Karlsruhe Institute of Technology

Semi-automatische Generierung von Aktiven Ontologien aus Datenbankschemata

Masterarbeit
von

Kevin Angele

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf Reussner
Betreuender Mitarbeiter:	Dipl.-Inform. Sebastian Weigelt
Zweiter betr. Mitarbeiter:	Dipl.-Inform. Martin Blersch

Bearbeitungszeit: 13.05.2018 – 12.11.2018

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 12.11.2018

. . . . . .

(**Kevin Angele**)

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt haben.
Insbesondere möchte ich mich bei meinen Betreuern Sebastian Weigelt und Martin Blersch bedanken. Ihre Unterstützung verhalf mir zu einem durchdachten und gut strukturierten Konzept zur Bewältigung der Aufgabe der Masterarbeit. Dank der Rahmenarchitektur EASIER von Martin Blersch konnte ich die erzeugten Aktiven Ontologien ausführen und evaluieren. Daneben gilt mein Dank Dr. Ioan Toma, er war mir eine große Hilfe bei der Zusammenstellung der Testdaten und der Agenten aus Dialogflow.
Nicht zuletzt gebührt all denjenigen Dank, die meine Arbeit in zahlreichen Stunden Korrektur gelesen und mich auf Schwächen in der Formulierungen hingewiesen haben. Abschließend möchte ich mich bei Herrn Prof. Dr. Walter Tichy und Herrn Prof. Dr. Ralf Reussner bedanken, die als Gutachter das Zustandekommen dieser Arbeit erst ermöglicht haben.

Kurzfassung

Es wird prognostiziert, dass in Zukunft die Hälfte der Firmenausgaben für mobile Anwendungen in die Entwicklung von Chatbots oder intelligenten Assistenten fließt. In diesem Bereich benötigt es zur Zeit viel manuelle Arbeit zur Modellierung von Beispielfragen. Diese Beispielfragen werden benötigt, um natürlichsprachliche Anfragen zu verstehen und in Datenbankanfragen umsetzen zu können. Im Rahmen dieser Arbeit wird ein Ansatz vorgestellt, welcher die manuelle Arbeit reduziert. Dazu wird mittels der Daten aus der Datenbank und Formulierungen, inklusive Synonymen, aus Dialogflow (ein intelligenter Assistent von Google) eine Aktive Ontologie erzeugt. Diese Ontologie verarbeitet anschließend die natürlichsprachlichen Anfragen und extrahiert die Parameter, welche für die Anfrage an die Datenbank benötigt werden. Die Ergebnisse der Aktiven Ontologie werden mit den Ergebnissen aus Dialogflow verglichen. Bei der Evaluation fällt auf, dass die Aktiven Ontologien fehleranfällig sind. Es werden zusätzliche, unerwünschte Parameter extrahiert, welche das Ergebnis verschlechtern. Die Übereinstimmungsrate bei einem Eins-zu-Eins-Vergleich mit Dialogflow liegt bei etwa 40%. Zukünftig könnte durch das Hinzufügen einer zusätzlichen selektiven Schicht innerhalb der Aktiven Ontologien die Parameterextraktion verbessert werden.

Inhaltsverzeichnis

1. Einleitung **1**
1.1. Zielsetzung . . . 2
1.2. Struktur der Arbeit . . . 2

2. Grundlagen **3**
2.1. Aktive Ontologien . . . 3
2.1.1. Aufbau von Aktiven Ontologien . . . 3
2.1.2. Active-Rahmenarchitektur . . . 5
2.1.3. Auswertung natürlichsprachlicher Eingaben . . . 6
2.2. Dialogflow . . . 8
2.2.1. Einführung . . . 8
2.2.2. Umwandlung natürlicher Sprache in strukturierte Daten . . . 12
2.3. Schema.org . . . 12
2.4. Deduktive Datenbanken . . . 14
2.4.1. Ursprüngliche Definition deduktiver Datenbanken . . . 14
2.4.2. SemReasoner . . . 15
2.5. Computