut für Nachrichtentechnik
Labor für Informatik

Master-Arbeit

Thema: **Wissensakquisition und maschinelle Task-Generierung für einen virtuellen Küchenroboter.**

Student : **Henning Budde**

Matrikelnummer: 11038659

Referent : Prof. Dr. phil. G. Büchel
Koreferent : Prof. Dr.-Ing. G. Hartung

Abgabedatum : 28. Februar 2009

Hiermit versichere ich, dass ich die Master-Arbeit selbstständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

_____

Henning Budde

Inhaltsverzeichnis

1 Inhalt dieser Arbeit 6
1.1 Thema 6
1.2 Motivation 6
1.3 Ziel dieser Arbeit 7

2 Grundlagen 8
2.1 Semantic Web 8
2.2 Konzepte zur Aufbereitung von Daten 9
2.2.1 Strukturierte Daten 9
2.2.2 Semistrukturierte Daten 10
2.2.3 Unstrukturierte Daten 11
2.3 Konzept semantisch-orientierter Beschreibung 11
2.4 Konzept einer Beschreibungslogik für Web Objekte 13
2.4.1 OWL – Web Ontology Language 13
2.4.2 Aufbau der OWL 14
2.4.3 Funktionen der OWL 14
2.5 Software Agenten 16
2.5.1 Eigenschaften eines Software Agenten 18
2.5.2 Grundlegende Strukturen eines Software Agenten 20
2.5.2.1 3-Phasen Zyklus 20
2.5.2.2 Gedächtnis eines Agenten 22
2.5.2.3 Umwelt eines Agenten 22
2.5.2.4 Multi Agenten Systeme 23

3 Weltmodell, Wissensrepräsentation und Wissensakquisition 25
3.1 Weltmodell der Küche und der Kochrezepte 25
3.1.1 Aufbau eines Rezepts 25
3.1.2 ERD der Küchenwelt 26
3.1.3 Grundidee zur Verarbeitung der Arbeitsanweisung 26
3.1.3.1 Taskbezeichnung zur Beschreibung eines Schritts 27
3.1.3.2 Tasknummer zur fortlaufenden Nummerierung 28
3.1.3.3 Argumente eines Tasks 28
3.1.3.4 Referenzen 29
3.1.3.5 Das Problem der „stummen“ Referenzen 29
3.1.4 Probleme während der Analyse 30
3.2 Wissensakquisition 31
3.3 Informationsextraktion aus WorldWideWeb 31
3.3.1 Patternvergleich 34
3.4 Syntaktische Verarbeitung maschinenlesbarer Texte der deutschen Sprache 34
3.4.1 Anwendung der Satzgliedanalyse 34
3.4.2 Wortanalyse 35
3.4.3 Phrasenanalyse und Neubewertung 37
3.4.4 Protokolle 37
3.4.5 Änderungen innerhalb der Satzgliedanalyse 37

3.4.5.1 Sprachanalyse – Main Klasse der Satzgliedanalyse....38
3.4.5.2 Benutzerverwaltung der Satzgliedanalyse....38
3.4.5.3 Erstellen der Protokolldateien....38
3.4.5.4 Verzeichnisstruktur und Pakettierung....39
3.4.5.5 Datenbank....39
3.4.5.6 Die Klasse „Debug.java“....39
3.4.5.7 Änderungen innerhalb der Klassen....40

4 Task Generierung....42
4.1 Die XML-Task Sprache mit semantischer Anreicherung....42
4.1.1 Aufbau der XML-Task Sprache....42
4.2 Prozess Beschreibungs-Sprachen / Task Definition Language....45
4.2.1 Übersicht einiger Task-Sprachen....46
4.3 Task Definition Language for Virtual Agents....46
4.3.1 Konzept und Aufbau der TDL....47
4.3.1.1 Datentypen....47
4.3.1.2 Entitäten....47
4.3.1.3 Definition eines Tasks....47
4.3.1.4 Der Taskname....48
4.3.1.5 Variablen....48
4.3.1.6 Body-Block....49
4.3.1.7 Kontrollstrukturen innerhalb von Body-Blöcken....49
4.3.1.8 Erweiterung der TDL....52

5 Software Engineering....54
5.1 Software Agenten Design....54
5.1.1 Eigenschaften des Software Agenten „Agent Cook“....54
5.1.2 Informationsflüsse des Software Agenten....56
5.1.3 Modularisierung des Agenten....57
5.2 Klassendiagramme....57
5.2.1 Module der Wissensakquisition und -extraktion....57
5.2.2 Module und Erweiterungen des SAX-Parsers....58
5.2.3 Module der Satzgliedanalyse....58
5.2.4 Module der XML-Task-Sprachen Generierung....59
5.2.5 Module der TDL Generierung....59

6 Der Software Agent - „Agent Cook“....60
6.1 Beschreibung....60
6.2 Ablaufplan Agent Cook....61
6.3 GUI – Grafische Benutzeroberfläche....62
6.4 Funktionen des Agenten....63
6.4.1 Auswahl der Kochrezeptseite....63
6.4.2 Wissensakquisition eines Rezepts....64
6.4.3 Wissensextraktion aus HTML-Code....65
6.4.4 Starten der Satzgliedanalyse....67
6.4.5 Anzeigen und Korrektur der Satzgliedanalyse Resultate....67
6.4.6 Generierung eines XML-Rezepts mit Taskanweisungen....69
6.4.7 Generierung von Tasks mit Hilfe der Task Definition Language (TDL) ….75
6.4.7.1 Ablaufplan TDL-Generierung....76

6.5 Installationsanleitungen....77
7 Resultate und Ausschau....78
7.1 Analyse der Ergebnisse....78
7.1.1 Analyse der XML und TDL Dateien....78
7.1.2 Einführung von „stummen“ Tasks....79
7.2 Ausschau....82
7.2.1 Möglichkeiten zur Verbesserung der maschinellen Task Generierung....82
7.2.1.1 Einpflegen bestimmter Wörter....82
7.2.1.2 Verwalten einer Blacklist....83
7.2.1.3 Möglichkeiten zum Eingriff zur Laufzeit des Agenten....83
7.2.2 Möglichkeiten zur Weiterentwicklung und Verbesserung des Agenten....84
7.2.2.1 Wissensakquisition....84
7.2.2.2 Wissensextraktion und Nachbearbeitung der Ergebnisse....84
7.2.2.3 Satzgliedanalyse....85
7.2.2.4 Verbesserung der Mechanismen zur Erstellung der XML- und TDL-Dateien....85
7.2.3 Benutzen der Funktionalität der TDL....86
7.2.3.1 Parallelität....86
7.2.3.2 Schleifen....87
7.3 Fazit....88
8 Anhang....90
8.1 Quellenverzeichnis....90
8.2 Tabellenverzeichnis....91
8.3 Abbildungsverzeichnis....91
8.4 Beispiel Parametrierung für Chefkoch.de....92
8.5 DTD der Parameter Dateien....92
8.6 Beispiel für XML-Protokoll-Datei....93
8.7 DTD der XML-Protokoll-Dateien (Wortanalyse.dtd)....93
8.8 Beispiel Rezept in XML....93
8.9 DTD der Rezepte in XML (Rezept.dtd)....96
8.10 Rezept in TDL....96
9 Quellcode....100

1 Inhalt dieser Arbeit

1.1 Thema

Inhalt dieser Arbeit ist die Entwicklung eines Software Agenten, der auf Basis des World Wide Web Wissen akquiriert, extrahiert und dieses so aufbereitet, dass es maschinell weiter zu verarbeiten ist. Eine Task Sprache, die gesammelte Daten entsprechend aufbereitet, soll gefunden, entwickelt und beispielhaft verwendet werden. Diese Task Sprache soll als Meta-Sprache für Roboter so zu verwenden sein, dass sie in einem Roboter in ausführbaren Code umgewandelt wird und dieser die enthaltenen Arbeitsschritte mit vorhandenen Materialien und Arbeitsgeräten durchführen kann.
Wie aus dem Titel dieser Arbeit zu entnehmen ist, basiert die Akquisition, Extraktion und Task-Generierung auf dem Bereich der Küche. Kochrezepte werden als neues Wissen aus dem World Wide Web von verschiedensten Anbietern heruntergeladen, Zutaten und Zubereitungsschritte extrahiert und als XML beziehungsweise in einer Task Definition Language aufbereitet.
Der Software Agent wird in der Programmiersprache JAVA geschrieben. Die Teile eines Kochrezepts, die sprachlich untersucht werden müssen sollen durch eine vorangegangene Diplomarbeit zum Thema „Satzgliedanalyse“ untersucht werden, so dass die Sprachanalyse in den Software Agenten integriert werden muss.

1.2 Motivation

Das Thema Wissensakquisition und Wissensrepräsentation ist bei den Entwicklern des World Wide Web in aller Munde. In Zeiten von stark wachsenden Inhalten im Internet wird es immer wichtiger Wissen und Informationen zu sammeln und entsprechend aufzubereiten. Durch den sogenannten „User Generated Content“ werden täglich so viele neue Informationen und neues Wissen generiert, dass es Menschen allein unmöglich ist dieses alles aufzuarbeiten und in eine geordnete Form zu bringen, die für jeden Menschen, aber eben auch für Maschinen zugänglich und lesbar sein sollen. Die Anreicherung von Wissen mit semantischen Informationen wird eines der großen Themen in den nächsten Jahren sein und wird als nächste große Stufe in der Weiterentwicklung des World Wide Web gesehen. Auch Tim Berners Lee, der Begründer des Webs und Erfinder von HTML will das Web in diese Richtung lenken.

Das Thema Küche, welches in dieser Arbeit beispielhaft verwendet wird, repräsentiert einen großen Teil an unterschiedlichen Arbeitsanweisungen, die ein Roboter für einen Menschen übernehmen soll.
Ein weitere Fragestellung dieser Arbeit ist die Möglichkeit des vollständigen Parsens der deutschen Sprache, denn nur eine vollständig von Computern verstandene und auf Korrektheit überprüfte Sprache kann zu einem korrekten Ergebnis in der maschinellen Task Generierung führen.

1.3 Ziel dieser Arbeit

Ziel dieser Arbeit ist es den Weg der Wissensrepräsentation voranzutreiben und neue Wege zu beschreiten, die nicht nur für die Küchenwelt eine relevante Bedeutung haben sondern auch auf viele andere Themengebiete übertragbar sind. Dabei soll das Wissen als Resultat so präsentiert werden, dass es genügend semantische Informationen enthält um dieses maschinell weiter zu verarbeiten.
Ein weiteres Ziel ist es einen möglichst autonomen Software Agenten zu implementieren, der alle geforderten Aufgaben möglichst selbstständig und korrekt ausführt und sein Ergebnis in einer Task Sprache erzeugt, die als Meta-Sprache für einen Roboter weiter zu verwenden ist.

Hinweis:
Alle Quellen werden in dieser Arbeit in eckigen Klammern und fortlaufender Nummerierung angegeben. BSP: Quelle: [1]

2 Grundlagen

2.1 Semantic Web

Das Semantic Web ist im Moment der Begriff, wenn es darum geht in welche Richtung sich das Internet in den nächsten Jahren entwickeln wird. Oft wird es als Erweiterung des derzeitigen Internets bezeichnet, da die derzeitige „Version“ durchaus auch so bestehen bleiben soll. Ziel des Semantic Webs ist es den Inhalten des Netzes eine Bedeutung zuzuordnen. Diese Bedeutung soll so aufgearbeitet und präsentiert werden, dass die Inhalte maschinell von Computern zu begreifen und zu interpretieren sind und zur Lösung von Problemstellungen herangezogen werden können. Ziel ist es darüber hinaus sämtliche Informationen miteinander zu vernetzten und daraus ein riesiges Informationsnetz zu bilden. Das Semantic Web wird häufig auch als Web 3.0 bezeichnet, da dies der nächste große Entwicklungsschritt nach der Entwicklung hin zu „User Generated Content“ und „Social Networking“, die derzeit als Web 2.0 bezeichnet werden, sein wird. Offiziell sind weder Web 2.0 noch Web 3.0 eingeführte Begriffe, aber selbst Web-Erfinder Tim Berners-Lee akzeptiert diese Begriffe, sieht sie aber kritisch, da er das Web seit der Entstehung schon immer in der Entwicklung dorthin gesehen hat.
Im Semantic Web werden Informationen mit weiteren Informationen gekennzeichnet. Dies wird häufig als Annotation bzw. das Versehen von Daten mit Metadaten bezeichnet. Die grundsätzliche Technik so etwas durchzuführen gibt es schon lange. Ein Beispiel dafür ist die Einführung von XML Ende der 90er Jahre (XML: Recommendation 1 vom 10.02.1998; Aktuell: Recommendation 5 vom 26.11.2008). XML repräsentiert Daten strukturiert und erlaubt über Attribute Daten mit zusätzlichen Informationen zu versehen. XML selbst wurde so frei gestaltet, dass Informationen sehr allgemein gespeichert werden und Datentypen für Informationen nicht vorgesehen sind. Die Zeit brachte Erweiterungen zu XML und auch völlig neue Ansätze.
Berners Lee versucht das Web für die Möglichkeit einer semantischen Suchmaschine vorzubereiten. Er möchte weg von der einfachen zeichenbasierten Suchmaschine, die über eine Stichwort Suche funktioniert hin zu einer Maschine die Antworten auf gestellte Fragen geben kann.

BSP: Die Strecke Frankfurt - Köln

- Suchmaschine mit Stichwortsuche
  Gibt man heute in eine Suchmaschine die Suchanfrage „Länge der Strecke Frankfurt – Köln“ ein so bekommt man nur eine Auswahl an Links die die Stichworte der Suche beinhalten.
- Semantische Suchmaschine
  Gibt man die selbe Suche in Zukunft in eine semantische Suchmaschine ein, so soll als Ergebnis die Länge der Strecke zwischen Frankfurt und Köln als Resultat heraus kommen. Natürlich können darüber hinaus auch Informationen zu Köln und Frankfurt, zu Reisemöglichkeiten oder zur Routenplanung geliefert

werden.
Schon heute gibt es für den deutschen Raum eine semantische Suchmaschine: www.semager.de. Diese liefert aber trotzdem keine direkte Antwort auf die Frage sondern Verweise auf Seiten, die mit höhere Wahrscheinlichkeit eine Antwort auf die gestellte Frage liefern könnten.

Google bemüht sich auch heute schon direkte Antworten auf Fragen zu liefern. Sucht man zum Beispiel nach der „Höhe des Mount Everest“ so ist die erste Antwort:

Mount Everest – Höhe: 8.850m

Bis es soweit ist, dass einfache Fragen immer zuerst mit einer Antwort und nicht mit Verweisen auf Webseiten beantwortet werden ist noch ein längerer Weg zu beschreiten. Tim Berners Lee beschreibt das zu erreichende Ziel so:

„Das Internet wäre dann weitaus mehr als die Summe seiner Dokumente, es wäre sozusagen ein globales Gehirn.“
Tim Berners Lee

Quelle: [1] Internetquelle: Social Media Blog
Quelle: [8] Internetquelle: W3C – XML

2.2 Konzepte zur Aufbereitung von Daten

Um solch eine semantische Suchmaschine zu starten müssen Konzepte zur Aufbereitung von Daten durchgesetzt werden.

A. Das Konzept strukturierter Daten
B. Das Konzept semantisch-orientierter Beschreibung von Web Ressourcen
C. Das Konzept einer Beschreibungslogik für Web-Objekte

Man unterscheidet drei Arten von strukturierten Daten:
strukturierte Daten
semistrukturierte Daten
unstrukturierte Daten

2.2.1 Strukturierte Daten

Kennzeichen strukturierter Daten
(a) Daten sind in Entitäten gegliedert
BSP: Tabellenzeilen in Datenbanken
Datensätze in Dateien
Instanzen persistenter Klassen

(b) Gliederung der Entitäten in Attributen und Beschreibung durch ein kontrolliertes Vokabular

(c) Zuordnung von Datentypen zu allen Attributen und Entitäten

BSP: Tabellenzeilen in Datenbanken

$T = \{A_i, dt_i, W_i\}$

mit:

T: Tabelle einer Datenbank

A_i: i-ter Spaltenname (i-tes Attribut) einer Tabelle

dt_i: Datentyp des i-ten Attributs einer Tabelle

W_i: Wertebereich des i-ten Attributs

Auch die Tabelle T selbst bekommt durch die Beschreibung der einzelnen Attribute automatisch einen eigenen Datentyp zugewiesen.

$$dt(T) = \text{TUPLE OF } (A_1 : dt_1; A_2 : dt_2; \ldots ; A_n : dt_n)$$

2.2.2 Semistrukturierte Daten

Daten werden als semistrukturiert bezeichnet, wenn für sie zwei der drei Kennzeichen von strukturierter Daten gilt. Sie müssen in Entitäten angeordnet und als Attribute eines kontrollierten Vokabulars gegliedert sein. Semistrukturierte Daten besitzen keine Datentypen.

```
+    in Entitäten untergliedert
+    Beschreibung in Attributen durch
ein   kontrolliertes Vokabular
-    Attribute besitzen Datentypen
```

Ein Beispiel für semistrukturierte Daten sind XML-Dateien. In ihnen werden Daten in Entitäten und Attribute gegliedert.

BSP : Adressdaten

```
<ADRESSDATEN>
 <PERSON>
  <NAME>Mustermann</NAME>
  <VORNAME>Max</VORNAME>
  <STRASSE>Musterweg 5</STRASSE>
  <PLZ>55555</PLZ>
  <ORT>Koeln</ORT>
 </PERSON>
 <PERSON>
  <NAME>MÜLLER</NAME>
  ...
 </PERSON>
</ADRESSDATEN>
```

Hinweis: Die semistrukturierten Daten einer XML-Datei lassen sich mit Hilfe von XML-S auch als strukturierte Daten ablegen.

2.2.3 Unstrukturierte Daten

Als unstrukturierte Daten bezeichnet man Daten bzw. Datensätze für die keine der Eigenschaften strukturierter Daten gilt oder aber auch Datensätze, die nur in Entitäten unterteilt sind aber nicht in Attributen gegliedert sind. Ein Beispiel hierfür sind HTML-Dateien (Hyper Text Markup Language). In HTML-Dateien werden Daten gesichert, die von HTML-Tags umklammert sind, also in Entitäten zusammen gefasst werden, aber nicht alle Daten werden zusätzlich in Attribute oder Elemente gegliedert.

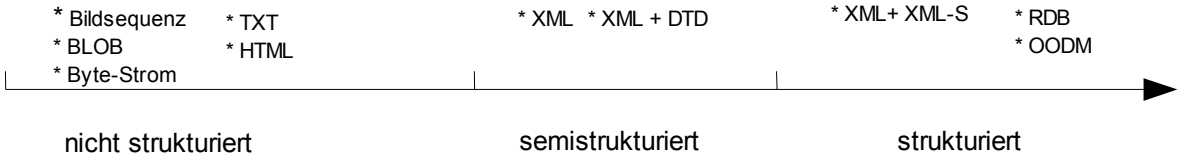

Abbildung 1: nicht strukturierte zu strukturierte Daten

2.3 Konzept semantisch-orientierter Beschreibung

Um die Struktur einer Webseite so zu erstellen, dass auf ihre verschiedenen Entitäten direkt zugegriffen werden kann ist es nötig Stellen genauer zu kennzeichnen. Damit dies möglich ist, wurde das Konzept der URL (Unified Ressource Location) erweitert. Diese Erweiterung heißt Unified Ressource Identifier (kurz: URI[1]). Mit dem URI Konzept ist es möglich innerhalb einer HTML Seite direkt an eine Stelle zu springen und nur

1 URI: RFC2396 – „Unified Ressource Identifiier – Generic Syntax“ (http://www.ietf.org/rfc/rfc2396.txt)

gewünschte Daten auszulesen. Für Software Agenten ist das eine wichtige Funktion, da diese meist nicht an Bildern oder zusätzlichen Texten interessiert sind sondern nur auf Grund einer bestimmten Information die Seite verarbeiten möchten.
Noch nicht ausreichend ist das URI Konzept wenn Metadaten, also zusätzliche Informationen, an Daten gehaftet werden sollen. Metadaten werden als zusätzliche Ressource im Dateisystem hinterlegt. Sie müssen im Ressource Description Framework (RDF) Format erstellt werden.
In RDF werden Ressourcen durch Eigenschaften (Properties) mit Namen und Werten beschrieben. Die Namen unterliegen wiederum einem kontrollierten Vokabular. In RDF wird jede Ressource R durch eine Property P beschrieben, die einen Wert W hat oder auf eine andere Ressource W verweist.

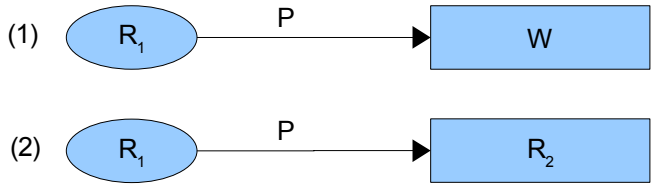

Abbildung 2: RDF-(1) Ressource hat Wert W /(2) Ressource verweist auf Ressource

Um das kontrollierte Vokabular selbstständig zu erweitern wird wiederum ein zusätzlicher Standard benötigt. Dieser Standard heißt Ressource Description Framework – Schema (RDFS). In ihm können neue Ressourcen angelegt und genau beschrieben werden, welche Werte diese Ressourcen annehmen dürfen. Zur Beschreibung dieser Werte verwendet man dafür die Begriffe Domain (≙ Definitionsbereich) und Range (≙Wertebereich).

Domain und Range für Adressdaten-Beispiel:
Die „Domain“ für eine Person kann folgende Domain-Werte annehmen:

- NAME
- VORNAME
- STRASSE
- PLZ
- ORT

Die „Range“ für Personendaten werden zum Beispiel über primitive Datentypen angeben:

- Range(NAME) ϵ String
- Range(VORNAME) ϵ String
- Range(STRASSE) ϵ String
- Range(PLZ) ϵ Integer
- Range(ORT) ϵ String

In XML-S sind eine Vielzahl von verschiedenen Datentypen vordefiniert.

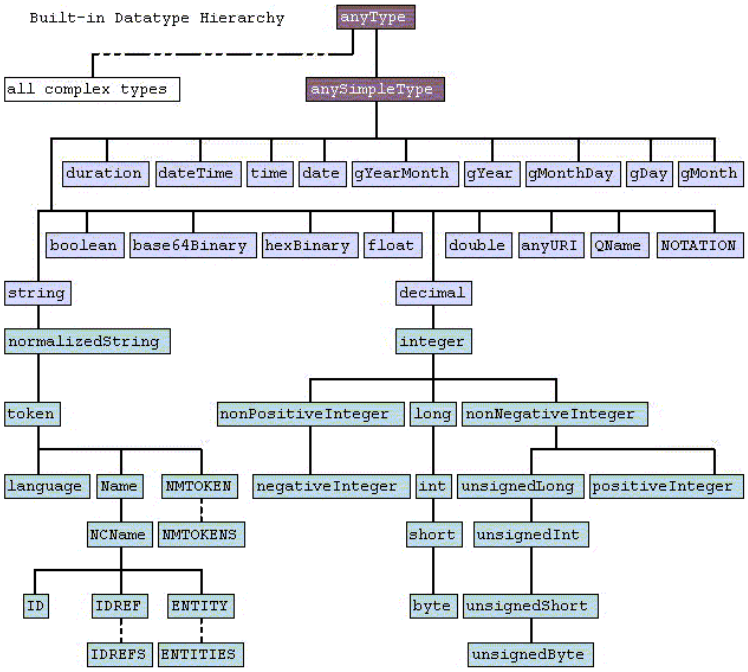

Abbildung 3: Definierte XML-S-Datentypen

Quelle: [2] Internetquelle: W3C - XML-Schema

Diese Datentypen können über einen URI direkt angesprochen werden. Der Datentyp integer kann beispielsweise über folgenden URI angesprochen werden:

```
http://www.w3.org/2001/XMLSchema#integer
```

2.4 Konzept einer Beschreibungslogik für Web Objekte

2.4.1 OWL – Web Ontology Language

Die **W**eb **O**ntology **L**anguage ist ein weiterer Schritt hin zu den von Tim Berners Lee geforderten Veränderungen für das Web. Hier sind wirkliche Beziehungen zwischen Daten möglich. Obwohl die Web Ontology Language eigentlich mit WOL abgekürzt werden müsste, entschieden sich die Entwickler für die Bezeichnung OWL. Eine wirkliche offizielle Begründung warum dies so ist wurde nicht getätigt, aber im Web findet man des öfteren folgende Gründe:

1. OWL ist einfach und leicht auszusprechen (in Englisch wie die englische Eule)
2. die Eule zeugt von Weisheit und würde gut als eine Art Maskottchen für die Wissensrepräsentation stehen
3. eine vorhergehende Entwicklung einer Sprache zur Wissensrepräsentation mit dem Name „One World Language“

Quelle: [3] Internetquelle: W3C – OWL
Quelle: [10] Internetquelle: Wikipedia - OWL

2.4.2 Aufbau der OWL

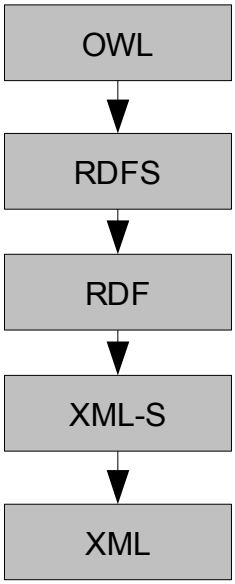

Abbildung 4: Aufbau von XML zu OWL

Die OWL baut auf vorhandenen Sprachen zur Sicherung von Daten auf. Zunächst entstand XML, das durch XML-Schema (XML-S) erweitert wurde. Aber auch XML-Schema genügte nicht um Daten sinnvoll zu strukturieren. RDF setzte als nächste Stufe auf XML und XML-S auf. Einfache Bezüge von Daten und Informationen waren nun möglich. Komplexere Abhängigkeiten konnten aber noch nicht beschrieben werden. Dafür wurde RDFS hinzugefügt, das ein einfaches Klassensystem mit sich bringt. OWL ist bis heute der Baustein der „on Top“ auf das Gerüst aufgesetzt wurde.

2.4.3 Funktionen der OWL

Ein kurzer Überblick über Funktionen von OWL

- Subklassen/Unterklassen
  Diese sind schon in RDFS vorhanden, aber in OWL um Eigenschaften erweitert,

die eine genauere Beschreibung zulassen, so dass Klassen zum Beispiel auch gleichzeitig Entitäten anderer Klassen sein können.

- Kardinalitäten
  Wie in Datenbanken können Kardinalitäten gesetzt werden, so dass genau spezifiziert werden kann wie viele Entitäten eine Klasse besitzen darf.
  BSP:
  1 Woche hat 7 Tage -> eine Woche darf maximal 7 Entitäten der Klasse Tage besitzen

- Zusammenhalten einer Entität mit unterschiedlichen Bezeichnungen
  Stellt Beziehung zwischen zwei unterschiedlichen Bezeichnern her, die aber auf die selbe Entität hinweisen.
  BSP: Handy <=> Mobiltelefon

- Unterscheidung von Entitäten
  Stellt eindeutigen Unterschied zwischen zwei Entitäten dar.

- Funktionale Beziehung
  Von einer Entität kann auf eine oder mehrere Entitäten geschlossen werden.
  BSP: Ausgehend von einer Mutter kann auf ihre Kinder geschlossen werden.

- Inverse Funktionale Beziehung
  Der Rückschluss zu einer funktionalen Beziehung.
  BSP: Von der Mutter kann auf die Kinder geschlossen werden, aber umgekehrt auch von den Kindern auf die Mutter.

- Symmetrische Beziehung
  Beschreibt zwei ähnliche Entitäten die gleiche Eigenschaften haben
  BSP: Mutter hat zwei Kinder, Kinder haben beide gleiche Mutter -> Kinder sind symmetrisch zueinander

- Transitive Beziehung
  Beschreibt die Beziehung einer Entität zu seinen Vorgängern oder Nachkommen
  BSP: Großmutter hat eine Tochter die wiederum Mutter eines Sohnes ist -> Mutter = Tochter je nachdem von welcher Entität die Sicht ausgeht

Es existieren zur Zeit drei verschiedene Dialekte der OWL

Dialektname	Beschreibung
OWL Lite	Für einfache Ontologien, einige der zuvor beschriebenen Funktionen stehen nicht zur Verfügung
OWL DL	Wird häufig eingesetzt, benutzt aber nicht den vollen, komplexen Umfang der OWL

Dialektname	Beschreibung
	(BSP: Klassen dürfen nicht Entitäten einer anderen Klasse sein)
OWL Full	Volle Unterstützung der OWL Syntax und aller Konstrukte

Tabelle 1: OWL Dialekte

Die OWL liegt derzeit in einer Version vom 10.02.2004 vor.

Quelle: [4] Internetquelle: W3C – OWL-Semantics
Quelle: [9] Datenbanken und Wissensrepräsentation

2.5 Software Agenten

Definitionen für einen Software Agenten:

> „Ein Software-Agent ist ein längerfristig arbeitendes Programm, dessen Arbeit als eigenständiges Erledigen von Aufträgen oder Verfolgen von Zielen in Interaktion mit einer Umwelt beschrieben werden kann."
> [Handbuch der KI, G. Görz, C.-R. Rollinger, J. Schneeberger (Hrsg.), 4. Auflage, Oldenbourg 2003, S.951]

> „Ein Agent bezeichnet in der Regel ein Programm, dem eine gewissen Eigenständigkeit bei der Ausführung von „Aufträgen" zugeschrieben wird".
> [Handbuch der KI, G. Görz, C.-R. Rollinger, J. Schneeberger (Hrsg.), 4. Auflage, Oldenbourg 2003, S.944]

Für einen Software Agenten findet man diverse unterschiedliche Definitionen, die die Strukturen und Aufgaben eines solchen Programms beschreiben. Grundsätzlich soll ein Software Agent automatisch und selbstständig Aufgaben für den Menschen übernehmen. Vergleicht man den Software Agenten mit einem Roboter, so ist der Roboter zuständig für motorische und sensorische Aufgaben in der realen Welt, also den materiellen Dingen, die man mit den Sinnen erfassen kann.
Innerhalb dieses Kapitels wird häufiger der Begriff des Roboters benutzt und zum Vergleich herangezogen. Roboter stehen hierbei für Maschinen, die im Gegensatz zu Agenten nicht nur kognitive Fähigkeiten besitzen, sondern auch physische Arbeit verrichten können. Ein Software Agent ist für die virtuelle Welt zuständig. Auch dort muss er Dinge beschaffen, wie Anfragen an einen Webserver zum Senden einer HTML

Seite oder aber die sensorische Erfassung seiner virtuellen Umwelt und bestückt diese Informationen mit zusätzlichen Daten.
Agenten besitzen dabei zwei unterschiedliche Aktivitätsformen:

- Wissensaktualisierung
  Hierbei kommuniziert der Agent mit Mensch oder der virtuellen Welt. Ein Mensch kann ihm neue Regeln zur Wissensakquirierung oder -extraktion beibringen. Automatisch kann er sich durch virtuelle Welten bewegen um dort sein Wissen zu erweitern oder altes Wissen zu überprüfen, beziehungsweise neu zu bewerten. Ein Software Agent kann dabei auch mit einem anderen Software Agenten in Kontakt treten um Wissen zu erfragen. Ein Agent zum Buchen einer Reise könnte als Beispiel mit Agenten von verschiedenen Reisebüros kommunizieren um Preise und Termine für eine ausstehende Reise zu erfragen.
  Auch zur Wissensaktualisierung gehört die Aktualisierung der Liste der Agenten mit der ein Agent in Kontakt steht, d.h. er muss überprüfen, existiert ein anderer Agent noch oder sendet ein ihm bekannter Agent Informationen über einen neuen Agenten. Auch ein Benutzer des Agenten kann ihn manuell mit neuen Agenten Informationen bestücken.

- Kundendialog
  Im Kundendialog erfragt der Agent Fakten vom Kunden, so dass er seine nächste Suche genau nach den Wünschen des Kunden anpassen kann und die Ausgabe auf nur gewünschte Informationen filtert. Im Beispiel des Reisebüro Agenten würde er bevor er sich auf die Suche nach Angeboten macht, Informationen zur Reise wie Ort, Datum, Anzahl Personen, etc. vom Kunden erfragen und seine Suche nur gezielt nach diesen Kriterien starten.
  Höher entwickelte Agenten betrachten den Kundendialog nicht nur als Eingabeaufforderung von Fakten und dem späteren Darstellen der Suchergebnisse, sondern versuchen den Dialog auf zwischenmenschliche Kommunikationsformen abzubilden. Hierfür kann ein Kunde zum Beispiel anhand von Kundendaten analysiert werden und spezielle Anforderungen individuell erstellt werden. Aktuell werden Raster entwickelt die es erlauben Kunden in grobe Kategorien einzuteilen. Der momentane Stand der zwischenmenschlichen Kommunikation beschränkt sich aber eher auf Erscheinungsform der Ergebnispräsentation, das Ausdrücken von Emotionen über die geschriebene Sprache oder einfache kleine Symbole und Zeichnungen.
  Ein Beispiel hierfür ist das Ausdrücken von Bedauern bei nicht möglicher Erfüllung eines Kundenwunsches, wenn beispielsweise alle Reisen im eingeschränkten Suchkreis ausgebucht sind.
  Die Google Suchmaschine versucht ihre Ergebnisse auch möglichst an den jeweiligen Kunden anzupassen und sammelt deswegen für jeden Kunden seine bisherigen Suchanfragen und versucht diese in die Auswertung der aktuellen Suchanfrage mit zu integrieren.

Quelle: [5] Handbuch der KI S. 948

2.5.1 Eigenschaften eines Software Agenten

Ein Software Agent kann durch zahlreiche Eigenschaften genauer spezifiziert und beschrieben werden. Diese Beschreibungen sind dem Handbuch der KI entnommen und sollen nur als Dimensionen zu verstehen sein, was heißt, dass diese nur eine grobe Beschreibung des Systems aufzeigen sollen. Einige dieser Eigenschaften überschneiden sich stark und lassen sich nicht genau auseinander differenzieren:

- Andauernde Verfügbarkeit
  Ist der Agent 24 Stunden / 7 Tage die Woche zu erreichen bzw. arbeitet er in dieser Zeit selbstständig weiter oder wird der Agent nur auf Bedarf eingeschaltet, erledigt die zu bearbeitenden Aufträge und wird danach wieder beendet.

- Interaktion mit einer Umwelt
  Interagiert der Agent auf Grund seiner Umwelt oder verhält er sich immer gleich unabhängig von seiner gegebenen und veränderlichen Welt.

- Situiertheit
  Ähnlich der Interaktion mit der Umwelt, aber betrachtet den Aspekt genauer ob der Agent sein Verhalten als direkte Reaktion auf Umwelteinflüsse verändert.

- Eigenständigkeit
  Ist der Agent autark und kann sich selbstständig und ohne neue Befehle in einer virtuellen Umwelt bewegen, seine Dienste auftragsgemäß erledigen oder wartet er darauf, dass ein Nutzer ihn mit Informationen für einen neuen Auftrag versorgt.

- Reaktivität
  Reagiert der Agent unmittelbar auf Umweltereignisse oder wartet er auf manuelle Anpassungen

- Reaktives Verhalten
  Wie wird Reaktionsverhalten bestimmt? Wird sein Verhalten durch eine Turing Maschine bestimmt oder werden einfache Zuordnungen in Tabellen getätigt, die besagen auf Ereignis A folgt Zustand B

- Zielgerichtetheit
  Arbeitet der Agent langfristig auf ein Ziel hin oder wird er für jeden Auftrag manuell gestartet bzw. mit neuen Informationen bestückt.

- Pro-Aktivität
  Ähnlich der Zielgerichtetheit mit Betonung auf „Eigeninitiative“ des Agenten.

- Intelligenz
  Wie stark ist die Intelligenz des Agenten ausgeprägt? Kann der Agent sich neues Regelwissen selbstständig beibringen? Erfolgt eine Neubewertung von Sachverhalten nach neuen Erkenntnissen?

- Rationalität
  Führt der Agent seine Aufgaben auch unter Berücksichtigungen von eigenen beschränkten Ressourcen aus? Überprüft er Zeitvorgaben?

- Komplexität
  Sind dem Agenten einfache Zustände zugeordnet (z.B. in Tabellenform) oder muss er komplexe Durchläufe vollziehen, da er anhand von vielen Variablen eine Entscheidung treffen muss.

- (Persistente) Zustände
  Besitzt der Agent ein Gedächtnis in dem Informationen über vergangene Ereignisse, Ziele und Pläne gesichert werden. Persistenz beschreibt dabei den Vorgang der langfristigen Speicherung, zum Beispiel über das Programm Ende hinaus.

- Lernfähigkeit
  Passt ein Agent seine Fähigkeiten oder Entscheidungsprozesse an die Umwelt und lernt dauerhaft für neue Situationen.

- Mobilität
  Ist eine Migration auf eine andere Plattform/Rechner möglich?

- Kooperation
  Kommuniziert und kooperiert der Agent mit anderen Agenten oder Menschen?

- Wohlwollen (Benevolenz)
  Führt der Agent Aufträge in Sinne des Menschen oder anderen Agenten aus?

- Soziales Verhalten
  Hält sich ein Agent an Regeln, Normen und Protokolle egal ob im Umgang mit Menschen oder mit anderen Agenten?

- Emotionales Verhalten
  Ist der Agent auf emotionales Verhalten trainiert/programmiert um zur Verbesserung der Interaktion mit Menschen beizutragen.

- Glaubwürdigkeit
  Ist das emotionale Verhalten glaubwürdig oder wirkt es maschinell aufgesetzt.

Quelle: [5] Handbuch der KI, S. 951ff

2.5.2 Grundlegende Strukturen eines Software Agenten

2.5.2.1 3-Phasen Zyklus

Die Arbeitsschritte, die ein Agent durchläuft werden typischerweise in 3 Phasen unterteilt, die sich zyklisch wiederholen, je nachdem ob der Agent sich im Dauerbetrieb befindet oder ob er manuell vom Benutzer gestartet wird.
Folgende drei Phasen lassen sich dabei beobachten:

- Phase der Informationsaufnahme
- Phase der Wissensverarbeitung und Entscheidung
- Phase der Aktionsausführung

Diese Phasen können nicht nur auf Software Agenten angewendet werden, sondern auch auf Roboter, die einen ähnlichen Lebenszyklus durchlaufen.

Phase der Informationsaufnahme
In dieser Phase erhält der Agent seine Informationen. Dabei kann er diese durch manuelle Eingaben eines Benutzers, durch automatisches Sammeln im Internet, durch andere Agenten oder auch durch externe Sensoren erhalten. Aber auch die Beobachtung seiner eigenen Handlung kann ihm als Informations- und Wissensquelle dienen. Informationen können neue Regeln, Sensordaten, neue Aufträge oder auch neues Wissen sein.

Phase der Wissensverarbeitung und Entscheidung
In dieser Phase aktualisiert ein Agent sein Wissen mit Hilfe der in der Phase der Informationsaufnahme neu gewonnenen Informationen. Anhand der neuen Daten analysiert und bewertet der Agent die Situation, die bestehenden und die frisch eingetroffenen Aufträge neu. Entscheidungen über aktuelle und zukünftige Aktionen werden getroffen.

Phase der Aktionsausführung
Aktionen werden ausgeführt und Aufträge abgehandelt. Diese Phase kann daraus bestehen, dass ein Agent einen neuen Auftrag generiert, Berechnungen durchführt, Ergebnisse präsentiert oder Nachrichten an Menschen oder andere Agenten verschickt.

Auch wenn diese Phasen eigentlich eher zyklisch ablaufen, kann man diese auch als nebenläufig betrachten. Eine Realisierung auf einem Mehrprozessor System auf dem jede dieser Phase in einem eigenen Prozessor nebenläufig laufen ist möglich. Trotzdem ist es bei Software Agenten meist so, dass die Phasen nacheinander und streng zyklisch ablaufen. Dieser Zyklus wird auch als „observation - thought - action“-Zyklus bezeichnet.

„Observation – thought – action“ - Zyklus
In der Phase „Observation“ (dtsch: Überwachung) nimmt der Agent über seine Sensoren (Tastatur, Netzwerk, Scanner, Maus, etc.) Daten S aus einer Menge *sensoryinputs*. Die

eingehenden Daten S verarbeitet der Agent zu „Wahrnehmungen“ W aus der Menge vieler Wahrnehmungen (engl. Perceptions).

sense: sensoryinputs → perceptions

Aus diesen Wahrnehmungen kann der Agent nun mit Hilfe seines internen Zustands (internal state) in einen neuen Zustand in die Phase „thought“ (dtsch: denken) übergehen. Dieser Schlussfolgerungsprozess kann mit folgender Funktion beschrieben werden:

thought: internalstates x perceptions → internalstates

Einer „thought“-Phase können sich weitere „thought“-Phase anschließen um so zu einem zunächst vorläufigen Ergebnis zu kommen. Über den Zustand in dem sich der Agent gerade befindet, wird in jeder dieser Phasen neu entschieden und vergangene Zustände können über ein Gedächtnis über zukünftige Zustände mit entscheiden.

Nachdem die „thought“-Phase vorerst beendet wurde, geht der Agent in die Phase des Agierens über. Er schließt in dieser Phase aus seinem letzten Zustand und seinen letzten Informationen wie er nun schlussendlich handeln muss.

act: internalstates → actions

Wie die Aktionen den Zuständen zugeordnet sind, kann von Agent zu Agent unterschiedlich sein. Er könnte direkt aus seinem letzten Zustand eine Aktion ableiten, zum Beispiel aus einer Tabelle oder aber aus der Betrachtung seiner vorhergehenden Zustände einen neuen Zustand erreichen und daraus eine direkte Aktion ableiten.

Die folgende Darstellung zeigt die Handlungen eines Agenten zur Lebenszeit

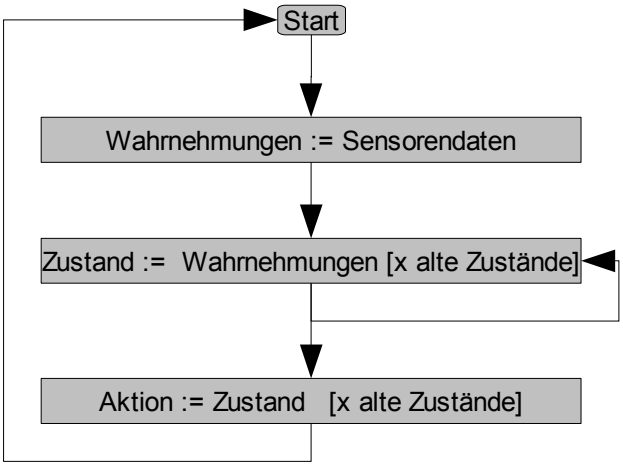

Abbildung 5: Handlungen eines Agenten zur Lebenszeit

2.5.2.2 Gedächtnis eines Agenten

Zustände in denen sich ein Agent befindet können unterschiedlich komplex implementiert und kombiniert werden. Dabei ist es für die Komplexität von großer Bedeutung ob der Agent über ein Gedächtnis verfügt oder ob neue Zustände unmittelbar aus eingegangenen Sensordaten ermittelt bzw. berechnet werden.
Informationen müssen daher je nach Komplexität persistent über einen Zeitraum (Länge des Zeitraums ist von Aufgabe des Agenten abhängig) gesichert werden. Dabei kann die Länge der persistenten Sicherung in zwei Zeiträume unterschieden werden:

1. langfristige Informationssicherung
   Wichtig um Kundeninformationen, Artikeldaten, angesammeltes Wissen, neues Regelwissen und ähnliches zu sichern.

2. kurzfristige Informationssicherung
   Wichtig um aktuelle Sensordaten, die letzten Zustände, Informationen über den laufenden Dialog mit Menschen oder anderen Agenten zu sichern.

Wie und wo ein Agent seine Informationen sichert ist je nach Aufgabe unterschiedlich zu implementieren. Für Kundenstammdaten oder Artikeldaten empfiehlt sich meist ein Datenbank Management System (DBMS), da dies große Mengen an unterschiedlichen Datensätzen aufnehmen kann und diese für den Agenten schnell und direkt über ein Netzwerk zugreifbar sind.
Zur Sicherung von internen Zuständen eines Agenten wäre ein DBMS viel zu überdimensioniert. Hier empfiehlt es sich Zustandsvariablen im Agenten selbst zu implementieren.
Ergebnisse (auch Zwischenergebnisse) können wiederum in einem DBMS oder im Filesystem dauerhaft gesichert werden.

2.5.2.3 Umwelt eines Agenten

Die Umwelt eines Agenten bezeichnet die für ihn über seine Sensoren wahrnehmbaren Informationen. Für den Agenten entsteht dadurch eine für ihn „relevante Welt“, die genau auf seine Bedürfnisse hin angepasst ist.
Die Steuerung eines Roboters ist an die materielle Welt mit Hindernissen, Wänden, Neigungen, Höhenunterschieden und Strukturen auf denen er sich bewegt gebunden. Für einen Software Agenten sollten ähnliche Bedingungen gelten, da auch dieser sich nur in seiner „relevanten Welt“ zurechtfinden kann und soll. Im Detail bedeutet das, dass einem Agenten der sich im Internet bewegt, genau beschrieben werden muss, wohin er sich bewegen soll (genaue Angabe einer URL) und wie weit er sich in dieser Welt bewegen kann und darf. Hierbei muss er wissen, ob er weitere Links auf einer Seite verfolgen und wenn, wie viele Links in welcher Tiefe er verarbeiten soll? Auch die Struktur einer WWW Seite muss dem Agenten beschrieben werden und Angaben gemacht werden, wo seine „Trennwände“ sind und wie diese aussehen (Beispiel: HTML Tags).
Da sich eine Umwelt immer wieder verändern kann, sollte ein Agent je nach Komplexität auch in der Lage sein, sich an seine Welt anzupassen. Dabei ist es nicht

unbedingt von Nöten, dass sich ein Agent automatisch und selbstständig an die Umwelt anpasst, die Möglichkeit zur manuellen Anpassung sollte aber in jedem Fall gegeben sein.
Um die Umwelt eines Agenten zu beschreiben, ist es sinnvoll den Begriff „Annahmen“ (engl.: belief) zu benutzen um klar zustellen, dass diese Annahmen kein konkretes Wissen (engl. knowledge) darstellen und sich durchaus verändern können. Für den Agenten entwickelt sich ein internes Abbild der Umwelt aus einprogrammiertem Wissen und den über seine Sensoren ermittelten Daten.

Quelle: [5] Handbuch der KI, S.977,978

2.5.2.4 Multi Agenten Systeme

Agenten können ähnlich wie Roboter auch gemeinsam versuchen Ziele zu erreichen und gegebene Aufträge ausführen. Ein deutlicher Unterschied dabei ist, dass Roboter nicht nur auf den Austausch von Nachrichten bzw. Wissen beschränkt sind, sondern sie können auch gemeinschaftlich physische Arbeit verrichten. Ein weiterer großer Unterschied ist, dass Roboter zwangsweise immer die gleichen Ziele verfolgen und nicht „gegeneinander“ arbeiten. Software Agenten die als Multi Agenten System arbeiten können theoretisch gegeneinander arbeiten, wenn sie jeweils für ein Zwischenziel unterschiedliche Wege gehen und dort zu unterschiedlichen Ergebnissen kommen. Deshalb sollten Agenten, die für Kooperation mit anderen Agenten vorgesehen sind mit Konfliktlösungsansätzen ausgestattet werden, damit diese in der Lage sind Interessenkonflikte (BSP: Ressourcen, Aufgabenverteilung, Zwischenziele) zu lösen.
Ein Multi Agenten System bezeichnet meist ein Verbund von Agenten, die als Einheit auftreten aber jeder für sich mit individuellen Aufgaben betreut wird. Dies steht ein wenig im Gegensatz zu den „verteilten Problemlösern“, also mehrere Agenten die gemeinsam an der Lösung eines globalen Problems sitzen. Diese Unterscheidung wird aber nicht so scharf getroffen und kann fließend ineinander übergehen.
Die Koordination der Agenten wird meist mit Hilfe entsprechender Protokolle getätigt. Häufig wird dazu das IP-Protokoll verwendet um eine Kommunikation über das Internet zu gewährleisten. Aber auch andere Netzwerkprotokolle sind durchaus vorstellbar um andere Netzwerk Topologien zu integrieren. Die Protokolle regeln dabei Aufgabenverteilungen, Kommunikationsformen, „soziale Beziehungen“ und das gemeinsame Problemlösen.
Um ein Problem gemeinsam zu lösen werden meist folgende Schritte abgehandelt:

1. Zerlegung einer Aufgabe in kleinere Teilaufgaben, die separat zu lösen sind
2. Delegation einer Teilaufgabe an einen einzelnen Agenten
3. Lösung der Teilaufgabe durch einen Agenten
4. Zusammenführen der gelösten Teilaufgabe

Die Möglichkeit das eine Aufgabe nicht gelöst werden konnte, sollte dabei berücksichtigt werden und eine Möglichkeit geschaffen werden, dieses Problem mit Hilfe der anderen Teillösungen in einer weiteren Bearbeitungsphase zu lösen.
Das Kosten-Nutzen Management muss in einem Multi Agenten System durchaus

bedacht werden. Im Internet erwartet ein Benutzer, dass er möglichst schnell Ergebnisse präsentiert bekommt. Deswegen sollte ein Verkaufsagent nicht zwangsläufig auf Antwort aller Agenten warten bis er dem Benutzer Ergebnisse präsentiert. Besser wartet er eine adäquate Zeit ab und präsentiert daher zeitnahe Ergebnisse auf Kosten der Quantität der Ergebnismenge.

Quelle: [5] Handbuch der KI, S. 999ff

3 Weltmodell, Wissensrepräsentation und Wissensakquisition

3.1 Weltmodell der Küche und der Kochrezepte

In diesem Kapitel wird die grundlegende Struktur der Umwelt für den Agenten analysiert. Diese Analyse ist wichtig um zu wissen, welche Informationen für den Software Agenten relevant sind und welche er ignorieren kann. Die erste Frage, die sich dabei stellt ist die grundsätzliche Frage nach dem Aufbau eines Rezepts.

3.1.1 Aufbau eines Rezepts

Die meisten Rezepte beginnen mit einer Zutatenliste auf der mindestens eine Zutatenbezeichnung steht. Zusätzliche Informationen können Mengen- und Einheiten-Informationen sein.

```
3    ml      Öl
             Salz
1            Paprika
200  g       Mehl
```

Zusätzlich gehört zu einem Rezept eine im Fließtext geschriebene Arbeitsanweisung, wie ein Koch dieses Rezept zu kochen hat.

Beispiel Rezept: Zwiebelkuchen

```
Hefeteig herstellen. Die Zwiebeln in Ringe schneiden und
in Öl andünsten. Etwas abkühlen lassen. Den Speck würfeln,
den Käse reiben. Die Sahne mit den beiden Eigelb
verquirlen. Den Teig ausrollen. Darauf die Zwiebeln
verteilen. Mit Pfeffer würzen. Den Speck und den Käse in
einer Schüssel miteinander vermischen. Auf die Zwiebeln
verteilen. Die Sahne darübergeben. Bei 50° im Backofen bei
geöffneter Tür ca. 15 - 20 Minuten gehen lassen. Das Blech
herausnehmen und den Backofen auf 200° aufheizen. Die
Backzeit beträgt etwa 30 Minuten.
```

Quelle: [11] Internetquelle: Zwiebelkuchen Rezept

3.1.2 ERD der Küchenwelt

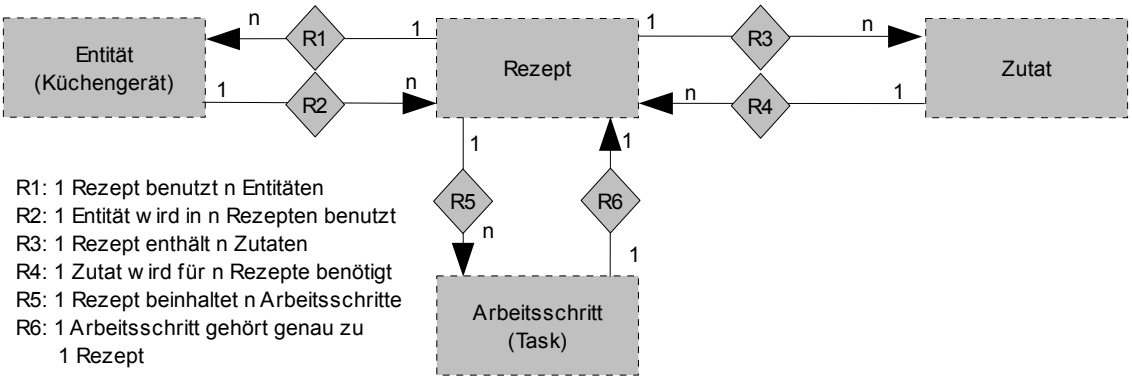

Abbildung 6: ERD der Küchenwelt

Die Zutatenliste kann dabei relativ leicht von einem Computer maschinell verarbeitet werden. Die Arbeitsanweisungen hingegen brauchen eine genauere Analyse.

3.1.3 Grundidee zur Verarbeitung der Arbeitsanweisung

Jeder Satz in einem Rezept beschreibt eine Tätigkeit, die ausgeführt werden muss, egal ob aktiv (z. B. etwas schneiden) oder passiv (z. B. auf etwas warten). Zusätzlich zu einer Tätigkeit kommen noch verschiedene Zutaten, Küchengeräte und andere Parameter, die beachtet werden müssen. Parameter können zum Beispiel die Angabe von einer Menge in einer bestimmten Einheit sein (BSP: 10 Minuten oder 200 °C). Diese Parameter werden im folgenden Argumente genannt.

Diese grobe Unterteilung in Sätze reicht aber noch nicht aus um die gesamte Komplexität eine Rezepts zu beschreiben, denn innerhalb eines Satzes können sich weitere Tätigkeiten verbergen.

Ein weiteres Problem ist die Zusammengehörigkeit von unterschiedlichen Arbeitsanweisungen. Es kann durchaus vorkommen, dass in einem ersten Arbeitsschritt etwas hergestellt wird, das erst in einem späteren Schritt weiterverarbeitet wird oder der aktuelle Arbeitsschritt durch ein Wort den Bezug auf einen letzten Arbeitsschritt herstellt.

Zusätzlich besteht immer die Gefahr von Irritationen für einen Computer, die vom Menschen ausgehen. Jeder Mensch schreibt seine Rezepte in einem eigenen Stil. Dabei kann ein Stil sehr blumig wirken und viele beschreibende Adjektive enthalten oder er kann sehr sachlich alle Anweisungen angeben. Auch Probleme durch Tippfehler oder Anweisungen, die nicht für einen Roboter geeignet sind, treten durchaus häufig auf.

Häufig treten bei Kochrezepten ähnliche Strukturen bei der Beschreibung auf. Zumeist beginnt ein Satz mit einem Artikel oder Pronomen, dem dazugehörenden Substantiv, darauf folgt in einigen Fällen ein Adjektiv und beendet wird der Satz mit einem Verb

und falls erforderlich einem Hilfsverb.

Einige Beispiele:

```
Die Paprika schneiden.
```

```
Den Käse reiben.
```

```
Die Nudeln weich kochen.
```

```
Den Salat abtropfen lassen.
```

Die deutsche Sprache erlaubt aber so viele Möglichkeiten um solche Sachverhalte auszudrücken, so dass man leider nicht über Position eines Wortes eine Entscheidung zur Verwendung eines Wortes treffen kann, sondern nur zusätzliche Informationen gewonnen werden können.

Ein Rezept in Task-Sprache
Jeder Arbeitsschritt (Task) benötigt folgende Eigenschaften:

i. Taskbezeichnung zur Beschreibung eines Schritts
ii. Tasknummer zur fortlaufenden Nummerierung
iii. ein oder mehrere Argumente
iv. eine oder mehrere Referenzen auf vorherige Tasks

3.1.3.1 Taskbezeichnung zur Beschreibung eines Schritts

Die Taskbezeichnung soll eindeutig den Arbeitsschritt beschreiben. Sie kann aus einem Verb und falls vorhanden einem zugehörigen Hilfsverb gebildet werden. In manchen Fällen benutzt der Autor eines Rezepts auch Adverbien, die zusätzlich in die Tasbezeichnung eingefügt werden
Jedes Verb in einem Satz bildet einen eigenen Task, d.h. aus einem einzelnen Satz können mehrere Tasks entstehen.

BSP 1: Verb als Taskbezeichnung

```
Hefeteig herstellen.
```

Taskbezeichnung: herstellen

BSP 2: Verb und Hilfsverb als Taskbezeichnung

```
Etwas abkühlen lassen.
```

Taskbezeichnung: abkühlen lassen

BSP 3: Satz enthält mehrere Verben, die zu mehreren Tasks führen

```
Den Speck würfeln, den Käse reiben.
```

Taskbezeichnung 1: würfeln
Taskbezeichnung 2: reiben

3.1.3.2 Tasknummer zur fortlaufenden Nummerierung

Eine einfache Art der Nummerierung wäre es jedem Satz eine fortlaufende Nummer zu geben und daraus auch die Tasknummern abzuleiten. Beispiel 3 würde aber mit Hilfe dieser Methode ein Problem aufwerfen, denn dort sind zwei Tasks innerhalb eines Satzes und die würden die gleiche Nummer bekommen. Eine Möglichkeit ist es nun, die Tasks fortlaufend durchzunummerieren oder Tasks, die sich auf den selben Satz beziehen eine weitere Untergliederung mit Hilfe von weiteren Zahlen oder Buchstaben anzuhängen.
Da ein direkter Bezug auf den Satz nicht unbedingt von Nöten ist, arbeitet der Agent mit der einfacheren Methode der Durchnummerierung.

3.1.3.3 Argumente eines Tasks

Ein Task benötigt zur Durchführung seiner Arbeitsanweisung entweder ein Küchengerät und/oder Zutaten, die er verarbeiten kann. Diese Argumente sind durch Substantive beschrieben. Deswegen wird jedes Substantiv zunächst als Argument markiert und in eine Argumentliste geschrieben.
In der Küchenwelt existieren für diese Arbeit drei unterschiedliche Arten von Argumenten:

- Zutaten
- Küchengeräte (oder Zutaten die nicht in der Liste stehen) werden im folgenden auch als **Entitäten** bezeichnet werden
- Argumente die nur einmalig verwendet werden wie eine Zeit- oder Mengenangabe

Hinweis: Die Zuteilung eines Substantivs geschieht über einen Abgleich mit der Zutatenliste oder durch einen Benutzer innerhalb des hier in dieser Arbeit zu entwickelten Programms.

BSP 4: Substantive sind Zutaten und Entitäten

```
Den Speck und den Käse in einer Schüssel miteinander
vermischen
```

Argument: Speck (ist eine Zutat)
Argument: Käse (ist eine Zutat)
Argument: Schüssel (ist eine Entität)

BSP 5:

```
Bei 50° im Backofen bei geöffneter Tür ca. 15 - 20 Minuten
gehen lassen.
```

Argument: Backofen (ist eine Entität)
Argument: Tür (gehört zu Backofen, ist deswegen keine Entität)
Argument: Minuten (ist ein einmaliger Parameter, der auch vorangehende Zahlenwerte benötigt)

3.1.3.4 Referenzen

Zutaten und Küchengeräte (Entitäten) müssen für einen späteren Bezug in Erinnerung behalten werden und sind deswegen in eine Liste mit einer Referenznummer abzulegen. Argumente hingegen, die nur einmalig verwendet werden, müssen nicht weiter behandelt werden und können direkt nach Schreiben in die Argumentliste wieder verworfen werden.
Nimmt ein Argument auf ein vorheriges Argument Bezug, so wird die Referenznummer zu diesem (schon verarbeiteten) Argument hervorgeholt und nach Verarbeitung auf die aktuelle Tasknummer gesetzt.

BSP 6:

```
Den Speck würfeln, den Käse reiben.
```

Argument: Speck (Referenz: 5)
Argument: Käse (Referenz: 6)

```
Den Speck und den Käse in einer Schüssel miteinander
vermischen
```

Argument: Schüssel
Referenz: 5 (Referenz von Speck wird auf aktuelle Tasknummer gesetzt)
Referenz: 6 (Referenz von Käse wird auf aktuelle Tasknummer gesetzt)

3.1.3.5 Das Problem der „stummen“ Referenzen

Einige Referenzen in Rezeptanweisungen sind nicht durch das wiederholte aufführen eines Substantivs zu erkennen. Diese Referenzen benutzen entweder Konjunktionen oder Präpositionen um auf ein vorherigen Task, eine Zutat oder eine Entität hinzu deuten. Präpositionen, die auf einen vorherigen Task hindeuten treten meist zu Beginn des Satzes auf.

BSP 7: Präposition zu Beginn eines Satzes mit Bezug auf vorherigen

```
Den Teig ausrollen.
```

```
Darauf die Zwiebeln verteilen.
```

Argument: Zwiebeln
Referenz: vorheriger Task, da Präposition „Darauf“ zu Beginn des Satzes

BSP 8: Konjunktion nimmt Bezug auf vorherigen Task

```
Die Zwiebeln in Ringe schneiden und in Öl andünsten.
```

Task A: schneiden
Task B: andünsten
Referenz: Durch Konjunktion „und“ wird Bezug von Task B auf Task A genommen.

3.1.4 Probleme während der Analyse

Eine Vielzahl von Problemen können während der Analyse eine Kochrezepts zum Vorschein treten, die eine maschinelle Erzeugung erschweren:

- Abkürzungen, die dem Menschen leicht verständlich sind, sind für den Computer nicht zu interpretieren

- Tippfehler werden im menschlichen Gehirn automatisch korrigiert und führen zu keinen Problemen, der Computer hingegen stellt einen Wortvergleich (Pattern Analyse) an und findet bei Tippfehlern keine Übereinstimmung.
  BSP: Zucini
  Mensch erkennt den Tippfehler und macht Zucchini daraus
  Computer versucht Wortvergleich (auch mit Zucchini) aber findet keine Übereinstimmung

- Präpositionen zu Beginn des Satzes müssen nicht zwangsläufig einen Bezug auf den vorherigen Task herstellen

  ```
  Auf den Teig Salz geben.
  ```

- Dinge können innerhalb eines Rezepts aus dem Nichts auftauchen und auch wieder verschwinden.

  ```
  Bei 50° im Backofen bei geöffneter Tür ca. 15 - 20 Minuten
  gehen lassen. Das Blech herausnehmen und den Backofen auf
  200° aufheizen.
  ```

Für Menschen ist die Zusammengehörigkeit von einem Blech und dem Backofen klar, dem Computer muss dieser Zusammenhang aber klar gemacht werden. Besonders wichtig ist, dass dem Computer klar ist, dass das Zutatengemisch auf dem Blech ist, da er dieses samt Zutaten herausnehmen soll.

- Wünsche oder Meinungen tauchen in manchen Rezepten auf. Der Computer versucht daraus aber auch Arbeitsanweisungen zu machen.

```
Ich nehme frische Kräuter, da diese besser schmecken.
```

```
Ich wünsche einen guten Appetit.
```

Hinweis:
Lösungsansätze für einige von diesen Problemen befinden sich in Kapitel 7.2.1

3.2 Wissensakquisition

Das Wissen, welches in dieser Arbeit angesammelt und verarbeitet werden soll, sind gegebene Kochrezepte. Nun gäbe es mehrere Möglichkeiten diese gegebenen Rezepte dem Agenten zugänglich zu machen. Zunächst könnte jedes Rezept über die Tastatur eingeben werden, was aber für die Entwicklung hin zu einem autonomen Agenten ein großes Hindernis wäre. Eine zweite Möglichkeit wäre das Bereitstellen eines Rezepts in einem bestimmten Format, wie z.B. XML. Die für einen autonomen Agenten aber sinnvollste Lösung ist die Beschaffung der Rezepte aus einer immer selbstständig weiter wachsenden Informationsquelle, dem Internet.

Die XML Lösung, also das Bereitstellen der Rezepte in solch einem Format ist trotzdem gegeben, da der Agent akquirierte Internet Rezepte maschinell in ein solches XML Format konvertiert.

3.3 Informationsextraktion aus WorldWideWeb

Um die größte Sammlung von Informationen schlechthin, das Internet, für den Agenten als Wissensquelle benutzen zu können, muss der Agent über Parameter auf diese Umwelt eingestellt werden. Da jede WWW-Seite einen unterschiedlichen Aufbau hat, muss jede Seite bzw. jeder Anbieter global für seine Seiten dem Agenten bekannt gemacht werden.
In dieser Arbeit wird der Software Agent Informationen akquirieren, die aus der Welt der Kochrezepte sind. Für ihn sind daher folgende Informationen relevant:

- Rezeptname
- Zutaten
- Menge der Zutaten
- Einheit der Zutaten
- Arbeitsanweisung

Diese Informationen sind Grundbestandteil eines jeden Rezepts und befinden sich zunächst für den Agenten wild versteckt innerhalb des HTML-Codes vermischt mit anderen Informationen der WWW-Seite.
Die Strategie, die in dieser Arbeit verwendet wird um dem Agenten zu beschreiben, wo die für ihn relevanten Informationen stehen, ist ihm zu sagen, was steht genau vor einer relevanten Information und was genau steht danach. Jede Information muss genau so beschrieben sein um ein extrahieren der gewünschten Information zu gewährleisten.

Die Angabe der Parameter kann nur durch eine jeweils einmalige Analyse des HTML-Quelltexts für jeden Anbieter von Kochrezepten im Internet erfolgen. Ein Beispiel erläutert hier die Vorgehensweise:

HTML-Quellcode (Vorverarbeitet wie in Kapitel 6.4.3 beschrieben)

BSP1:

```
...
#!-- google_ad_section_start --#
#h1 style=#padding-bottom:10px;##Zwiebelkuchen#/h1#
#h3 style=#color:black;###img
src=#http://img.chefkoch.de/img/tag.gif# width=#13# height=#12#
alt=### Vergebene Tags / Schlagworte:#/h3#
...
```

BSP2:

```
...
#!-- google_ad_section_start --#
#h1 style=#padding-bottom:10px;##Hackfleischpfanne mit Paprika
und Reis#/h1#
#h3 style=#color:black;###img
src=#http://img.chefkoch.de/img/tag.gif# width=#13# height=#12#
alt=### Vergebene Tags / Schlagworte:#/h3#
...
```

Für den Agenten, wie auch für den Menschen ist dies zunächst ein nicht so schnell zu überblickender Teil ASCII-Zeichen. Schaut man genauer hin, sieht man aber in diesen beiden Auszügen jeweils eine Stelle, die für den Menschen klar als Rezeptname auszumachen ist. In beiden Auszügen ist der Rezeptname genau gleich umgeben von HTML-Tags und anderen Steuerzeichen. Genau diese Tags und Steuerzeichen sind nun wichtig für die „Orientierung“ des Agenten. Ihm müssen genau diese Stellen bekannt gemacht werden um ihn so wissen zu lassen, dass die Informationen zwischen diesen beiden Stellen für ihn relevant sind. Die Parameter für den Rezeptnamen sehen für diese Beispiele folgendermaßen aus:

```
<rezeptname-start>
  #h1 style=#padding-bottom:10px;##
</rezeptname-start>
```

```
<rezeptname-end>#/h1#</rezeptname-end>
```

Diese Parameter werden nun für alle zu gewinnenden Informationen benötigt. Das XML-Dokument besitzt dadurch nachfolgende Struktur in DTD Schreibweise:

```
<!ELEMENT parameter (beginn,rezeptname,personen,zutaten,rezept-
description,end)>
<!ELEMENT rezeptname (rezeptname-start,rezeptname-end)>
<!ELEMENT personen (personen-start, personen-end)>
<!ELEMENT zutaten (zutaten-start,zutat,zutaten-end)>
<!ELEMENT rezept-description (rezept-description-start,rezept-
description-start-foto,rezept-description-end)>
<!ELEMENT zutat (zutat-start, zutat-menge, zutat-einheit,
zutat-bez-start, zutat-bez-end)>

<!ELEMENT beginn (#PCDATA)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT rezeptname-start (#PCDATA)>
<!ELEMENT rezeptname-end (#PCDATA)>
<!ELEMENT personen-start (#PCDATA)>
<!ELEMENT personen-end (#PCDATA)>
<!ELEMENT zutaten-start (#PCDATA)>
<!ELEMENT zutaten-end (#PCDATA)>
<!ELEMENT zutat-start (#PCDATA)>
<!ELEMENT zutat-menge (#PCDATA)>
<!ELEMENT zutat-einheit (#PCDATA)>
<!ELEMENT zutat-bez-start (#PCDATA)>
<!ELEMENT zutat-bez-end (#PCDATA)>
<!ELEMENT rezept-description-start (#PCDATA)>
<!ELEMENT rezept-description-start-foto (#PCDATA)>
<!ELEMENT rezept-description-end (#PCDATA)>
```

Ein weiterer Parameter, der angibt ob das Rezept ein Bild beinhaltet oder nicht, kann über eine Checkbox innerhalb der Benutzeroberfläche angegeben werden. Dies ist nötig, da Bilder zusätzlichen HTML Code benötigen und deswegen Einfluss auf den Parameter für die Rezeptbeschreibung hat (<rezept-description-start> bzw. <rezept-description-start-foto>).

Eine vollständige Beispiel-Parametrierung für die Seite www.chefkoch.de ist im Anhang zu finden.

3.3.1 Patternvergleich

Um diese Informationen aus dem HTML Dokument ausschneiden zu können, ist für jede zu gewinnende Information ein Patternvergleich (=Mustervergleich) durchzuführen. Jede Zeile der HTML Seite wird mit dem Pattern aus der XML Datei verglichen, sollte dabei eine Übereinstimmung zustande kommen, so wird die entsprechende Textzeile zur weiteren Verarbeitung übergeben.
Übergebene Zeilen werden einem weiteren Patternvergleich unterzogen, die das abschließende Pattern suchen um so zwischen 1. Pattern und 2. Pattern die relevanten Informationen extrahieren können.

BSP:

```
#h1 style=#padding bottom:10px;##Zwiebelkuchen#/h1#
```

1. Pattern: #h1 sytle=#padding bottom:10px;##
2. Pattern: #/h1#
Extrahierte Information: Zwiebelkuchen

Mit Hilfe dieser Parameter und der HTML-Seite ist der Agent in der Lage, die für ihn relevanten Informationen auszuschneiden und in Variablen und einer externen XML-Datei zu sichern.

3.4 Syntaktische Verarbeitung maschinenlesbarer Texte der deutschen Sprache

Ein Großteil der in dieser Arbeit verwendeten Mechanismen zur Verarbeitung der deutschen Sprache stammen aus der Diplomarbeit: „Das System Satzgliedanalyse: Portierung der Anwendung von Pro*C nach JDBC und ihre Erweiterung auf ein Oracle DBMS“ von Herrn S. Rudolf. Die vorhandenen JAVA-Klassen sind für den in dieser Arbeit entwickelten Software Agenten angepasst und in kleinen Teilen verändert worden. In Kapitel 3.4.5 gehe ich auf diese Änderungen genauer ein.

3.4.1 Anwendung der Satzgliedanalyse

Die Satzgliedanalyse wird in dieser Arbeit verwendet um die natürliche Sprache eines Kochrezepts, genauer die Arbeitsanweisung oder auch -beschreibung eines Rezepts für den Agenten mit semantischen Informationen anzureichern. Dabei ist es für den

Agenten sehr wichtig, zunächst einmal die einzelnen Wortarten zu kennen. So kann ihm beispielsweise die Wortart Verb sagen, dass er irgendetwas zu tun hat oder die Wortart „Substantiv“, dass es sich um einen Gegenstand, einen Ort, eine Zeit oder ähnliches handeln kann.
Die Satzgliedanalyse arbeitet nach folgendem Prinzip:

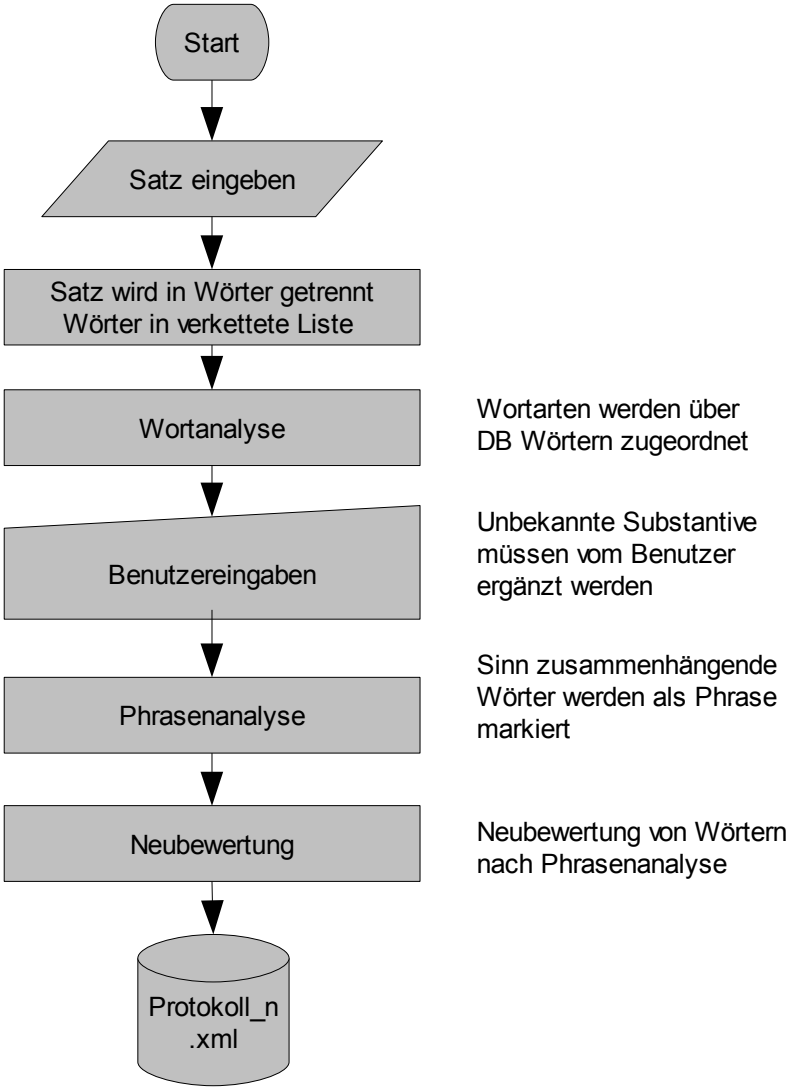

Abbildung 7: Sequenzdiagramm Satzgliedanalyse

3.4.2 Wortanalyse

Die Wortanalyse analysiert zunächst jedes Wort einzeln unabhängig von den anderen. Sie versucht über unterschiedliche Mechanismen einzelne Wortarten zu erkennen:

Mechanismus	Bedeutung des Worts
Wort ist groß geschrieben	Wort ist Substantiv, wenn nicht am Satzanfang

Mechanismus	Bedeutung des Worts
Wort ist klein geschrieben	Standard Endung eines Worts wird versucht abzutrennen (-e, -en, -s, -st, -t ...) und in DB Tabelle nach geschaut, ob Infinitiv des Verbs vorhanden. Nur Verben, die in DB Tabelle vorhanden können gefunden werden.
Wort ist klein geschrieben und nicht in Verb Tabelle	Wort wird mit anderen Tabellen verglichen und überprüft, ob Wort dort vorhanden

Hinweis: Dies ist nur eine sehr grobe zur Übersicht dienende Beschreibung der Satzgliedanalyse Mechanismen. Das System ist in Wirklichkeit viel komplexer. Genauere Informationen zu Mechanismen und Datenbank-Aufbau sind in der Diplomarbeit von Herrn S. Rudolf selbst zu finden.

Benutzereingaben

Die Benutzereingaben in der Satzgliedanalyse (und damit auch im Agenten) sind immer dann notwendig, wenn die Wortanalyse zu keinem konkreten Ergebnis gekommen ist und eine Wortart nicht identifizieren konnte. Dies ist hauptsächlich der Fall bei ihr unbekannten Substantiven, die über eine Eingabe in der Konsole angelernt werden können.

BSP: Wortanalyse erhält als Wort „Gabel“ und das Wort ist noch nicht in DB Tabelle vorhanden

Wortanalyse fragt den Benutzer zunächst:

```
Substantiv Gabel nicht gefunden?
                  [y|n] Substantiv eintragen?
```

Verneint der Benutzer die Frage, so wird die Wortart auf unbekannt gesetzt und zum nächsten Wort fortgeschritten.
Möchte der Benutzer aber das Wort eintragen, so verlangt die Analyse weitere Angaben:

```
Substantiv Nominativ Singular:
```

```
Substantiv Nominativ Plural:
```

```
Genus (Geschlecht) des Substantivs :
```

```
Handelt es sich bei dem Substantiv um ein(e)/(en)
     [L] Lebewesen
     [O] Ort
     [G] Gegenstand
     [A] Abstraktum
     [Z] Zeit
     [N] Namen
     [k] weiß nicht / keine Angabe
```

Hinweis:
Diese Benutzereingabeaufforderungen werden vom Software Agenten genauso übernommen und müssen im Gegensatz zum restlichen Programm innerhalb des Konsolen Fensters der JAVA Anwendung bearbeitet werden.

3.4.3 Phrasenanalyse und Neubewertung

Die Phrasenanalyse soll die einzelnen Wörter wieder mit einander verbinden und bei einer Neubewertung von Wortarten helfen, was wichtig sein kann bei der Unterscheidung von Adjektiven und Adverbien.
Die Phrasenanalyse stützt sich auf vorgegebene Muster von Aneinanderreihungen von Wortarten. Sie versucht nach vorgegebenen Kriterien abzuschätzen wie eine nicht sofort zu erkennende Wortart in den Kontext passen könnte. Zusammenhängende Satzteile werden mit einer Phrasennummer versehen um so ihre Zusammengehörigkeit zu kennzeichnen.

Der Software Agent dieser Arbeit benutzt die Phrasenanalyse nur zur Erkennung von Wörtern, nicht aber um deren Zusammengehörigkeit zu ermitteln, da in diesem Teil der Satzgliedanalyse noch erhebliche Schwächen zum Vorschein treten und oftmals falsche Wörter als zusammengehörig markiert werden.

3.4.4 Protokolle

Ursprünglich wurden die Resultate der Satzgliedanalyse als unstrukturierte Daten in ASCII-Text Dateien geschrieben. In Kapitel 3.4.5.3 Erstellen der Protokolldateien wird auf die Protokolldateien und deren Änderung auf eine semistrukturierte XML-Datei genauer eingegangen.

3.4.5 Änderungen innerhalb der Satzgliedanalyse

Einige Änderungen mussten vorgenommen werden um die Satzgliedanalyse in den Software Agenten dieser Arbeit zu integrieren. Die Überlegung, die Satzgliedanalyse nicht in den Software Agenten zu integrieren sondern sie als Netzwerkdienst und somit als weiteren Agenten zu betreiben wurde durchaus in Betracht gezogen, hätte aber so

große Änderungen an der Satzgliedanalyse verlangt, dass diese nicht mehr im Rahmen dieser Arbeit zu bewältigen gewesen wären.

3.4.5.1 Sprachanalyse – Main Klasse der Satzgliedanalyse

Der Main-Klasse der Satzgliedanalyse wurde die „main“-Methode entfernt und durch die Methode „analyse“ mit ähnlicher Funktionalität ersetzt. Der „analyse“-Methode wird als Parameter der zu analysierende Satz (auch mehrere Sätze) in einem String Objekt übergeben. Zuvor forderte die „main“-Methode über die Konsole den Benutzer auf einen oder mehrere Sätze einzugeben.

3.4.5.2 Benutzerverwaltung der Satzgliedanalyse

Die Klasse Benutzer.java der Satzgliedanalyse wurde so angepasst das ein fester Benutzer innerhalb der Sprachanalyse ausgewählt wird, dieser sich nicht erst über die Konsole anmelden muss und der Ablauf automatisch fortgeführt wird. Hauptgründe dafür sind, dass der Software Agent hauptsächlich über eine GUI gesteuert werden soll und es so nicht zu einem ständigen Wechsel zwischen Konsolen und GUI Fenster kommt. Hierdurch ist es natürlich auch nicht möglich einen neuen Nutzer über die Satzgliedanalyse anzulegen, was für die Verwendung innerhalb des Agenten aber auch nicht unbedingt benötigt wird. Auch die Begrüßung des Benutzers fällt auf Grund der Funktionalität und Geschwindigkeit weg.

3.4.5.3 Erstellen der Protokolldateien

Die Resultate der Satzgliedanalyse werden als ASCII-Text Datei mit fortlaufenden Protokollnummern im Dateisystem abgelegt. Zusätzlich werden die Ergebnisse in der Datenbank abgelegt, was aber für den Agenten irrelevant ist, da der Agent selbst keine Datenbank Zugriffe tätigt.
Die Resultate in den ASCII-Dateien sind für eine maschinelle Weiterverarbeitung aber eher ungeeignet, da die semantische Anreicherung nicht ausreichend strukturiert ist um sie automatisch und maschinell einzulesen. Die sinnvollste Lösung für dieses Problem ist die Erstellung von XML-Protokollen, da dort alle Werte in strukturierter Form persistent gesichert werden.

DTD der XML-Protokolle (Wortanalyse.dtd)

```
<!ELEMENT satz (satzteil+)>
<!ELEMENT satzteil (wort, wortart, phrase, umstand)>

<!ELEMENT wort (#PCDATA)>
<!ELEMENT wortart (#PCDATA)>
<!ELEMENT phrase (#PCDATA)>
<!ELEMENT umstand (#PCDATA)>
```

Wie man an der DTD erkennen kann ist die Struktur, in der die Daten abgelegt werden, simpel, aber ausreichend genau für eine maschinelle Weiterverarbeitung.

Beispiel XML-Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE satz SYSTEM "Wortanalyse.dtd">
<satz>
     <satzteil>
      <wort>Hefeteig</wort>
      <wortart>Substantiv</wortart>
      <phrase>1</phrase>
      <umstand></umstand>
     </satzteil>
     <satzteil>
      <wort>herstellen</wort>
      <wortart>Verb im Infinitiv</wortart>
      <phrase>0</phrase>
      <umstand></umstand>
     </satzteil>
     <satzteil>
      <wort>.</wort>
      <wortart>Satzzeichen</wortart>
      <phrase>0</phrase>
      <umstand></umstand>
     </satzteil>
</satz>
```

3.4.5.4 Verzeichnisstruktur und Pakettierung

Die Verzeichnisstruktur der Satzgliedanalyse habe ich aufgebrochen, da die gegebene Struktur nur mit vielen Import-Anweisungen ohne eine Eclipse-Entwicklungsumgebung zum Laufen zu bringen wäre. Da der Agent aber möglichst portabel und einfach zu bedienen sein soll, wurden sämtlich JAVA Packages aufgelöst und alle Quellcode Dateien in einen gemeinsamen Ordner verschoben.

3.4.5.5 Datenbank

Die Datenbankstruktur ist nicht verändert worden. Auch die Zugriffe auf die Datenbank wurden nur an wenigen Stellen überarbeitet, da diese ansonsten zu fehlerhaften Ergebnissen führten.
Eine Kopie der Datenbank inklusive der Daten wurde erstellt und speziell für den Agenten neu aufgesetzt.
Hinweis: Kopie der Agenten Datenbank als SQL Skript inklusive Daten auf CD vorhanden.

3.4.5.6 Die Klasse „Debug.java“

Die Debug Klasse ist nur für den Programmierer des Agenten wichtig. Sie erlaubt über eine statische boolean Variable Debug-Ausgaben ein- und auszuschalten. Diese Klasse

wird nicht nur von der Satzgliedanalyse verwendet, sondern auch von einigen Anderen. Über das Setzen der statischen Variable debug auf den bool'schen Wert „true“ lassen sich einige Konsolen Ausgaben einschalten, die bei der Fehlersuche helfen können.

3.4.5.7 Änderungen innerhalb der Klassen

Klasse: Substantiv.java , Methode: vormark

```
//Durchlauf Liste ab 2.Knoten
//Budde: ab 1. Knoten
for (int i=0; i<list.size(); i++) {
...
```

Alt:	Liste wurde erst ab 2. Knoten durchlaufen
Auswirkung:	Substantive zu Beginn eines Satzes wurden nicht als solche erkannt
Neu:	Liste wird ab 1. Knoten durchlaufen

Klasse: Wortsteuer.java , Methode: pruefersteswort

```
do { //Eingabe y/n des Benutzers, ob Substantiv
  c = IO1.einchar();
  System.out.println(c);
  if (c!='y' || c!='n') System.out.println("Bitte [y | n]
      waehlen!");
} while (c!='y' && c!='n');          // ZEILE VON BUDDE
GEÄNDERT ( || ersetzt durch && )
```

Alt:	Schleifen-Abbruch-Bedingung war mit Oder-Zeichen verknüpft („||“)
Auswirkung:	Schleife konnte niemals abgebrochen werden, da Zeichen niemals gleichzeitig 'y' oder 'n' sein konnte
Neu:	Abbruch-Bedingung durch Und-Zeichen verknüpft („&&“)

Klasse: Benutzer.java , Methode: benliste

```
..
do {
 //  k = IO1.einint();     // geändert von Budde
 k = 5;
 if (k<0 && k>benid) System.out.println("Bitte gueltige ID
 eingeben!");
} while (k<0 && k>benid);
...
```

Alt:	Benutzer ID wurde von Eingabeaufforderung verlangt
Auswirkung:	Programm wartet auf Eingabe
Neu:	Benutzer ID fest auf 5 gelegt und Ausgaben in Konsole auskommentiert

Klasse: Sprachanalyse.java , Methode: main

```
public static void analyse(String a) {
...
```

Alt:	Main-Methode der Satzgliedanalyse
Auswirkung:	keine, wird nicht mehr als Main benötigt
Neu:	zur Methode „analyse“ mit „String“-Parameter zur Übergabe eines Satzes (oder mehrerer) umbenannt

Quelle: [7] Das System: Satzgliedanalyse

4 Task Generierung

4.1 Die XML-Task Sprache mit semantischer Anreicherung

Ein Ergebnis des Software Agenten ist die Sicherung eines Rezepts als XML-Datei mit zusätzlichen semantischen Informationen. Diese Tasksprache ist speziell für diese Arbeit entwickelt worden und würde sich durch geringfügige Änderungen auf andere Themengebiete und Welten anpassen lassen.

4.1.1 Aufbau der XML-Task Sprache

Der Aufbau der XML-Task Sprache orientiert sich stark am Aufbau eines Kochrezepts.

- Rezeptname
- Zutatenliste
- Arbeitsschritte

Die XML-Datei wird in UTF-8 Kodiert und lässt damit auch Umlaute und andere Sonderzeichen zu die in deutschen Rezepten häufig gegeben sind. Der Aufbau der XML Datei wird durch nachstehende DTD (Rezept.dtd) beschrieben:

```
<!ELEMENT REZEPT (REZEPTNAME,ZUTATENLISTE,ARBEITSSCHRITTE)>
<!ELEMENT ZUTATENLISTE (ZUTAT*)>
<!ELEMENT ZUTAT (BEZEICHNUNG,MENGE,EINHEIT)>
<!ELEMENT ARBEITSSCHRITTE (SCHRITT*)>
<!ELEMENT SCHRITT (TXT,TASK,ARGUMENT*,REFERENZ*)>

<!ELEMENT REZEPTNAME (#PCDATA)>
<!ELEMENT BEZEICHNUNG (#PCDATA)>
<!ELEMENT MENGE (#PCDATA)>
<!ELEMENT EINHEIT (#PCDATA)>
<!ELEMENT TXT (#PCDATA)>
<!ELEMENT TASK (#PCDATA)>
<!ELEMENT ARGUMENT (#PCDATA)>
<!ELEMENT REFERENZ (#PCDATA)>

<!ATTLIST SCHRITT
   SID     CDATA       #REQUIRED
>
<!ATTLIST ARGUMENT
   AID     CDATA       #REQUIRED
>
<!ATTLIST ARGUMENT
   ZUT     CDATA       #IMPLIED
>
```

Beschreibung der XML-Elemente

XML-Element	Beschreibung
REZEPTNAME	Enthält den Namen des Rezepts
ZUTATENLISTE	Umschließt alle Zutaten
ZUTAT	Umschließt eine einzelne Zutat
BEZEICHNUNG	Bezeichnung einer Zutat, wie „Milch“ oder „Mehl“
MENGE	Menge einer Zutat
EINHEIT	Die Einheit in der die Menge einer Zutat gemessen wird
ARBEITSSCHRITTE	Umschließendes Element aller Arbeitsschritte
SCHRITT Attribut: SID	Einzelner Schritt Fortlaufende Nummer die Task eindeutig identifiziert
TXT	Satz(-teil) auf den sich aktueller Arbeitsschritt bezieht
TASK	Beschreibende Handlung, die den Task benennt
ARGUMENT Attribut: AID Attribut: ZUT	Ein Argument, das dem Task übergeben wird (BSP: eine Zutat) Argument-ID, die für jeden Task fortlaufend nummeriert wird Enthält Information ob Argument eine Zutat, eine Entität oder nichts von beiden ist. ZUT=“TRUE“ : Argument ist eine Zutat ZUT=“FALSE“ : Argument ist eine Entität
REFERENZ	Element zur Referenzierung von vorherigen Tasks

Tabelle 2: XML-Elemente eines Rezepts

Jeder Arbeitsschritt wird als Schritt in der XML-Datei geführt. Jeder Satz der Rezeptbeschreibung führt zu mindestens einem Task in der XML Datei. Jeder Task wird dabei durch ein Verb beschrieben, dem über die Argumente und Referenzen

unterschiedliche Parameter übergeben werden.
Argumente sind dabei entweder Zutaten, Entitäten oder nichts von beiden.

Zutaten: Argumente, die auf der Zutatenliste stehen
Entitäten: Argumente wie Küchengeräte oder Hilfsmittel (auch andere Lebensmittel, die nicht auf Zutatenliste stehen).
Andere: Argumente die Zeiten, Temperaturen oder andere Parameter angeben

Referenzen beziehen sich immer auf vorhergehende Schritte. Ein Referenz mit dem Wert 1 bezieht sich folglich auf das Resultat des ersten Schritts.

Auszug aus „Zwiebelkuchen.xml“

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE REZEPT SYSTEM "Rezept.dtd">
<REZEPT>
     <REZEPTNAME>Zwiebelkuchen</REZEPTNAME>
     <ZUTATENLISTE>
      <ZUTAT>
            <BEZEICHNUNG>Mehl</BEZEICHNUNG>
            <MENGE>400.0</MENGE>
            <EINHEIT>g</EINHEIT>
      </ZUTAT>
      <ZUTAT>
            <BEZEICHNUNG>Hefe</BEZEICHNUNG>
            <MENGE>20.0</MENGE>
            <EINHEIT>g</EINHEIT>
      </ZUTAT>
      ...
     </ZUTATENLISTE>
     <ARBEITSSCHRITTE>
      <SCHRITT SID="1">
            <TXT>Hefeteig herstellen .</TXT>
            <TASK>herstellen</TASK>
            <ARGUMENT AID="1" ZUT="FALSE">Hefeteig</ARGUMENT>
      </SCHRITT>
      <SCHRITT SID="2">
            <TXT>Die Zwiebeln in Ringe schneiden und</TXT>
            <TASK>schneiden</TASK>
            <ARGUMENT AID="1" ZUT="TRUE">Zwiebeln</ARGUMENT>
            <ARGUMENT AID="2">in Ringe</ARGUMENT>
      </SCHRITT>
      ...
     <SCHRITT SID="8">
            <TXT>Den Teig ausrollen .</TXT>
            <TASK>ausrollen</TASK>
            <REFERENZ>1</REFERENZ>
      </SCHRITT>
     ...
      </ARBEITSSCHRITTE>
</REZEPT>
```

Erklärung:
Schritt mit SID = 1:
In Schritt 1 soll wie aus dem Argument und dem Task zu entnehmen ist Hefeteig hergestellt werden. Der Hefeteig ist dabei eine Entität und keine Zutat, da er erstens kein elementares Nahrungsmittel ist, sondern auch ein aus verschiedenen Zutaten zubereitetes Rezept und zweitens nicht auf der Zutatenliste steht.

Schritt mit SID =2:
In diesem Schritt sollen Zwiebeln geschnitten werden. Als zusätzliches Argument, welches weder eine Zutat noch eine Entität ist, wird der Parameter „in Ringe“ übergeben, der die Art des Zuschneidens beschreibt.

Schritt mit SID = 8:
Die textuelle Beschreibung des Tasks lautet „den Teig ausrollen“. Im Task selbst wird der Teig nur noch als Referenz gesetzt, da dieser zuletzt in Schritt mit der Schrittnummer 1 behandelt wurde.

4.2 Prozess Beschreibungs-Sprachen / Task Definition Language

Die Suche und Auswahl einer geeigneten Task-Sprache für ein Projekt wie dieses muss einige wichtige Anforderungen erfüllen. Zum derzeitigen Zeitpunkt sind noch nicht allzu viele Task-Sprachen im Umlauf oder von einer Instanz standardisiert worden.
Die Anforderungen, die diese Arbeit an eine Task-Sprache hat, sollten möglichst deckungsgleich mit den Möglichkeiten einer bereits bestehenden Task-Sprache sein.

Anforderungen an eine Task-Sprache:

- Task-Sprache darf nicht auf Maschinensprache Level angesiedelt sein , sie muss auf einem höheren abstrakten Level arbeiten, so dass die Möglichkeit besteht, sie für jeden Roboter oder virtuellen Agenten einzusetzen
- Task-Sprache soll auch für Menschen noch lesbar sein um eventuelle Korrekturen von Hand aus zu tätigen
- Zeitliche Abläufe sollen abzubilden sein
- Tasks sollen aufeinander verweisen können (Referenzen setzen)
- Tasks sollen einfache Kontrollstrukturen abbilden können
- Tasks sollen andere Tasks aufrufen können

4.2.1 Übersicht einiger Task-Sprachen

Name der Task-Sprache	Beschreibung
RobotML (**Robo**tic **M**arkup **L**anguage) (Aktuelle Version 0.3.2)	Definiert in XML-Schema einige Standard Datentypen: * Bewegungen: Drehungen in Grad, Rad, etc * Fortbewegungen in cm, m Einheiten von Sensor-informationen wie Grad, Fahrenheit, etc * Zeiten: Sekunden, Minuten, etc * ...
Workflow Process Definition Interface - XML Process Definition Language (Aktuelle Version 2.1)	Ist eine Beschreibungssprache für Tasks und Prozesse innerhalb des Gebiets für Arbeitsablauf Beschreibungen und eher ungeeignet für die Abbildung auf Roboter
Microsoft Robotics	Proprietäre Programmiersprache von Microsoft zur Steuerung von Lego Mindstorms oder ähnlichen Robotersystemen
Task Definition Language for Virtual Agents (Aktuelle Version 2003b)	Einfache Task-Sprache die Sprachgrundgerüst mit Kontrollstrukturen bietet

Tabelle 3: Eigenschaften einiger Task-Sprachen

Auf Grund der sehr gut abzubildenden Kontrollstrukturen fiel die Wahl auf die zuletzt genannte Task Definition Language for Virtual Agents. Sie ermöglicht über spracheigene Konstrukte mit geringem Aufwand Schleifen, zeitliche Abläufe und Wenn-Dann-Strukturen abzubilden.

4.3 Task Definition Language for Virtual Agents

Die im folgenden nur noch als **T**ask **D**efinition **L**anguage (TDL) bezeichnete Task-Sprache zeichnet sich durch eine hohe Anpassungsfähigkeit aus. Sie wurde von der **W**inter **S**chool of **C**omputer **G**raphics (WSCG) derzeit bekannt als „International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision“ entwickelt und zunächst als Abstract im Journal of WSCG veröffentlicht. Die

WSCG ist Teil der Universität von West Bohemia in der Tschechischen Republik. Entwickelt wurde sie zur Definition von Tasks für eine virtuelle Welt in der sich virtuelle Menschen oder Wesen interagieren und bewegen können. Ein Beispiel für eine solche Welt ist „Second Life“, dass mit als eine der ersten großen Web 2.0 Anwendungen (Stichwort: Social Networking) gilt.
Die Task-Sprache ist derzeit immer noch in der Entwicklung und wurde von mir innerhalb dieser Arbeit ein wenig erweitert um sie auf das Weltmodell der Küche anzupassen. Zurzeit ist kein Compiler, Parser oder Linker für die Sprache verfügbar, deswegen ist die Benutzung dieser Sprache nur da um zu zeigen das es möglich ist ein Rezept in eine solche Task-Sprache umzusetzen und mit vorhandenen Kontrollstrukturen umzugehen.

4.3.1 Konzept und Aufbau der TDL

4.3.1.1 Datentypen

Die TDL beinhaltet einige Standard Datentypen die aus höheren Programmiersprachen bekannt sind.

- int
- double
- String
- float
- boolean
- char
- ...

Auch Funktionen zum Konvertieren (casten) von einem Datentyp in einen anderen sind in der TDL vorgesehen.

4.3.1.2 Entitäten

Sämtliche zu benutzende Gegenstände, die in der realen Welt existieren, werden als Entitäten bezeichnet. Diese Entitäten müssen dem Roboter oder dem virtuellen Agenten bekannt sein, nicht aber der TDL selbst, d.h. wird eine Entität Zwiebel erklärt nimmt die TDL dies einfach hin und muss sich darauf verlassen, dass diese Entität dem Roboter selbst aber bekannt ist.

4.3.1.3 Definition eines Tasks

In der TDL besteht ein Task aus drei Teilen

- Task-Bezeichnung
- Variablen und Entitäten Deklaration
- Body-Block mit Anweisungen und Kontrollstrukturen

Die Blöcke werden jeweils durch eine Art Kommentarzeile eingeleitet bzw. der Body-Block durch eine Ende-Bezeichnung abgeschlossen und die Task Definition beendet.

```
TASK name()
#VARIABLES
    <Deklaration von Variablen und Entitäten>
#BODY
    <Anweisungen die in diesem Task auszuführen sind>
#END
```

4.3.1.4 Der Taskname

Der Taskname wird eingeleitet durch das Schlüsselwort „TASK“ gefolgt von einer Zeichenkette, die als Beschreibung des Tasks dienen soll. Ein Paar Runde Klammern schließen den Bereich der Task-Bezeichnung ab.

```
TASK name()
```

BSP:

```
TASK schneiden()
```

4.3.1.5 Variablen

Variablen werden analog zu höheren Programmiersprachen mit einem Datentyp gefolgt von einem Variablenname deklariert. Sie können direkt zu Beginn mit einem Wert initialisiert werden oder aber auch erst zu einem späteren Zeitpunkt innerhalb des BODY-Blocks.
Entitäten werden direkt als Entität deklariert gefolgt von ihrer Bezeichnung und zwei Hochkommata.

BSP:

```
#VARIABLES
        int tasknr = 1
        int result
        entity Hefeteig"
```

- Integer Variable mit Bezeichnung tasknr wird mit Wert 1 initialisiert
- Integer Variable mit Bezeichnung result wird deklariert aber nicht initialisiert
- Entität Hefeteig wird deklariert und auch initialisiert

4.3.1.6 Body-Block

Der Body-Block ist der Teil eines Tasks mit der eigentlichen Logik. Er kann Funktionen aufrufen, andere Tasks benutzen, Kontrollstrukturen verarbeiten, Variablen Werte zuweisen und Werte an andere Tasks übergeben.
Der Body-Block wird durch die Kommentarzeile „#BODY“ eingeleitet und mit einem DO umschlossen.

BSP:

```
...
#BODY
  DO(
        result = schneiden(Zwiebeln, in_Ringe)
  )
...
```

- Task schneiden wird aufgerufen mit den Übergabeparametern „Zwiebeln“ und „in_Ringe“. Das Resultat dieses Tasks wird als Referenz in der zuvor deklarierten Variabel result abgelegt.
- Übergabeparameter „Zwiebeln“ ist eine Entität und muss vorher im Variables-Block auch so deklariert werden
- „in_Ringe“ ist eine Information, die für die Task-Sprache nicht zu verstehen ist. Die Task-Sprache kann diese Information nur an den Roboter weiterleiten und darauf vertrauen, dass er mit dieser Information umgehen kann.

4.3.1.7 Kontrollstrukturen innerhalb von Body-Blöcken

Die Task Definition Language bietet als Grundfunktionen einige Schlüsselwörter um bekannte Kontrollstrukturen aus anderen Programmiersprachen analog abzubilden. Für die Umsetzung ist wie immer in der TDL aber der Roboter selber zuständig, d.h. wird innerhalb der TDL ein Block programmiert, der als parallel durchzuführen gekennzeichnet wird, muss der Roboter sich selbst darum kümmern, dass die Anweisungen parallel ausgeführt werden.

Schleife
Die TDL beinhaltet in ihrer aktuellen Entwicklungsstufe nur ein Gerüst um Schleifen abzubilden. Sie benutzt dafür die Schlüsselwörter „DO“ und „UNTIL“. Mit diesen Wörtern ist es möglich, eine fußgesteuerte Schleife abzubilden, die analog zu einer Do-While Schleife funktioniert.

```
DO ( <Block mit Anweisungen>)
      UNTIL (<bool'scher Ausdruck>)
```

Analog zu Programmiersprachen wird die Schleife solange ausgeführt bis der bool'sche Ausdruck den Wert „false“ annimmt. Der Wert des bool'schen Ausdrucks kann dabei über einen Vergleich ermittelt werden, direkt über eine Variable des Typs boolean oder auch direkt über die Werte „true“ und „false“.

BSP 1:
i mit Wert 1 in Variablen Deklaration initialisiert

```
DO (
  DO (
    i = i +1
    umruehren(Topf, Löffel)
  ) UNTIL (i == 7)

)
```

Schleife läuft 6 mal durch und lässt Roboter einen Topf mit einem Löffel 6 mal umrühren.

BSP 2:

```
DO (
  DO (
    umruehren(Topf, Löffel)
  ) UNTIL (true)

)
```

Schleife läuft endlos und lässt Roboter ohne Ende Topf mit Löffel umrühren. Der Roboter hingegen könnte für so etwas eigene Abbruchbedingung implementiert haben, zum Beispiel „bis alle Klümpchen“ verschwunden sind.

Wenn-Dann-Struktur
Die Wenn-Dann-Struktur übernimmt analog zu herkömmlichen Programmiersprachen die Schlüsselwörter „IF“ und „ELSE“ kombiniert mit dem Schlüsselwort „THEN“, dass von vielen Programmiersprachen nicht benutzt wird.

```
DO (
  IF (<bool'scher Ausdruck>)
        THEN (<Block mit Anweisungen>)
  ELSE (<Block mit Anweisungen>)
)
```

BSP:

```
DO (
  IF (Temperatur  == 100 )
        THEN hinzufügen(Topf, Spaghetti)
  ELSE kochen(Topf)
)
```

So könnte eine Funktion aussehen, die überprüft ob die Wassertemperatur 100° C hat, wenn ja, dann sollen Spaghetti zum Topf hinzugefügt werden, wenn nicht, dann wird der Task kochen aufgerufen.
Natürlich kann auch hier hinter einem Else-Zweig wiederum eine neue If-Verzweigung geöffnet werden.

Parallelität

Eine Besonderheit der TDL ist die einfache Möglichkeit zwei Tasks als parallel ablaufend zu kennzeichnen. In Programmiersprachen wie C oder Java ist es häufig aufwendig einer CPU mitzuteilen, dass sie bitte zwei Methoden nebenläufig oder auf zwei unterschiedlichen Prozessoren parallel ausführen soll. In der TDL ist zumindest die Kennzeichnung von zwei Tasks, die parallel ablaufen sollen gegeben, was aber der Roboter daraus macht, also ob er die Parallelität ignoriert oder parallele Aufgaben an weitere Roboter delegiert ist ihm überlassen.
Zwei Tasks, die parallel ausgeführt werden, müssen in der TDL durch das Schlüsselwort „PAR“ gekennzeichnet werden.

```
DO (
  PAR (<Block mit Anweisunegn 1>, <Block mit Anweisungen 2>)
)
```

BSP:

```
DO (
  PAR ( würfeln(Speck) , reiben(Käse))
)
```

4.3.1.8 Erweiterung der TDL

Die Task-Sprache wurde von mir in kleinen Details erweitert um den von mir zu Beginn des Kapitels geforderten Kriterien zu genügen.

Entitätenbezeichnung

Die Entitätenbezeichnungen werden automatisch bei der Generierung von Tasks dynamisch angepasst, d.h. ein Gemisch aus der Entität „Käse“ und der Entität „Speck“ wird zur Entität „Käse/Speck“. Dies ist ein reiner Zugewinn für den Menschen, der dadurch die Inhalte der TDL-Dateien viel schneller erfassen kann und von Hand aus Korrekturen, wo sie nötig sind durchführen kann.

BSP:

```
entity Kaese"
entity Speck"
...
 result = vermischen(Kaese,Speck)
```

Nächster Task, der auf eine oder auf beide Entitäten Bezug nimmt erhält automatisch „Käse/Speck“-Enität

```
entity Kaese/Speck"
```

Referenzen

Jeder Task besitzt eine Integer Variable „tasknr“, die zu Referenzzwecken mit der aktuellen Tasknummer besetzt wird. Zusätzlich erhält jeder Task eine Integer Variable „result“ die eine Art Referenz auf das Resultat eines Tasks darstellt. Bezieht sich ein Task B auf einen anderen Task A so ist damit gemeint, dass er das Resultat von A, welches als Referenz in „result“ von A gesichert ist, in B weiter verwenden möchte.

BSP:

```
TASK herstellen
#VARIABLES
        int tasknr = 1
        int result
        entity Hefeteig "
#BODY
  DO(
          result = herstellen (Hefeteig)
  )
#END
```

```
TASK ausrollen
#VARIABLES
        int tasknr = 8
        int result
        entity Hefeteig_1 "
#BODY
#DO(
          result = ausrollen (1)
)
#END
```

Task ausrollen bekommt im BODY-Teil die Referenz 1 übergeben, d.h. er benutzt das Resultat aus Task 1 (hergestellter Hefeteig) und verarbeitet es weiter. Die Entität bekommt zusätzlich eine Referenznummer angehangen, die auf den letzten Task referenziert in dem diese Entität benutzt wurde.

5 Software Engineering

5.1 Software Agenten Design

5.1.1 Eigenschaften des Software Agenten „Agent Cook“

Die in Kapitel 2.5.1 Eigenschaften eines Software Agenten beschriebenen Eigenschaften, die einen Agenten genauer klassifizieren sind wie folgt auf den Software Agenten „Agent Cook“ anzuwenden.

Eigenschaft	Agent Cook
Andauernde Verfügbarkeit	Der Agent ist zunächst nicht für einen Dauerbetrieb ausgelegt, da er immer wieder Benutzereingaben erwartet.
Interaktion mit der Umwelt	Die Umwelt des Agenten wird gegeben durch die verschiedenen Kochseiten. Diese Seite holt sich der Agent zunächst vom Anbieter unabhängig davon ob sich an der Umwelt etwas geändert hat oder nicht.
Situiertheit	Der Agent verändert sein Verhalten nicht auf Grund von Änderungen seiner Umwelt. Nur manuelle Eingriffe können Agenten auf neue Situationen einstellen.
Eigenständigkeit	Agent arbeitet nicht autark und wartet immer wieder auf Benutzereingaben. In Kapitel 7.2.1 sind mögliche Funktionen beschrieben, die zukünftig ein eigenständiges Arbeiten ermöglichen würden.
Reaktivität	Agent wartet auf manuelle Anpassung.
Reaktives Verhalten	Agent verarbeitet unterschiedliche Zustände durch IF-Kontrollstrukturen.
Zielgerichtetheit	Jeder Auftrag wird manuell gestartet und mit neuen Informationen (URL, Kochseite) bestückt.
Pro-Aktivität	Eigeninitiative ist zur Zeit nicht gegeben, ist aber möglich diese zukünftig dem Agenten hinzuzufügen.
Intelligenz	Die Intelligenz des Agenten ist stark begrenzt. Nur im Modul der Satzgliedanalyse ist der Agent fähig neues Regelwissen selbstständig zu erarbeiten. (Neubewertung von Worten). In allen anderen Teilbereichen

Eigenschaft	Agent Cook
	kann neue Intelligenz nur über manuelle Eingaben eingepflegt werden.
Rationalität	Nicht vorhanden.
Zustände	Zustände werden nicht langfristig gesichert. Vorherige Ergebnisse haben keinen direkten Einfluss auf folgende.
Lernfähigkeit	Selbstständig ist der Agent nicht in der Lage neue Dinge (neues Regelwissen, Umweltveränderungen) zu erlernen. Durch einen Benutzer können ihm diese Dinge aber angelernt werden.
Mobilität	Da der Agent in JAVA implementiert ist, ist eine Migration ohne jegliche Schwierigkeiten auf alle Systeme mit einer JVM möglich.
Kooperation	Wenn man den Kochrezept-Webserver als Agenten bezeichnen wollte (was heute immer wieder getan wird) kooperiert er mit Mensch und zusätzlichen Agenten.
Wohlwollen	Agent arbeitet nur nach Vorgaben des Menschen.
Soziales Verhalten	Zur Kommunikation mit anderen Agenten benutzt er standardmäßige Protokolle.
Emotionales Verhalten	Satzgliedanalyse besaß Begrüßungsfloskeln, die aber auf Grund der GUI entfernt wurden.
Glaubwürdigkeit	Da kein emotionales Verhalten vorhanden, kann auch die Eigenschaft der Glaubwürdigkeit nicht bewertet werden.

Tabelle 4: Eigenschaften des "Agent Cook"s

5.1.2 Informationsflüsse des Software Agenten

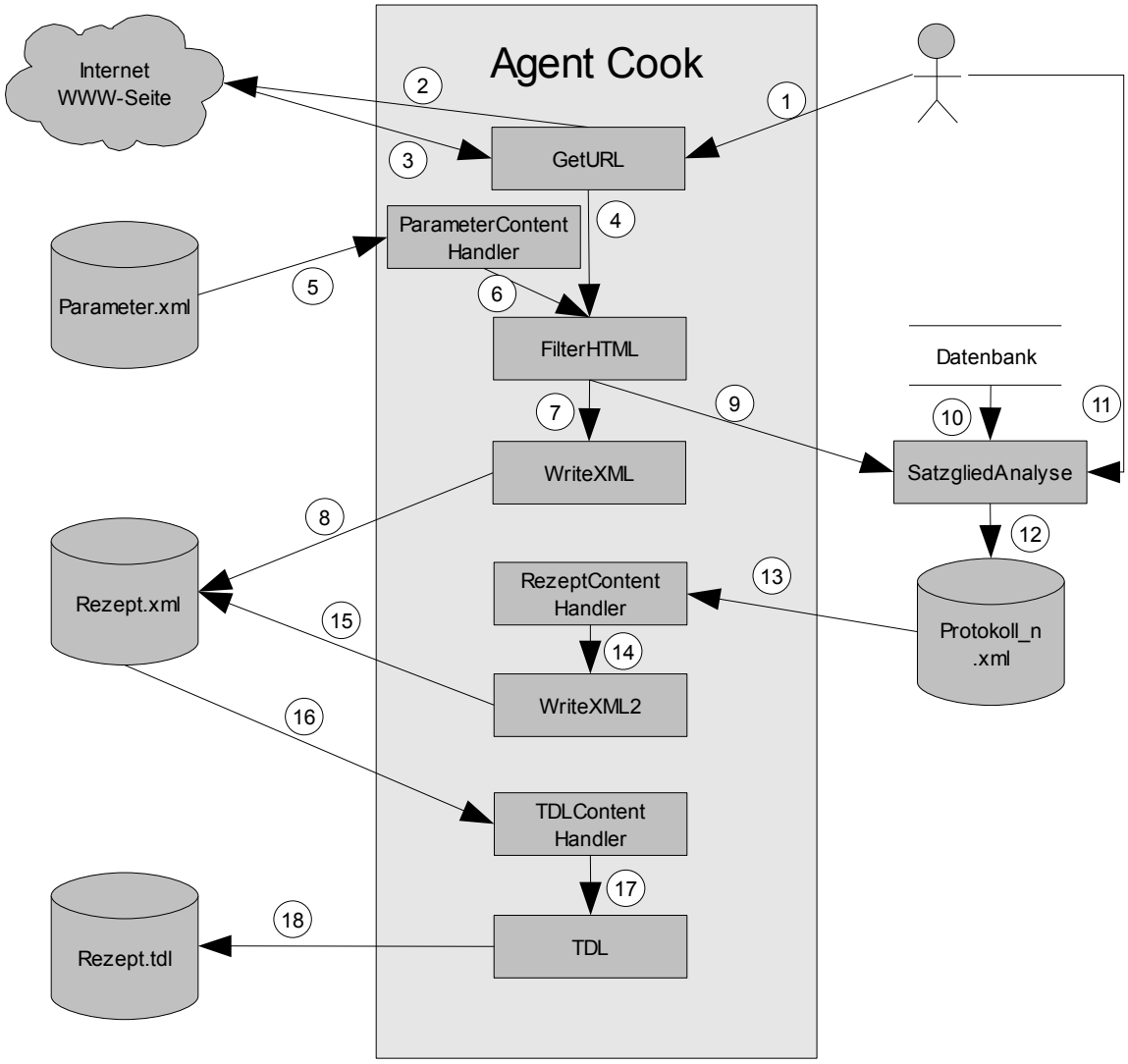

Abbildung 8: Informationsflussdiagramm des Software Agenten

1. Eingabe einer URL vom Benutzer des Agenten
2. Anfrage an Webserver die gewünschte Ressource zu senden
3. Gewünschte Ressource (Web-Seite) wird an Agenten gesendet
4. Empfangene Webseite wird an FilterHTML weitergereicht und verarbeitet
5. Parameter XML Datei wird eingelesen
6. XML Parser stellt Parameter aus XML-Datei zur Verfügung
7. Relevante Information aus Webseite (Rezept mit Zutatenliste) wird an WriteXML weitergereicht
8. XML Rezept ohne Arbeitsschritte wird geschrieben
9. Rezeptbeschreibung wird an Satzgliedanalyse weitergereicht
10. Rezeptbeschreibung wird mit Datenbank Information semantisch angereichert
11. Mögliche Benutzereingaben in der Satzgliedanalyse (neue Worte integrieren)

12. Protokolle der Satzgliedanalyse werden in fortlaufenden nummerierten XML-Protokoll-Dateien geschrieben
13. Protokolle werden von SAX Parser verarbeitet
14. Verarbeitete Protokolle werden als Tasks eines Rezept angelegt und an WriteXML2 weitergereicht
15. XML-Rezept Datei mit Arbeitsschritten wird angelegt
16. XML-Rezept Datei wird mittels SAX-Parser wieder eingelesen
17. Tasks in TDL Sprache erzeugen
18. Erzeugte Tasks im TDL Format in Datei persistent sichern

5.1.3 Modularisierung des Agenten

Die Module des Agenten sind so implementiert, dass die Hauptfunktionen austauschbar sind und für die Funktion nur die in Abbildung 8 erwähnten Informationsflüsse jeweils benötigt werden.
Die Benutzeroberfläche (GUI) des Agenten ist von den Funktionen getrennt und beinhaltet nur Aufrufe von Methoden der selbigen. Das Sammeln von Informationen ist meist gelöst von einer Weiterverarbeitung und sorgt so für eine klare Trennung von Information und deren Verarbeitung.

5.2 Klassendiagramme

Die Klassendiagramme werden jeweils als Ausschnitt der entsprechenden Module dargestellt.

5.2.1 Module der Wissensakquisition und -extraktion

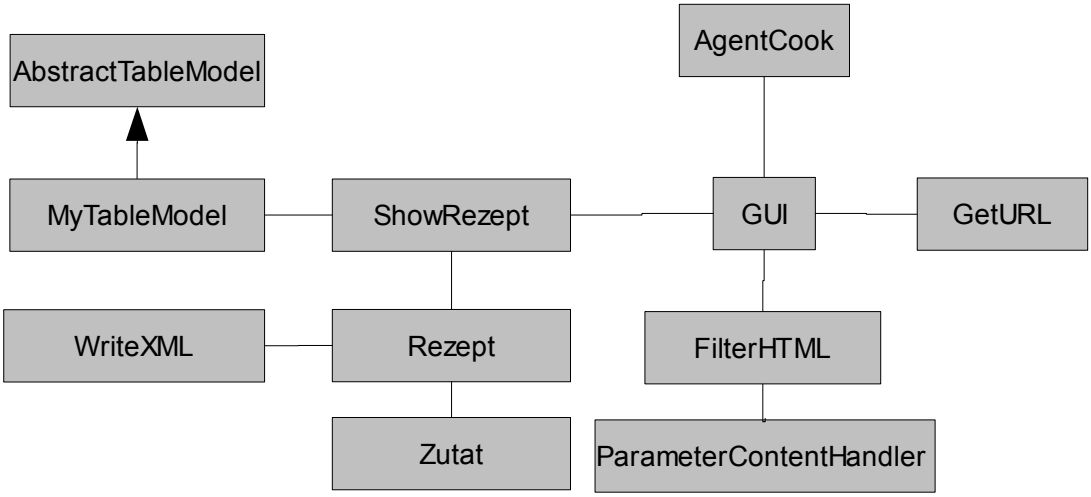

Abbildung 9: Module der Wissensakquisition und -extraktion

5.2.2 Module und Erweiterungen des SAX-Parsers

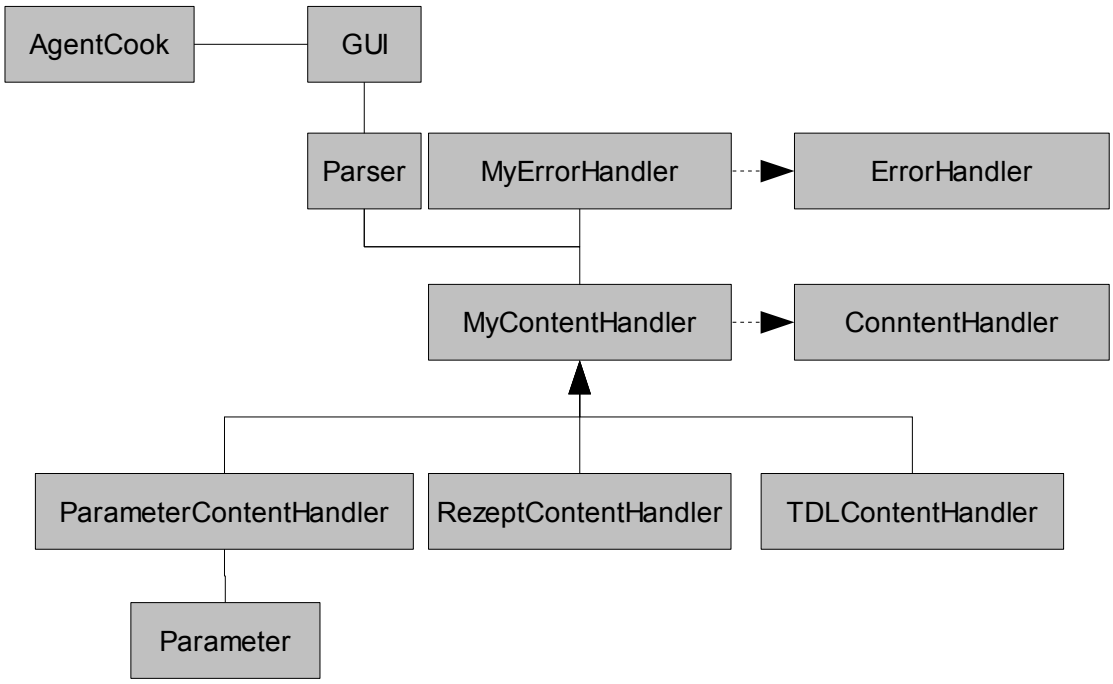

Abbildung 10: Module und Erweiterungen des SAX-Parsers

5.2.3 Module der Satzgliedanalyse

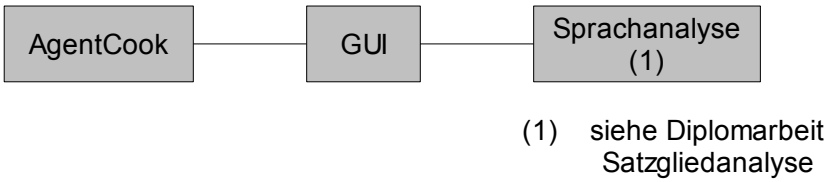

Abbildung 11: Module der Satzgliedanalyse

5.2.4 Module der XML-Task-Sprachen Generierung

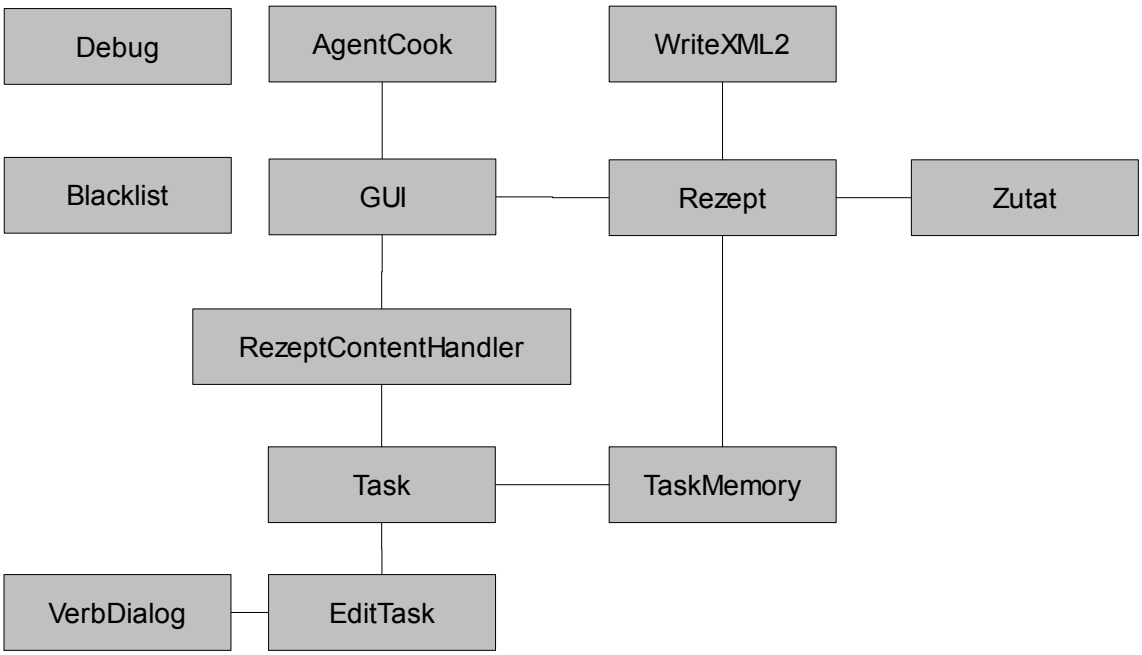

Abbildung 12: Module der XML-Task-Sprachen Generierung

5.2.5 Module der TDL Generierung

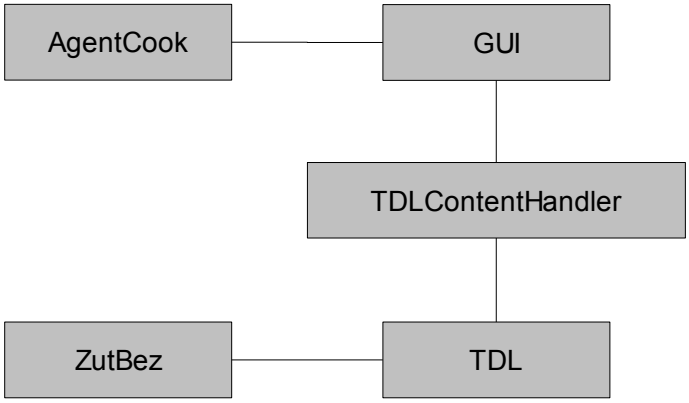

Abbildung 13: Module der TDL Generierung

6 Der Software Agent - „Agent Cook“

6.1 Beschreibung

Das im Zusammenhang mit dieser Arbeit erstellte Software Agent „Agent Cook“ ist in der Programmiersprache JAVA (Version 1.6.0.3) programmiert um eine möglichst große Plattformunabhängigkeit zu gewährleisten. Ein zusätzlicher wichtiger Aspekt zur Entscheidung den Agenten in JAVA zu implementieren gab die Integration der Satzgliedanalyse aus einer vorangegangenen Diplomarbeit[2].
„Agent Cook“ arbeitet mit einer intuitiv zu bedienenden Benutzeroberfläche die den Benutzer durch die Steuerung des Software Agenten hilft. Der Software Agent läuft in seiner derzeitigen Entwicklungsstufe nicht durchgängig und muss vom Benutzer immer wieder zum nächsten Arbeitsschritt durch Drücken eines Buttons oder Angabe verschiedener Daten (URL, Checkbox Auswahl) angeregt werden. Eine genaue Beschreibung des Ablaufs befindet sich in Abbildung 14: Ablaufplan Agent Cook. Die Ergebnisse, die der Agent erstellt sind in Form von XML und TDL Dateien (beides in UTF-8 kodierten und für Menschen lesbaren Formaten) in entsprechenden Ordnern zu finden.

Hinweis:
Im folgenden werden die Klassen für den SAX-Parser zum Einlesen von XML-Dokumenten als „XML-Parser“ bezeichnet und nur die spezialisierten ContentHandler angegeben. Zum „XML-Parser“ gehören: Parser, MyErrorHandler, MyContentHandler

2 Diplomarbeit von Dipl. Ing. Sascha Klaus Rudolf : „Das System Satzgliedanalyse: Portierung der Anwendung von Pro*C nach JDBC und ihre Erweiterung auf ein Oracle DBMS“ vom 02.Juli.2007

6.2 Ablaufplan Agent Cook

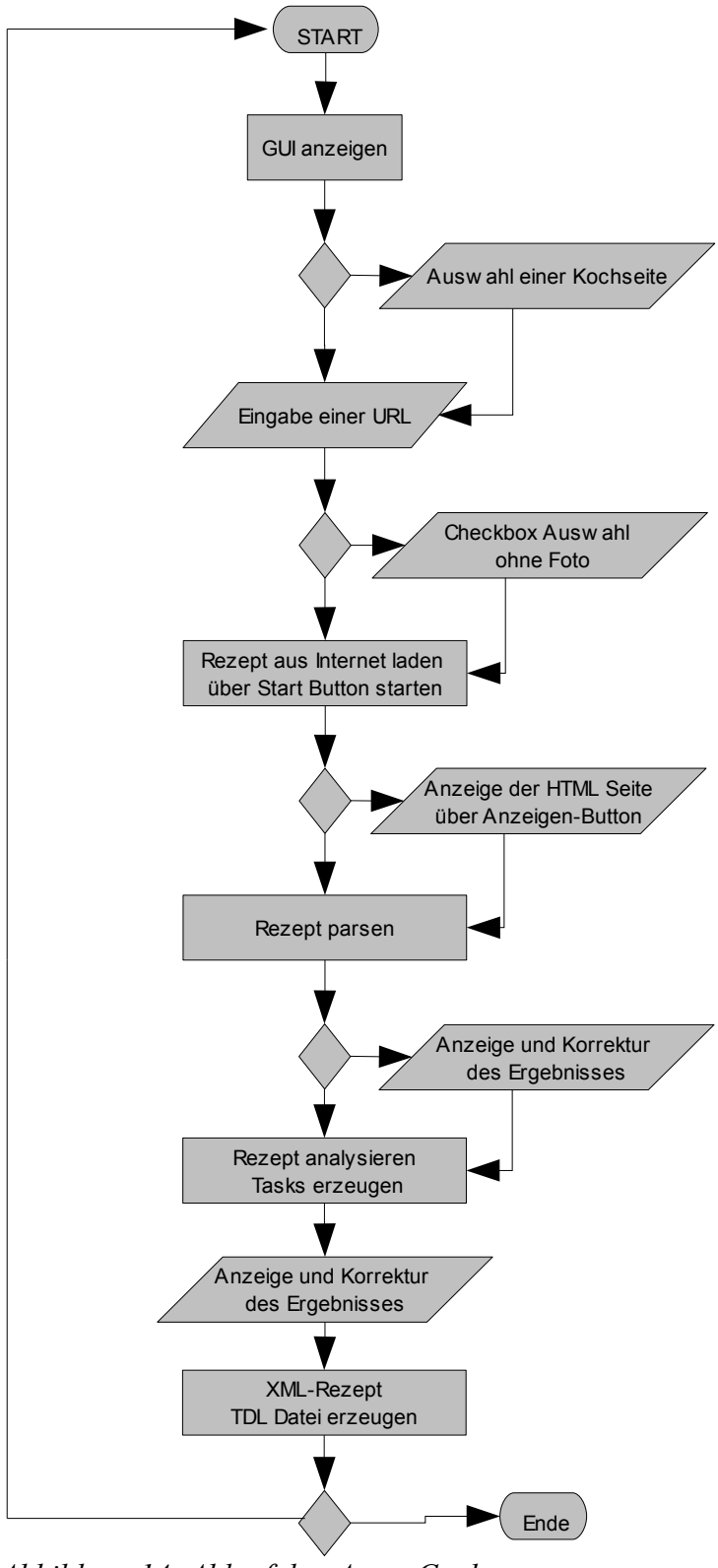

Abbildung 14: Ablaufplan Agent Cook

6.3 GUI – Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche ist in JAVA Swing geschrieben und lässt sich intuitiv von oben nach unten benutzen. Der Benutzer wird sequentiell von einer Funktion zur nächsten geführt und kann sich Zwischenergebnisse jederzeit anschauen. Das Hauptmenü des Agenten ist in der Klasse „GUI“ programmiert.

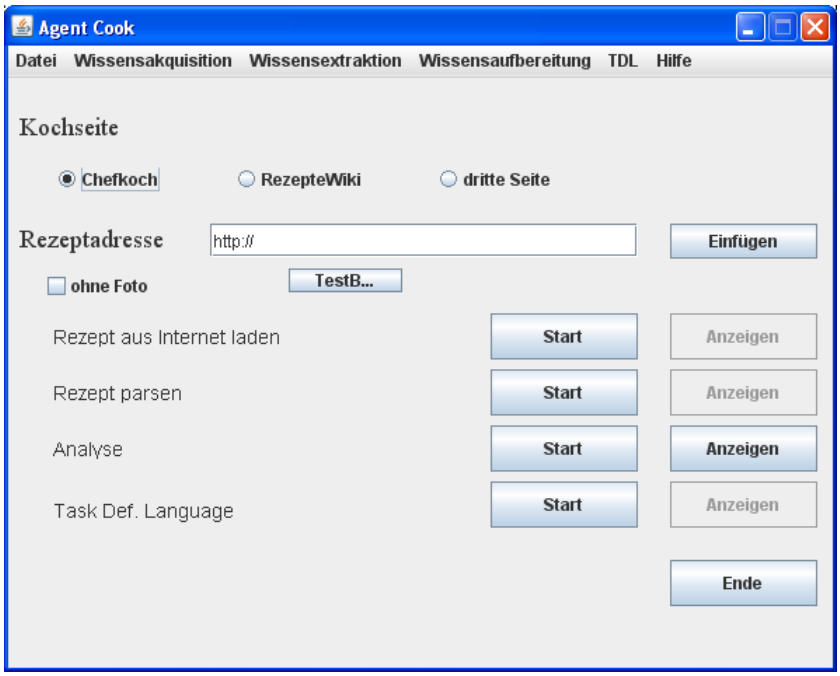

Abbildung 15: Hauptmenü Agent Cook

Anzeige Dialoge werden über die Klasse „ErrorDialog“ für Fehlerdialoge (Datenbank-Fehler, Eingabefehler, Dateifehler) und über die Klasse „ShowResultDialog“ für Zwischenergebnisse realisiert. Für verschiedene Korrekturen und Präsentationen von Ergebnissen, die nicht standardisiert angezeigt werden können, sind zusätzlich Klassen implementiert.

Klassenname	Aufgabe der Klasse
GUI	Hauptmenü des Agenten (siehe Abbildung 15)
ErrorDialog	Anzeige von Fehlern (Datenbank, Eingabe/Ausgabe, Datei, ...)
ShowResultDialog	Anzeige von Resultaten, Zwischenergebnissen
EditTask	Korrigieren und Anzeigen von Tasks

Klassenname	Aufgabe der Klasse
VerbDialog	Möglichkeit zur Sicherung von Verben in der Datenbank, die falsch erkannt wurden
ShowRezept	Ansicht eines Rezepts mit Möglichkeit zur Korrektur und Bearbeitung einer Rezeptdurchführung

Tabelle 5: GUI-Klassen

Von der Klasse GUI aus werden alle Funktionen, die im nächsten Kapitel aufgeführt sind aufgerufen und Variablen mit Standardwerten vorbelegt.

6.4 Funktionen des Agenten

Die Funktionen des Agenten sind weitestgehend modularisiert und würden im Falle einer Aktualisierung einzelner Programmfunktionen, soweit sie das Ergebnis in Repräsentation und persistenter Sicherung nicht verändern, ohne Aufwand weiter funktionieren. Alle Funktionen sind daher möglichst unabhängig voneinander programmiert und erwarten als Informationseingang einen vordefinierten Standard, der beim Überarbeiten von einzelnen Modulen beachtet werden muss.
Folgende Funktionen stellt der Agent zur Verfügung:

1. Auswahl einer Kochrezeptseite
2. Wissensakquisition eines Rezepts und Anzeigen des heruntergeladenen HTML Codes
3. Wissensextraktion aus HTML-Code und Anzeigen des Rezepts
4. Starten der Satzgliedanalyse
5. Anzeige und Korrektur der Satzgliedanalyse
6. Generierung eines XML-Rezepts mit Taskanweisungen
7. Generierung von Tasks mit Hilfe der Task Definition Language (TDL)

Diese Funktionen können als grobe Struktur des Agenten angesehen werden und laufen sequentiell nach Anweisung des Benutzers ab.

6.4.1 Auswahl der Kochrezeptseite

Der Agent ist so aufgebaut, dass er theoretisch jede Kochrezeptseite, die im Internet existiert, als Wissensquelle benutzen könnte. Da jeder Kochrezeptanbieter im Internet einen eigenständig, meist völlig verschiedenen Aufbau der Seite hat, muss dem Agent über Parameter (siehe Kapitel 3.3) beschrieben werden, wie dieser Aufbau aussieht, um dem Agenten seine Umwelt bekannt zu machen.
Im Moment ist der Agent nur auf die Umwelt von www.chefkoch.de trainiert. In der Benutzeroberfläche sind RadioButtons für drei unterschiedliche Internet Rezeptesammlungen vorgesehen. Gedacht ist die Anwendung von zwei fest

programmierten Sammlungen und einer frei verfügbaren, die jederzeit geändert werden kann.

Zuständige Klasse(n): GUI

6.4.2 Wissensakquisition eines Rezepts

Nach Auswahl eines Rezeptanbieters wartet der Agent auf die Eingabe einer konkreten URL die für ihn „den Weg durch seine Umwelt“ beschreibt. Dazu begibt sich der Benutzer in einen Browser und ruft die Seite eines konkreten Rezepts auf.

Beispiel - Rezept: Zwiebelkuchen

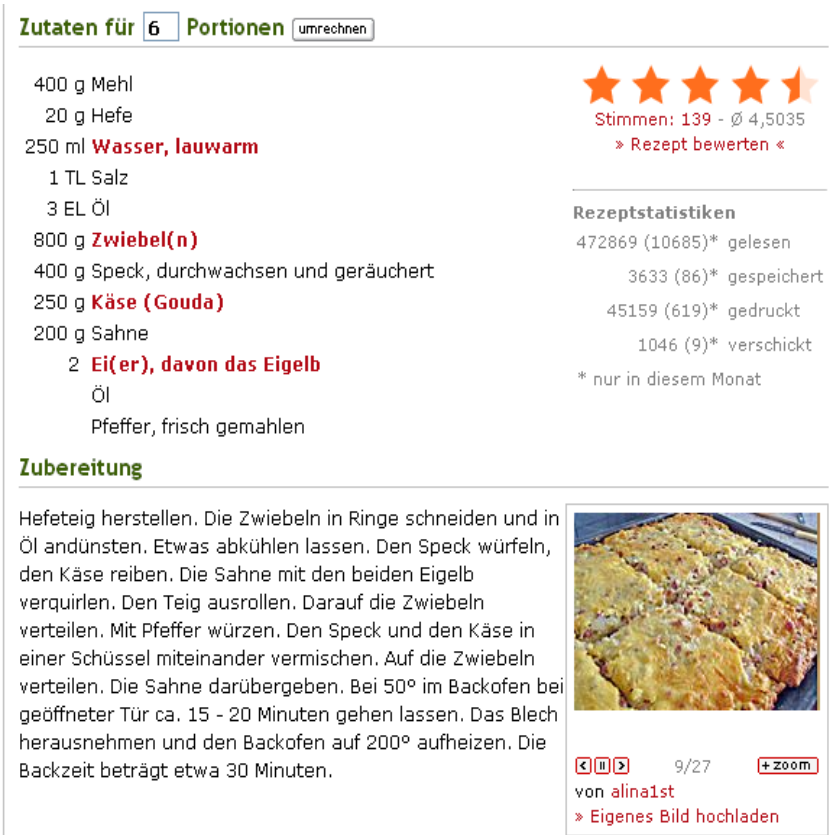

Abbildung 16: Zwiebelkuchen Rezept

Quelle: [11] Internetquelle: Zwiebelkuchen Rezept

Im Browserfenster wird die URL aus der Adresszeile des Browsers in die Zwischenablage kopiert und im Agenten in die Adresszeile über den Einfüge-Button übergeben. Zusätzlich braucht der Agent die Angabe, ob das Rezept ein Foto besitzt oder nicht, da dies die Struktur der HTML-Datei beeinflusst. Diese Angabe muss der Benutzer über eine Checkbox auswählen (Default-Wert ist „Seite mit Foto“, da Checkbox nicht ausgewählt ist).
Über den Start-Button wird Akquisition gestartet und das Rezept in HTML-Form mit

sämtlichen Formatierungen, Steuerzeichen und anderen HTML-Tags als Datei[3] gesichert. Diese Datei ist Grundlage für die weiteren Schritte, d.h möchte man an den Modulen der GUI oder Akquisition etwas verändern, ist es wichtig, dass diese Datei trotz aller Veränderungen am Ende erstellt wird.

Über den Anzeige-Button kann die heruntergeladene HTML-Kochseite als ASCII-Text angesehen werden.

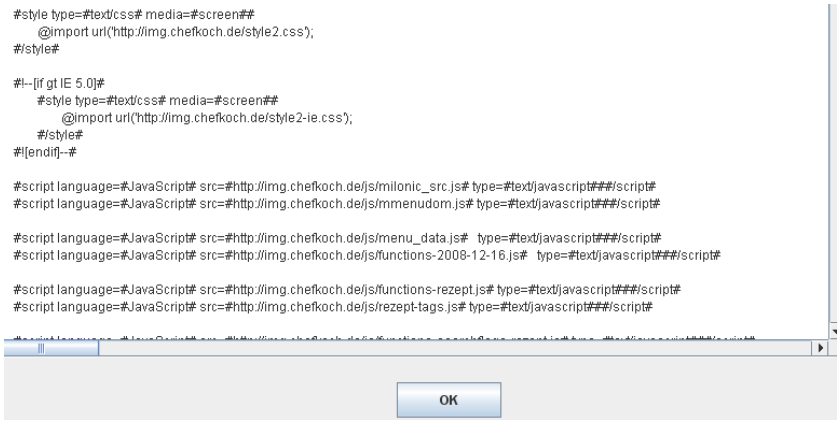

Abbildung 17: Anzeige des HTML-Codes in Agent Cook

Zuständige Java Klasse(n): GetURL, ShowResultDialog

Klassenname	Funktion
GetURL	In dieser Klassen baut JAVA eine TCP Verbindung mit Hilfe eines DataInputStreams auf und schreibt Zeilenweise die empfangenen HTML-Daten in eine Datei („htmlseite.txt“ im Source Ordner). GetURL macht eine Vorverarbeitung der HTML Seite und ersetzt Sonderzeichen durch '#'.
ShowResultDialog	Diese Klasse ist zuständig für die Präsentation des HTML-Codes, wie in Abbildung 17 zu sehen ist.

6.4.3 Wissensextraktion aus HTML-Code

Über den Start-Button des Menüpunktes „Rezept parsen“ startet der Agent die Extraktion der Rezeptdaten, die für ihn relevant sind. Die genaue Arbeitsweise der Extraktion wird in Kapitel 3.3 erläutert.

3 Name der Datei: htmlseite.txt

Die Ergebnisse der Extraktion werden in der JAVA Klasse Rezept gesichert und gleichzeitig auch in eine XML-Datei unter Rezeptname.xml (Rezeptname steht hier für einen tatsächlichen Rezeptnamen, z.B. „Zwiebelkuchen.xml“) im Ordner Rezepte gesichert.
Das Resultat der Extraktion kann über den Anzeige-Button aufgerufen werden.

Die Wissensextraktion verlangt eine geeignete Vorverarbeitung der zuvor empfangenen HTML-Quelltext-Seite, da diese einige Sonderzeichen enthält, die in XML besondere Zeichen sind und so zu verfälschten Ergebnissen führen würden. Die Zeichen '<', '>', '&', ' ' ' und ' “ ' werden durch '#' innerhalb des Quelltexts ersetzt.

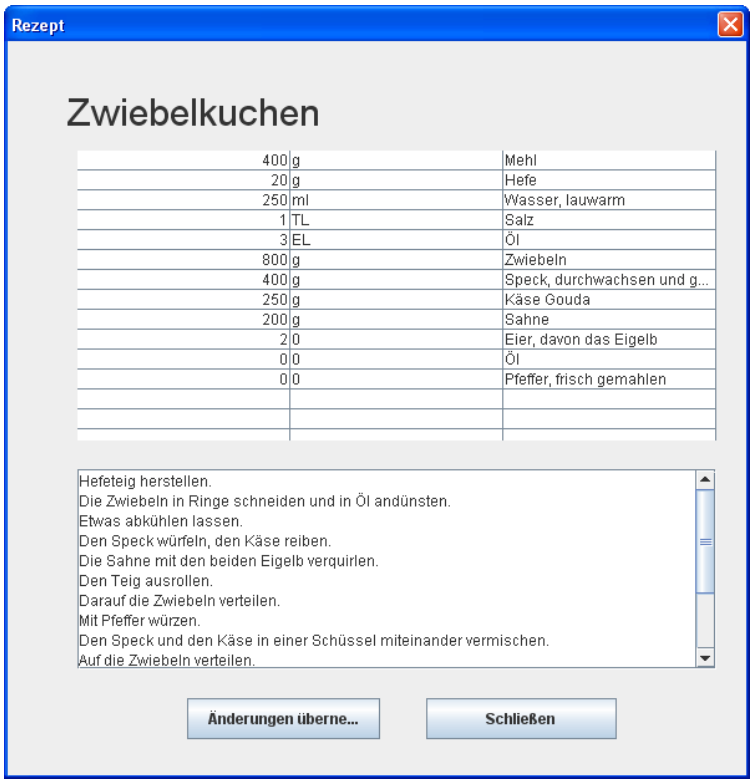

Abbildung 18: Anzeige und Korrektur eines Rezepts

Hinweis: Bestandteil dieser Arbeit ist eine XML-Parameter Beispiel Datei für die Seite www.chefkoch.de , die unter dem Namen „chefkoch.xml“ im Source Ordner zu finden ist.

Zuständige Klasse(n): FilterHTML, XML-Parser mit ParameterContentHandler, ShowRezept, Rezept, WriteXML

Klassenname	Funktion
FilterHTML	Klasse schneidet nach gegebenen Parametern relevante Informationen aus
ParameterContentHandler	Liest aus XML-Datei Parameter ein

ShowRezept	Zeigt die Zutaten in Tabellenform und Beschreibung in Textfeld an, welches Korrekturen zulässt (siehe Abbildung 18)
WriteXML	Klasse zur Zwischenspeicherung eines temporären Resultats im Dateisystem
Rezept	Klasse zur Aufnahme der relevanten Rezeptinformationen

6.4.4 Starten der Satzgliedanalyse

Über den Start-Button des Menüpunkts Analyse wird die Satzgliedanalyse gestartet. Diese teilt jedem Wort eine Wortart, eine Phrasennummer und eine Umstandsbeschreibung zu. Eine genauere Beschreibung dieses Moduls ist im Kapitel 3.4 zu finden.
Während der Satzgliedanalyse können Reaktionen des Benutzers verlangt werden. Dies kann in unterschiedlichen Fällen möglich sein:

- Substantiv noch nicht vorhanden und soll in unterschiedlichen Formen (Nominativ, singular/plural) eingetragen werden
- Ist Wort ein Substantiv?
- Welche Wortart gehört zum Wort am Satzbeginn?

Ohne die Pflege des Benutzers kann die Satzgliedanalyse nicht beendet werden und der Agent verharrt solange in Warteposition bis die Analyse erfolgreich abgeschlossen ist.
Die Resultate der Analyse werden in XML-Dateien für jedes Rezept fortlaufend nummeriert im Ordner Protokolle abgelegt.

Zuständige Klasse(n): Sprachanalyse (die wiederum ihre eigenen Klassen besitzt)

Klassenname	Funktion
Sprachanalyse	Main Klasse der Sprachanalyse der Diplomarbeit von S. Rudolf (benutzt eigene Klassen zur Analyse)

6.4.5 Anzeigen und Korrektur der Satzgliedanalyse Resultate

Die Ergebnisse der Satzgliedanalyse liegen in externen XML-Dateien (s. vorheriger Abschnitt) und sind nicht direkt im Agenten abzurufen, da die Analyse eine externe Komponente ist, die in den Agenten integriert wurde aber keine direkte Schnittstelle mit Informationsfluss in Richtung des Agenten bietet. Die externen XML-Dateien müssen deshalb durch einen SAX-Parser in den Agenten geladen werden. Der entsprechende SAX-Parser ist in die Klasse RezeptContentHandler implementiert. Er entscheidet anhand der XML-Tags um welche Art von Daten es sich handelt und schreibt diese wiederum in Objekte der Klasse Task.
Durch Drücken des Anzeigen-Buttons der Analyse startet die Klasse Rezept den Parser

mit Hilfe der Methode createTasks().
Anschließend wird ein neuer Dialog zur Korrektur von Wörtern angezeigt. Zunächst zeigt dieser Dialog sämtliche Wörter untereinander fortlaufend an. Neben jedem Wort befindet sich eine Auswahl Box (Combo Box) zur Korrektur der Wortart, zwei Checkboxen, die erste als Kennzeichen für eine Zutat, die zweite als Kennzeichen für eine Entität. Abschließend hat jedes Wort einen Anwenden-Button um getätigte Änderungen zu bestätigen. Jede Änderung muss neben jedem Wort einzeln bestätigt werden um inkonsistente Zustände zu vermeiden.
Die Änderungen, die durchgeführt werden können unterschiedliche Ereignisse auslösen:

- Änderung der Wortart zu Verb
  Ein Dialog zum Einpflegen des Verbs in die Wortanalyse Datenbank erscheint
  (Unique Constraint Exception wird angezeigt falls Wort schon in DB vorhanden)

- Änderung der Wortart zu Adjektiv
  Adjektiv wird automatisch versucht in die Wortanalyse Datenbank einzutragen
  (Unique Constraint Exception wird angezeigt falls Wort schon in DB vorhanden)

- Änderung der Wortart zu Subjektiv
  Eine automatische Überprüfung ob Substantiv eine Zutat ist wird durchgeführt

- jede andere Änderung aktualisiert den jeweiligen Task und merkt sich diese Änderung

In 7.2 sind einige Ideen, die zukünftig diese Wortänderungen noch erweitern können.

Abbildung 19: Wörter anzeigen und ändern

Zuständige Klasse(n): Rezept, Task, EditTask, XML-Parser mit RezeptContentHandler

Klassenname	Funktion
Rezept	Verwaltet sämtliche Rezeptdaten (Zutaten, Taskliste, Entitäten)
Task	Klasse zur Sicherung eines einzelnen Tasks
EditTask	GUI Klasse zum Editieren von Tasks
RezeptContentHandler	Liest aus XML-Dateien Wörter und deren semantische Anreicherung aus und schreibt diese in Task Objekte, die in einer ArrayListe in der Klasse Rezept verwaltet werden

6.4.6 Generierung eines XML-Rezepts mit Taskanweisungen

Diese Funktion generiert ein erstes Endresultat. Betätigt man den Start-Button neben dem Menüpunkt „Task Definition Language“ wird diese und auch direkt die nächste Funktion (Generierung der TDL-Datei) gestartet.
Das erste Endresultat ist eine XML-Datei, die für eine Tasksprache alle relevanten Informationen beinhaltet. Der Aufbau der XML-Datei mit zugehöriger DTD ist in Kapitel 4.1.1 zusehen.

Die XML-Datei wird über ein komplexes System aus if-Kontrollstrukturen erzeugt. Grob gesehen läuft eine Schleife über alle Tasks (an dieser Stelle sind damit noch einzelne Sätze gemeint, denn erst die Klasse selbst entscheidet, ob innerhalb eines Satzes auch mehrere Tasks zu erstellen sind). Jedes Wort innerhalb der Sätze wird geprüft zu welchem Task es gehört und ob dieser Task auch der ist, der gerade bearbeitet wird. Wird ein Wort aus einem Task Objekt ausgelesen, das eine Tasknummer beinhaltet welche gerade nicht zur Bearbeitung ansteht, wird der aktuelle Task erzeugt und mit dem Wort, welches die Taskerzeugung ausgelöst hat ein neuer Task Schritt erzeugt, der nun die aktuell zu bearbeitende Tasknummer vorgibt.
Jedes Wort wird nach seiner Wortart in einer zur Wortart entsprechend benannten Variable gesichert und bis zur nächsten Taskerzeugung auch nicht wieder gelöscht. Von dieser Regel gibt es je nach Wortart Ausnahmen, da diese nicht immer für den ganzen Satz relevant sind sondern sich nur auf einzelne Satzteile beziehen.

Adjektive müssen gesondert behandelt werden, da diese in der deutschen Sprache meist nur das nächste Substantiv beschreiben und zum darauf folgenden Substantiv nicht passend sind. Ein Adjektiv wird deswegen zwischengespeichert bis das nächste Substantiv auftritt, dort wird es verwendet indem es dem Substantiv vorangestellt wird und die Variable danach geleert wird um die doppelte Verwendung eines Adjektivs zu vermeiden.

Adverbien sind ähnlich zu behandeln wie Adjektive, nur dass diese sich nicht auf ein

Substantiv sondern auf ein Verb (mit oder ohne Hilfsverb) beziehen. Analog zu Adjektiven werden diese in einer Variablen zwischengespeichert, dem Verb vorangestellt und anschließend aus der Variablen wieder gelöscht um eine doppelte Verwendung zu vermeiden.

Die Wortart **Konjunktion** entscheidet über neue Taskerzeugung oder Weiterführung des aktuellen Tasks. Verbindet die Konjunktion zwei Substantive, so wird kein neuer Task erzeugt und die Konjunktion kann außer acht gelassen werden. Verbindet sie aber zwei Sätze miteinander, die unterschiedliche Verben haben, so muss die Kontrollstruktur für eine Konjunktion dafür sorgen, dass der aktuelle Task erzeugt wird und ein neuer für den restlichen Satz erstellt wird.

BSP:
Konjunktion die zwei Substantive aneinanderreiht:

> Den Speck und den Käse miteinander vermischen.

Konjunktion die zwei unterschiedliche Satzteile verbindet:

> Die Zwiebeln in Ringe schneiden und in Öl andünsten.

Satzzeichen können unterschiedliche Ereignisse auslösen. Der '.' ist dabei relativ einfach zu behandeln, falls dem Agenten die gängigsten Abkürzungen wie „bzw.“, „Min.“, „ca.“ und ähnliche bekannt sind und er diese einfach missachtet. Diese sind grundsätzlich auch keine Satzzeichen, aber sie werden teilweise von der Satzgliedanalyse als solche erkannt.
Der Punkt als Satzendezeichen ist ein sicheres Merkmal für den Abschluss eines Tasks. Für ein ',' hingegen sind wiederum zwei Fälle zu unterscheiden: zunächst als Aufzählungszeichen von zwei oder mehreren Substantiven (die meist mit einer Konjunktion beendet wird) oder als Trenner von zwei unterschiedlichen Satzteilen.

BSP:
Das Komma als Zeichen zur Trennung mehrere Elemente in einer Aufzählung:

> Salz, Pfeffer und Curry hinzugeben.

Das Komma als Trenner zweier Satzteile:

> Den Speck würfeln, den Käse reiben.

Hinweis: Das Semikolon und der Doppelpunkt werden in dieser Arbeit nicht genauer betrachtet. Zurzeit führt dies zu einer neuen Taskerzeugung.

Zahlen werden zunächst nicht besonders behandelt, sondern erst, wenn ein darauf folgendes Substantiv auftritt. Diese werden analog zu Adjektiven und Adverbien direkt nach ihrer Verwendung gelöscht.

Präpositionen sind wiederum komplexere Gebilde und können Hinweise auf vorherige Tasks sein. Sie können aber auch nur beschreiben wie zwei unterschiedliche Zutaten (oder Gemische von Zutaten) „zusammen gefügt“ oder getrennt werden (in Kapitel 7.1 wird dies als join/split Bezeichnet). Steht eine Präposition am Satzanfang ist dies mit großer Wahrscheinlichkeit ein Zeichen dafür, dass dieser Task sich auf den Vorherigen bezieht.

BSP:
Präposition am Satzanfang → Bezug auf vorherigen Task:

> Mit Pfeffer würzen.

Präposition mitten im Satz → Beschreibung eines Umstands

> Die Sahne mit den beiden Eigelb verquirlen.

Obwohl **Verben** für die Taskgenerierung existentiell wichtig sind, denn ohne Tätigkeit kann kein Task bzw. nur ein unzureichender Task erstellt werden, werden diese nur in einer Variablen zwischengespeichert und nicht besonders behandelt. Hilfsverben werden analog behandelt.

Substantive benötigen die größte Beachtung, da hier zwei Fälle differenziert betrachtet werden, die wiederum in zwei Fälle untergliedert werden müssen:

1 Substantiv ist eine Zutat oder eine Entität
   - 1.1 Zutat/Entät wurde noch nicht verwendet im Rezept und bekommt deswegen die aktuelle Tasknummer als Referenznummer zugewiesen und wird als Argument in eine Argumentliste eingetragen. Für jeden Task wird eine neue Argumentliste gepflegt.
   - 1.2 Zutat/Entität wurde bereits verwendet. Referenz der Zutat/Entität wird ausgelesen und Referenzliste angehängt. Referenz der Zutat/Entität wird auf aktuellen Task geändert. Für jeden Task wird eine neue Referenzliste angelegt.

2 Substantiv ist augenscheinlich keine Zutat oder Entität
   Substantiv wird durch eine einfache Morphemanalyse überprüft ob es nicht doch eine Zutat/Entität ist
   - 2.1 wenn ja, dann verfahren wie unter 1
   - 2.2 wenn nein, dann Eintrag in Argumentliste, aber keine Pflege von Referenzen.

Für einzelne Substantive wird eine spezielle Verarbeitung eingepflegt:

- Minuten
  Falls dieses Substantiv auftritt, wird eine zuvor gesicherte Zahl davor geschrieben
- Watt
  Auch hier wird eine Zahl, die zuvor in einer Variablen gesichert wurde vor das Substantiv geschrieben

Abbildung 20 und Abbildung 21 zeigen eine vereinfachte Form dieser komplexen Kontrollstruktur.

Zuständige Klasse(n): WriteXML2, Task, TaskMemory

Klassenname	Funktion
WriteXML2	Verarbeitet über if-Kontrollstrukturen die Tasks
Task	Klasse zur Sicherung einzelner Tasks
TaskMemory	Klasse zur Sicherung von Substantiven (Zutat/Entität) und Verwaltung derer Referenzen
RezeptContentHandler	Liest aus XML-Dateien Tasks aus und schreibt diese in Task Objekte, die in einer ArrayListe in der Klasse Rezept verwaltet werden

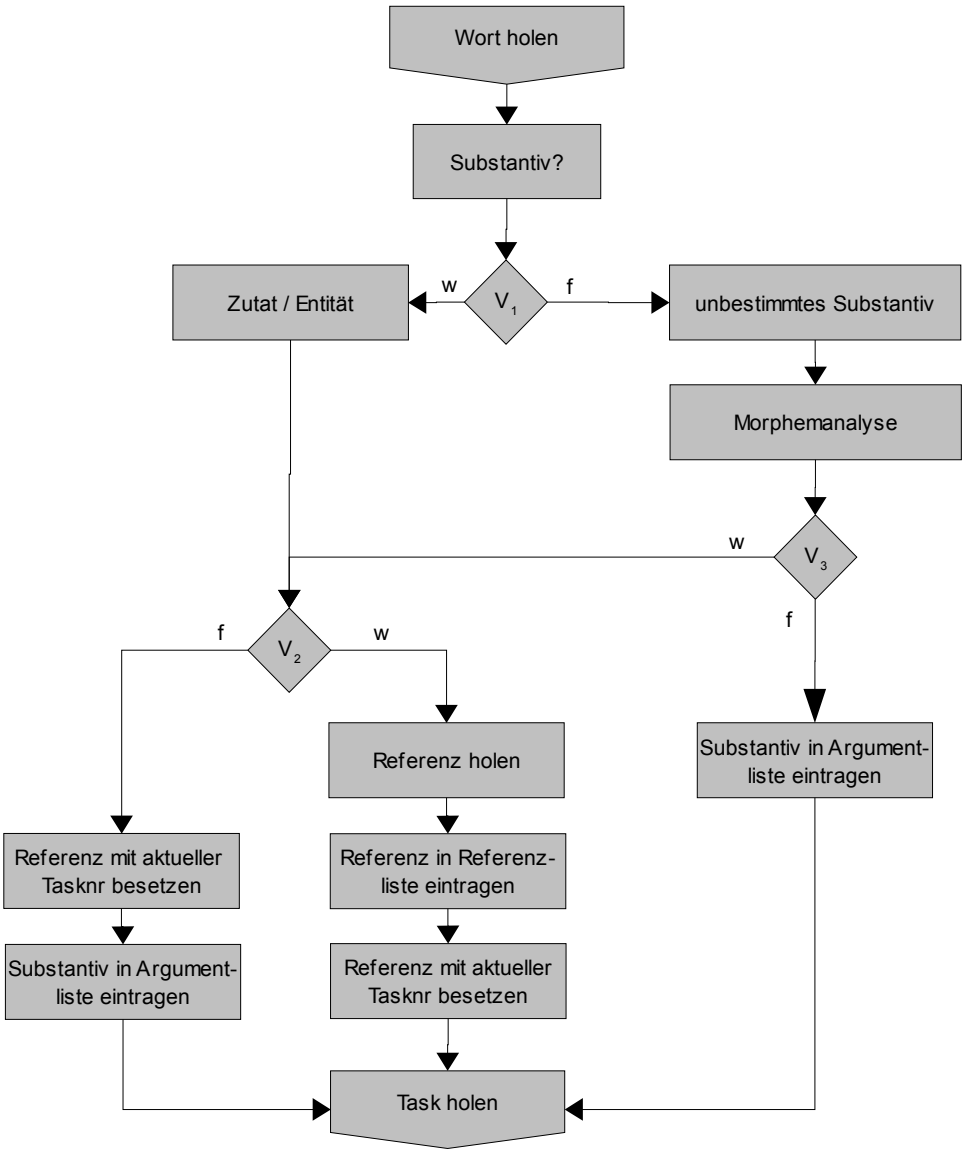

Abbildung 20: Spezielle Behandlung von Substantiven in WriteXML2

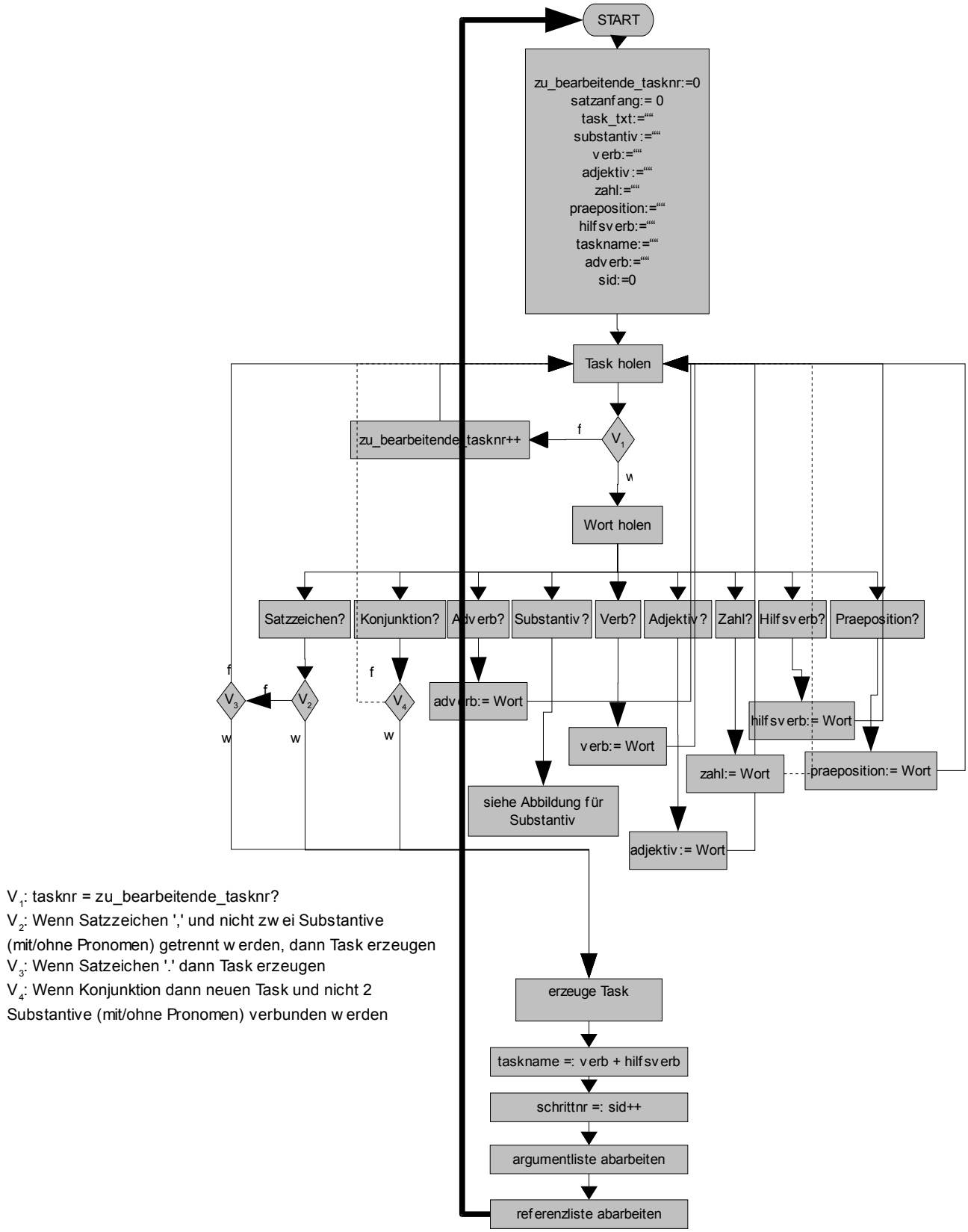

Abbildung 21: Ablaufplan WriteXML2

6.4.7 Generierung von Tasks mit Hilfe der Task Definition Language (TDL)

Die Task Definition Language wird wie zuvor erwähnt auch über den „Start-Button“ neben dem Menüpunkt „Task Definition Language“ gestartet. Die Generierung der Tasks beruht komplett auf Inhalt und Aufbau der XML-Datei, d.h. auch hier ist die Modularität voll gegeben. Wichtig als Informationsquelle ist hier also nur, dass eine XML-Datei mit Aufbau wie in Kapitel 4.1.1 gegeben ist.
Zunächst wird wieder ein XML-Sax-Parser gestartet, der nach und nach die Informationen für die Task Generierung einliest. Der ContentHandler dieses Parsers (TDLContentHandler) ist ein wenig anders als die vorherigen, da dieser nicht nur auf die End-Tags reagiert sondern auch auf Start-Tags. Dies ist nötig, da Attribute innerhalb der XML-Tags verwendet werden und die sofort nach Sax-Parser Ereignissen verarbeitet werden müssen. Dies betrifft die XML-Tags „SCHRITT“ und „ARGUMENT“.
Die Steuerung zur Erzeugung eines Tasks innerhalb der TDL übernimmt das **schließende** XML-Tag „SCHRITT“. Tritt dieses auf, wird der aktuelle Task in TDL Form geschrieben und anschließend alle Variablen und Listen geleert. Das Schreiben eines Tasks in die Datei (Rezeptname.tdl im Ordner TDL) übernimmt der Klasse TDL. In der Klasse TDL befinden sich auch alle Variablen und Listen, die vom ContentHandler gefüllt werden.

Verwendete Klasse(n): XML-Parser mit TDLContentHandler, TDL

Klassenname	Funktion
TDLContentHandler	XML-Parser zum Parsen der XML-Rezept Datei
TDL	Verantwortliche Klasse zum Schreiben der Tasks in Form der TDL

6.4.7.1 Ablaufplan TDL-Generierung

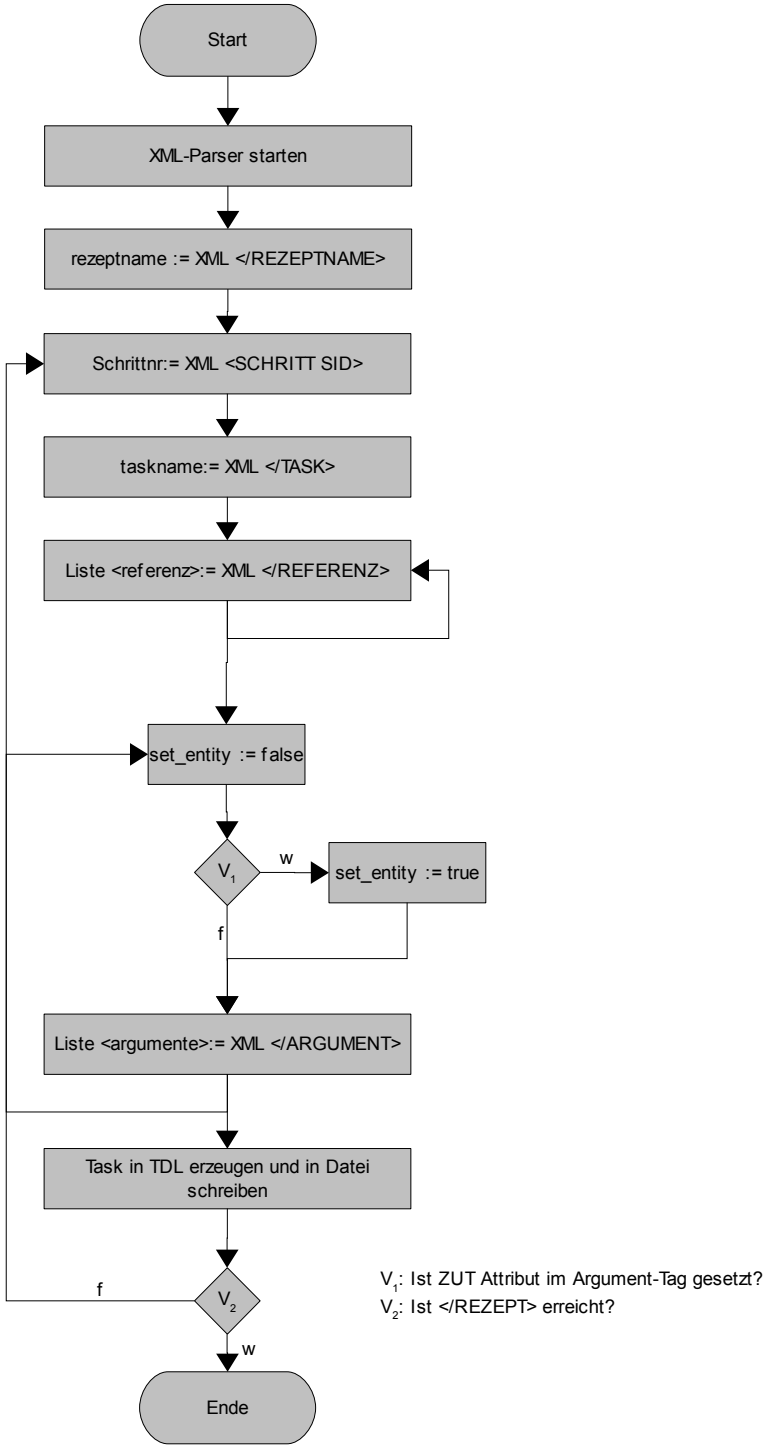

Abbildung 22: Ablaufplan TDL Generierung

6.5 Installationsanleitungen

Der Software Agent kann auf einer beliebigen Plattform zum Einsatz gebracht werden. Er benötigt eine Java Virtual Maschine ab Version 1.4 um die Java-Bytecode Dateien zum Laufen zu bringen. Zusätzlich wird ein Betriebssystem benötigt, welches eine grafische Oberfläche bereitstellt um die GUI des Agenten anzeigen zu können.
Die Klassen-Dateien, die zum Agenten gehören befinden sich alle im Verzeichnis Source auf der beiliegenden CD. Die Klassen-Dateien, die zur Satzgliedanalyse gehören befinden sich innerhalb des Source Verzeichnisses im Verzeichnis Sourcecode getrennt von den Agenten-Dateien.
Für die Satzgliedanalyse wird zusätzlich ein Oracle DBMS benötigt auf dem die zur Satzgliedanalyse zugehörige Datenbank installiert ist. Alle Data Dictionary Anweisungen liegen im ASCII-Text im SQL-Format auf der beiliegenden CD vor, genauso wie alle bisher eingefügten Datensätze.
Für die JDBC Schnittstelle in der Satzgliedanalyse wird ein beiliegender Oracle Datenbank-Treiber benötigt.
Eine Netzwerkanbindung ist dringend notwendig, da der Agent auf eine Datenbank und das Internet Zugriff benötigt.

Hinweis:
Der Agent ist nicht von der CD aus zu benutzen, da er seine Resultate im Dateisystem ablegen möchte.

Schnellübersicht:

Modul	Benötigt:
Agent Cook	- Grafisches Betriebssystem - JAVA ab Version 1.4 - 1 MB Festplatten Platz
Satzgliedanalyse	-Oracle DBMS ab Version 8g (auch XE Versionen) -Tabellenstruktur und Inhalte der Satzgliedanalyse -Oracle Datenbank Treiber (ojdbc14.jar)
Netzwerk	Internet und Verbindung zur Datenbank

7 Resultate und Ausschau

7.1 Analyse der Ergebnisse

Die Ergebnisse (XML- und TDL-Dateien) des Agenten zeigen noch einige Schwierigkeiten bei der Analyse eines vom Menschen geschriebenen Rezepts auf. Die deutsche Sprache ist so vielfältig in ihren Regeln und Möglichkeiten, einen Satz zu bilden, so dass eine kontextabhängige Analyse unbedingt von Nöten ist.

Es gibt zwei Möglichkeiten die Resultate des Agenten zu bewerten:

- Die Funktionalität eines kompletten Rezepts
- Die Funktionalität eines Rezepts unterteilt in seine Tasks

Die Funktionalität eines kompletten Rezepts
Hier muss man ganz klar sagen, dass der Agent noch an seine Grenzen stößt. Er vermag nur sehr wenige Rezepte in ihrer Gesamtheit in eine Task-Sprache zu überführen. Im Durchschnitt werden etwa nur 5-10 Prozent aller Rezepte in eine korrekte Form gebracht, die ein realer Roboter ohne Störung nachkochen könnte. Häufig scheitert ein Rezept an Kleinigkeiten, die aber Folgefehler auslösen und so den ganzen Ablauf durcheinander bringen.

Die Funktionalität eines Rezepts unterteilt in seine Tasks
Hier entwickelt der Agent seine Stärken, denn im Gegensatz zur Betrachtung eines kompletten Rezepts sieht man nach diesen Kriterien, dass ein jedes Rezept zu 80-90 Prozent korrekt in die Task-Sprache übersetzt wird. Meist baut sich ab Mitte des Rezepts eine Komplexität auf, die sich ohne die Fähigkeit des Menschen, selber neue Schlüsse zu ziehen, nur sehr schwierig direkt in eine Task-Sprache umsetzen lässt.
Dieses hohe Ergebnis zeigt aber, dass es möglich ist, ein Rezept maschinell in eine Arbeitsanweisung für einen Roboter umzusetzen.

7.1.1 Analyse der XML und TDL Dateien

Die resultierenden Dateien beschreiben eine Art Sequenz, die nach und nach abgearbeitet werden muss. Hierbei bilden die Zutaten und auch Entitäten eine Art Einzel Sequenzen, die bis zu einem Zeitpunkt t_i austauschbar wären, d.h. es würde keinen Unterschied machen, ob nun zuerst Zwiebeln geschnitten werden oder zunächst der Käse gerieben wird (siehe Abbildung 23, Einzel Sequenz (1,8), (2,3,4), (5), (6), (7)) . Erst wenn verschiedene verarbeitete Zutaten miteinander kombiniert werden, muss darauf geachtet werden, dass eine bestimmte zeitliche Abfolge eingehalten wird (z.B. (4,8) oder (5,6)). Vorher könnte ein Roboter auf Grund dieser Eigenschaft eigenmächtig entscheiden, ob er vielleicht einen anderen Task auf Grund von mangelnden Ressourcen oder auf Grund zweier ähnlicher Tasks, etwa das Schneider zweier Zutaten, bevorzugt um ein erneutes Heranschaffen eines Arbeitsgerätes zu vermeiden.

- Das Vermischen von Zutaten (oder Entitäten), von denen mindestens schon eine zuvor verarbeitet wurde, wird im folgenden als „**Join**“ bezeichnet

- Das Trennen von Zutaten (oder Entitäten) wird im folgenden als „**Split**“ bezeichnet

In Abbildung 23: Sequenzdiagramm des Rezepts "Zwiebelkuchen" ist dieser Zusammenhang an Hand des „Zwiebelkuchen“ Rezepts grafisch dargestellt.

7.1.2 Einführung von „stummen“ Tasks

Durch die Einführung der Splits und Joins tritt in einigen Rezepten auch eine neue Art von Tasks auf. Diese Tasks bezeichne ich als sogenannte „stumme“ Tasks, denn sie existieren in der Rezeptbeschreibung nicht, sind aber für die Weiterführung des Rezepts sinnvoll bzw. sogar sehr wichtig.
Manche „stumme“ Tasks werden nicht mehr benötigt und können deswegen auch ausgelassen werden, auf andere hingegen wird zu einem späteren Zeitpunkt wieder Bezug genommen.

BSP: Stummer Task aus Zwiebelkuchen Rezept

```
<SCHRITT SID="15">
  <TXT>Das Blech herausnehmen und</TXT>
  <TASK>herausnehmen</TASK
  <ARGUMENT AID="1">Blech</ARGUMENT>
</SCHRITT>
```

```
<SCHRITT SID="16">
  <TXT>den Backofen auf 200° aufheizen</TXT>
  <TASK>aufheizen</TASK
  <ARGUMENT AID="1">Backofen</ARGUMENT>
  <ARGUMENT AID="2">200°</ARGUMENT>
</SCHRITT>
```

In Schritt 15 wird das Blech (samt Zutaten-Gemisch) aus dem Backofen herausgenommen und für eine spätere Weiterverarbeitung zur Seite gestellt. In Schritt 16 wird der leere Backofen aufgeheizt. Das Blech aus 15 wird zu einem „stummen“ Task, der als „warten auf Tasknummer 16“ bezeichnet werden könnte. In Abbildung 23 ist er als Tasknummer 15' angegeben.
Auch Tasknummer 17 könnte man einen „stummen“ Task voran schieben:

```
<SCHRITT SID="17">
  <TXT>Die Backzeit beträgt etwa 30 Minuten.</TXT>
  <TASK>beträgt</TASK
  <ARGUMENT AID="1">Backzeit</ARGUMENT>
  <ARGUMENT AID="2">30 Minuten</ARGUMENT>
</SCHRITT>
```

Hier ist natürlich mit gesundem Menschenverstand zu verstehen, dass zunächst das Blech mit dem Zutaten-Gemisch wieder in den Backofen eingeschoben werden muss. Da davon aber nichts im Rezept steht, würde ein Roboter das ohne Vorwissen nach Bearbeitung der Tasks nicht tun. Auch hier müsste also ein „stummer“ Task eingeführt werden, der folgendermaßen aussehen sollte:

```
<SCHRITT SID="17'">
  <TXT>stummer Task</TXT>
  <TASK>join</TASK
  <REFERENZ>15'</REFERENZ>
  <REFERENZ>16</REFERENZ>
</SCHRITT>
```

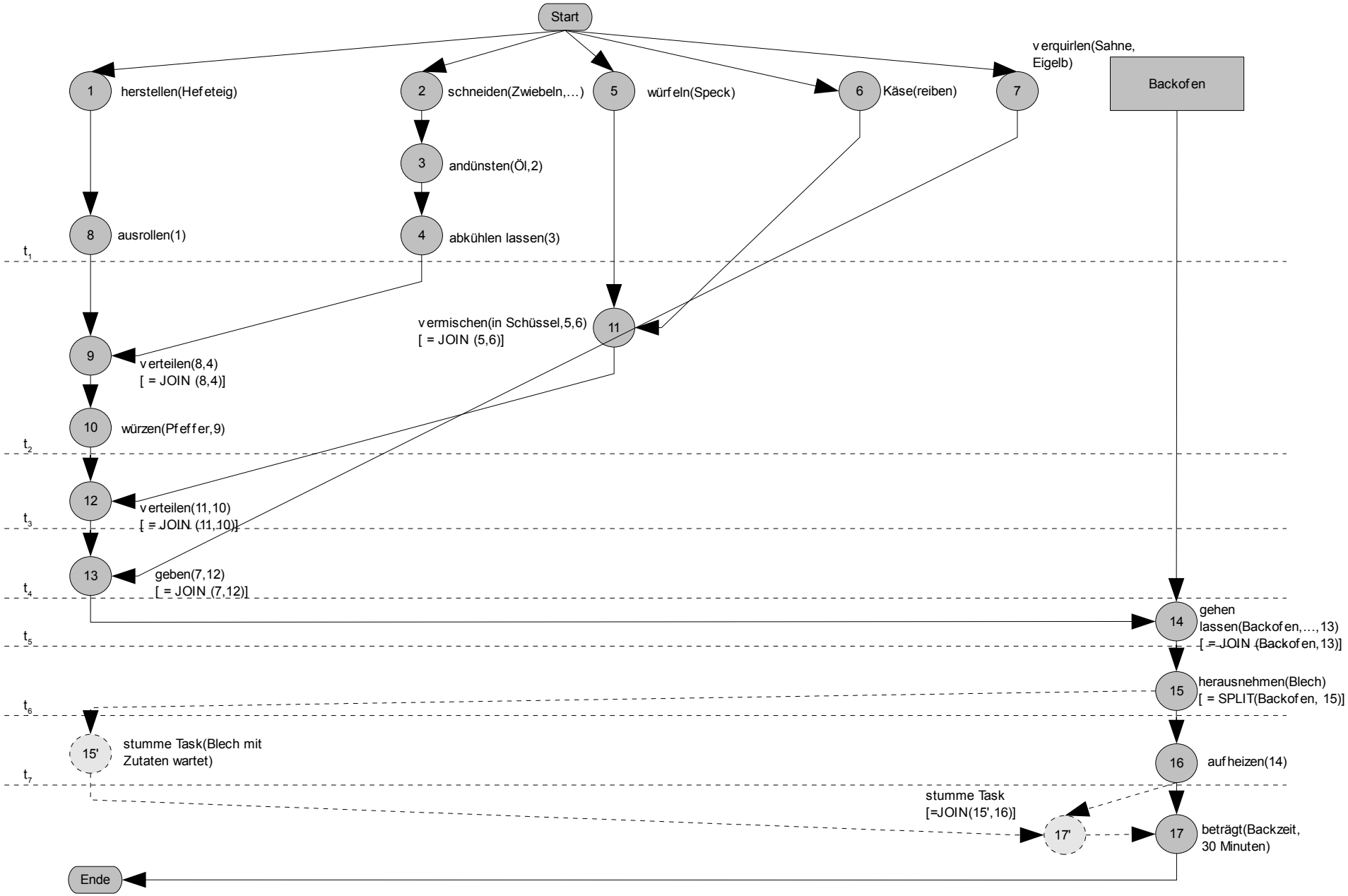

Abbildung 23: Sequenzdiagramm des Rezepts "Zwiebelkuchen"

7.2 Ausschau

7.2.1 Möglichkeiten zur Verbesserung der maschinellen Task Generierung

Innerhalb des Agenten sind schon einige Schritte getätigt um die Ergebnisse der bisher in dieser Arbeit beschriebenen Methoden zu verbessern. Meist sind dies in den Quellcode einprogrammierte feste Vorgaben, die dem Agenten direkt sagen, was er in einer bestimmten Situation zu tun hat.

7.2.1.1 Einpflegen bestimmter Wörter

Im Quellcode sind einige Wörter eingepflegt, die außerhalb der grundsätzlichen Programmlogik funktionieren. Diese funktionieren als Korrektur und Hilfe für den Agenten wie auch für den Benutzer.

Wort	Funktion
darüber	Das Wort „darüber“ lässt automatisch eine Referenz auf den vorherigen Task anlegen
Minuten	stellt automatisch Zahlenwerte wenn vorhanden vorne an
Watt	stellt automatisch Zahlenwerte wenn vorhanden vorne an
Backofen	wird automatisch als Entität gekennzeichnet
Pfanne	wird automatisch als Entität gekennzeichnet
vermischen	wird als Verb gekennzeichnet (wird von Satzgliedanalyse als Adjektiv gekennzeichnet)
Öl	wird als Substantiv gekennzeichnet (wird von Satzgliedanalyse als nicht erkanntes Wort gekennzeichnet)
garen	wird als Verb gekennzeichnet (wird von Satzgliedanalyse als Adjektiv gekennzeichnet)
untermischen	wird als Verb gekennzeichnet (wird von Satzgliedanalyse als Adjektiv gekennzeichnet)
Anschließend	wird als Präposition gekennzeichnet

Wort	Funktion
	(wird von Satzgliedanalyse als nicht erkanntes Wort gekennzeichnet)
darübergeben	Schreibfehler wird korrigiert in „darüber geben“

Tabelle 6: Wörter-Korrekturen des Software Agenten

7.2.1.2 Verwalten einer Blacklist

Aktuell verfügt der Agent über eine Klasse „Blacklist.java“, die über eine ASCII-Text Datei („Blacklist.txt“) Wörter einliest, die er grundsätzlich ignorieren soll. Derzeit stehen folgende Wörter in dieser Liste:

Wort in Blacklist	Warum?
Für	Das Wort „für“ zu Beginn eines Satzes ist für die Task Generierung irrelevant und führt in manchen Fällen zu falschen Referenzen
Appetit	Das Substantiv „Appetit“ soll nicht in Argumentliste oder Referenzliste aufgenommen werden, da es keine Bedeutung für die Herstellung des Rezepts hat
Geschmack(	„Geschmack“ mit anschließender Klammer steht hier beispielhaft für Wörter in Rezepten die eine Klammerung nach sich ziehen. Dies kann zur Zeit noch nicht vom Agenten verarbeitet werden und soll deswegen ignoriert werden

Tabelle 7: Inhalt der Blacklist.txt

7.2.1.3 Möglichkeiten zum Eingriff zur Laufzeit des Agenten

Zur Laufzeit des Agenten lassen sich zur Zeit an zwei Stellen Korrekturen durchführen:

- Eine erste Korrekturmöglichkeit besteht unter dem Menüpunkt „Rezept parsen – Anzeigen“. Hier ist es möglich manuelle Anpassungen zu tätigen, wie Rechtschreibfehler zu korrigieren, Sonderzeichen zu entfernen oder auch ganze Sätze, die für die Verarbeitung eines Rezepts irrelevant sind, zu löschen. Erst nach dieser Korrektur wird die Satzgliedanalyse mit der korrigierten Version des Rezepts gestartet.

- Eine zweite Korrekturmöglichkeit verbirgt sich hinter dem Menüpunkt „Analyse – Anzeigen“. Hier können Ergebnisse der Satzgliedanalyse verbessert und Substantive als Zutat oder Entität gekennzeichnet werden, die nicht als solche erkannt wurden bzw. die Kennzeichnung kann entfernt werden, wenn diese fälschlicherweise als solche markiert wurden.

Hinweis: Natürlich ist auch ein Eingriff in die erstellten Dateien zur Laufzeit möglich. Dies ist aber nicht zu empfehlen, da dies eher zu inkonsistenten Datensätzen führt.

7.2.2 Möglichkeiten zur Weiterentwicklung und Verbesserung des Agenten

7.2.2.1 Wissensakquisition

Im Modul der Wissensakquisition sind nur „kosmetische“ Verbesserungen wenn überhaupt von Nöten, da dieses Modul funktionell genau das leistet, was von ihm verlangt wird.

Automatische Rezeptanbieter Auswahl
Verbessern könnte man die automatische Erkennung einer Webseite aus der URL. Zur Zeit wird eine URL eingegeben und zusätzlich die Rezeptseite über einen Radio Button ausgewählt. Das Extrahieren einer Seite aus der URL ist aber durchaus möglich und würde dem Anwender des Agenten einen Arbeitsschritt mehr ersparen und den Agenten einen Schritt autonomer machen.

Web Crawler Funktionalität
Eine zusätzliche Funktionalität könnte das automatische Akquirieren von neuen Rezepten sein, dazu müsste der Agent entsprechende Links aus der Webseite extrahieren und selbstständig die Daten holen und verarbeiten.

7.2.2.2 Wissensextraktion und Nachbearbeitung der Ergebnisse

Innerhalb der Wissensextraktion können einige Mechanismen zur Verbesserung der Ergebnisse führen.

Eingabe-Maske für Parameter-XML Dateien
Eine grafisch unterstützte Benutzeroberfläche zur Eingabe neuer Parameter für einen neuen Anbieter von Rezepten.

Nachbearbeitung der Zutatenliste
Innerhalb des Agenten könnte eine Funktion zur Bearbeitung der Zutatenliste implementiert werden. Hier könnten fehlerhafte Angaben korrigiert und fehlende Zutaten hinzugefügt werden.

Automatische Korrektur von Rechtschreibfehlern
Diese Funktion könnte ähnlich implementiert werden wie sie in Office Systemen integriert ist, so dass mögliche Fehler entsprechend in der Anzeige hervorgehoben werden oder direkt und automatisch durch die wahrscheinlichste Korrektur ersetzt werden. Offene und frei zu verwendende Wörterbücher zur Korrektur sind im Internet zu erhalten.

Entfernung von Meinungen und Wünschen
Sätze in denen Worte wie „guten Appetit“, „ich wünsche“, „viel Spaß“ oder ähnliche Phrasen enthalten sind, könnten automatisch als irrelevant markiert werden und nicht mehr weiter verarbeitet werden.

7.2.2.3 Satzgliedanalyse

Da dieser Teil nur bedingt Inhalt dieser Arbeit war, ist hier bis auf zuvor erwähnte Änderungen nicht weiterentwickelt worden. Dass die Satzgliedanalyse aber noch zu verbessern ist, steht außer Frage. Die allgemeine Verbesserung von Wörtern sollte dabei natürlich oberste Priorität haben. Zusätzlich könnten folgende Punkte in Zukunft implementiert werden:

- zusätzlich semantische Anreicherung der Ergebnisse mit
  - Satznummer
  - Wortnummer innerhalb eines Satzes
  - Wortnummer innerhalb des gesamten zu analysierenden Textes
  - Wort gehört zu einer Aussage, zu einer Frage oder zu einem Ausruf
  - Konjunktion verbindet zwei Satzteile oder eine Aufzählung
  - Satzzeichen ',' verbindet zwei Satzteile oder eine Aufzählung
- Überarbeitung der Logik zur Erkennung, da Worte die korrekt in Datenbank stehen, trotzdem falsch erkannt werden
- Hinzufügen einer grafischen Benutzeroberfläche
- Benutzen der Satzgliedanalyse als verteiltes System, z.B. Senden eines Textes an den Dienst und Empfangen der Protokolldateien als Antwort
- integrierte Benutzerschnittstelle für Datenbankzugriffe (Wörter einpflegen, Regeln erstellen, ...)

7.2.2.4 Verbesserung der Mechanismen zur Erstellung der XML- und TDL-Dateien

Dieser Teil der Verbesserungen ist abhängig von der vorhergehenden Satzgliedanalyse, denn sollte diese bessere und vor allem sicherere Ergebnisse liefern, so könnte die Auswertung der Satzglieder nicht nur über die Wortarterkennung laufen, sondern auch über die Phrasenerkennung.

Phrasenerkennung
Aktuell schaut der Agent auf vorhergehende Wortarten und kann so zum Beispiel Adjektive oder Adverbien entsprechend einem Substantiv oder Verb zuordnen. Würde

man an dieser Stelle (gesicherte) Informationen der Phrasenerkennung haben, müsste dieser Abgleich mit vorherigen und folgenden Worten nicht mehr stattfinden.

Umstandsbestimmung zur Referenzenbildung
Auch eine Verbesserung der Umstandsbestimmung einer Präposition würde im Zusammenhang mit der verbesserten Phrasenerkennung die Referenzenbildung verbessern und zu weniger falschen oder nicht erzeugten Referenzen führen.

7.2.3 Benutzen der Funktionalität der TDL

In Kapitel 4.2 sind einige Funktionen als Anforderung genannt, die in dieser Arbeit nicht praktisch umgesetzt wurden. Hierbei sind besonders die Funktionen zur Abbildung von Parallelitäten und Schleifen hervorzuheben.

7.2.3.1 Parallelität

In Rezepten existieren einige Schlüsselwörter, die Parallelitäten, also Arbeitsschritte die gleichzeitig erledigt werden können, andeuten. Da diese Funktionalität in der TDL von Hause aus gegeben ist, lassen sich Schritte, die gleichzeitig behandelt werden sollen auch als solche abbilden. Der Agent müsste dafür nur auf die entsprechenden Schlüsselwörter reagieren. Hauptsächlich sind diese Schlüsselwörter Präpositionen, die einen temporalen Hintergrund haben, aber es können auch andere Gebilde, die einen zeitlichen Ablauf beschreiben, sein.

- während
- währenddessen
- in der Zwischenzeit
- gleichzeitig
- parallel dazu
- im selben Augenblick
- nebenher
- unterdessen
- solange
- ...

Taucht eines dieser Wörter zu Beginn einer Arbeitsanweisung auf, so ist die Wahrscheinlichkeit, dass hier zwei Schritte parallel abzuarbeiten sind sehr groß.

BSP:

```
Die Nudeln kochen.
Währenddessen die Paprika schälen.
```

Aus dieser Arbeitsanweisung könnte folgende XML-Task Anweisung entstehen:

```
...
<PAR>
  <SCHRITT SID="1">
    <TXT>Die Nudeln kochen.</TXT>
    <TASK>kochen</TASK>
    <ARGUMENT AID="1" ZUT="TRUE">Nudeln</ARGUMENT>
  </SCHRITT>
  <SCHRITT SID="2">
    <TXT>Währenddessen die Paprika schälen.</TXT>
    <TASK>schälen</TASK>
    <ARGUMENT AID="1" ZUT="TRUE">Paprika</ARGUMENT>
  </SCHRITT>
</PAR>
...
```

Die TDL-Erzeugung könnte über die <PAR>-Elemente die Parallelität feststellen und wie folgt generieren:

```
...
TASK kochen/schaelen
#VARIABLES
entity Paprika''
entity Nudeln''
int task_nr
int result_kochen
int result_schaelen
#BODY
DO (
  PAR (
    result_kochen = kochen(Nudeln),
    result_schaelen = schaelen(Paprika)
  )
#END
...
```

7.2.3.2 Schleifen

Analog zu parallelen Abläufen können auch Arbeitsanweisungen so lange wiederholt werden, bis eine zuvor bestimmte Bedingung eintritt.

BSP:

```
Nudeln bissfest kochen.
```

Aus dieser Arbeitsanweisung könnte folgende XML-Task Anweisung entstehen:

```
...
<WHILE BED="BISSFEST">
  <SCHRITT SID="1">
    <TXT>Nudeln bissgest kochen.</TXT>
    <TASK>kochen</TASK>
    <ARGUMENT AID="1" ZUT="TRUE">Nudeln</ARGUMENT>
  </SCHRITT>
</WHILE>
...
```

Oder mit DO-UNTIL Konstrukt in TDL

```
...
TASK kochen
#VARIABLES
entity Nudeln''
int task_nr
int result
#BODY
DO (
  DO (
    result_kochen = kochen(Nudeln),
  )UNTIL (bissfest)
#END
...
```

7.3 Fazit

Wie wichtig das Thema der Wissensakquisition und Wissensrepräsentation in den nächsten Jahren sein wird ist wahrscheinlich noch gar nicht auszumachen. Das es aber einen großen Teil der Forschung beanspruchen wird steht für mich außer Frage. Die Informationstechnologie beschäftigt sich nun schon seit einigen Jahren mit dem Thema der Wissensrepräsentation und zeigt erste Wege mit der Einführung von OWL und, der in dieser Arbeit nicht erwähnten, XML Topic Maps.
Die geplante Entwicklung des Internets hin zu einem gigantischen semantischen Netz, in welchem wenn möglich alle Informationen miteinander verbunden sind, wird sicherlich noch ein paar Jahre auf sich warten lassen, doch das Fundament ist längst gelegt. Nur so ist die geplante Technologiesierung und Vereinfachung von alltäglichen Arbeitsschritten möglich und lässt Maschinen, Robotern oder auch Software Agenten, wie der in dieser Arbeit, mit dem uns vorhandenen Wissen umgehen.

Innerhalb dieser Arbeit sind einige interessante Problemstellungen aufgetreten, die zukünftig noch weiter erforscht werden sollten.

- Inwiefern kann ein solcher Roboter/Software Agent irgendwann einmal autark und autonom agieren?
- Wie kann er sich selbst mit neuem Wissen versorgen und daraus lernen?
- Wie kann er sein Wissen teilen oder geteiltes Wissen von anderen Agenten benutzen?

Auch stellt sich die Frage, ob die deutsche Sprache wirklich zu 100% zu Parsen ist oder aber sind die Kontextbezüge, die benötigt werden so stark gestreut, dass manche semantische Information einfach nicht maschinell gewonnen werden kann. Die Informatik ist da heute schon sehr weit, aber ob sie in naher Zukunft am Ziel ankommen wird, ist noch lange nicht klar.

Die Arbeit an diesem Projekt und besonders die Ergebnisse dieses Projekts geben ein eindeutiges Zeichen, dass hier weitergearbeitet werden sollte. Im vorherigen Kapitel sind einige Vorschläge zur Weiterentwicklung gemacht, die den Agenten um einige wichtige Funktionen verbessern würden und auch die Ergebnisresultate steigern könnten.
Die Möglichkeiten diesen Agenten auf viele andere Themengebiete anzusetzen sind natürlich gegeben. Vorstellbar sind jedenfalls einige:

- Verarbeitung von Bauanleitungen (Modellbau, Auto, etc.)
- Verarbeitung von Rezepturen in der Medizin (für Apotheken Roboter)
- Verarbeitung von Installationsanleitungen diverser Software
- Verarbeitung von Reparaturanleitungen

Im Grunde lässt sich fast alles mit einem ähnlich implementierten Agenten verarbeiten was als Information irgendwo vorhanden ist und in physische Arbeitsschritte zu unterteilen und zu bewältigen ist.

Als Ergebnis dieser Arbeit bleibt ein funktionstüchtiger Software Agent, der bei weitem noch nicht an seine Grenzen gestoßen ist, aber sich auf einem guten Weg befindet, selbstständig Wissen zu akquirieren und dies so aufzuarbeiten, dass es maschinell weiter zu verarbeiten ist.

8 Anhang

8.1 Quellenverzeichnis

[1] Internetquelle: Social Media Blog
zuletzt besucht: 13.02.2009
http://www.social-media-blog.de/tag/semantisches-web/

[2] Internetquelle: W3C - XML-Schema
zuletzt besucht: 13.02.2009
http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

[3] Internetquelle: W3C – OWL
zuletzt besucht: 13.02.2009
http://www.w3.org/2004/OWL/

[4] Internetquelle: W3C – OWL-Semantics
zuletzt besucht: 13.02.2009
http://www.w3.org/TR/owl-semantics/

[5] Handbuch der KI
G. Görz, C.-R. Rollinger, J. Schneeberger (Hrsg.)
4. Auflage, Oldenbourg 2003

[6] A Task Definition Language for Virtual Agents
Journal of WSCG, Vol. 11 No. 1
ISSN: 1213-6972
Plzen, Czech Republic

[7] Das System: Satzgliedanalyse: Portierung der Anwendung von Pro*C
nach JDBC und ihre Erweiterung auf ein Oracle DBMS
Diplomarbeit Sascha Klaus Rudolf
02.07.2007

[8] Internetquelle: W3C – XML
zuletzt besucht: 13.02.2009
http://www.w3.org/XML/

[9] Datenbanken und Wissensrepräsentation
Vorlesungsmitschrift
Sommersemester 2008 , Prof. Dr. phil. G. Büchel

[10] Internetquelle: Wikipedia - OWL
zuletzt besucht: 13.02.2009
http://de.wikipedia.org/wiki/Web_Ontology_Language

[11] Internetquelle: Zwiebelkuchen Rezept
Chefkoch.de
zuletzt besucht: 13.02.2009
http://www.chefkoch.de/rezepte/835988123291/Zwiebelkuchen.html

8.2 Tabellenverzeichnis

Tabellenverzeichnis

Tabelle 1: OWL Dialekte....16
Tabelle 2: XML-Elemente eines Rezepts....43
Tabelle 3: Eigenschaften einiger Task-Sprachen....46
Tabelle 4: Eigenschaften des "Agent Cook"s....55
Tabelle 5: GUI-Klassen....63
Tabelle 6: Wörter-Korrekturen des Software Agenten....83
Tabelle 7: Inhalt der Blacklist.txt....83

8.3 Abbildungsverzeichnis

Abbildungsverzeichnis

Abbildung 1: nicht strukturierte zu strukturierte Daten....11
Abbildung 2: RDF-(1) Ressource hat Wert W /(2) Ressource verweist auf Ressource. .12
Abbildung 3: Definierte XML-S-Datentypen....13
Abbildung 4: Aufbau von XML zu OWL....14
Abbildung 5: Handlungen eines Agenten zur Lebenszeit....21
Abbildung 6: ERD der Küchenwelt....26
Abbildung 7: Sequenzdiagramm Satzgliedanalyse....35
Abbildung 8: Informationsflussdiagramm des Software Agenten....56
Abbildung 9: Module der Wissensakquisition und -extraktion....57
Abbildung 10: Module und Erweiterungen des SAX-Parsers....58
Abbildung 11: Module der Satzgliedanalyse....58
Abbildung 12: Module der XML-Task-Sprachen Generierung....59
Abbildung 13: Module der TDL Generierung....59
Abbildung 14: Ablaufplan Agent Cook....61
Abbildung 15: Hauptmenü Agent Cook....62
Abbildung 16: Zwiebelkuchen Rezept....64
Abbildung 17: Anzeige des HTML-Codes in Agent Cook....65
Abbildung 18: Anzeige und Korrektur eines Rezepts....66
Abbildung 19: Wörter anzeigen und ändern....68
Abbildung 20: Spezielle Behandlung von Substantiven in WriteXML2....73
Abbildung 21: Ablaufplan WriteXML2....74
Abbildung 22: Ablaufplan TDL Generierung....76
Abbildung 23: Sequenzdiagramm des Rezepts "Zwiebelkuchen"....81

8.4 Beispiel Parametrierung für Chefkoch.de

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE parameter SYSTEM "AgentCookParams.dtd">
<parameter>
  <beginn>!-- BEGIN REZEPT --#</beginn>
  <rezeptname>
    <rezeptname-start>#h1 style=#padding-bottom:10px;##</rezeptname-start>
    <rezeptname-end>#/h1#</rezeptname-end>
  </rezeptname>
  <personen>
    <personen-start>size=#2# value=#</personen-start>
    <personen-end># style=</personen-end>
  </personen>
  <zutaten>
    <zutaten-start>table class=#zutaten##</zutaten-start>
    <zutat>
      <zutat-start>#td align=#right##</zutat-start>
      <zutat-menge>#nbsp;#nobr#</zutat-menge>
      <zutat-einheit>#/nobr#</zutat-einheit>
      <zutat-bez-start>#td#</zutat-bez-start>
      <zutat-bez-end>#/td#</zutat-bez-end>
    </zutat>
    <zutaten-end>/table#</zutaten-end>
  </zutaten>
  <rezept-description>
    <rezept-description-start>/SCRIPT#</rezept-description-start>
    <rezept-description-start-foto>#span class=#n##</rezept-description-start-foto>
    <rezept-description-end>#!-- google_ad</rezept-description-end>
  </rezept-description>
  <end></end>
</parameter>
```

8.5 DTD der Parameter Dateien

```
<!ELEMENT parameter (beginn,rezeptname,personen,zutaten,rezept-description,end)>
<!ELEMENT rezeptname (rezeptname-start,rezeptname-end)>
<!ELEMENT personen (personen-start, personen-end)>
<!ELEMENT zutaten (zutaten-start,zutat,zutaten-end)>
<!ELEMENT     rezept-description     (rezept-description-start,rezept-description-start-
foto,rezept-description-end)>

<!ELEMENT zutat (zutat-start, zutat-menge, zutat-einheit, zutat-bez-start, zutat-bez-
end)>

<!ELEMENT beginn (#PCDATA)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT rezeptname-start (#PCDATA)>
<!ELEMENT rezeptname-end (#PCDATA)>
<!ELEMENT personen-start (#PCDATA)>
<!ELEMENT personen-end (#PCDATA)>
<!ELEMENT zutaten-start (#PCDATA)>
<!ELEMENT zutaten-end (#PCDATA)>
<!ELEMENT zutat-start (#PCDATA)>
<!ELEMENT zutat-menge (#PCDATA)>
<!ELEMENT zutat-einheit (#PCDATA)>
<!ELEMENT zutat-bez-start (#PCDATA)>
<!ELEMENT zutat-bez-end (#PCDATA)>
<!ELEMENT rezept-description-start (#PCDATA)>
<!ELEMENT rezept-description-start-foto (#PCDATA)>
<!ELEMENT rezept-description-end (#PCDATA)>
```

8.6 Beispiel für XML-Protokoll-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE satz SYSTEM "Wortanalyse.dtd">
<satz>
	<satzteil>
		<wort>Den</wort>
		<wortart>Artikel</wortart>
		<phrase>1</phrase>
		<umstand></umstand>
	</satzteil>
	<satzteil>
		<wort>Teig</wort>
		<wortart>Substantiv</wortart>
		<phrase>1</phrase>
		<umstand></umstand>
	</satzteil>
	<satzteil>
		<wort>ausrollen</wort>
		<wortart>Verb im Infinitiv</wortart>
		<phrase>0</phrase>
		<umstand></umstand>
	</satzteil>
	<satzteil>
		<wort>.</wort>
		<wortart>Satzzeichen</wortart>
		<phrase>0</phrase>
		<umstand></umstand>
	</satzteil>
</satz>
```

8.7 DTD der XML-Protokoll-Dateien (Wortanalyse.dtd)

```
<!ELEMENT satz (satzteil+)>
<!ELEMENT satzteil (wort, wortart, phrase, umstand)>

<!ELEMENT wort (#PCDATA)>
<!ELEMENT wortart (#PCDATA)>
<!ELEMENT phrase (#PCDATA)>
<!ELEMENT umstand (#PCDATA)>
```

8.8 Beispiel Rezept in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE REZEPT SYSTEM "Rezept.dtd">
<REZEPT>
	<REZEPTNAME>Zwiebelkuchen</REZEPTNAME>
	<ZUTATENLISTE>
		<ZUTAT>
			<BEZEICHNUNG>Mehl</BEZEICHNUNG>
			<MENGE>400.0</MENGE>
			<EINHEIT>g</EINHEIT>
		</ZUTAT>
		<ZUTAT>
			<BEZEICHNUNG>Hefe</BEZEICHNUNG>
			<MENGE>20.0</MENGE>
			<EINHEIT>g</EINHEIT>
		</ZUTAT>
		<ZUTAT>
			<BEZEICHNUNG>Wasser, lauwarm</BEZEICHNUNG>
			<MENGE>250.0</MENGE>
			<EINHEIT>ml</EINHEIT>
		</ZUTAT>
```

```
        <ZUTAT>
              <BEZEICHNUNG>Salz</BEZEICHNUNG>
              <MENGE>1.0</MENGE>
              <EINHEIT>TL</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Öl</BEZEICHNUNG>
              <MENGE>3.0</MENGE>
              <EINHEIT>EL</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Zwiebeln</BEZEICHNUNG>
              <MENGE>800.0</MENGE>
              <EINHEIT>g</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Speck, durchwachsen und geräuchert</BEZEICHNUNG>
              <MENGE>400.0</MENGE>
              <EINHEIT>g</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Käse Gouda</BEZEICHNUNG>
              <MENGE>250.0</MENGE>
              <EINHEIT>g</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Sahne</BEZEICHNUNG>
              <MENGE>200.0</MENGE>
              <EINHEIT>g</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Eier, davon das Eigelb</BEZEICHNUNG>
              <MENGE>2.0</MENGE>
              <EINHEIT>0</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Öl</BEZEICHNUNG>
              <MENGE>0.0</MENGE>
              <EINHEIT>0</EINHEIT>
        </ZUTAT>
        <ZUTAT>
              <BEZEICHNUNG>Pfeffer, frisch gemahlen</BEZEICHNUNG>
              <MENGE>0.0</MENGE>
              <EINHEIT>0</EINHEIT>
        </ZUTAT>
  </ZUTATENLISTE>
  <ARBEITSSCHRITTE>
        <SCHRITT SID="1">
              <TXT>Hefeteig herstellen .</TXT>
              <TASK>herstellen</TASK>
              <ARGUMENT AID="1" ZUT="FALSE">Hefeteig</ARGUMENT>
        </SCHRITT>
        <SCHRITT SID="2">
              <TXT>Die Zwiebeln in Ringe schneiden und</TXT>
              <TASK>schneiden</TASK>
              <ARGUMENT AID="1" ZUT="TRUE">Zwiebeln</ARGUMENT>
              <ARGUMENT AID="2">in Ringe</ARGUMENT>
        </SCHRITT>
        <SCHRITT SID="3">
              <TXT>in Öl andünsten .</TXT>
              <TASK>andünsten</TASK>
              <REFERENZ>2</REFERENZ>
        </SCHRITT>
        <SCHRITT SID="4">
              <TXT>Etwas abkühlen lassen .</TXT>
              <TASK>abkühlen lassen</TASK>
              <REFERENZ>3</REFERENZ>
        </SCHRITT>
        <SCHRITT SID="5">
              <TXT>Den Speck würfeln ,</TXT>
              <TASK>würfeln</TASK>
              <ARGUMENT AID="1" ZUT="TRUE">Speck</ARGUMENT>
```

```
            </SCHRITT>
            <SCHRITT SID="6">
                  <TXT>den Käse reiben .</TXT>
                  <TASK>reiben</TASK>
                  <ARGUMENT AID="1" ZUT="TRUE">Käse</ARGUMENT>
            </SCHRITT>
            <SCHRITT SID="7">
                  <TXT>Die Sahne mit den beiden Eigelb verquirlen .</TXT>
                  <TASK>verquirlen</TASK>
                  <ARGUMENT AID="1" ZUT="TRUE">Sahne</ARGUMENT>
                  <ARGUMENT AID="2" ZUT="TRUE">Eigelb</ARGUMENT>
            </SCHRITT>
            <SCHRITT SID="8">
                  <TXT>Den Teig ausrollen .</TXT>
                  <TASK>ausrollen</TASK>
                  <REFERENZ>1</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="9">
                  <TXT>Darauf die Zwiebeln verteilen .</TXT>
                  <TASK>verteilen</TASK>
                  <REFERENZ>8</REFERENZ>
                  <REFERENZ>4</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="10">
                  <TXT>Mit Pfeffer würzen .</TXT>
                  <TASK>würzen</TASK>
                  <ARGUMENT AID="1" ZUT="TRUE">Pfeffer</ARGUMENT>
                  <REFERENZ>9</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="11">
                  <TXT>Den Speck und den Käse in einer Schüssel miteinander
vermischen .</TXT>
                  <TASK>vermischen</TASK>
                  <ARGUMENT AID="1">in Schüssel</ARGUMENT>
                  <REFERENZ>5</REFERENZ>
                  <REFERENZ>6</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="12">
                  <TXT>Auf die Zwiebeln verteilen .</TXT>
                  <TASK>verteilen</TASK>
                  <REFERENZ>11</REFERENZ>
                  <REFERENZ>10</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="13">
                  <TXT>Die Sahne darübergeben .</TXT>
                  <TASK>darübergeben</TASK>
                  <REFERENZ>7</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="14">
                  <TXT>Bei 50° im Backofen bei geöffneter Tür circa 15 - 20 Minuten
gehen lassen .</TXT>
                  <TASK>gehen lassen</TASK>
                  <ARGUMENT AID="1" ZUT="FALSE">Backofen</ARGUMENT>
                  <ARGUMENT AID="2">bei geöffneter Tür</ARGUMENT>
                  <ARGUMENT AID="3">20 Minuten</ARGUMENT>
                  <REFERENZ>13</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="15">
                  <TXT>Das Blech herausnehmen und</TXT>
                  <TASK>herausnehmen</TASK>
                  <ARGUMENT AID="1">Blech</ARGUMENT>
            </SCHRITT>
            <SCHRITT SID="16">
                  <TXT>den Backofen auf 200° aufheizen .</TXT>
                  <TASK>aufheizen</TASK>
                  <REFERENZ>14</REFERENZ>
            </SCHRITT>
            <SCHRITT SID="17">
                  <TXT>Die Backzeit beträgt etwa 30 Minuten .</TXT>
                  <TASK>beträgt</TASK>
                  <ARGUMENT AID="1">Backzeit</ARGUMENT>
                  <ARGUMENT AID="2">30 Minuten</ARGUMENT>
```

```
            </SCHRITT>
      </ARBEITSSCHRITTE>
</REZEPT>
```

8.9 DTD der Rezepte in XML (Rezept.dtd)

```
<!ELEMENT REZEPT (REZEPTNAME,ZUTATENLISTE,ARBEITSSCHRITTE)>
<!ELEMENT ZUTATENLISTE (ZUTAT*)>
<!ELEMENT ZUTAT (BEZEICHNUNG,MENGE,EINHEIT)>
<!ELEMENT ARBEITSSCHRITTE (SCHRITT*)>
<!ELEMENT SCHRITT (TXT,TASK,ARGUMENT*,REFERENZ*)>

<!ELEMENT REZEPTNAME (#PCDATA)>
<!ELEMENT BEZEICHNUNG (#PCDATA)>
<!ELEMENT MENGE (#PCDATA)>
<!ELEMENT EINHEIT (#PCDATA)>
<!ELEMENT TXT (#PCDATA)>
<!ELEMENT TASK (#PCDATA)>
<!ELEMENT ARGUMENT (#PCDATA)>
<!ELEMENT REFERENZ (#PCDATA)>

<!ATTLIST SCHRITT
   SID     CDATA       #REQUIRED
>
<!ATTLIST ARGUMENT
   AID     CDATA       #REQUIRED
>
<!ATTLIST ARGUMENT
   ZUT     CDATA       #IMPLIED
>
```

8.10 Rezept in TDL

```
TASK herstellen()
#VARIABLES
       int tasknr = 1
       int result
       entity Hefeteig ''
#BODY
  DO(
             result = herstellen (Hefeteig)
  )
#END

TASK schneiden()
#VARIABLES
       int tasknr = 2
       int result
       entity Zwiebeln ''
#BODY
  DO(
             result = schneiden (Zwiebeln, in Ringe)
  )
#END

TASK andünsten()
#VARIABLES
       int tasknr = 3
       int result
       entity Zwiebeln_2 ''
#BODY
  DO(
             result = andünsten (2)
  )
#END

TASK abkühlen lassen()
```

```
#VARIABLES
       int tasknr = 4
       int result
       entity Zwiebeln_3 ''
#BODY
  DO(
             result = abkühlen lassen (3)
  )
#END

TASK würfeln()
#VARIABLES
       int tasknr = 5
       int result
       entity Speck ''
#BODY
  DO(
             result = würfeln (Speck)
  )
#END

TASK reiben()
#VARIABLES
       int tasknr = 6
       int result
       entity Käse ''
#BODY
  DO(
             result = reiben (Käse)
  )
#END

TASK verquirlen()
#VARIABLES
       int tasknr = 7
       int result
       entity Sahne ''
       entity Eigelb ''
#BODY
  DO(
             result = verquirlen (Sahne, Eigelb)
  )
#END

TASK ausrollen()
#VARIABLES
       int tasknr = 8
       int result
       entity Hefeteig_1 ''
#BODY
  DO(
             result = ausrollen (1)
  )
#END

TASK verteilen()
#VARIABLES
       int tasknr = 9
       int result
       entity Hefeteig_8 ''
       entity Zwiebeln_4 ''
#BODY
  DO(
             result = verteilen (8, 4)
  )
#END

TASK würzen()
#VARIABLES
       int tasknr = 10
       int result
       entity Pfeffer ''
```

```
       entity Hefeteig/Zwiebeln_9 ''
#BODY
  DO(
             result = würzen (Pfeffer, 9)
  )
#END

TASK vermischen()
#VARIABLES
       int tasknr = 11
       int result
       entity Speck_5 ''
       entity Käse_6 ''
#BODY
  DO(
             result = vermischen (in Schüssel, 5, 6)
  )
#END

TASK verteilen()
#VARIABLES
       int tasknr = 12
       int result
       entity Speck/Käse_11 ''
       entity Pfeffer/Hefeteig/Zwiebeln_10 ''
#BODY
  DO(
             result = verteilen (11, 10)
  )
#END

TASK darübergeben()
#VARIABLES
       int tasknr = 13
       int result
       entity Sahne/Eigelb_7 ''
#BODY
  DO(
             result = darübergeben (7)
  )
#END

TASK gehen lassen()
#VARIABLES
       int tasknr = 14
       int result
       entity Backofen ''
       entity Sahne/Eigelb_13 ''
#BODY
  DO(
             result = gehen lassen (Backofen, bei geöffneter Tür, 20 Minuten, 13)
  )
#END

TASK herausnehmen()
#VARIABLES
       int tasknr = 15
       int result
#BODY
  DO(
             result = herausnehmen (Blech)
  )
#END

TASK aufheizen()
#VARIABLES
       int tasknr = 16
       int result
       entity Backofen/Sahne/Eigelb_14 ''
#BODY
  DO(
             result = aufheizen (14)
```

```
  )
#END

TASK beträgt()
#VARIABLES
       int tasknr = 17
       int result
#BODY
  DO(
             result = beträgt (Backzeit, 30 Minuten)
  )
#END

TASK Rezept_Zwiebelkuchen
#BODY
#DO(
       task herstellen()
       task schneiden()
       task andünsten()
       task abkühlen lassen()
       task würfeln()
       task reiben()
       task verquirlen()
       task ausrollen()
       task verteilen()
       task würzen()
       task vermischen()
       task verteilen()
       task darübergeben()
       task gehen lassen()
       task herausnehmen()
       task aufheizen()
       task beträgt()
)
#END
```

9 Quellcode

```
 1: /**
 2:   *
 3:   * Main Klasse des Software Agenten "Agent Cook"
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: public class AgentCook {
 9:     /**
10:       * Hauptprogramm.
11:       *
12:       * @param args Kommandozeilenparameter
13:       */
14:     public static void main(String[] args) {
15:         // GUI aufrufen und mit Titel "Agent Cook" initialisieren
16:         GUI gui = new GUI("Agent Cook");
17:     }
18: }
19:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Verwaltung einer Blacklist
 4:   * Klasse holt sich Blacklist Wörter aus Blacklist.txt
 5:   *
 6:   * @version 1.0 vom 25.02.2009
 7:   * @author Henning Budde
 8:   */
 9:
10: import java.util.ArrayList;
11: import java.io.*;
12:
13: public class Blacklist {
14:     // Anfang Attribute
15:     private static ArrayList<String> blacklist = new
    ArrayList<String>();
16:
17:     // Ende Attribute
18:
19:     // Anfang Methoden
20:     public Blacklist() {
21:         try {
22:             BufferedReader br = new BufferedReader(new
    InputStreamReader(
23:                         new FileInputStream("Blacklist.txt"), "UTF8"));
24:             String line;
25:             line = br.readLine();
26:
27:             while (line != null) {
28:                 blacklist.add(line);
29:                 line = new String();
30:                 line = br.readLine();
31:             }
32:         } catch (IOException e) {
33:         }
34:     }
35:
36:   /**
37:     * Get Methode der Blacklist ArrayList
38:     *
39:     * @return ArrayList mit Wörtern, die in Blacklist stehten
40:     */
41:     public ArrayList<String> getBlacklist() {
42:         return blacklist;
43:     }
44:
45:   /**
46:     * Set Methode der Blacklist ArrayList
47:     *
48:     * @param blacklist mit Wörtern übergeben
49:     */
50:     public void setBlacklist(ArrayList<String> blacklist) {
51:         this.blacklist = blacklist;
52:     }
53:
54:     /**
55:     * Prüft ob ein übergebener String in Blacklist existiert
56:     *
57:     * @param s Zeichenkette die zu überprüfen ist
58:     * @return true wenn Wort in Blacklist
59:     */
60:     public static boolean inBlacklist(String s) {
61:         for (String a : blacklist) {
```

```
62:             if (a.equals(s)) {
63:                 return true;
64:             }
65:         }
66:
67:         return false;
68:     }
69:
70:     /**
71:     * Anzeige Methode der Blacklist ArrayList
72:     *
73:     */
74:     public static void showBlacklist() {
75:         for (String a : blacklist) {
76:             System.out.println(a);
77:         }
78:     }
79:
80:     // Ende Methoden
81: }
82:
```

```
 1: /**
 2:   *
 3:   * GUI zum Editieren von Wörtern, Wortarten, Entitäten, Zutaten
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.awt.*;
 9: import java.awt.event.*;
10:
11: import java.sql.*;
12:
13: import java.util.ArrayList;
14:
15: import javax.swing.*;
16: import javax.swing.event.*;
17:
18:
19: public class EditTask extends JDialog {
20:     // Anfang Attribute
21:     private Rezept rezept = Rezept.getInstance();
22:     private ArrayList<JTextField> wort = new ArrayList<JTextField>();
23:     private ArrayList<JTextField> wortart = new
    ArrayList<JTextField>();
24:     private ArrayList<JButton> buttons = new ArrayList<JButton>();
25:     private ArrayList<JComboBox> combolist = new
    ArrayList<JComboBox>();
26:     private ArrayList<JCheckBox> checkboxlist = new
    ArrayList<JCheckBox>();
27:     private ArrayList<JCheckBox> checkboxlist2 = new
    ArrayList<JCheckBox>();
28:     private JScrollPane jScrollPane1 = new JScrollPane();
29:     private JPanel jPanel1 = new JPanel(null);
30:     private JButton CloseButton = new JButton();
31:     private JComboBox combo = new JComboBox();
32:     private JCheckBox checkbox = new JCheckBox();
33:     private JCheckBox checkbox2 = new JCheckBox();
34:     private JLabel label1;
35:     private JLabel label2;
36:     private JLabel label3;
37:     private JLabel label4;
38:     private JLabel label5;
39:
40:     // Ende Attribute
41:     /**
42:     * Konstruktor der Klasse
43:     *
44:     * @param owner übergeordneter Frame
45:     * @param title Titel des Frames
46:     * @param modal soll Fenster modal oder nicht modal sein?
47:     */
48:     public EditTask(JFrame owner, String title, boolean modal) {
49:         // Dialog-Initialisierung
50:         super(owner, title, modal);
51:
52:         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
53:
54:         int frameWidth = 640;
55:         int frameHeight = 691;
56:         setSize(frameWidth, frameHeight);
57:
58:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
59:         int x = (d.width - getSize().width) / 2;
```

```
 60:         int y = (d.height - getSize().height) / 2;
 61:         setLocation(x, y);
 62:
 63:         Container cp = getContentPane();
 64:         cp.setLayout(null);
 65:
 66:         JTextField textfeld1;
 67:         JTextField textfeld2;
 68:         JTextField textfeld3;
 69:         JTextField textfeld4;
 70:         JButton button;
 71:
 72:         // Anfang Komponenten
 73:         int county = 0;
 74:         GridBagLayout grid = new GridBagLayout();
 75:         jPanel1.setLayout(grid);
 76:
 77:         GridBagConstraints c = new GridBagConstraints();
 78:         label1 = new JLabel("Wort");
 79:         label2 = new JLabel("Wortart");
 80:         label3 = new JLabel("Zutat ?");
 81:         label4 = new JLabel("Ändern");
 82:         label5 = new JLabel("Entität?");
 83:         //         System.out.println(rezept.getTasks().size()+1);
 84:         c.anchor = GridBagConstraints.WEST;
 85:         c.fill = GridBagConstraints.BOTH;
 86:         c.gridheight = 1;
 87:
 88:         c.insets = new Insets(1, 30, 1, 1);
 89:
 90:         c.gridx = 0;
 91:         c.gridy = county;
 92:         jPanel1.add(label1, c);
 93:         c.gridx = 4;
 94:         jPanel1.add(label2, c);
 95:         c.gridx = 5;
 96:         jPanel1.add(label3, c);
 97:         c.gridx = 6;
 98:         jPanel1.add(label5, c);
 99:         c.gridx = 7;
100:         jPanel1.add(label4, c);
101:
102:         for (Task t : rezept.getTasks()) {
103:             county++;
104:             c.gridx = 0;
105:             c.gridy = county;
106:
107:             textfeld1 = new JTextField();
108:
109:             combo = new JComboBox();
110:             checkbox = new JCheckBox();
111:             checkbox2 = new JCheckBox();
112:             textfeld1.setText(t.getWort());
113:
114:             c.insets = new Insets(10, 30, 10, 10);
115:             jPanel1.add(textfeld1, c);
116:             wort.add(textfeld1);
117:
118:             combo.addItem(t.getWortart());
119:             combo.addItem("Substantiv");
120:             combo.addItem("Verb");
121:             combo.addItem("Hilfsverb");
122:             combo.addItem("Zahl");
```

```
123:             combo.addItem("Adjektiv");
124:             combo.addItem("Praeposition");
125:             combo.setToolTipText(t.getUmstand());
126:             combolist.add(combo);
127:             combo.setEditable(true);
128:
129:             c.gridx = 4;
130:             jPanel1.add(combo, c);
131:
132:             checkboxlist.add(checkbox);
133:             checkboxlist2.add(checkbox2);
134:
135:             c.gridx = 5;
136:             jPanel1.add(checkbox, c);
137:             c.gridx = 6;
138:             jPanel1.add(checkbox2, c);
139:
140:             button = new JButton("anwenden");
141:
142:             c.gridx = 7;
143:             jPanel1.add(button, c);
144:
145:             button.addActionListener(new ActionListener() {
146:                     public void actionPerformed(ActionEvent evt) {
147:                         pushedbutton(evt.getSource());
148:                     }
149:                 });
150:             buttons.add(button);
151:
152:             if (t.getIsZutat()) {
153:                 checkbox.setSelected(true);
154:             }
155:
156:             if (t.getIsentity()) {
157:                 checkbox2.setSelected(true);
158:             }
159:         }
160:
161:         jPanel1.setBounds(0, 0, 640, 561);
162:         jScrollPane1.setBounds(0, 0, 630, 561);
163:         jScrollPane1.add(jPanel1);
164:         jScrollPane1.setViewportView(jPanel1);
165:         cp.add(jScrollPane1);
166:
167:         CloseButton.setBounds(224, 592, 153, 41);
168:         CloseButton.setText("Schließen");
169:         CloseButton.addActionListener(new ActionListener() {
170:                 public void actionPerformed(ActionEvent evt) {
171:                     CloseButton_ActionPerformed(evt);
172:                 }
173:             });
174:         cp.add(CloseButton);
175:         // Ende Komponenten
176:         setResizable(false);
177:         setVisible(true);
178:     }
179:
180:     // Anfang Methoden
181:
182:     /**
183:     * Ein Anwenden-Button wurde gedrückt
184:     *
185:     * @param evt Event
```

```
186:     */
187:     private void pushedbutton(Object evt) {
188:         JButton b = (JButton) evt;
189:         int index = buttons.indexOf(b);
190:         Task t = rezept.getTasks().get(index);
191:         t.setIsZutat(checkboxlist.get(index).isSelected());
192:         t.setIsentity(checkboxlist2.get(index).isSelected());
193: 
194:         if (!t.getWortart()
195:                     .equals((String)
     combolist.get(index).getSelectedItem())) {
196:             System.out.print("Task wurde geändert: " + t.getWort() +
197:                 "  wurde von " + t.getWortart() + " auf ");
198:             t.setWortart((String)
     combolist.get(index).getSelectedItem());
199:             System.out.println(t.getWortart() + " geändert");
200:         }
201: 
202:         if (t.getWortart().equals("Verb")) {
203:             new VerbDialog(null, "Verb aktualisieren", false,
     t.getWort());
204:         }
205: 
206:         // wenn neues Adjektiv
207:         if (t.getWortart().equals("Adjektiv")) {
208:             Connection con = DBzugriff.connect();
209: 
210:             String SQL = "INSERT INTO ADJ_TAB(Form1) VALUES('" +
     t.getWort() +
211:                 "~')";
212: 
213:             try {
214:                 Statement stmt = con.createStatement();
215:                 stmt.executeUpdate(SQL);
216:                 con.close();
217:             } catch (SQLException e) {
218:                 new ErrorDialog(null, "Adjektiv Fehler", true,
     e.getMessage());
219:             }
220:         }
221:     }
222: 
223:   /**
224:     * Fenster schliessen Methode
225:     *
226:     * @param evt Event -> Fenster soll sich schließen
227:     */
228:     public void CloseButton_ActionPerformed(ActionEvent evt) {
229:         this.setVisible(false);
230:         this.dispose();
231:     }
232: 
233:     // Ende Methoden
234: }
235: 
```

```
 1: /**
 2:   *
 3:   * Dialog GUI Komponenten zum Anzeigen verschiedener Fehlermeldungen
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.awt.*;
 9: import java.awt.event.*;
10:
11: import javax.swing.*;
12: import javax.swing.event.*;
13:
14:
15: public class ErrorDialog extends JDialog {
16:     // Anfang Attribute
17:     private JLabel jLabelErrorText = new JLabel();
18:     private JButton jButtonOK = new JButton();
19:
20:     // Ende Attribute
21:     public ErrorDialog(JFrame owner, String title, boolean modal,
    String msg) {
22:         // Dialog-Initialisierung
23:         super(owner, title, modal);
24:         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
25:
26:         int frameWidth = 363;
27:         int frameHeight = 260;
28:         setSize(frameWidth, frameHeight);
29:
30:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
31:         int x = (d.width - getSize().width) / 2;
32:         int y = (d.height - getSize().height) / 2;
33:         setLocation(x, y);
34:
35:         Container cp = getContentPane();
36:         cp.setLayout(null);
37:         // Anfang Komponenten
38:         jLabelErrorText.setBounds(8, 16, 340, 124);
39:         jLabelErrorText.setText(msg);
40:         jLabelErrorText.setFont(new Font("MS Sans Serif", Font.PLAIN,
    15));
41:         cp.add(jLabelErrorText);
42:         jButtonOK.setBounds(112, 152, 121, 33);
43:         jButtonOK.setText("OK");
44:         jButtonOK.addActionListener(new ActionListener() {
45:                 public void actionPerformed(ActionEvent evt) {
46:                     jButtonOK_ActionPerformed(evt);
47:                 }
48:             });
49:
50:         jButtonOK.setToolTipText("Fehler gelsen und bestätigen");
51:         cp.add(jButtonOK);
52:         // Ende Komponenten
53:         setResizable(false);
54:         setVisible(true);
55:     }
56:
57:     // Anfang Methoden
58:   /**
59:     * Bestätigungs-Button wurde gedrückt -> Fenster soll sich schließen
60:     *
61:     * @param evt Event: Button wurde gedrückt
```

```
62:     */
63:     public void jButtonOK_ActionPerformed(ActionEvent evt) {
64:         setVisible(false);
65:         dispose();
66:     }
67:
68:     // Ende Methoden
69: }
70:
```

```
 1: /**
 2:   *
 3:   * Klasse die Rezeptinformationen über Pattern Vergleich
 4:   * aus HTML Datei herausfiltert / ausschneidet
 5:   *
 6:   * @version 1.0 vom 25.02.2009
 7:   * @author Henning Budde
 8:   */
 9:
10: import java.io.*;
11: import java.util.ArrayList;
12: import java.util.regex.Matcher;
13: import java.util.regex.Pattern;
14:
15:
16: public class FilterHTML {
17:     // Anfang Attribute
18:     private FileWriter fw;
19:     private PrintWriter pw;
20:     private FileReader fr;
21:     private BufferedReader buf;
22:     private boolean beginn = false;
23:     private boolean name = false;
24:     private boolean anzahl = false;
25:     private boolean zutaten = false;
26:     private Parameter params;
27:     private String FilterFileName = "FilterHTML";
28:     private Rezept rezept;
29:     private Zutat zutat;
30:     private int kochseite = 1; // 1: chefkoch, 2: rezeptewiki 3: xxx
31:     private ArrayList<Zutat> Zutatenliste = new ArrayList<Zutat>();
32:     private ArrayList<String> desc = new ArrayList<String>();
33:     String a = "";
34:
35:     // Ende Attribute
36:     public FilterHTML(String filename, Parameter params) {
37:         this.params = params;
38:         rezept = Rezept.getInstance();
39:
40:         try {
41:             fr = new FileReader(filename);
42:             buf = new BufferedReader(fr);
43:
44:             fw = new FileWriter(FilterFileName);
45:             pw = new PrintWriter(fw);
46:         } catch (IOException e) {
47:             new ErrorDialog(null, "IOException im Konstruktor
    FilterHTML",
48:                     true, e.getMessage());
49:         } finally {
50:         }
51:     }
52:
53:     // Anfang Methoden
54:   /**
55:     * Methode  scannt HTML Datei und schreibt relevante Informationen
56:     * in ein Objekt der Klasse Rezept
57:     *
58:     */
59:     public void scanHTML() {
60:         int zustand = 0;
61:         zutat = new Zutat();
62:
```

```
 63:         String html = "";
 64:
 65:         try {
 66:             html = buf.readLine();
 67:
 68:             Pattern p;
 69:             Matcher m;
 70:             p = Pattern.compile(".*" + params.getRezeptname_start() +
    ".*");
 71:
 72:             while (html != null) {
 73:
 74:                 //                System.out.println(p.pattern());
 75:                 switch (zustand) {
 76:                 case 0:
 77:                     m = p.matcher(html);
 78:
 79:                     if (m.matches()) {
 80:                         //
    System.out.println("Debug 0");
 81:                         rezept.setRezeptname(this.split(
 82:                                 params.getRezeptname_start(),
 83:                                 params.getRezeptname_end(), html));
 84:
 85:                         System.out.println(rezept.getRezeptname());
 86:
 87:                         zustand = 1;
 88:                     }
 89:
 90:                     break;
 91:
 92:                 case 1:
 93:                     p = Pattern.compile(".*" +
    params.getPersonen_start() +
 94:                             ".*");
 95:                     m = p.matcher(html);
 96: //                     System.out.println("IN Zustand 1");
 97:                     if (m.matches()) {
 98:                         rezept.setPersonen(this.split(
 99:                                 params.getPersonen_start(),
100:                                 params.getPersonen_end(), html));
101:
    rezept.setPersonen((rezept.getPersonen()).replace("â??","-"));
102:
103:                         System.out.println("Anzahl Personen: " +
104:                             rezept.getPersonen());
105:                         zustand = 2;
106:                     }
107:
108:                     break;
109:
110:                 case 2:
111:                     //                     zutat = new Zutat();
112:                     p = Pattern.compile(".*" +
    params.getZutat_start() + ".*");
113:                     m = p.matcher(html);
114:
115:                     if (m.matches()) {
116:                         try {
117:
    zutat.setMenge(Integer.parseInt(this.split(
118:                                     params.getZutat_start(),
119:                                     params.getZutat_menge(),
    html)));
```

```
120:
121:                             //
    System.out.println(zutat.getMenge());
122:                         } catch (NumberFormatException e) {
123:                             zutat.setMenge(0);
124:                             zutat.setEinheit("-");
125:                         }
126:
127:                         //System.out.println("DEBUG 2");
128:                         if (kochseite == 1) {
129:                             zutat.setEinheit((this.split(
130:                                     params.getZutat_menge(),
131:                                     params.getZutat_einheit(),
    html)));
132:                             //
    System.out.println(html);
133:                             //
    zutat.setEinheit(zutat.getEinheit().replace(" ",""));
134:                             //
    System.out.print("Zutaten Einheit: "+zutat.getEinheit() + " & Zutaten
    Menge: "+ zutat.getMenge() );
135:                             zustand = 3;
136:                         }
137:                     }
138:
139:                     p = Pattern.compile(".*" +
    params.getZutaten_end() + ".*");
140:                     m = p.matcher(html);
141:
142:                     if (m.matches()) {
143: //                       System.out.println("ZUSTAND =3");
144:                         zustand = 4;
145:                         desc = new ArrayList<String>();
146:                     }
147:
148:                     break;
149:
150:                 case 3:
151:                     //                    buf.readLine();
152:                     //                    buf.readLine();
153:                     html = buf.readLine();
154:                     html = html.replace("(","");
155:                     html = html.replace(")","");
156:
157:                     zutat.setBez(html.trim());
158:                     System.out.println(zutat.getMenge() + "\t " +
159:                         zutat.getEinheit() + "\t " + zutat.getBez());
160:                     Zutatenliste.add(zutat);
161:                     zutat = new Zutat();
162:                     zustand = 2;
163:
164:                     break;
165:
166:                 case 4:
167:                     rezept.setZutaten(Zutatenliste);
168:                     if (GUI.getMitFoto()) {
169:                         p = Pattern.compile(".*" +
170:
    params.getRezept_description_start_foto() +
171:                                 ".*");
172:                         m = p.matcher(html);
173: //                       System.out.println(p.pattern());
174:                     } else {
```

```
175:                         p = Pattern.compile(".*" +
176:                                 params.getRezept_description_start()
    + ".*");
177:                         m = p.matcher(html);
178:                     }
179:
180:                     //                    System.out.println(html);
181:                     //                    System.out.println("ZUSTAND
    = 4");
182:                     int i = 0;
183:
184:                     if (m.matches()) {
185:                         do {
186: //                           System.out.println(p.pattern());
187:
188:                             html = buf.readLine();
189:                             //
    System.out.println(html);
190:                             p = Pattern.compile(".*" +
191:
    params.getRezept_description_end() + ".*");
192:                             m = p.matcher(html);
193:                             html = html.trim();
194:                             html = html.replace("#br", "");
195:                             html = html.replace("/#", "");
196:                             if (!html.equals("") && !m.matches()) {
197:                                 a = a + html;
198:
199:                                 //
    desc.add(html);
200:                             }
201:
202:                             zustand = 5;
203:                         } while (!m.matches());
204:                     }
205:
206:                     //                    System.out.println("zustand
    = 4");
207:                     break;
208:
209:                 case 5:
210:                     System.out.println("DEBUG 5");
211:                     a = a.replace('!','.');  // wenn Ausrufezeichen
    im Rezept
212:                     //                     System.out.println(a);
213:                     String punkt = "\\.";
214:                     String[] arr = a.split(punkt);
215:
216:                     //
    System.out.println(arr.length);
217:                     for (int j = 0; j < arr.length; j++) {
218:
219:                         // "Ca.", "ca.", "Mind.", "mind.", "Min.",
    "min." abzufangen
220:                         if (arr[j].endsWith("ca") ||
    arr[j].endsWith("Ca") ||
221:                                 arr[j].endsWith("mind") ||
222:                                 arr[j].endsWith("Mind") ||
223:                                 arr[j].endsWith("Min") ||
224:                                 arr[j].endsWith("min")) {
225:                             arr[j + 1] = arr[j] + "." + arr[j + 1];
226:                             j++;
227:                         }
```

```
228:
229:                         if (arr[j].endsWith("ca") ||
    arr[j].endsWith("Ca") ||
230:                                 arr[j].endsWith("mind") ||
231:                                 arr[j].endsWith("Mind") ||
232:                                 arr[j].endsWith("Min") ||
233:                                 arr[j].endsWith("min")) {
234:                             arr[j + 1] = arr[j] + "." + arr[j + 1];
235:                             j++;
236:                         }
237:                         // Um "z.B." abzufangen
238:                         if (arr[j].endsWith("z") ||
    arr[j].endsWith("B")
239:
240:                                 ) {
241:                             arr[j + 1] = arr[j] + "." + arr[j + 1];
242:                             j++;
243:                         }
244:                         if (arr[j].endsWith("z") ||
    arr[j].endsWith("B")
245:                                 ) {
246:                             arr[j + 1] = arr[j] + "." + arr[j + 1];
247:                             j++;
248:                         }
249:                        if (arr[j].endsWith("Std") ||
    arr[j].endsWith("std")
250:
251:                                 ) {
252:                             arr[j + 1] = arr[j] + "." + arr[j + 1];
253:                             j++;
254:                         }
255:                         if (arr[j].endsWith("Std") ||
    arr[j].endsWith("std")
256:                                 ) {
257:                             arr[j + 1] = arr[j] + "." + arr[j + 1];
258:                             j++;
259:                         }
260:
261:                         //
    System.out.println(arr[j]);
262:                         desc.add((arr[j] + ".").trim());
263:                     }
264:
265:                     for (String z : desc) {
266:                         System.out.println(z);
267:                     }
268:                     rezept.setDescription(desc);
269:                     zustand = 6;
270:
271:                     break;
272:
273:                 default:
274:                     break;
275:                 }
276:
277:                 html = buf.readLine();
278:             }
279:         } catch (IOException e) {
280:             new ErrorDialog(null, "IOException in Methode scanHTML",
    true,
281:                 e.getMessage());
282:         } finally {
283:         }
```

```
284:     }
285:   /**
286:     * Schreibt XML Header
287:     *
288:     * @param filename Dateiname der zu schreibenden XML Datei
289:     */
290:     public void write_xml_header(String filename) {
291:         try {
292:             File file1 = new File(filename);
293:
294:             if (file1.exists()) {
295:                 if (file1.delete()) {
296:                     System.out.println("File wurde gelöscht");
297:                 }
298:
299:                 ;
300:             }
301:
302:             FileWriter fw = new FileWriter(file1);
303:             PrintWriter pw = new PrintWriter(fw);
304:         } catch (IOException e) {
305:             new ErrorDialog(null, "IOException", true,
306:                 "<html> Fehler in write_xml_header: <br>" +
    e.getMessage() +
307:                 "</html>");
308:         }
309:     }
310:
311:   /**
312:     * Methode schneidet relevane Informationen zwischen String before
313:     * und after aus zeile aus
314:     *
315:     * @param before Zeichenkette die vor relevanter Info steht
316:     * @param after Zeichenkette die hinter relevanter Info steht
317:     * @param zeile Zeile, die durchsucht werden soll
318:     */
319:     public String split(String before, String after, String zeile) {
320:         String cut;
321:         String[] array1;
322:         String[] array2;
323:         //        System.out.println(before +" und " + after );
324:         array1 = zeile.split(before);
325:         array2 = array1[1].split(after);
326:
327:         //        System.out.println(array2[0]);
328:         if (array2[0].equals("")) {
329:             return "0";
330:         }
331:
332:         //        System.out.println("HIER ARRAY 2 "+ array2[0]);
333:         return array2[0];
334:     }
335:
336:   /**
337:     * Get Methode der Zutatenliste
338:     *
339:     * @return Objekte der Klasse Zutat
340:     */
341:     public ArrayList<Zutat> getZutatenliste() {
342:         return Zutatenliste;
343:     }
344:   /**
345:     * Set Methode der Zutaten ArrayList
```

```
346:     *
347:     * @param Zutatenliste Objekte der Klasse Zutat
348:     */
349:     public void setZutatenliste(ArrayList<Zutat> Zutatenliste) {
350:         this.Zutatenliste = Zutatenliste;
351:     }
352:
353:     // Ende Methoden
354: }
355:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Kommunikation mit dem Internet.
 4:   * Klasse empfängt Webseite von Server.
 5:   *
 6:   * @version 1.0 vom 25.02.2009
 7:   * @author Henning Budde
 8:   */
 9: import java.io.*;
10: import java.net.*;
11: import java.util.regex.Matcher;
12: import java.util.regex.Pattern;
13:
14: public class GetURL {
15:   /**
16:     * Statische Methode fordert Webseite von Server an
17:     * und schreibt diese in eine Datei.
18:     *
19:     * @param filename Dateiname
20:     * @param userURL URL der Seite
21:     */
22:     public static void getSite(String filename, String userURL) {
23:         URL u;
24:         InputStream is = null;
25:         DataInputStream dis;
26:         String s;
27:         FileWriter o;
28:         PrintWriter pw = null;
29:
30:         try {
31:             o = new FileWriter(filename, false);
32:             pw = new PrintWriter(o);
33:
34:             u = new URL(userURL);
35:
36:             is = u.openStream(); // throws an IOException
37:
38:             dis = new DataInputStream(new BufferedInputStream(is));
39:
40:             while ((s = dis.readLine()) != null) {
41:                 s = replaceSpecialValues(s);
42:
43:                 Pattern p;
44:                 Matcher m;
45:
46:                 p = Pattern.compile(".*#a href.*");
47:                 m = p.matcher(s);
48:
49:                 if (m.matches()) {
50:                     continue;
51:                 }
52:
53:                 p = Pattern.compile(".*#/a#.*");
54:                 m = p.matcher(s);
55:
56:                 if (m.matches()) {
57:                     continue;
58:                 }
59:
60:                 pw.println(s);
61:
62:                 //            System.out.println(s);
63:             }
```

```
64:         } catch (MalformedURLException mue) {
65:             ErrorDialog er = new ErrorDialog(null,
    "MalformedURLException",
66:                     true, mue.getMessage());
67:         } catch (IOException ioe) {
68:             ErrorDialog er = new ErrorDialog(null, "IOLException",
    true,
69:                     ioe.getMessage());
70:         } finally {
71:             try {
72:                 is.close();
73:                 pw.close();
74:             } catch (IOException ioe) {
75:                 new ErrorDialog(null, "IOExceptionm", true,
76:                     "<html> Fehler beim Schliessen des InputStreams:
    <br>" +
77:                     ioe.getMessage() + "</html>");
78:             }
79:         }
80:     }
81:
82:   /**
83:     * Methode zur Vorverarbeitung der HTML Seite
84:     * Umlaute und Sonderzeichen werden ersetzt
85:     *
86:     * @param s Zeile aus HTML Seite
87:     * @return vorverarbeiteter String
88:     */
89:     public static String replaceSpecialValues(String s) {
90:         s = s.replaceAll("&auml;", "ä");
91:         s = s.replaceAll("&Auml;", "Ä");
92:         s = s.replaceAll("&ouml;", "ö");
93:         s = s.replaceAll("&Ouml;", "Ö");
94:         s = s.replaceAll("&uuml;", "ü");
95:         s = s.replaceAll("&Uuml;", "Ü");
96:         s = s.replaceAll("&slig;", "ß");
97:         s = s.replaceAll("&aacute;", "a");
98:
99:         s = s.replace('<', '#');
100:        s = s.replace('>', '#');
101:        s = s.replace('"', '#');
102:        s = s.replace('&', '#');
103:        s = s.replace("darübergeben", "darüber geben");
104:
105:        return s;
106:    }
107: }
108:
```

```
 1: /**
 2:   *
 3:   * GUI Klasse des Software Agenten
 4:   * Alle Haupfunktionen werden von dieser Klasse aus aufgerufen
 5:   *
 6:   * @version 1.0 vom 25.02.2009
 7:   * @author Henning Budde
 8:   */
 9: import java.awt.*;
10: import java.awt.datatransfer.*;
11: import java.awt.event.*;
12:
13: import java.io.*;
14:
15: import java.util.ArrayList;
16:
17: import javax.swing.*;
18: import javax.swing.event.*;
19:
20:
21: public class GUI extends JFrame {
22:     private static boolean mitFoto;
23:
24:     // Anfang Attribute
25:     final String filename = "htmlseite.txt";
26:     final String chefkochxml = "Chefkoch.xml";
27:     final String rezeptewikixml = "RezepteWiki.xml";
28:     private JPanel jPanel1 = new JPanel(null);
29:     private JMenuBar menubar = new JMenuBar();
30:     private JMenu menufile = new JMenu("Datei");
31:     private JMenu menuknow = new JMenu("Wissensakquisition");
32:     private JMenu menuparse = new JMenu("Wissensextraktion");
33:     private JMenu menuresult = new JMenu("Wissensaufbereitung");
34:     private JMenu menurtd = new JMenu("TDL");
35:     private JMenu menuhelp = new JMenu("Hilfe");
36:     private JMenuItem fileend = new JMenuItem("Ende");
37:     private JMenuItem knowstart = new JMenuItem("Start");
38:     private JMenuItem knowshow = new JMenuItem("Anzeigen");
39:     private JMenuItem parsestart = new JMenuItem("Start");
40:     private JMenuItem parseshow = new JMenuItem("Anzeigen");
41:     private JMenuItem resultstart = new JMenuItem("Start");
42:     private JMenuItem resultshow = new JMenuItem("Anzeigen");
43:     private JMenuItem knowext = new JMenuItem("Start");
44:     private JMenuItem knowextshow = new JMenuItem("Anzeigen");
45:     private JMenuItem tdlstart = new JMenuItem("Start");
46:     private JMenuItem tdlshow = new JMenuItem("Anzeigen");
47:     private JMenuItem helpshow = new JMenuItem("Hilfe");
48:     private JMenuItem helpabout = new JMenuItem("Über");
49:     private JLabel Kochseite = new JLabel();
50:     private JRadioButton jRadioButton1 = new JRadioButton();
51:     private JRadioButton jRadioButton2 = new JRadioButton();
52:     private JRadioButton jRadioButton3 = new JRadioButton();
53:     private ButtonGroup radiogroup = new ButtonGroup();
54:     private JLabel labeladress = new JLabel();
55:     private JTextField jTextFieldAdress = new JTextField();
56:     private JButton jButtonKnowStart = new JButton();
57:     private JButton jButtonParseStart = new JButton();
58:     private JButton jButtonOWL = new JButton();
59:     private JLabel jLabel1 = new JLabel();
60:     private JLabel jLabel2 = new JLabel();
61:     private JLabel jLabel3 = new JLabel();
62:     private JButton jButtonKnowShow = new JButton();
63:     private JButton jButtonParseShow = new JButton();
```

```
64:     private JButton jButtonResultShow = new JButton();
65:     private JButton jButtonEnde = new JButton();
66:     private JButton jButtonPaste = new JButton();
67:     private Parser parser;
68:     private JButton TestButton = new JButton();
69:     private JCheckBox fotocheck = new JCheckBox();
70:     private JButton jAnalyseStart = new JButton();
71:     private JLabel jLabel4 = new JLabel();
72:     private JButton ShowAnalyse = new JButton();
73:
74:     // Ende Attribute
75:     public GUI(String title) {
76:         // Frame-Initialisierung
77:         super(title);
78:         setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
79:
80:         int frameWidth = 594;
81:         int frameHeight = 476;
82:         setSize(frameWidth, frameHeight);
83:
84:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
85:         int x = (d.width - getSize().width) / 2;
86:         int y = (d.height - getSize().height) / 2;
87:         setLocation(x, y);
88:
89:         menufile.add(fileend);
90:         fileend.addActionListener(new ActionListener() {
91:                 public void actionPerformed(ActionEvent evt) {
92:                     setVisible(false);
93:                     dispose();
94:                     System.exit(0);
95:                 }
96:             });
97:
98:         menubar.add(menufile);
99:
100:        menuknow.add(knowstart);
101:        knowstart.addActionListener(new ActionListener() {
102:                public void actionPerformed(ActionEvent evt) {
103:                    jButtonKnowStart_ActionPerformed(evt);
104:                }
105:            });
106:        menuknow.add(knowshow);
107:        knowshow.addActionListener(new ActionListener() {
108:                public void actionPerformed(ActionEvent evt) {
109:                    jButtonKnowShow_ActionPerformed(evt);
110:                }
111:            });
112:        menubar.add(menuknow);
113:
114:        menuparse.add(parsestart);
115:        parsestart.addActionListener(new ActionListener() {
116:                public void actionPerformed(ActionEvent evt) {
117:                    jButtonParseStart_ActionPerformed(evt);
118:                }
119:            });
120:
121:        menuparse.add(parseshow);
122:        parseshow.addActionListener(new ActionListener() {
123:                public void actionPerformed(ActionEvent evt) {
124:                    jButtonParseShow_ActionPerformed(evt);
125:                }
126:            });
```

```
127:
128:         menubar.add(menuparse);
129:
130:         menubar.add(menuresult);
131:         menuresult.add(this.resultstart);
132:         resultstart.addActionListener(new ActionListener() {
133:                 public void actionPerformed(ActionEvent evt) {
134:                     jAnalyseStart_ActionPerformed(evt);
135:                 }
136:             });
137:
138:         menuresult.add(resultshow);
139:         resultshow.addActionListener(new ActionListener() {
140:                 public void actionPerformed(ActionEvent evt) {
141:                     ShowAnalyse_ActionPerformed(evt);
142:                 }
143:             });
144:
145:         menubar.add(this.menurtd);
146:         menurtd.add(this.tdlstart);
147:         menurtd.add(this.tdlshow);
148:
149:         menubar.add(menuhelp);
150:         menuhelp.add(helpshow);
151:         menuhelp.add(helpabout);
152:         helpabout.addActionListener(new ActionListener() {
153:                 public void actionPerformed(ActionEvent evt) {
154:                     new ErrorDialog(null, "Über dieses Programm",
     true,
155:                         "<html> <b>Agent Cook</b> <br> Autor: Henning
     Budde <br> (c) 2008 FH Köln <br> Version 0.1</hmtl>");
156:                 }
157:             });
158:
159:         this.setJMenuBar(menubar);
160:
161:         Container cp = getContentPane();
162:         cp.setLayout(null);
163:         // Anfang Komponenten
164:         setLayout(new BorderLayout());
165:
166:         jPanel1.setBounds(0, 8, 793, 489);
167:         cp.add(jPanel1);
168:
169:         Kochseite.setBounds(8, 24, 79, 22);
170:         Kochseite.setText("Kochseite");
171:         Kochseite.setFont(new Font("Times New Roman", Font.PLAIN,
     18));
172:         Kochseite.setToolTipText("Welche Kochseite soll aquisiert
     werden?");
173:         jPanel1.add(Kochseite);
174:         jRadioButton1.setBounds(32, 64, 81, 17);
175:         jRadioButton1.setText("Chefkoch");
176:         jRadioButton1.setToolTipText("www.chefkoch.de");
177:         //        jRadioButton1.setBorderPainted(false);
178:         radiogroup.add(jRadioButton1);
179:         jRadioButton1.setSelected(true);
180:
181:         jPanel1.add(jRadioButton1);
182:         jRadioButton2.setBounds(160, 64, 113, 17);
183:         jRadioButton2.setText("RezepteWiki");
184:         jRadioButton2.setToolTipText("www.rezeptewiki.de");
185:         radiogroup.add(jRadioButton2);
```

```
186:
187:         jPanel1.add(jRadioButton2);
188:         jRadioButton3.setBounds(304, 64, 121, 17);
189:         jRadioButton3.setText("dritte Seite");
190:         radiogroup.add(jRadioButton3);
191:         jPanel1.add(jRadioButton3);
192:         labeladress.setBounds(8, 104, 114, 22);
193:         labeladress.setText("Rezeptadresse");
194:         labeladress.setFont(new Font("Times New Roman", Font.PLAIN,
     18));
195:         labeladress.setToolTipText("Hier die exakte Rezeptadresse
     angeben");
196:         jPanel1.add(labeladress);
197:         jTextFieldAdress.setBounds(144, 104, 305, 24);
198:         jTextFieldAdress.setText("http://");
199:         jTextFieldAdress.setToolTipText(
200:             "Hier exakte Rezeptadresse eintragen. (Aus Browser
     kopiert)");
201:         jPanel1.add(jTextFieldAdress);
202:         jButtonKnowStart.setBounds(344, 168, 105, 33);
203:         jButtonKnowStart.setText("Start");
204:         jButtonKnowStart.addActionListener(new ActionListener() {
205:                 public void actionPerformed(ActionEvent evt) {
206:                     jButtonKnowStart_ActionPerformed(evt);
207:                 }
208:             });
209:         jPanel1.add(jButtonKnowStart);
210:         jButtonParseStart.setBounds(344, 208, 105, 33);
211:         jButtonParseStart.setText("Start");
212:         jButtonParseStart.addActionListener(new ActionListener() {
213:                 public void actionPerformed(ActionEvent evt) {
214:                     jButtonParseStart_ActionPerformed(evt);
215:                 }
216:             });
217:         jButtonParseStart.setToolTipText("Startet Parsen des
     Rezepts");
218:         jButtonParseStart.setEnabled(false);
219:         jPanel1.add(jButtonParseStart);
220:         jButtonOWL.setBounds(344, 288, 105, 33);
221:         jButtonOWL.setText("Start");
222:         jButtonOWL.addActionListener(new ActionListener() {
223:                 public void actionPerformed(ActionEvent evt) {
224:                     jButtonOWL_ActionPerformed(evt);
225:                 }
226:             });
227:         jButtonOWL.setToolTipText("In TDL umwandeln");
228:
229:         jButtonOWL.setEnabled(false);
230:         jPanel1.add(jButtonOWL);
231:         jLabel1.setBounds(32, 176, 183, 16);
232:         jLabel1.setText("Rezept aus Internet laden");
233:         jLabel1.setFont(new Font("MS Sans Serif", Font.PLAIN, 14));
234:
235:         jPanel1.add(jLabel1);
236:
237:         jLabel2.setBounds(32, 216, 152, 16);
238:         jLabel2.setText("Rezept parsen");
239:         jLabel2.setFont(new Font("MS Sans Serif", Font.PLAIN, 14));
240:
241:         jPanel1.add(jLabel2);
242:
243:         jLabel3.setBounds(32, 296, 154, 24);
244:         jLabel3.setText("Task Def. Language");
```

```
245:         jLabel3.setFont(new Font("MS Sans Serif", Font.PLAIN, 14));
246: 
247:         jPanel1.add(jLabel3);
248: 
249:         jButtonKnowShow.setBounds(472, 168, 105, 33);
250:         jButtonKnowShow.setText("Anzeigen");
251:         jButtonKnowShow.setEnabled(false);
252:         jButtonKnowShow.addActionListener(new ActionListener() {
253:                 public void actionPerformed(ActionEvent evt) {
254:                     jButtonKnowShow_ActionPerformed(evt);
255:                 }
256:             });
257: 
258:         jPanel1.add(jButtonKnowShow);
259: 
260:         jButtonParseShow.setBounds(472, 208, 105, 33);
261:         jButtonParseShow.setText("Anzeigen");
262:         jButtonParseShow.setEnabled(false);
263:         jButtonParseShow.addActionListener(new ActionListener() {
264:                 public void actionPerformed(ActionEvent evt) {
265:                     jButtonParseShow_ActionPerformed(evt);
266:                 }
267:             });
268: 
269:         jPanel1.add(jButtonParseShow);
270: 
271:         jButtonResultShow.setBounds(472, 288, 105, 33);
272:         jButtonResultShow.setText("Anzeigen");
273:         jButtonResultShow.setEnabled(false);
274:         jButtonResultShow.addActionListener(new ActionListener() {
275:                 public void actionPerformed(ActionEvent evt) {
276:                     jButtonResultShow_ActionPerformed(evt);
277:                 }
278:             });
279: 
280:         jPanel1.add(jButtonResultShow);
281: 
282:         jButtonEnde.setBounds(472, 344, 105, 33);
283:         jButtonEnde.setText("Ende");
284:         jButtonEnde.addActionListener(new ActionListener() {
285:                 public void actionPerformed(ActionEvent evt) {
286:                     jButtonEnde_ActionPerformed(evt);
287:                 }
288:             });
289: 
290:         jPanel1.add(jButtonEnde);
291: 
292:         jButtonPaste.setBounds(472, 104, 105, 25);
293:         jButtonPaste.setText("Einfügen");
294:         jButtonPaste.addActionListener(new ActionListener() {
295:                 public void actionPerformed(ActionEvent evt) {
296:                     jButtonPaste_ActionPerformed(evt);
297:                 }
298:             });
299:         jButtonPaste.setToolTipText("Adresse aus Zwischenablage
    einfügen");
300: 
301:         jPanel1.add(jButtonPaste);
302: 
303:         TestButton.setBounds(200, 136, 81, 17);
304:         TestButton.setText("TestButton");
305:         TestButton.addActionListener(new ActionListener() {
306:                 public void actionPerformed(ActionEvent evt) {
```

```
307:                     TestButton_ActionPerformed(evt);
308:                 }
309:             });
310:         jPanel1.add(TestButton);
311:         fotocheck.setBounds(24, 136, 137, 25);
312:         fotocheck.setText("ohne Foto");
313:         jPanel1.add(fotocheck);
314:         jAnalyseStart.setBounds(344, 248, 105, 33);
315:         jAnalyseStart.setText("Start");
316:         jAnalyseStart.addActionListener(new ActionListener() {
317:                 public void actionPerformed(ActionEvent evt) {
318:                     jAnalyseStart_ActionPerformed(evt);
319:                 }
320:             });
321:         jAnalyseStart.setToolTipText("Starten der Analyse");
322:         jAnalyseStart.setEnabled(false);
323:         jPanel1.add(jAnalyseStart);
324:         jLabel4.setBounds(32, 256, 73, 16);
325:         jLabel4.setText("Analyse");
326:         jLabel4.setFont(new Font("MS Sans Serif", Font.PLAIN, 14));
327:         jPanel1.add(jLabel4);
328:         ShowAnalyse.setBounds(472, 248, 105, 33);
329:         ShowAnalyse.setText("Anzeigen");
330:         ShowAnalyse.setEnabled(false);
331:         ShowAnalyse.addActionListener(new ActionListener() {
332:                 public void actionPerformed(ActionEvent evt) {
333:                     ShowAnalyse_ActionPerformed(evt);
334:                 }
335:             });
336:         jPanel1.add(ShowAnalyse);
337:         // Ende Komponenten
338:         setResizable(false);
339:         setVisible(true);
340:     }
341:
342:     // Anfang Methoden
343:
344:     /**
345:       * Methode ruft getSite Methode der GetURL Klasse auf
346:       * Objekt der Klasse Rezept wird erzeugt
347:       * Objekt der Klasse WriteXML wird erzeugt
348:       *
349:       * @param evt Rezept aus Internet laden wurde gedrückt
350:       */
351:     public void jButtonKnowStart_ActionPerformed(ActionEvent evt) {
352:         jButtonParseStart.setEnabled(true);
353:
354:         Rezept r = Rezept.getInstance();
355:         r.reset();
356:
357:         WriteXML xml = WriteXML.getInstance();
358:         xml.reset();
359:
360:         jButtonKnowShow.setEnabled(true);
361:
362:         if (this.jRadioButton1.isSelected()) {
363:             System.out.println(this.jTextFieldAdress.getText());
364:             GetURL.getSite("htmlseite.txt",
    this.jTextFieldAdress.getText());
365:         }
366:
367:         if (this.jRadioButton2.isSelected()) {
368:             System.out.println(this.jTextFieldAdress.getText());
```

```
369:             GetURL.getSite("htmlseite.txt",
    this.jTextFieldAdress.getText());
370:         }
371:
372:         if (this.jRadioButton3.isSelected()) {
373:             System.out.println(this.jTextFieldAdress.getText());
374:             GetURL.getSite("htmlseite.txt",
    this.jTextFieldAdress.getText());
375:         }
376:     }
377:
378:     /**
379:      * Methode startet scanHTML der FilterHTML Klasse
380:      *
381:      * @param evt Rezept Parsen Button wurde gedrückt
382:      */
383:     public void jButtonParseStart_ActionPerformed(ActionEvent evt) {
384:         this.jAnalyseStart.setEnabled(true);
385:         this.jButtonParseShow.setEnabled(true);
386:
387:         if (this.jRadioButton1.isSelected()) {
388:             mitFoto = this.fotocheck.isSelected();
389:             parser = new Parser(chefkochxml, 1);
390:             parser.getParameter();
391:             parser.filterHTML();
392:         }
393:
394:         if (this.jRadioButton2.isSelected()) {
395:             mitFoto = this.fotocheck.isSelected();
396:             parser = new Parser(this.rezeptewikixml, 1);
397:             parser.getParameter();
398:             parser.filterHTML();
399:         }
400:
401:         if (this.jRadioButton3.isSelected()) {
402:             mitFoto = this.fotocheck.isSelected();
403:             parser = new Parser("custom_parameter.txt", 1);
404:             parser.getParameter();
405:             parser.filterHTML();
406:         }
407:
408:         WriteXML xml = WriteXML.getInstance();
409:         xml.reset();
410:         xml = WriteXML.getInstance();
411:         xml.xml_header();
412:         xml.write_xml_rezept(false);
413:     }
414:
415:     /**
416:      * Methode startet XML Parser zum einlesen des XML-Rezepts
417:      *
418:      * @param evt Rezept Task Def. Language wurde gedrückt
419:      */
420:     public void jButtonOWL_ActionPerformed(ActionEvent evt) {
421:         this.jButtonKnowShow.setEnabled(true);
422:
423:         WriteXML2 xml2 = WriteXML2.getInstance();
424:         xml2 = xml2.reset();
425:         xml2 = WriteXML2.getInstance();
426:         xml2.xml_header();
427:         xml2.write_xml_rezept(true);
428:
429:         parser = new Parser("../Rezepte/" +
```

```
430:                 Rezept.getInstance().getRezeptname() + ".xml", 4);
431:         parser.getParameter();
432:     }
433:
434:     /**
435:       * Methode erzeugt neuen Dialog zur Anzeige der
     heruntergeladenen HTML Seite
436:       *
437:       * @param evt Anzeigen wurde gedrückt
438:       */
439:     public void jButtonKnowShow_ActionPerformed(ActionEvent evt) {
440:         new ShowResultDialog(this, "Heruntergeladene HTML Seite",
     false,
441:             filename);
442:     }
443:
444:     /**
445:       * Methode erzeugt neuen Dialog zur Anzeige des aktuellen Rezepts
446:       *
447:       * @param evt Anzeigen wurde gedrückt
448:       */
449:     public void jButtonParseShow_ActionPerformed(ActionEvent evt) {
450:         new ShowRezept(this, "Rezept", false);
451:     }
452:
453:     /**
454:       * Methode zeigt Ergebnis an
455:       *
456:       * @param evt Anzeigen wurde gedrückt
457:       */
458:     public void jButtonResultShow_ActionPerformed(ActionEvent evt) {
459:         // TODO hier Quelltext einfügen
460:     }
461:
462:     /**
463:       * Methode zum Beenden des Programs
464:       *
465:       * @param evt Ende Button wurde gedrückt
466:       */
467:     public void jButtonEnde_ActionPerformed(ActionEvent evt) {
468:         WriteXML xml = WriteXML.getInstance();
469:         xml.closeFile();
470:
471:         File file;
472:         Rezept rezept = Rezept.getInstance();
473:         setVisible(false);
474:         dispose();
475:         System.exit(0);
476:     }
477:
478:     /**
479:       * Methode fügt Werte aus der Zwischenablage in Textfield ein
480:       *
481:       * @param evt Einfügen wurde gedrückt
482:       */
483:     public void jButtonPaste_ActionPerformed(ActionEvent evt) {
484:         Clipboard sysClip =
     Toolkit.getDefaultToolkit().getSystemClipboard();
485:         Transferable transfer = sysClip.getContents(null);
486:         String clipboard = "";
487:
488:         try {
489:             clipboard = (String)
     transfer.getTransferData(DataFlavor.stringFlavor);
```

```
490:         } catch (UnsupportedFlavorException e) {
491:             new ErrorDialog(this, "Einfüge Fehler", true,
492:                 "<html> Fehler beim Einfügen aus der Zwischenablage:
    <br>" +
493:                 e.getMessage() + "</html>");
494:         } catch (IOException e) {
495:             new ErrorDialog(this, "IOException", true,
496:                 "<html> Fehler beim Einfügen aus der Zwischenablage:
    <br>" +
497:                 e.getMessage() + "</html>");
498:         }
499:
500:         this.jTextFieldAdress.setText(clipboard);
501:     }
502:
503:     /**
504:       * Methode fügt eine Test URL in Textfield ein
505:       *
506:       * @param evt TestB Button wurde gedrückt
507:       */
508:     public void TestButton_ActionPerformed(ActionEvent evt) {
509:         this.jTextFieldAdress.setText(
510:
    "http://www.chefkoch.de/rezepte/835988123291/Zwiebelkuchen.html");
511:     }
512:
513:     /**
514:       * Methode überprüft ob Checkbox mitFoto gesetzt ist oder nicht
515:       *
516:       * @return boolean mitFoto gesetzt?
517:       */
518:     public static boolean getMitFoto() {
519:         return mitFoto;
520:     }
521:
522:     /**
523:       * Methode setzt Checkbox mitFoto
524:       *
525:       * @param mitFoto markiert Checkbox als gesetzt
526:       */
527:     public void setMitFoto(boolean mitFoto) {
528:         this.mitFoto = mitFoto;
529:     }
530:
531:     /**
532:       * Methode startet die Satzgliedanalyse
533:       *
534:       * @param evt Start wurde gedrückt
535:       */
536:     public void jAnalyseStart_ActionPerformed(ActionEvent evt) {
537:         this.ShowAnalyse.setEnabled(true); // Button freischalten
538:
539:         Rezept rezept = Rezept.getInstance();
540:         String b = "";
541:
542:         for (String a : rezept.getDescription()) {
543:             b = b + a;
544:         }
545:
546:         // Sprachanalyse initialisieren (DA: Rudolf)
547:         Sprachanalyse ana = new Sprachanalyse();
548:         // wenn neues Rezept, dann Protokollnr wieder auf 0 reseten
549:         Protokoll.setCount(0);
```

```
550:         ana.analyse(b); //  Sprachanalyse starten
551:                         //         rezept.showAllTasksOnConsole();
552:     }
553:
554:     /**
555:       * Methode erzeugt neues Objekt der Klasse EditTask
556:       * und generiert für die Klasse die einzelnen Schritte
557:       *
558:       * @param evt Anzeigen wurde gedrückt
559:       */
560:     public void ShowAnalyse_ActionPerformed(ActionEvent evt) {
561:         Rezept rezept = Rezept.getInstance();
562:         rezept.setTasks(new ArrayList<Task>());
563:         rezept.createTasks(); // Tasks aus Rudolf XML Dateien erzeugen
564:         new EditTask(this, "Tasks anzeigen und ändern", false);
565:         this.jButtonOWL.setEnabled(true);
566:     }
567:
568:     // Ende Methoden
569: }
570:
```

```
 1: /**
 2:   *
 3:   * XML Content Handler Oberklasse von der die anderen abgeleitet
   werden
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8:
 9: import org.xml.sax.*;
10: import org.xml.sax.helpers.*;
11: import java.io.*;
12: import javax.xml.parsers.*;
13:
14:
15: public class MyContentHandler implements ContentHandler {
16:     String insertAnf = new String("");
17:     String tabelle = new String();
18:     String spaltseq = new String();
19:     String values = new String("");
20:     String wertseq = new String();
21:     int za = 0; /* Zeile aktiv <=> za=1                */
22:     int m = 0; /* lfd. Spaltenr.i.d. Zeile           */
23:     int cm = 0; /* DTYP(Spalte) ist SQL-char-aehnlich */
24:     String aktwert; /* Wert des aktuellen XML-Elements    */
25:     FileWriter pta; /* Zeichenorientierte Ausgabedatei    */
26:     PrintWriter pd1; /* Methodeninventar fuer Ausgabedatei */
27:     String dsn; /* Name der SQL-Ausgabedatei          */
28:     String wert1 = "";
29:     String wert2 = "";
30:     String wert3 = "";
31:     String wert4 = "";
32:     Task task;
33:
34:   /**
35:     * Methode startet Parsen von Dokument
36:     *
37:     */
38:     public void startDocument() {
39:     }
40:   /**
41:     * Methode beendet Parsen von Dokument
42:     *
43:     */
44:     public void endDocument() {
45:     }
46:   /**
47:     * Methode liest Wert nach dem Begin Element aus
48:     *
49:     */
50:     public void startElement(String uri, String localName, String
   qName,
51:         Attributes attributes) throws SAXException {
52:         int i;
53:         String gVl = null;
54:         String gTy = null;
55:         String gNam = null;
56:         String gVl4;
57:         AttributesImpl a1 = new AttributesImpl(attributes);
58:         int l1 = a1.getLength();
59:
60:         // System.out.println("Element: "+qName+" Attributanzahl:
   "+l1);
```

```
 61:         for (i = 0; i < l1; i++) {
 62:             gVl = a1.getValue(i);
 63:             gTy = a1.getType(i);
 64:             gNam = a1.getQName(i);
 65:
 66:             // System.out.println("++"+i+". Attribut: "+gNam+"
    ("+gTy+") : "+gVl);
 67:         }
 68:
 69:         /* Zeilenanfangsverarbeitung            */
 70:         if (qName.compareTo("zeile") == 0) {
 71:
 72:             qName = "";
 73:
 74:             return;
 75:         }
 76:
 77:         /* Verarbeitung der Elemente in einer Zeile */
 78:         if (za == 1) /* za==1 <=> Zeile ist aktiv    */ { /*
    Verarbeitung von char-aehnlichen SQL-Datentypen */
 79:
 80:             if (a1.getQName(0).compareTo("DT") == 0) {
 81:                 if (gVl.length() >= 4) {
 82:                     gVl4 = gVl.substring(0, 4);
 83:
 84:                     if (gVl4.compareTo("char") == 0) {
 85:                         cm = 1;
 86:                     }
 87:                 }
 88:             }
 89:
 90:             if (m == 0) /* 1. Spalte in der Tabellenzeile */ {
 91:                 spaltseq = spaltseq + qName;
 92:             } else {
 93:                 spaltseq = spaltseq + "," + qName;
 94:             }
 95:         }
 96:     }
 97:   /**
 98:     * Methode liest Zeichen für Zeichen in Variabel aktwert
 99:     *
100:     */
101:     public void characters(char[] ch, int start, int length)
102:         throws SAXException {
103:         String h = null;
104:         h ="";
105:         h = new String(ch, start, length);
106:
107:         wert4 = wert3;
108:         wert3 = wert2;
109:         wert2 = wert1;
110:         wert1 = aktwert;
111:           aktwert = "";
112:
113:         aktwert = h;
114:     }
115:
116:     public void skippedEntity(String name) throws SAXException {
117:     }
118:
119:     public void processingInstruction(String target, String data)
120:         throws SAXException {
121:     }
```

```
122:
123:     public void ignorableWhitespace(char[] ch, int start, int length)
124:         throws SAXException {
125:     }
126:   /**
127:     * Methode liest Wert vor dem Ende Element aus
128:     *
129:     */
130:     public void endElement(String uri, String localName, String qName)
131:         throws SAXException {
132:
133:         if (qName.compareTo("tabname") == 0) {
134:             tabelle = aktwert;
135:
136:         }
137:
138:         /* Zeilenendeverarbeitung          */
139:         if (qName.compareTo("beginn") == 0) {
140:         }
141:
142:         /* Verarbeitung der Elemente in einer Zeile */
143:         if (za == 1) /* za==1 <=> Zeile ist aktiv    */ {
144:             String hoch = new String();
145:
146:             if (cm == 1) /* DTYP(Spalte) ist SQL-char-aehnlich */ {
147:                 hoch = "'";
148:                 cm = 0;
149:             } else {
150:                 hoch = "";
151:             }
152:
153:             if (m == 0) {
154:                 m = 1;
155:                 wertseq = wertseq + hoch + aktwert + hoch;
156:             } else {
157:                 m = m + 1;
158:                 wertseq = wertseq + "," + hoch + aktwert + hoch;
159:             }
160:         }
161:     }
162:
163:     public Parameter getParams(){
164:       return null;
165:     }
166:     public void endPrefixMapping(String prefix) throws SAXException {
    //System.out.println("-Pr-> Praefix: "+prefix);
167:     }
168:
169:     public void startPrefixMapping(String prefix, String uri)
170:         throws SAXException { //System.out.println("-PrS-> Praefix:
    "+prefix);
171:     }
172:
173:     public void setDocumentLocator(Locator locator) {
    //System.out.println("-L-> Locator: "+locator.toString());
174:     }
175: }
176:
```

```
 1: /**
 2:   *
 3:   * XML Error Handler
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8:
 9: import org.xml.sax.*;
10: import org.xml.sax.helpers.*;
11: import java.io.*;
12: import javax.xml.parsers.*;
13:
14:
15: public class MyErrorHandler implements ErrorHandler {
16:     public void warning(SAXParseException ep) throws SAXException {
17:         new ErrorDialog(null, "Warning", true,
18:             "<html> Parser meldet WARNUNG: " + ep.toString() +
19:             "<br> an der Entity        : " + ep.getPublicId() +
20:             "<br> Zeile,Spalte         : " + ep.getLineNumber() + "," +
21:             ep.getColumnNumber() + "</html>");
22:
23:         System.out.println("Parser meldet WARNUNG: " + ep.toString());
24:         System.out.println("an der Entity        : " +
    ep.getPublicId());
25:         System.out.println("Zeile,Spalte         : " +
    ep.getLineNumber() +
26:             "," + ep.getColumnNumber());
27:     }
28:
29:     public void error(SAXParseException ep) throws SAXException {
30:         new ErrorDialog(null, "Fehler", true,
31:             "<html> Parser meldet Fehler: " + ep.toString() +
32:             "<br> an der Entity        : " + ep.getPublicId() +
33:             "<br> Zeile,Spalte         : " + ep.getLineNumber() + "," +
34:             ep.getColumnNumber() + "</html>");
35:         System.out.println("Parser meldet FEHLER : " + ep.toString());
36:         System.out.println("an der Entity        : " +
    ep.getPublicId());
37:         System.out.println("Zeile,Spalte         : " +
    ep.getLineNumber() +
38:             "," + ep.getColumnNumber());
39:     }
40:
41:     public void fatalError(SAXParseException ep) throws SAXException {
42:         new ErrorDialog(null, "Fataler FEHLER", true,
43:             "<html> Parser meldet Fataler Fehler !!!: " +
    ep.toString() +
44:             "<br> an der Entity        : " + ep.getPublicId() +
45:             "<br> Zeile,Spalte         : " + ep.getLineNumber() + "," +
46:             ep.getColumnNumber() + "</html>");
47:         System.out.println("Fataler FEHLER !!!   : " + ep.toString());
48:         System.out.println("an der Entity        : " +
    ep.getPublicId());
49:         System.out.println("Zeile,Spalte         : " +
    ep.getLineNumber() +
50:             "," + ep.getColumnNumber());
51:     }
52: }
53:
```

```
 1: import java.util.ArrayList;
 2:
 3: import javax.swing.event.*;
 4:
 5: /**
 6:   *
 7:   * Klasse erzeugt eigenes Table Model für Anzeige der Zutaten in der
    Rezept
 8:   * Anzeige
 9:   *
10:   * @version 1.0 vom 25.02.2009
11:   * @author Henning Budde
12:   */
13: import javax.swing.table.*;
14:
15:
16: class MyTableModel extends AbstractTableModel {
17:     private Rezept rezept = Rezept.getInstance();
18:     ArrayList<Zutat> zutaten = rezept.getZutaten();
19:     int zutatensize = zutaten.size();
20:     private String[] columnNames = { "Menge", "Einheit", "Bezeichnung"
    };
21:     private Object[][] data = new Object[20][20];
22:     int row = -1;
23:
24:     public MyTableModel() {
25:         for (Zutat z : zutaten) {
26:             System.out.println(row++);
27:             this.setValueAt(z.getMenge(), row, 0);
28:             this.setValueAt(z.getEinheit(), row, 1);
29:             this.setValueAt(z.getBez(), row, 2);
30:
31:             System.out.println(z.getBez());
32:         }
33:     }
34:
35:     public int getColumnCount() {
36:         return columnNames.length;
37:     }
38:
39:     public int getRowCount() {
40:         return data.length;
41:     }
42:
43:     public String getColumnName(int col) {
44:         switch (col) {
45:         case 0:
46:             return "Menge";
47:
48:         case 1:
49:             return "Einheit";
50:
51:         case 2:
52:             return "Bezeichnung";
53:
54:         default:
55:             return null;
56:         }
57:     }
58:
59:     public Object getValueAt(int row, int col) {
60:         return data[row][col];
61:     }
```

```
62:
63:     public Class getColumnClass(int c) {
64:         return getValueAt(0, c).getClass();
65:     }
66:
67:     public boolean isCellEditable(int row, int col) {
68:         if (col < 2) {
69:             return false;
70:         } else {
71:             return true;
72:         }
73:     }
74:
75:     public void setValueAt(Object value, int row, int col) {
76:         data[row][col] = value;
77:         fireTableCellUpdated(row, col);
78:     }
79: }
80:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Aufname der Rezeptparameter
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8:
 9: public class Parameter {
10:     // Anfang Attribute
11:     private String beginn;
12:     private String rezeptname_start;
13:     private String rezeptname_end;
14:     private String personen_start;
15:     private String personen_end;
16:     private String zutaten_start;
17:     private String zutat_start;
18:     private String zutat_menge;
19:     private String zutat_einheit;
20:     private String zutat_bez_start;
21:     private String zutat_bez_end;
22:     private String zutaten_end;
23:     private String rezept_description_start;
24:     private String rezept_description_end;
25:     private String end;
26:     private String rezept_description_start_foto;
27:
28:     // Ende Attribute
29:     // Anfang Methoden
30:   /**
31:     * Alle Parameter in GUI anzeigen
32:     *
33:     */
34:     public void showAll() {
35:         new ShowResultDialog(null, "Parameter", false, this);
36:     }
37:
38:     public String getBeginn() {
39:         return beginn;
40:     }
41:
42:     public void setBeginn(String beginn) {
43:         this.beginn = beginn;
44:     }
45:
46:     public String getRezeptname_start() {
47:         return rezeptname_start;
48:     }
49:
50:     public void setRezeptname_start(String rezeptname_start) {
51:         this.rezeptname_start = rezeptname_start;
52:     }
53:
54:     public String getRezeptname_end() {
55:         return rezeptname_end;
56:     }
57:
58:     public void setRezeptname_end(String rezeptname_end) {
59:         this.rezeptname_end = rezeptname_end;
60:     }
61:
62:     public String getPersonen_start() {
63:         return personen_start;
```

```
 64:     }
 65:
 66:     public void setPersonen_start(String personen_start) {
 67:         this.personen_start = personen_start;
 68:     }
 69:
 70:     public String getPersonen_end() {
 71:         return personen_end;
 72:     }
 73:
 74:     public void setPersonen_end(String personen_end) {
 75:         this.personen_end = personen_end;
 76:     }
 77:
 78:     public String getZutaten_start() {
 79:         return zutaten_start;
 80:     }
 81:
 82:     public void setZutaten_start(String zutaten_start) {
 83:         this.zutaten_start = zutaten_start;
 84:     }
 85:
 86:     public String getZutat_start() {
 87:         return zutat_start;
 88:     }
 89:
 90:     public void setZutat_start(String zutat_start) {
 91:         this.zutat_start = zutat_start;
 92:     }
 93:
 94:     public String getZutat_menge() {
 95:         return zutat_menge;
 96:     }
 97:
 98:     public void setZutat_menge(String zutat_menge) {
 99:         this.zutat_menge = zutat_menge;
100:     }
101:
102:     public String getZutat_einheit() {
103:         return zutat_einheit;
104:     }
105:
106:     public void setZutat_einheit(String zutat_einheit) {
107:         this.zutat_einheit = zutat_einheit;
108:     }
109:
110:     public String getZutat_bez_start() {
111:         return zutat_bez_start;
112:     }
113:
114:     public void setZutat_bez_start(String zutat_bez_start) {
115:         this.zutat_bez_start = zutat_bez_start;
116:     }
117:
118:     public String getZutat_bez_end() {
119:         return zutat_bez_end;
120:     }
121:
122:     public void setZutat_bez_end(String zutat_bez_end) {
123:         this.zutat_bez_end = zutat_bez_end;
124:     }
125:
126:     public String getZutaten_end() {
```

```
127:         return zutaten_end;
128:     }
129:
130:     public void setZutaten_end(String zutaten_end) {
131:         this.zutaten_end = zutaten_end;
132:     }
133:
134:     public String getRezept_description_start() {
135:         return rezept_description_start;
136:     }
137:
138:     public void setRezept_description_start(String
     rezept_description_start) {
139:         this.rezept_description_start = rezept_description_start;
140:     }
141:
142:     public String getRezept_description_end() {
143:         return rezept_description_end;
144:     }
145:
146:     public void setRezept_description_end(String
     rezept_description_end) {
147:         this.rezept_description_end = rezept_description_end;
148:     }
149:
150:     public String getEnd() {
151:         return end;
152:     }
153:
154:     public void setEnd(String end) {
155:         this.end = end;
156:     }
157:
158:     public String getRezept_description_start_foto() {
159:         return rezept_description_start_foto;
160:     }
161:
162:     public void setRezept_description_start_foto(
163:         String rezept_description_start_foto) {
164:         this.rezept_description_start_foto =
     rezept_description_start_foto;
165:     }
166:
167:     // Ende Methoden
168: }
169:
```

```
 1: /**
 2:   * XML Content Handler für Rezept Parameter
 3:   *
 4:   * @version 1.0 vom 25.02.2009
 5:   * @author Henning Budde
 6:   */
 7: import java.io.*;
 8: import javax.xml.parsers.*;
 9: import org.xml.sax.*;
10: import org.xml.sax.helpers.*;
11:
12: public class ParameterContentHandler extends MyContentHandler {
13:
14:   private Parameter params = new Parameter();
15:
16:     public ParameterContentHandler(String filename) {
17:         super();
18:     }
19:     // Anfang Attribute
20:     // Ende Attribute
21:
22:     // Anfang Methoden
23:
24:   /**
25:     * {@inheritDoc}
26:     */
27:     public void endElement(String uri, String localName, String qName)
28:         throws SAXException {
29:
30:         if (qName.compareTo("beginn") == 0) {
31:             this.params.setBeginn(aktwert);
32:         }
33:
34:         if (qName.compareTo("rezeptname-start") == 0) {
35:
36:             this.params.setRezeptname_start(aktwert);
37:         }
38:
39:         if (qName.compareTo("rezeptname-end") == 0) {
40:
41:             this.params.setRezeptname_end(aktwert);
42:         }
43:
44:         if (qName.compareTo("personen-start") == 0) {
45:
46:             this.params.setPersonen_start(aktwert);
47:         }
48:
49:         if (qName.compareTo("personen-end") == 0) {
50:             this.params.setPersonen_end(aktwert);
51:         }
52:
53:         if (qName.compareTo("zutaten-start") == 0) {
54:
55:                 this.params.setZutaten_start(aktwert);
56:         }
57:
58:         if (qName.compareTo("zutat-start") == 0) {
59:
60:             this.params.setZutat_start(aktwert);
61:         }
62:
63:         if (qName.compareTo("zutat-einheit") == 0) {
```

```
 64:
 65:             this.params.setZutat_einheit(aktwert);
 66:         }
 67:
 68:         if (qName.compareTo("zutat-menge") == 0) {
 69:
 70:             this.params.setZutat_menge(aktwert);
 71:         }
 72:
 73:         if (qName.compareTo("zutat-bez-start") == 0) {
 74:
 75:             this.params.setZutat_bez_start(aktwert);
 76:         }
 77:
 78:         if (qName.compareTo("zutat-bez-end") == 0) {
 79:
 80:             this.params.setZutat_bez_end(aktwert);
 81:         }
 82:
 83:         if (qName.compareTo("zutaten-end") == 0) {
 84:
 85:             this.params.setZutaten_end(aktwert);
 86:         }
 87:
 88:         if (qName.compareTo("rezept-description-start") == 0) {
 89:             this.params.setRezept_description_start(aktwert);
 90:         }
 91:         if (qName.compareTo("rezept-description-start-foto") == 0) {
 92:             this.params.setRezept_description_start_foto(aktwert);
 93:         }
 94:
 95:         if (qName.compareTo("rezept-description-end") == 0) {
 96:
 97:             this.params.setRezept_description_end(aktwert);
 98:         }
 99:
100:         if (qName.compareTo("parameter") == 0) {
101:
102:         }
103:     }
104:
105:   public Parameter getParams() {
106:     return params;
107:   }
108:
109:   public void setParams(Parameter params) {
110:     this.params = params;
111:   }
112:
113:     // Ende Methoden
114: }
115:
```

```
 1:
 2:
 3: /**
 4:   *
 5:   * XML Parser Klasse
 6:   *
 7:   * @version 1.0 vom 25.02.2009
 8:   * @author Henning Budde
 9:   */
10: import java.io.*;
11: import org.xml.sax.*;
12: import org.xml.sax.helpers.*;
13: import javax.xml.parsers.*;
14:
15: public class Parser {
16:     // Anfang Attribute
17:     private String filename;
18:     private MyContentHandler handler;
19: //    private Handler
20:     private MyErrorHandler ehandler;
21:     private Parameter params;
22:   private static int count;
23:
24:     public Parser(String filename, int i) { // i kodiert die Auswahl
    des Parsers
25:         this.filename = filename;
26:         switch (i) {
27:
28:         case 1:
29:             this.handler = new ParameterContentHandler(filename);
30:
31:             break;
32:
33:         case 2:        // Content Handler für Rudolf XML Dateien
34: //             System.out.println("Parser 2 wurde gestartet");
35:
36:              this.handler = new RezeptContentHandler(filename,
    count++);
37: //             count++;
38:              break;
39:
40: //         case 3: this.handler = new TaskContentHandler(filename);
41: //              break;
42:
43:          case 4: this.handler = new TDLContentHandler(filename);  //
    schreiben der *.TDL Datei
44:               break;
45:
46:         }
47:
48:         this.ehandler = new MyErrorHandler();
49:     }
50:
51:     // Ende Attribute
52:
53:     // Anfang Methoden
54:     public void getParameter() {
55:         try {
56:             SAXParserFactory factory = SAXParserFactory.newInstance();
57:             factory.setValidating(true);
58:
59:             SAXParser saxpars1 = factory.newSAXParser();
60:             XMLReader read1 = saxpars1.getXMLReader();
```

```
 61:             read1.setContentHandler(this.handler);
 62:             read1.setErrorHandler(ehandler);
 63:
 64:             boolean w1 = saxpars1.isValidating();
 65:
 66:
 67:
 68:             String h1 = new File(filename).toURL().toString();
 69: //            System.out.println("URI = " + h1);
 70:             read1.parse(new File(filename).toURL().toString());
 71:         } catch (SAXParseException ep) { // A parsing error occurred;
    the xml input is not valid
 72:             new ErrorDialog(null, "SAXParseException", true,
 73:                 "<html> Parser meldet Fehler" + ep.toString() +
 74:                 "<br> an der Entity:  " + ep.getPublicId() +
 75:                 "<br>Zeile, Spalte: " + ep.getLineNumber() + "," +
 76:                 ep.getColumnNumber() + "</html>");
 77:         } catch (SAXException e) { // A parsing error occurred; the
    xml input is not valid
 78:             new ErrorDialog(null, "SAXException", true,
 79:                 "Da ist eine XML-Invaliditaet in " + filename + "
    :\n" +
 80:                 e.getMessage());
 81:         } catch (ParserConfigurationException e) {
 82:             new ErrorDialog(null, "ParserConfigurationError", true,
 83:                 "<htm>Ein Parser-Konfigurationsproblem: <br>" +
    e.getMessage()+"</html>");
 84:         } catch (IOException e) {
 85:             new ErrorDialog(null, "IOException", true,
 86:                 "<html>XML-File = " + filename +
 87:                 " konnte nicht geoeffnet werden <br> " +
    e.getMessage() +"</hmtl>");
 88:                 System.out.println(e.getMessage());
 89:         }
 90:     }
 91:
 92:     public void filterHTML() {
 93:         params = this.handler.getParams();
 94:         System.out.println(params.getBeginn());
 95:
 96:         FilterHTML html = new FilterHTML("htmlseite.txt", params);
 97:         html.scanHTML();
 98:         html.getZutatenliste();
 99:     }
100:
101:   public  void setCount(int count) {
102:     this.count = count;
103:   }
104:
105:     // Ende Methoden
106: }
107:
```

```java
/**
  *
  * Rezept Klasse zur Sicherung aller Rezeptinformationen
  *
  * @version 1.0 vom 25.02.2009
  * @author Henning Budde
  */
import java.util.ArrayList;


public class Rezept {
    // Anfang Attribute
    private static String rezeptname;
    private static String personen;
    private static ArrayList<Zutat> zutaten;
    private static ArrayList<String> description;
    private static ArrayList<Task> tasks;
    public static boolean instance = false;

    // Ende Attribute
    public static Rezept rezept;

    private Rezept() {
        zutaten = new ArrayList<Zutat>();
        description = new ArrayList<String>();
        tasks = new ArrayList<Task>();
    }

    /**
      * Singleton Muster
      *
      * @return Rezept
      */
    public static Rezept getInstance() {
        if (instance) {
            return rezept;
        } else {
            instance = true;
            rezept = new Rezept();

            return rezept;
        }
    }

    // Anfang Methoden
    public String getRezeptname() {
        return rezeptname;
    }

    /**
      * Rezeptname wird gesetzt und Umlaute, Sonderzeichen entfernt
      *
      * @param rezeptname uebergebener Rezeptname
      */
    public void setRezeptname(String rezeptname) {
        System.out.println(rezeptname);
        rezeptname = rezeptname.replace("ä", "ae");
        rezeptname = rezeptname.replace("ü", "ue");
        rezeptname = rezeptname.replace("ß", "ss");
        rezeptname = rezeptname.replace("ö", "oe");
        rezeptname = rezeptname.replace("Ä", "Ae");
        rezeptname = rezeptname.replace("Ü", "Ue");
        rezeptname = rezeptname.replace("Ö", "Oe");
```

```
 64:         rezeptname = rezeptname.replace(" ", "_");
 65:         System.out.println(rezeptname);
 66:
 67:         this.rezeptname = rezeptname;
 68:     }
 69:
 70:     public String getPersonen() {
 71:         return personen;
 72:     }
 73:
 74:     public void setPersonen(String personen) {
 75:         this.personen = personen;
 76:     }
 77:
 78:     public ArrayList<Zutat> getZutaten() {
 79:         //    return Zutaten ;
 80:         return zutaten;
 81:     }
 82:
 83:     public void setZutaten(ArrayList<Zutat> Zutaten) {
 84:         this.zutaten = Zutaten;
 85:     }
 86:
 87:     public ArrayList<String> getDescription() {
 88:         return description;
 89:     }
 90:
 91:     public void setDescription(ArrayList<String> description) {
 92:         this.description = description;
 93:     }
 94:
 95:     /**
 96:       * Alle Tasks auf der Konsole anzeigen
 97:       *
 98:       */
 99:     public void showAllTasksOnConsole() {
100:         for (Task t : tasks) {
101:             System.out.println("Wort: \t" + t.getWort());
102:             System.out.println("Wortart: \t" + t.getWortart());
103:             System.out.println("Phrase: \t" + t.getPhrase());
104:             System.out.println("Umstand: \t" + t.getUmstand());
105:         }
106:     }
107:
108:     public ArrayList<Task> getTasks() {
109:         return tasks;
110:     }
111:
112:     public void setTasks(ArrayList<Task> tasks) {
113:         this.tasks = tasks;
114:     }
115:
116:     /**
117:       * Tasks erzeugen
118:       *
119:       */
120:     public void createTasks() {
121:         rezept.setTasks(new ArrayList<Task>());
122:
123:         Parser p = null;
124:
125:         for (int i = 0; i < this.description.size(); i++) {
126:             System.out.println("Protokoll: " + i + " wird
bearbeitet");
```

```
127:             p = new Parser(".\\Protokolle\\protokoll_" + i + ".xml",
    2);
128:             p.getParameter();
129:         }
130:
131:         p.setCount(0); // Nachdem alle Tasks erzeugt wurden, Task
    Zähler wieder auf Null setzen
132:     }
133:
134:     /**
135:       * Überprüfen ob ein String eine Zutat ist
136:       *
137:       * @param vergleich Zeichenkette zur Überprüfung
138:       * @return true wenn String eine Zutat ist
139:       */
140:     public static boolean isZutat(String vergleich) {
141:         for (Zutat z : zutaten) {
142:             if (z.getBez().contains(vergleich)) {
143:                 return true;
144:             }
145:         }
146:
147:         return false;
148:     }
149:
150:     /**
151:       * Methode erzeugt neues Rezept
152:       *
153:       */
154:     public static void reset() {
155:         rezept = new Rezept();
156:     }
157:
158:     public void resetRef() {
159:         for (Zutat z1 : getZutaten()) {
160:             z1.setReferenz(-1);
161:         }
162:     }
163:
164:     // Ende Methoden
165: }
166:
```

```
 1: /**
 2:   * XML Content Handler für Rezept Inhalte
 3:   *
 4:   * @version 1.0 vom 25.02.2009
 5:   * @author Henning Budde
 6:   */
 7: import org.xml.sax.*;
 8: import org.xml.sax.helpers.*;
 9:
10: import java.util.ArrayList;
11: import java.util.regex.Matcher;
12: import java.util.regex.Pattern;
13:
14:
15: public class RezeptContentHandler extends MyContentHandler {
16:     private Rezept r;
17:     private ArrayList<Task> tasks;
18:     private int count;
19:
20:     public RezeptContentHandler(String filename, int count) {
21:         super();
22:         this.count = count;
23:         r = Rezept.getInstance();
24:         tasks = r.getTasks();
25:
26:         //        System.out.println("RezeptContentHandler wurder
    erzeugt");
27:     }
28:
29:     // Anfang Attribute
30:
31:     // Ende Attribute
32:
33:     // Anfang Methoden
34:
35:     /**
36:       * {@inheritDoc}
37:       *
38:       */
39:     public void endElement(String uri, String localName, String qName)
40:         throws SAXException {
41:         if (qName.compareTo("wort") == 0) {
42:             task = new Task();
43:             task.setTasknr(count);
44:             task.setWort(aktwert);
45:
46:             //          System.out.println("ausdruck: "+aktwert);
       // für Debug Ausgabe
47:         }
48:
49:         if (qName.compareTo("wortart") == 0) {
50:             if (aktwert.contains("Verbpartikel")) {
51:                 task.setWortart("Verbpartikel");
52:             } else if (aktwert.contains("Verb")) {
53:                 task.setWortart("Verb");
54:             } else if (aktwert.contains("Hilfsverb")) {
55:                 task.setWortart("Hilfsverb");
56:             } else if (task.getWort().contains("°")) {
57:                 task.setWortart("Zahl");
58:             } else if (aktwert.contains("Wortart nich")) {
59:                 task.setWortart("");
60:             } else {
61:                 task.setWortart(aktwert);
```

```
 62:             }
 63:         }
 64:         if (qName.compareTo("phrase") == 0) {
 65:             try {
 66:                 task.setPhrase(Integer.parseInt(aktwert));
 67:             } catch (NumberFormatException e) {
 68:                 new ErrorDialog(null, "RezeptContentHandler: Fehler
    in Phrase",
 69:                         true, e.getMessage());
 70:             } finally {
 71:             }
 72:         }
 73:         if (qName.compareTo("umstand") == 0) {
 74:             task.setUmstand(aktwert);
 75:
 76:             if (aktwert.equals("" + task.getPhrase())) {
 77:                 task.setUmstand("");
 78:             }
 79:
 80:             // HANDKORREKTUREN
 81:             if (task.getWort().equals("vermischen")) {
 82:                 task.setWortart("Verb");
 83:             }
 84:             if (task.getWort().equals("Öl")) {
 85:                 task.setWortart("Substantiv");
 86:             }
 87:             if (task.getWort().equals("Hefeteig")) {
 88:                 task.setIsentity(true);
 89:             }
 90:             if (task.getWort().equals("Backofen")) {
 91:                 task.setIsentity(true);
 92:             }
 93:             if (task.getWort().equals("garen")) {
 94:                 task.setWortart("Verb");
 95:             }
 96:             if (task.getWort().equals("untermischen")) {
 97:                 task.setWortart("Verb");
 98:             }
 99:             if (task.getWort().equals("Anschließend")) {
100:                 task.setWortart("Praeposition");
101:             }
102:
103:             tasks.add(task);
104:         }
105:     }
106:
107:     public ArrayList<Task> getTasks() {
108:         return tasks;
109:     }
110:
111:     public void setTasks(ArrayList<Task> tasks) {
112:         this.tasks = tasks;
113:     }
114:
115:     // Ende Methoden
116: }
117:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Anzeige verschiedener Ergebnisse
 4:   * Enthält eine JTextArea für Inhalte
 5:   *
 6:   * @version 1.0 vom 25.102.2009
 7:   * @author Henning Budde
 8:   */
 9:
10: import java.awt.*;
11: import java.awt.event.*;
12: import java.io.*;
13: import javax.swing.*;
14: import javax.swing.event.*;
15:
16:
17: public class ShowResultDialog extends JDialog {
18:     // Anfang Attribute
19:     private JButton OKButton = new JButton();
20:     private FileReader fr = null;
21:     private BufferedReader buf = null;
22:
23:   private JScrollBar jScrollbar1 = new JScrollBar();
24:   private JScrollPane jScrollPane1 = new JScrollPane();
25:   private JTextArea jTextArea1 = new JTextArea("");
26:     // Ende Attribute
27:     public ShowResultDialog(JFrame owner, String title, boolean modal,
28:         String filename) {
29:         super(owner, title, modal);
30:         jTextArea1.setTabSize(2);
31:         try {
32:             fr = new FileReader(filename);
33:             buf = new BufferedReader(fr);
34:             String text=buf.readLine();
35:             do {
36:
37:                 jTextArea1.append("\t"+text+"\n");
38:                 text= buf.readLine();
39:             } while (text != null);
40:         } catch (IOException e) {
41:             new ErrorDialog(null, "Datei öffnen Fehler", true,
42:                 "<html> Fehler beim Öffnen der Datei : <br>" +
    e.getMessage() +
43:                 "</html>");
44:         } finally {
45:         }
46:
47:         // Dialog-Initialisierung
48:         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
49:
50:     int frameWidth = 822;
51:     int frameHeight = 686;
52:         setSize(frameWidth, frameHeight);
53:
54:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
55:         int x = (d.width - getSize().width) / 2;
56:         int y = (d.height - getSize().height) / 2;
57:         setLocation(x, y);
58:
59:         Container cp = getContentPane();
60:         cp.setLayout(null);
61:         // Anfang Komponenten
62:
```

```
 63:     OKButton.setBounds(400, 576, 97, 33);
 64:         OKButton.setText("OK");
 65:         OKButton.addActionListener(new ActionListener() {
 66:                 public void actionPerformed(ActionEvent evt) {
 67:                     OKButton_ActionPerformed(evt);
 68:                 }
 69:             });
 70:         cp.add(OKButton);
 71:     jScrollPane1.setBounds(0, 0, 817, 553);
 72:     cp.add(jScrollPane1);
 73: 
 74:     jTextArea1.setBounds(-2, -2, 793, 553);
 75:     jTextArea1.setCaretPosition(0);
 76:     jScrollPane1.setViewportView(jTextArea1);
 77:     jTextArea1.setEditable(false);
 78:         // Ende Komponenten
 79:         setResizable(false);
 80:         setVisible(true);
 81:     }
 82:     public ShowResultDialog(JFrame owner, String title, boolean modal,
 83:         Parameter params) {
 84:         super(owner, title, modal);
 85:         jTextArea1.setTabSize(15);
 86:         jTextArea1.append("beginn: \t"+params.getBeginn()+"\n");
 87:         jTextArea1.append("rezeptname_start:
    \t"+params.getRezeptname_start()+"\n");
 88:         jTextArea1.append("rezeptname_end:
    \t"+params.getRezeptname_end()+"\n");
 89:         jTextArea1.append("personen_start:
    \t"+params.getPersonen_start()+"\n");
 90:         jTextArea1.append("personen_end:
    \t"+params.getPersonen_end()+"\n");
 91:         jTextArea1.append("zutaten-start:
    \t"+params.getZutaten_start()+"\n");
 92:         jTextArea1.append("zutat-start:
    \t"+params.getZutat_start()+"\n");
 93:         jTextArea1.append("zutat-menge:
    \t"+params.getZutat_menge()+"\n");
 94:         jTextArea1.append("zutat-einheit:
    \t"+params.getZutat_einheit()+"\n");
 95:         jTextArea1.append("zutat-bez-start:
    \t"+params.getZutat_bez_start()+"\n");
 96:         jTextArea1.append("zutat-bez-end:
    \t"+params.getZutat_bez_end()+"\n");
 97:         jTextArea1.append("zutaten-end:
    \t"+params.getZutaten_end()+"\n");
 98:         jTextArea1.append("rezept-description-start:
    \t"+params.getRezept_description_start()+"\n");
 99:         jTextArea1.append("rezept-description-start-foto:
    \t"+params.getRezept_description_start_foto()+"\n");
100:         jTextArea1.append("rezept-description-end:
    \t"+params.getRezept_description_end()+"\n");
101:         jTextArea1.append("end: \t"+params.getEnd()+"\n");
102: //         System.out.println(params.getRezeptname_start());
103:         // Dialog-Initialisierung
104:         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
105: 
106:     int frameWidth = 822;
107:     int frameHeight = 686;
108:         setSize(frameWidth, frameHeight);
109: 
110:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
111:         int x = (d.width - getSize().width) / 2;
```

```
112:         int y = (d.height - getSize().height) / 2;
113:         setLocation(x, y);
114:
115:         Container cp = getContentPane();
116:         cp.setLayout(null);
117:         // Anfang Komponenten
118:
119:     OKButton.setBounds(240, 584, 97, 33);
120:         OKButton.setText("OK");
121:         OKButton.addActionListener(new ActionListener() {
122:                 public void actionPerformed(ActionEvent evt) {
123:                     OKButton_ActionPerformed(evt);
124:                 }
125:             });
126:         cp.add(OKButton);
127:     jScrollPane1.setBounds(0, 0, 817, 553);
128:     cp.add(jScrollPane1);
129:
130:     jTextArea1.setBounds(-2, -2, 793, 553);
131:     jTextArea1.setCaretPosition(0);
132:     jScrollPane1.setViewportView(jTextArea1);
133:     jTextArea1.setEditable(false);
134:         // Ende Komponenten
135:         setResizable(false);
136:         setVisible(true);
137:  }
138:
139:     // Anfang Methoden
140:     public void OKButton_ActionPerformed(ActionEvent evt) {
141:         setVisible(false);
142:         dispose();
143:     }
144:
145:     // Ende Methoden
146: }
147:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Anzeige eines Rezepts
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.awt.*;
 9: import java.awt.event.*;
10:
11: import java.util.ArrayList;
12:
13: import javax.swing.*;
14: import javax.swing.event.*;
15: import javax.swing.table.*;
16:
17:
18: public class ShowRezept extends JDialog {
19:     // Anfang Attribute
20:     private JLabel jLabel1 = new JLabel();
21:     private JTable jTable1;
22:     private Rezept rezept = Rezept.getInstance();
23:     private JScrollPane jScrollPane1 = new JScrollPane();
24:     private JTextArea jTextArea1 = new JTextArea("");
25:     private JButton jButton1 = new JButton();
26:     private JButton change = new JButton();
27:
28:     // Ende Attribute
29:     public ShowRezept(JFrame owner, String title, boolean modal) {
30:         // Dialog-Initialisierung
31:         super(owner, title, modal);
32:         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
33:
34:         int frameWidth = 599;
35:         int frameHeight = 622;
36:         setSize(frameWidth, frameHeight);
37:
38:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
39:         int x = (d.width - getSize().width) / 2;
40:         int y = (d.height - getSize().height) / 2;
41:         setLocation(x, y);
42:
43:         Container cp = getContentPane();
44:         setLayout(null);
45:         // Anfang Komponenten
46:         //    String column[]= {"Menge","Einheit","Bezeichnung"};
47:         jTable1 = new JTable(new MyTableModel());
48:         jLabel1.setBounds(48, 32, 820, 48);
49:         jTable1.setPreferredScrollableViewportSize(new Dimension(500,
    70));
50:         jTable1.setFillsViewportHeight(true);
51:
52:         //Create the scroll pane and add the table to it.
53:         JScrollPane scrollPane = new JScrollPane(jTable1);
54:         JTableHeader header = jTable1.getTableHeader();
55:
56:         jLabel1.setText(rezept.getRezeptname());
57:         jLabel1.setFont(new Font("MS Sans Serif", Font.PLAIN, 31));
58:         cp.add(jLabel1);
59:         jTable1.setBounds(56, 88, 513, 233);
60:         cp.add(jTable1);
61:
62:         //    ("Menge","Einheit","Zutat");
```

```
 63:         jScrollPane1.setBounds(56, 344, 513, 161);
 64:
 65:         //    jTextArea1.setText("text");
 66:         jTextArea1.setBounds(-2, -2, 513, 161);
 67:         jScrollPane1.setViewportView(jTextArea1);
 68:         jButton1.setBounds(336, 528, 153, 33);
 69:
 70:         for (String desc : rezept.getDescription()) {
 71:             // Handkorrektur
 72:             desc = desc.replace("darübergeben", "darüber geben");
 73:             jTextArea1.append(desc + "\n");
 74:         }
 75:
 76:         jButton1.setText("Schließen");
 77:
 78:         jTextArea1.setCaretPosition(0);
 79:
 80:         jButton1.addActionListener(new ActionListener() {
 81:                 public void actionPerformed(ActionEvent evt) {
 82:                     jButton1_ActionPerformed(evt);
 83:                 }
 84:             });
 85:         cp.add(jButton1);
 86:         cp.add(scrollPane);
 87:         cp.add(jScrollPane1);
 88:         change.setBounds(144, 528, 155, 33);
 89:         change.setText("Änderungen übernehmen");
 90:         change.addActionListener(new ActionListener() {
 91:                 public void actionPerformed(ActionEvent evt) {
 92:                     change_ActionPerformed(evt);
 93:                 }
 94:             });
 95:         change.setEnabled(true);
 96:         cp.add(change);
 97:         // Ende Komponenten
 98:         setResizable(false);
 99:         setVisible(true);
100:     }
101:     // Anfang Methoden
102:     public void jButton1_ActionPerformed(ActionEvent evt) {
103:         this.setVisible(false);
104:         this.dispose();
105:     }
106:     public void change_ActionPerformed(ActionEvent evt) {
107:         ArrayList<String> desc = new ArrayList<String>();
108:         int col = jTextArea1.getLineCount();
109:         int start;
110:         int ende;
111:         String a = "";
112:         a = jTextArea1.getText();
113:
114:         String[] b = a.split("\n");
115:
116:         for (int c = 0; c < b.length; c++) {
117:             desc.add(b[c]);
118:         }
119:         rezept.setDescription(desc);
120:     }
121:     // Ende Methoden
122: }
123:
```

```java
/**
  *
  * Klasse zur Aufnahme der Protokoll Ergebnisse
  *
  * @version 1.0 vom 25.02.2009
  * @author Henning Budde
  */

public class Task {

  // Anfang Attribute
  private String wort;
  private String wortart;
  private boolean iszutat;
  private boolean isentity;
  private int phrase;
  private String umstand;
  private int tasknr;
  // Ende Attribute

  // Anfang Methoden
  public String getWort() {
    return wort;
  }

  public void setWort(String wort) {
    this.wort = wort;
  }

  public String getWortart() {
    return wortart;
  }

  public void setWortart(String wortart) {
    if (wortart.equals("Substantiv")){
      iszutat = Rezept.isZutat(wort);
    }
    this.wortart = wortart;
  }

  public int getPhrase() {
    return phrase;
  }

  public void setPhrase(int phrase) {
    this.phrase = phrase;
  }

  public String getUmstand() {
    return umstand;
  }

  public void setUmstand(String umstand) {
    this.umstand = umstand;
  }

  public int getTasknr() {
    return tasknr;
  }

  public void setTasknr(int tasknr) {
    this.tasknr = tasknr;
  }
```

```
64:
65:   public boolean getIsZutat() {
66:     return iszutat;
67:   }
68:
69:   public void setIsZutat(boolean zutat) {
70:     this.iszutat = zutat;
71:   }
72:
73:   public boolean getIsentity() {
74:     return isentity;
75:   }
76:
77:   public void setIsentity(boolean isentity) {
78:     this.isentity = isentity;
79:   }
80:
81:   // Ende Methoden
82: }
83:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Referenzsicherung der Tasks
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.util.ArrayList;
 9:
10: public class TaskMemory {
11:
12:   // Anfang Attribute
13:   private String Substantiv;
14:   private boolean isEntity;
15:
16:   private int Referenz;
17:   private int Ref_old;
18:   // Ende Attribute
19:
20:   // Anfang Methoden
21:
22:   public int getReferenz() {
23:     return Referenz;
24:   }
25:
26:   public void setReferenz(int Referenz) {
27:     this.Referenz = Referenz;
28:   }
29:
30:   public boolean getIsEntity() {
31:     return isEntity;
32:   }
33:
34:   public void setIsEntity(boolean isEntity) {
35:     this.isEntity = isEntity;
36:   }
37:
38:
39:   public String getSubstantiv() {
40:     return Substantiv;
41:   }
42:
43:   public void setSubstantiv(String Substantiv) {
44:     this.Substantiv = Substantiv;
45:   }
46:
47:   public int getRef_old() {
48:     return Ref_old;
49:   }
50:
51:   public void setRef_old(int Ref_old) {
52:     this.Ref_old = Ref_old;
53:   }
54:
55:   // Ende Methoden
56: }
57:
```

```
 1: import java.io.*;
 2:
 3: /**
 4:   *
 5:   * Klasse zur Aufnahme von TDL Informationen  unc zum Schreiben der
    TDL Datei
 6:   *
 7:   * @version 1.0 vom 25.02.2009
 8:   * @author Henning Budde
 9:   */
10: import java.util.ArrayList;
11:
12:
13: public class TDL {
14:     // Anfang Attribute
15:     private String Rezeptname;
16:     private ArrayList<String> argumente = new ArrayList<String>();
17:     private ArrayList<String> referenz = new ArrayList<String>();
18:     private ArrayList<String> taskliste = new ArrayList<String>();
19:     private ArrayList<String> entity = new ArrayList<String>();
20:     private ArrayList<ZutBez> zutbez = new ArrayList<ZutBez>();
21:     private String[] entityarray = new String[100];
22:     private int Schrittnr;
23:     private String taskname;
24:     private PrintWriter pw;
25:
26:     // Ende Attribute
27:
28:     // Anfang Methoden
29:     public String getRezeptname() {
30:         return Rezeptname;
31:     }
32:
33:     public void setRezeptname(String Rezeptname) {
34:         this.Rezeptname = Rezeptname;
35:     }
36:
37:     public void setArgumente(ArrayList<String> argumente) {
38:         this.argumente = argumente;
39:     }
40:
41:     public int getSchrittnr() {
42:         return Schrittnr;
43:     }
44:
45:     public void setSchrittnr(int Schrittnr) {
46:         this.Schrittnr = Schrittnr;
47:     }
48:
49:     /**
50:      * Methode zum Anlegen der TDL Datei
51:      *
52:      */
53:     public void writeTDL() {
54:         try {
55:             pw = new PrintWriter("..\\TDL\\" + this.Rezeptname +
    ".tdl", "UTF8");
56:         } catch (IOException e) {
57:             new ErrorDialog(null, "WriteTDL IO", true,
    e.getMessage());
58:         }
59:     }
60:
```

```
 61:     /**
 62:     * Methode schreibt Taskname
 63:     *
 64:     */
 65:     public void writeTaskName() {
 66:         pw.print("TASK ");
 67:         //    System.out.println(taskname);
 68:         pw.println(this.taskname + "()");
 69:     }
 70:
 71:     /**
 72:       * Methode schreibt Variabeles Block der TDL
 73:       *
 74:       */
 75:     public void writeVariables() {
 76:         pw.println("#VARIABLES");
 77:         pw.println("\t int tasknr = " + this.Schrittnr);
 78:         pw.println("\t int result");
 79:     }
 80:
 81:     /**
 82:       * Methode schreibt Entitaeten
 83:       *
 84:       */
 85:     public void writeEntity() {
 86:         for (String e : this.entity) {
 87:             if (entity.size() != 0) {
 88:                 pw.println("\t entity " + e + " \'\'");
 89:             }
 90:
 91:             ZutBez a = new ZutBez();
 92:             //      System.out.println(e);
 93:             a.setBez(e);
 94:             a.setRef(this.Schrittnr);
 95:
 96:             if (Debug.tdl) {
 97:                 System.out.println("Bezeichnung: " + a.getBez() +
 98:                     "    Refnr: " + a.getRef());
 99:             }
100:
101:             if (entityarray[Schrittnr] != null) {
102:                 entityarray[Schrittnr] = entityarray[Schrittnr] + "/"
    + e;
103:             } else {
104:                 entityarray[Schrittnr] = e;
105:             }
106:
107:             zutbez.add(a);
108:         }
109:
110:         boolean was_null = false;
111:
112:         for (String ref2 : referenz) {
113:             try {
114:                 int refnr = Integer.parseInt(ref2);
115:                 String refbez = getRefBez(refnr);
116:
117:                 if (entityarray[Schrittnr] == null) {
118:                     System.out.println(refnr);
119:                     System.out.println(entityarray[refnr]);
120:                     entityarray[Schrittnr] = entityarray[refnr];
121:                     was_null = true;
122:                 }
```

```
123:
124:                 if (Debug.tdl) {
125:                     System.out.println(entityarray[Schrittnr]);
126:                 }
127:
128:                 pw.println("\t entity " + entityarray[refnr] + "_" +
    refnr +
129:                     " \'\'");
130:
131:                 if (!was_null) {
132:                     entityarray[Schrittnr] += ("/" +
    entityarray[refnr]);
133:                 }
134:
135:                 was_null = false;
136:
137:                 this.setRef(refbez, this.Schrittnr);
138:             } catch (NumberFormatException e) {
139:                 System.out.println(e.getMessage());
140:             }
141:         }
142:     }
143:
144:     /**
145:     * Methode schreibt Body-Block eines Tasks
146:     *
147:     */
148:     public void writeBody() {
149:         pw.println("#BODY");
150:         pw.println("  DO(");
151:
152:         String hilf = "\t result = " + this.taskname + " (";
153:
154:         for (String arg : argumente) {
155:             hilf += (", " + arg);
156:         }
157:
158:         for (String ref : referenz) {
159:             hilf += (", " + ref);
160:         }
161:
162:         hilf = hilf.replaceFirst(", ", "");
163:         hilf += ")";
164:         pw.println("\t " + hilf);
165:         pw.println("  )");
166:         pw.println("#END");
167:         pw.println("");
168:     }
169:
170:     /**
171:       * Methode schreibt Ende der TDL Datei
172:       *
173:       */
174:     public void writeEndTask() {
175:         pw.println("TASK Rezept_" + this.Rezeptname);
176:         pw.println("#BODY");
177:         pw.println("#DO(");
178:
179:         for (String t : this.taskliste) {
180:             pw.println("\t task " + t + "()");
181:         }
182:
183:         pw.println(")");
```

```
184:         pw.println("#END");
185:     }
186:
187:     public String getTaskname() {
188:         return taskname;
189:     }
190:
191:     public void setTaskname(String taskname) {
192:         this.taskname = taskname;
193:     }
194:
195:     /**
196:     * Methode resetet Listen und Zähler der TDL Datei
197:     *
198:     */
199:     public void reset() {
200:         argumente = new ArrayList<String>();
201:         referenz = new ArrayList<String>();
202:         Schrittnr = 0;
203:         taskname = "";
204:         this.entity = new ArrayList<String>();
205:     }
206:
207:     public ArrayList<String> getArgumente() {
208:         return argumente;
209:     }
210:
211:     public ArrayList<String> getReferenz() {
212:         return referenz;
213:     }
214:
215:     public void setReferenz(ArrayList<String> referenz) {
216:         this.referenz = referenz;
217:     }
218:
219:     public void close() {
220:         zutbez = new ArrayList<ZutBez>();
221:         pw.close();
222:     }
223:
224:     public ArrayList<String> getTaskliste() {
225:         return taskliste;
226:     }
227:
228:     public void setTaskliste(ArrayList<String> taskliste) {
229:         this.taskliste = taskliste;
230:     }
231:
232:     public ArrayList<String> getEntity() {
233:         return entity;
234:     }
235:
236:     public void setEntity(ArrayList<String> entity) {
237:         this.entity = entity;
238:     }
239:
240:     /**
241:       * Methode aktualisiert und setzt Referenz
242:       *
243:       * @param bez Bezeichnung einer Zutat
244:       * @param ref neue Referenznummer
245:       */
246:     public void setRef(String bez, int ref) {
```

```
247:         for (ZutBez z : zutbez) {
248:             if (ref == 7) {
249:                 System.out.println("Zutatenbezeichnung : " +
    z.getBez());
250:                 System.out.println("Bez: " + bez);
251:             }
252:
253:             if (z.getBez().equals(bez) ||
254:                     ((bez != null) && z.getBez().contains(bez))) {
255:                 z.setRef(ref);
256:             }
257:         }
258:     }
259:
260:     /**
261:      * Methode gibt Bezeichner einer Referenz zurück
262:      *
263:      * @param ref eine Referenznummer
264:      * @return Bezeichner der übergebenen Referenz
265:      */
266:     public String getRefBez(int ref) {
267:         for (ZutBez z : zutbez) {
268:             if (z.getRef() == ref) {
269:                 return z.getBez();
270:             }
271:         }
272:
273:         return null;
274:     }
275:
276:     // Ende Methoden
277: }
278:
```

```
 1: /**
 2:   * XML Content Handler für TDL
 3:   *
 4:   * @version 1.0 vom 25.02.2009
 5:   * @author Henning Budde
 6:   */
 7: import org.xml.sax.*;
 8: import org.xml.sax.helpers.*;
 9: import java.io.*;
10: import javax.xml.parsers.*;
11:
12:
13: public class TDLContentHandler extends MyContentHandler {
14:
15:   private Parameter params = new Parameter();
16:   boolean set_entity;
17:   private TDL tdl = new TDL();
18:     public TDLContentHandler(String filename) {
19:         super();
20:     }
21:     // Anfang Attribute
22:     // Ende Attribute
23:
24:     // Anfang Methoden
25:
26:     /**
27:       * {@inheritDoc}
28:       *
29:       */
30:     public void startElement(String uri, String localName, String
    qName,
31:         Attributes attributes) throws SAXException {
32:         int i;
33:         String gVl = null;
34:         String gTy = null;
35:         String gNam = null;
36:         String gVl4;
37:         AttributesImpl a1 = new AttributesImpl(attributes);
38:         int l1 = a1.getLength();
39:         if (Debug.tdl) System.out.println(l1);
40:          for (i = 0; i < l1; i++) {
41:             gVl = a1.getValue(i);
42:             gTy = a1.getType(i);
43:             gNam = a1.getQName(i);
44:
45:         }
46:
47:         if (qName.compareTo("SCHRITT") == 0) {
48:           if (a1.getQName(0).compareTo("SID") == 0) {
49:             if (Debug.tdl) System.out.println(gVl);
50:             tdl.setSchrittnr(Integer.parseInt(gVl));
51:           }
52:         }
53:
54:         if (qName.compareTo("ARGUMENT") == 0){
55:           if (a1.getQName(1) != null){
56:              System.out.println(a1.getQName(1));
57:              if (a1.getQName(1).compareTo("ZUT") ==0 ){
58:                 set_entity = true;
59:               }
60:           } else set_entity=false;
61:
62:         }
```

```
 63:      }
 64:
 65:     /**
 66:       * {@inheritDoc}
 67:       *
 68:       */
 69:     public void endElement(String uri, String localName, String qName)
 70:         throws SAXException {
 71:
 72:         System.out.println(aktwert);
 73:         if (qName.compareTo("SCHRITT") == 0) {
 74:           tdl.writeTaskName();
 75:           tdl.writeVariables();
 76:           tdl.writeEntity();
 77:           tdl.writeBody();
 78:           tdl.reset();
 79:
 80:         }
 81:
 82:         if (qName.compareTo("TXT") == 0) {
 83:             aktwert = "";
 84:         }
 85:
 86:         if (qName.compareTo("TASK") == 0) {
 87:            tdl.setTaskname(aktwert);
 88:
 89:            tdl.getTaskliste().add(new String(aktwert));
 90:         }
 91:
 92:         if (qName.compareTo("REFERENZ") == 0) {
 93:           tdl.getReferenz().add(new String(aktwert));
 94:         }
 95:
 96:         if (qName.compareTo("ARGUMENT") == 0) {
 97:           tdl.getArgumente().add(new String(aktwert));
 98:
 99:           if (set_entity){
100:               String [] neuaktwert = aktwert.split("_");
101:               if (neuaktwert.length == 2){
102:                 tdl.getEntity().add(new String(neuaktwert[1]));
103:                 System.out.println("NEUAKTWERT: " +neuaktwert[1]);
104:               } else tdl.getEntity().add(new String(aktwert));
105:
106:           }
107:
108:         }
109:
110:         if (qName.compareTo("REZEPTNAME") == 0) {
111:           tdl.setRezeptname(aktwert);
112:           tdl.writeTDL();
113:         }
114:         if (qName.compareTo("REZEPT") == 0) {
115:           tdl.writeEndTask();
116:           tdl.reset();
117:           tdl.close();
118:         }
119:
120:
121:     }
122:
123:   public Parameter getParams() {
124:     return params;
125:   }
```

```
126:
127:   public void setParams(Parameter params) {
128:     this.params = params;
129:   }
130:   public String change_umlaut(String umlaute){
131:     umlaute = umlaute.replace("ä","ae");
132:     umlaute = umlaute.replace("ü","ue");
133:     umlaute = umlaute.replace("ß","ss");
134:     umlaute = umlaute.replace("ö","oe");
135:     umlaute = umlaute.replace("Ä","Ae");
136:     umlaute = umlaute.replace("Ü","Ue");
137:     umlaute = umlaute.replace("Ö","Oe");
138:     umlaute = umlaute.replace(" ","_");
139:     return umlaute;
140:   }
141:
142:     // Ende Methoden
143: }
144:
```

```
 1: /**
 2:   *
 3:   * GUI Klasse zum Einpflegen von Verben
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.awt.*;
 9: import java.awt.event.*;
10:
11: import java.sql.*;
12:
13: import javax.swing.*;
14: import javax.swing.event.*;
15:
16:
17: public class VerbDialog extends Dialog {
18:     // Anfang Attribute
19:     private JTextField verbinf = new JTextField();
20:     private JLabel verbinflabel = new JLabel();
21:     private JLabel verbschwach = new JLabel();
22:     private JCheckBox checkBox1 = new JCheckBox();
23:     private JLabel label2 = new JLabel();
24:     private JButton okButton = new JButton();
25:
26:     // Ende Attribute
27:     public VerbDialog(Frame owner, String title, boolean modal,
    String verb) {
28:         // Dialog-Initialisierung
29:         super(owner, title, modal);
30:         addWindowListener(new WindowAdapter() {
31:                 public void windowClosing(WindowEvent evt) {
32:                     dispose();
33:                     setVisible(false);
34:                 }
35:             });
36:
37:         int frameWidth = 383;
38:         int frameHeight = 203;
39:         setSize(frameWidth, frameHeight);
40:
41:         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
42:         int x = (d.width - getSize().width) / 2;
43:         int y = (d.height - getSize().height) / 2;
44:         setLocation(x, y);
45:
46:         Panel cp = new Panel(null);
47:         add(cp);
48:         // Anfang Komponenten
49:         verbinf.setBounds(120, 72, 217, 24);
50:         verbinf.setText(verb);
51:         cp.add(verbinf);
52:         verbinflabel.setBounds(8, 72, 93, 24);
53:         verbinflabel.setText("Verb im Infintiv");
54:         verbinflabel.setFont(new Font("MS Sans Serif", Font.PLAIN,
    13));
55:         cp.add(verbinflabel);
56:         verbschwach.setBounds(8, 40, 106, 16);
57:         verbschwach.setText("schwaches Verb?");
58:         verbschwach.setFont(new Font("MS Sans Serif", Font.PLAIN,
    13));
59:         cp.add(verbschwach);
60:         checkBox1.setBounds(144, 40, 17, 17);
```

```
 61:         checkBox1.setText("checkBox1");
 62:         cp.add(checkBox1);
 63:         label2.setBounds(8, 8, 100, 24);
 64:         label2.setText(verb);
 65:         label2.setFont(new Font("MS Sans Serif", Font.BOLD, 13));
 66:         cp.add(label2);
 67:         okButton.setBounds(120, 120, 129, 33);
 68:         okButton.setText("OK");
 69:         okButton.addActionListener(new ActionListener() {
 70:                 public void actionPerformed(ActionEvent evt) {
 71:                     okButton_ActionPerformed(evt);
 72:                 }
 73:             });
 74:         cp.add(okButton);
 75:         // Ende Komponenten
 76:         setResizable(false);
 77:         setVisible(true);
 78:     }
 79:
 80:     // Anfang Methoden
 81:     public void okButton_ActionPerformed(ActionEvent evt) {
 82:         if (this.checkBox1.isSelected()) {
 83:             Connection con = DBzugriff.connect();
 84:
 85:             String SQL = "INSERT INTO verb_schwach(infinitiv,
    benutzer, eingetragen) " +
 86:                 "VALUES('" + this.verbinf.getText() +
 87:                 "','Agent Cook', sysdate)";
 88:
 89:             try {
 90:                 Statement stmt = con.createStatement();
 91:                 stmt.executeUpdate(SQL);
 92:                 con.close();
 93:             } catch (SQLException e) {
 94:                 new ErrorDialog(null, "Verb Fehler", true,
    e.getMessage());
 95:             } finally {
 96:             }
 97:         }
 98:
 99:         this.dispose();
100:         this.setVisible(false);
101:     }
102:
103:     // Ende Methoden
104: }
105:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Erzeugung eines XML Rezepts ohne Arbeitsschritte
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.io.*;
 9:
10: import java.util.ArrayList;
11:
12:
13: public class WriteXML {
14:     private static boolean singelton = false;
15:     private static WriteXML xml;
16:     private Rezept rezept;
17:     private ArrayList<Integer> Ref1 = new ArrayList<Integer>();
18:     private ArrayList<TaskMemory> Taskmem = new
    ArrayList<TaskMemory>();
19:     ArrayList<String> arguments = new ArrayList<String>();
20:     private String zahlenargument = "";
21:
22:     // Anfang Attribute
23:     private String filename;
24:     private PrintWriter pw;
25:     boolean preap = false;
26:     boolean noArg = false;
27:     boolean konjunktion = false;
28:     boolean zahlwort = false;
29:     String argument = "";
30:     String taskname = "";
31:     String adjektiv = "";
32:     String praeposition = "";
33:     String adverb = "";
34:     boolean set_adverb = false;
35:     boolean istzutat = false;
36:
37:     // Ende Attribute
38:
39:     // Anfang Methoden
40:     private WriteXML() {
41:         rezept = Rezept.getInstance();
42:         System.setProperty("file.encoding", "UTF8");
43:
44:         try {
45:             pw = new PrintWriter("..\\Rezepte\\" +
    rezept.getRezeptname() +
46:                     ".xml", "UTF8");
47:         } catch (IOException e) {
48:             new ErrorDialog(null, "WriteXML IO", true,
    e.getMessage());
49:         }
50:     }
51:
52:     // zur Singelton Erzeugung
53:     public static WriteXML getInstance() {
54:         if (singelton) {
55:             return xml;
56:         } else {
57:             xml = new WriteXML();
58:             singelton = true;
59:
60:             return xml;
```

```
 61:         }
 62:     }
 63:
 64:     /**
 65:       * Methode schreibt XML Header
 66:       *
 67:       */
 68:     public void xml_header() {
 69:         pw.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
 70:         pw.println("<!DOCTYPE REZEPT SYSTEM \"Rezept.dtd\">");
 71:     }
 72:
 73:     /**
 74:     * Methode schreibt XML Rezept
 75:     *
 76:     * @param secondRun markiert zweite Runde
 77:     */
 78:     public void write_xml_rezept(boolean secondRun) {
 79:         pw.println("<REZEPT>");
 80:         pw.print("\t<REZEPTNAME>");
 81:         pw.print(rezept.getRezeptname());
 82:         pw.println("</REZEPTNAME>");
 83:         pw.println("\t<ZUTATENLISTE>");
 84:
 85:         for (Zutat z : rezept.getZutaten()) {
 86:             pw.println("\t\t<ZUTAT>");
 87:             pw.println("\t\t\t<BEZEICHNUNG>" + z.getBez() +
    "</BEZEICHNUNG>");
 88:             pw.println("\t\t\t<MENGE>" + z.getMenge() + "</MENGE>");
 89:             pw.println("\t\t\t<EINHEIT>" + z.getEinheit() +
    "</EINHEIT>");
 90:             pw.println("\t\t</ZUTAT>");
 91:         }
 92:
 93:         pw.println("\t</ZUTATENLISTE>");
 94:         pw.println("\t<ARBEITSSCHRITTE>");
 95:
 96:         pw.println("\t</ARBEITSSCHRITTE>");
 97:         pw.println("</REZEPT>");
 98:         pw.close();
 99:     }
100:
101:
102:
103:     /**
104:     *
105:     * @deprecated ersetzt von WriteXML2 Methode resetLists
106:     */
107:     private void resetLists() {
108:         arguments = new ArrayList<String>();
109:         Ref1 = new ArrayList<Integer>();
110:
111:         zahlwort = false;
112:         argument = "";
113:         this.taskname = "";
114:         this.adjektiv = "";
115:         this.zahlenargument = "";
116:         this.istzutat = false;
117:         this.preap = false;
118:         this.adverb = "";
119:
120:         // ="";
121:     }
```

```
122:
123:     /**
124:      *
125:      * @deprecated ersetzt von WriteXML2 Methode reset
126:      */
127:     public WriteXML reset() {
128:         pw.close();
129:         resetLists();
130:         Taskmem = new ArrayList<TaskMemory>();
131:         Ref1 = new ArrayList<Integer>();
132:         this.singelton = false;
133:         this.xml = new WriteXML();
134:
135:         return this.xml;
136:     }
137:
138:     public void closeFile() {
139:         //      try {
140:         pw.close();
141:     }
142:
143:     // Ende Methoden
144: }
145:
```

```
 1: /**
 2:   *
 3:   * Diese Klasse erzeugt XML Rezepte mit Arbeitsschritten
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: import java.io.*;
 9:
10: import java.util.ArrayList;
11:
12:
13: public class WriteXML2 {
14:     private static boolean singelton = false;
15:     private static WriteXML2 xml;
16:     private Rezept rezept;
17:     private ArrayList<Integer> referenzliste = new
    ArrayList<Integer>();
18:     private ArrayList<TaskMemory> Taskmem = new
    ArrayList<TaskMemory>();
19:     ArrayList<String> arguments = new ArrayList<String>();
20:     private String zahlenargument = "";
21:
22:     // Anfang Attribute
23:     private String filename;
24:     private PrintWriter pw;
25:     boolean preap = false;
26:     boolean noArg = false;
27:     boolean konjunktion = false;
28:     boolean konjunktion_merk = false;
29:     boolean zahlwort = false;
30:     String argument = "";
31:     String taskname = "";
32:     String adjektiv = "";
33:     String praeposition = "";
34:     String adverb = "";
35:     String sub2 = "";
36:     boolean set_adverb = false;
37:     int satzanfang = 0;
38:     boolean istzutat = false;
39:
40:     // Ende Attribute
41:
42:     // Anfang Methoden
43:     private WriteXML2() {
44:         rezept = Rezept.getInstance();
45:         System.setProperty("file.encoding", "UTF8");
46:
47:         try {
48:             pw = new PrintWriter("..\\Rezepte\\" +
    rezept.getRezeptname() +
49:                     ".xml", "UTF8");
50:         } catch (IOException e) {
51:             new ErrorDialog(null, "WriteXML IO", true,
    e.getMessage());
52:         }
53:     }
54:
55:     // zur Singelton Erzeugung
56:     public static WriteXML2 getInstance() {
57:         if (singelton) {
58:             return xml;
59:         } else {
```

```
 60:             xml = new WriteXML2();
 61:             singelton = true;
 62:
 63:             return xml;
 64:         }
 65:     }
 66:
 67:     /**
 68:       * Methode schreibt XML Header
 69:       *
 70:       */
 71:     public void xml_header() {
 72:         pw.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
 73:         pw.println("<!DOCTYPE REZEPT SYSTEM \"Rezept.dtd\">");
 74:     }
 75:
 76:     /**
 77:     * Methode schreibt XML Rezept
 78:     *
 79:     * @param secondRun markiert zweite Runde
 80:     */
 81:     public void write_xml_rezept(boolean secondRun) {
 82:         pw.println("<REZEPT>");
 83:         pw.print("\t<REZEPTNAME>");
 84:         pw.print(rezept.getRezeptname());
 85:         pw.println("</REZEPTNAME>");
 86:         pw.println("\t<ZUTATENLISTE>");
 87:
 88:         for (Zutat z : rezept.getZutaten()) {
 89:             pw.println("\t\t<ZUTAT>");
 90:             pw.println("\t\t\t<BEZEICHNUNG>" + z.getBez() +
    "</BEZEICHNUNG>");
 91:             pw.println("\t\t\t<MENGE>" + z.getMenge() + "</MENGE>");
 92:             pw.println("\t\t\t<EINHEIT>" + z.getEinheit() +
    "</EINHEIT>");
 93:             pw.println("\t\t</ZUTAT>");
 94:         }
 95:
 96:         pw.println("\t</ZUTATENLISTE>");
 97:         pw.println("\t<ARBEITSSCHRITTE>");
 98:
 99:         if (secondRun) {
100:             generateTasks();
101:         }
102:
103:         pw.println("\t</ARBEITSSCHRITTE>");
104:         pw.println("</REZEPT>");
105:         pw.close();
106:     }
107:
108:     /**
109:     * Methode erzeugt Tasks
110:     *
111:     */
112:     public void generateTasks() {
113:         int aktuelle_tasknr = 1; // Variable für die aktuelle
    Tasknummer
114:         int zu_bearbeitende_tasknr = 0;
115:         String substantiv = "";
116:         String verb = "";
117:         String adjektiv = "";
118:         String praeposition = "";
119:         String hilfsverb = "";
```

```
120:         String adverb = "";
121:         String zahl = "";
122:         String taskname = "";
123:
124:         String task_txt = ""; // grundlegender Satz zur Beschreibung
     des Tasks
125:
126:         int phrase = 0;
127:         int last_phrase = 0;
128:         int sid = 1;
129:
130:         boolean konjunktion = false;
131:         boolean satzzeichen = false;
132:         boolean b_praeposition = false;
133:         task_txt = "";
134:
135:         for (int j = 0; j < rezept.getTasks().size(); j++) { // alle
     Tasks durchlaufen
136:
137:             Task task = rezept.getTasks().get(j); // einen Task holen
     und in Task Objekt schreiben
138:             aktuelle_tasknr = task.getTasknr(); // aktuelle Tasknr
     holen
139:             satzanfang++;
140:
141:             if (Debug.xml) {
142:                 System.out.println("Aktuelle_tasknr: " +
     aktuelle_tasknr +
143:                     "     zu_bearbeitende_tasknr: " +
     zu_bearbeitende_tasknr);
144:                 System.out.println(task.getWort());
145:             }
146:
147:             if (zu_bearbeitende_tasknr == aktuelle_tasknr) { // nur
     zusammengehörende Task (selber Satz) bearbeiten
148:                 task_txt += (" " + task.getWort());
149:
150:                 task_txt = task_txt.trim(); // Task Text aktuellen
     Wert anhängen
151:                 phrase = task.getPhrase(); // aktuelle Phrasen Nr
     holen
152:
153:
     /*****************************************************/
154:
155:                 /*                          VERBEN
     */
156:
157:
     /*****************************************************/
158:                 if (task.getWortart().equals("Verb")) { // Wenn
     aktuelles Wort Verb dann in zugehöriger Variable speichern
159:                     verb = task.getWort();
160:                 }
161:
162:
     /*****************************************************/
163:
164:                 /*                  HILFSVERBEN
     */
165:
166:
     /*****************************************************/
```

```
167:                 if (task.getWortart().equals("Hilfsverb")) {
168:                     hilfsverb = task.getWort();
169:                 }
170:
171:
    /*****************************************************/
172:
173:                 /*                SUBSTANTIVE
    */
174:
175:
    /*****************************************************/
176:                 if (task.getWortart().equals("Substantiv")) {
177:                     substantiv = task.getWort();
178:
179:                     if (task.getIsZutat() || task.getIsentity()) { //
    Ist Substantiv eine Zutat oder Entität ?
180:
181:                         TaskMemory temp_tm =
    this.inTaskmem(substantiv);                                          // Taskmem
    eingetragen
182:
       // und zurückgeben
183:
184:                         if (temp_tm == null) { // wenn Substantiv
    noch nicht eingetragen
185:
186:                             TaskMemory tm = new TaskMemory();
187:                             tm.setSubstantiv(new String(substantiv));
    // TaskMemory mit akteullem Wort belegen
188:                             tm.setReferenz(sid); // aktuelle Referenz
    setzten
189:                             tm.setIsEntity(task.getIsentity()); //
    setzten ob Wort Entität ist
190:                             this.Taskmem.add(tm); // in TaskMemory
    Liste ablegen
191:                             arguments.add(substantiv); // Substantiv
    in Argumentenliste speichern
192:                         }
193:                         // wenn Substantin schon eingetragen
    Referenzen aktualisieren
194:                         else {
195:
    temp_tm.setRef_old(temp_tm.getReferenz());                          // alte Refe
    speichern
196:                             temp_tm.setReferenz(sid); // neue
    Referenz eintragen
197:                             referenzliste.add(temp_tm.getRef_old());
    // alte Referenz in Referenzliste speichern
198:                         }
199:                     } else { // Substantiv ist weder Zutat noch
    Entität
200:                         sub2 += (" " + task.getWort());
201:                         sub2 = sub2.trim(); // überflüssige
    Leerzeichen entfernen
202:
203:                         TaskMemory tm2 = inTaskmem(sub2); //
    Morphemanalyse des Substantivs, wenn nicht vorhanden Rückgabewert null
204:
205:                         if (tm2 != null) { // wenn Substantiv
    vorhanden
206:                             referenzliste.add(tm2.getReferenz()); //
    Substantiv in Referenzlist eintragen
```

```
207:                                                                    //
                            System.out.println("OLD REF: " +
    tm2.getReferenz());
208:
209:                                tm2.setReferenz(sid); // neue Referenz
    setzten für Substantiv
210:                            } else { // wenn Substantiv noch nicht
    vorhanden
211:
212:                                String sub3 = praeposition + " " +
    adjektiv; // Präpositon und Adjektiv, wenn vorhanden anfügen
213:                                sub3 = sub3.trim(); // Leerzeichen vor
    und nach sub3 entfernen
214:
215:                                if (sub2.equals("Minuten")) { // wenn
    Substantiv Minuten
216:                                    sub2 = zahl + " " + sub2; // Zahlen
    zu String hinzufügen
217:                                    arguments.add(new String(sub2)); //
    Minuten mit Zahlen Argumenten hinzfügen
218:                                    praeposition = "";
219:                                    zahl = "";
220:                                    adjektiv = "";
221:                                    sub2 = "";
222:
223:                                    continue;
224:                                }
225:
226:                                if (sub2.equals("Watt")) {
227:                                    sub2 = zahl + " " + sub2;
228:                                    arguments.add(new String(sub2));
229:                                    praeposition = "";
230:                                    zahl = "";
231:                                    adjektiv = "";
232:                                    System.out.println("in Watt");
233:
234:                                    continue;
235:                                }
236:
237:                                sub3 += (" " + sub2);
238:                                sub3 = sub3.trim();
239:                                arguments.add(new String(sub3));
240:                                praeposition = "";
241:                                zahl = "";
242:                                adjektiv = "";
243:                            }
244:
245:                            sub2 = "";
246:                        }
247:                    }
248:
249:
    /*****************************************************/
250:
251:                    /*                ADJEKTIVE
     */
252:
253:
    /*****************************************************/
254:                    if (task.getWortart().equals("Adjektiv")) {
255:                        adjektiv = task.getWort();
256:                    }
257:
```

```
258:
    /*******************************************************/
259:
260:                /*                ADVERBEN
     */
261:
262:
    /*******************************************************/
263:                if (task.getWortart().equals("Adverb")) {
264:                    adverb = task.getWort();
265:                }
266:
267:
    /*******************************************************/
268:
269:                /*                ZAHLEN
     */
270:
271:
    /*******************************************************/
272:                if (task.getWortart().equals("Zahl")) {
273:                    zahl = task.getWort();
274:                }
275:
276:
    /*******************************************************/
277:
278:                /*                PRAEPOSITIONEN
     */
279:
280:
    /*******************************************************/
281:                if (task.getWortart().equals("Praeposition")) {
282:                    praeposition = task.getWort();
283:                    b_praeposition = true;
284:
285:                    //                System.out.println("IN
    Praposition : " +konjunktion_merk);
286:                    if (konjunktion_merk) {
287:                        updateReferenz(sid - 1, sid);
288:                        referenzliste.add(sid - 1);
289:                    }
290:
291:                    if (rezept.getTasks().get(j + 1).getWortart()
292:                                .equals("Substantiv") ||
293:                            rezept.getTasks().get(j + 1).getWortart()
294:                                    .equals("Artikel")) {
295:                        //                    sub2 += praeposition;
296:                    }
297:
298:                    System.out.println("satzanfang:" + satzanfang);
299:
300:                    if (satzanfang == 1) {
301:                        System.out.println("Satzanfang Preap");
302:                        referenzliste.add(sid - 1);
303:                    }
304:
305:                    if (task.getWort().equals("darüber")) { // wenn
    Wort darueber bezug auf letzten Task
306:                        referenzliste.add(sid - 1);
307:                    }
308:                }
309:
```

```
310:
    /*******************************************************/
311:
312:                /*                SATZZEICHEN
     */
313:
314:
    /*******************************************************/
315:                if (task.getWortart().equals("Satzzeichen")) {
316:                    satzzeichen = true;
317:
318:                    if (task.getWort().equals(".")) {
319:                        satzanfang = 0;
320:                    }
321:
322:                    if (task.getWort().equals(",")) {
323:                        if (previousnext(j) == 2) {
324:                            satzzeichen = false;
325:                        }
326:                    }
327:                }
328:
329:
    /*******************************************************/
330:
331:                /*                KONJUNKTIONEN
      */
332:
333:
    /*******************************************************/
334:                if (task.getWortart().equals("Konjunktion")) {
335:                    konjunktion = true;
336:
337:                    if (previousnext(j) == 2) {
338:                        konjunktion = false;
339:                    }
340:                }
341:
342:                taskname = verb + " " + hilfsverb;
343:                taskname = taskname.trim(); // überflüssige
    Leerzeichen entfernen
344:
345:                if (satzzeichen || konjunktion) {
346:                    writeTask(task_txt, sid, taskname, arguments,
    referenzliste);
347:                    task_txt = "";
348:                    substantiv = "";
349:                    verb = "";
350:                    adjektiv = "";
351:                    praeposition = "";
352:                    hilfsverb = "";
353:                    adverb = "";
354:                    zahl = "";
355:                    taskname = "";
356:                    sub2 = "";
357:                    satzanfang = 0;
358:                    referenzliste = new ArrayList<Integer>();
359:                    arguments = new ArrayList<String>();
360:
361:                    satzzeichen = false;
362:
363:                    konjunktion_merk = konjunktion; // alten
    Kinojunktionsstatus zwischenspeichern
```

```
364:                    konjunktion = false;
365:                    //
    System.out.println("KOnjunktion_merk: "+ konjunktion_merk);
366:                    //
    System.out.println("KOnjunktion: "+ konjunktion);
367:                    last_phrase = phrase; // aktuelle Phrasennr
    speichern, um in der nächsten Runde zugriff darauf zu erhalten
368:                    sid++;
369:                }
370:            } else {
371:                zu_bearbeitende_tasknr++;
372:                j--;
373:                satzanfang = 0;
374:            }
375:        }
376:    }
377:
378:    /**
379:      *
380:      * Teste ob die Konjunktion zwei Substantive (also Aufzählung)
    verbindet oder
381:      * ob unterschiedliche Tasks durch Konjunktion verbunden wurden
382:      * BSP: 1.  Speck und Zwiebeln würzen.  | 2. Käse reiben und
    Speck würfeln.
383:      * 1. BSP: return 2 | 2. BSP: return -1;
384:      *
385:      * @param j Aktuelle Tasknr
386:      */
387:    private int previousnext(int j) {
388:        boolean previous = rezept.getTasks().get(j - 1).getWortart()
389:                                .equals("Substantiv");
390:        boolean next = rezept.getTasks().get(j + 1).getWortart()
391:                            .equals("Substantiv");
392:
393:        if (!next) { // wenn nächstes Wort kein Substantiv, dann
    überprüfen ob übernächstes Wort Substantiv
394:            next = (rezept.getTasks().get(j +
    1).getWortart().equals("Artikel") &&
395:                rezept.getTasks().get(j +
    2).getWortart().equals("Substantiv"));
396:        }
397:
398:        if (previous && next) {
399:            return 2;
400:        }
401:
402:        previous = rezept.getTasks().get(j -
    1).getWortart().equals("Verb");
403:        next = rezept.getTasks().get(j +
    1).getWortart().equals("Verb");
404:
405:        if (previous && next) {
406:            return 3;
407:        } else {
408:            return -1;
409:        }
410:    }
411:
412:    /**
413:    * Schreibt Task in eine Datei
414:    *
415:    * @param TaskTXT Satz auf den dieser Task beruht
416:    * @param TaskID aktuelle Tasknummer
```

```
417:     * @param Task Taskbezeichner
418:     * @param arguments Argumentenliste zum aktuellen Task
419:     * @param referenzliste Liste mit Referenzen zum aktuellen Task
420:     */
421:     private void writeTask(String TaskTXT, int TaskID, String Task,
422:         ArrayList<String> arguments, ArrayList<Integer>
    referenzliste) {
423:         boolean noArgument = true;
424:         pw.println("\t\t<SCHRITT SID=\"" + TaskID + "\">");
425:         pw.println("\t\t\t<TXT>" + TaskTXT + "</TXT>");
426:         pw.println("\t\t\t<TASK>" + Task + "</TASK>");
427:         System.out.println("\t\t<SCHRITT SID=\"" + TaskID + "\">");
428:         System.out.println("\t\t\t<TXT>" + TaskTXT + "</TXT>");
429:         System.out.println("\t\t\t<TASK>" + Task + "</TASK>");
430:
431:         int aid = 1;
432:         boolean found = false;
433:
434:         //         int counter =0;
435:         for (String a : arguments) {
436:             if (Blacklist.inBlacklist(a)) {
437:                 continue; // wenn Argument in Blackliste ist
438:             }
439:
440:             pw.print("\t\t\t<ARGUMENT AID=\"" + aid + "\"");
441:
442:             for (Zutat z : rezept.getZutaten()) {
443:                 if
    (z.getBez().toLowerCase().contains(a.toLowerCase())) {
444:                     found = true;
445:                     pw.print(" ZUT=\"TRUE\"");
446:                 }
447:                 //               else if
    (a.toLowerCase().contains(z.getBez().toLowerCase())){
448:                 //                pw.print(" ZUTAT =\"TRUE\"");
449:                 //                break;
450:                 //              }
451:                 else {
452:                     String[] hilf = a.split(" ");
453:
454:                     for (int i = 0; i < hilf.length; i++) {
455:                         if (z.getBez().contains(hilf[i])) {
456:                             if (Debug.xml) {
457:                                 System.out.println("Z.getBez: " +
    z.getBez());
458:                             }
459:
460:                             if (Debug.xml) {
461:                                 System.out.println("HILF[i]: " +
    hilf[i]);
462:                             }
463:
464:                             pw.print(" ZUT=\"TRUE\"");
465:                             found = true;
466:                         }
467:                     }
468:                 }
469:
470:                 if (found) {
471:                     found = false;
472:
473:                     break;
474:                 }
```

```
475:             }
476:
477:             for (TaskMemory b : Taskmem) {
478:                 if (b.getSubstantiv().contains(a) && b.getIsEntity())
    {
479:                     pw.print(" ZUT=\"FALSE\"");
480:                     System.out.println(" ZUT=\"FALSE\"");
481:                 }
482:             }
483:
484:             pw.println(">" + a.trim() + "</ARGUMENT>");
485:
486:             if (Debug.xml) {
487:                 System.out.println("\t\t\t<ARGUMENT AID=\"" + aid +
    "\">" +
488:                     a.trim() + "</ARGUMENT>");
489:             }
490:
491:             aid++;
492:             noArgument = false;
493:         }
494:
495:         ArrayList<Integer> doppelt = new ArrayList<Integer>();
496:
497:         for (int r : referenzliste) {
498:             if ((r == 0) || (r == -1)) {
499:                 continue; // Referenz auf nicht vorhandenen Task 0
    niemals setzten
500:             }
501:
502:             if (doppelt.contains(r)) {
503:                 continue;
504:             }
505:
506:             doppelt.add(r);
507:
508:             if (r == TaskID) {
509:                 continue;
510:             }
511:
512:             pw.println("\t\t\t<REFERENZ>" + r + "</REFERENZ>");
513:
514:             if (Debug.xml) {
515:                 System.out.println("\t\t\t<REFERENZ>" + r +
    "</REFERENZ>");
516:             }
517:
518:             updateReferenz(r, TaskID);
519:
520:             noArgument = false;
521:
522:             // pw.println("referenzliste Update");
523:         }
524:
525:         if (noArgument) {
526:             pw.println("\t\t\t<REFERENZ>" + (TaskID - 1) +
    "</REFERENZ>");
527:
528:             if (Debug.xml) {
529:                 System.out.println("\t\t\t<REFERENZ>" + (TaskID - 1) +
530:                     "</REFERENZ>");
531:             }
532:
```

```
533:             updateReferenz(TaskID - 1, TaskID);
534:
535:             // pw.println("NoArgumentsUpdate");
536:         }
537:
538:         pw.println("\t\t</SCHRITT>");
539:
540:         //        this.showTaskmem();
541:     }
542:
543:     /**
544:      * Methode aktualisiert Referenzen
545:      *
546:      * @param alteRef alte Referenz
547:      * @param neueRef neue Referenz
548:      */
549:     public void updateReferenz(int alteRef, int neueRef) {
550:         for (Zutat z : rezept.getZutaten()) {
551:             if (z.getReferenz() == alteRef) {
552:                 z.setReferenz(neueRef);
553:             }
554:         }
555:
556:         for (TaskMemory tm : Taskmem) {
557:             if (tm.getReferenz() == alteRef) {
558:                 tm.setReferenz(neueRef);
559:             }
560:         }
561:     }
562:
563:     private void resetLists() {
564:     }
565:
566:     public WriteXML2 reset() {
567:         return this.xml;
568:     }
569:
570:     public void closeFile() {
571:         pw.close();
572:     }
573:
574:     /**
575:      * Methode überprüft ob ein Substantiv schon in TaskMemory
    eingetragen ist
576:      *
577:      * @param sub zu überprüfendes Substantiv
578:      * @return wenn Substantiv in Liste, dann diesen TaskMemory
    zurückgeben
579:      */
580:     private TaskMemory inTaskmem(String sub) {
581:         for (TaskMemory tm : this.Taskmem) {
582:             System.out.println("tm.getsubstantiv : " +
    tm.getSubstantiv() +
583:                 "        SUB: " + sub);
584:
585:             if (tm.getSubstantiv().equals(sub) ||
586:                     tm.getSubstantiv().contains(sub.toLowerCase())) {
587:                 System.out.println("Substantiv gefunden : " + sub);
588:                 System.out.println(tm.getSubstantiv() + "      " +
589:                     tm.getReferenz() + " <--- return");
590:
591:                 return tm;
592:             }
```

```
593:         }
594:
595:         return null;
596:     }
597:
598:     /**
599:       * TaskMemory in Konsole anzeigen
600:       *
601:       */
602:     private void showTaskmem() {
603:         for (TaskMemory tm : this.Taskmem) {
604:             System.out.println("SUBSTANTIV: " + tm.getSubstantiv() +
605:                 " Referenz :" + tm.getReferenz());
606:             ;
607:         }
608:     }
609:
610:     // Ende Methoden
611: }
612:
```

```
 1: /**
 2:   *
 3:   * Klasse zur Sicherung von Zutateninformationen
 4:   *
 5:   * @version 1.0 vom 25.02.2009
 6:   * @author Henning Budde
 7:   */
 8: public class Zutat {
 9:     // Anfang Attribute
10:     private String bez;
11:     private double menge;
12:     private String einheit;
13:
14:   private int Referenz=-1;
15:     // Ende Attribute
16:     public Zutat(String bez, double menge, String einheit) {
17:     }
18:
19:     public Zutat() {
20:     }
21:
22:     // Anfang Methoden
23:     public double getMenge() {
24:         return menge;
25:     }
26:
27:     public void setMenge(double menge) {
28:         this.menge = menge;
29:     }
30:
31:     public String getBez() {
32:         return bez;
33:     }
34:
35:     public void setBez(String bez) {
36:         this.bez = bez;
37:     }
38:
39:     public String getEinheit() {
40:         return einheit;
41:     }
42:
43:     public void setEinheit(String einheit) {
44:         this.einheit = einheit;
45:     }
46:
47:   public int getReferenz() {
48:     return Referenz;
49:   }
50:
51:   public void setReferenz(int Referenz) {
52:     this.Referenz = Referenz;
53:   }
54:
55:     // Ende Methoden
56: }
57:
```

```java
/**
  *
  * Klasse zur Sicherung der Beziehungen von Zuatten und Bezeichnern
  *
  * @version 1.0 vom 25.02.2009
  * @author Henning Budde
  */

public class ZutBez {

  // Anfang Attribute
  private String Bez;
  private int Ref;
  // Ende Attribute

  // Anfang Methoden
  public String getBez() {
    return Bez;
  }

  public void setBez(String Bez) {
    this.Bez = Bez;
  }

  public int getRef() {
    return Ref;
  }

  public void setRef(int Ref) {
    this.Ref = Ref;
  }

  // Ende Methoden
}

```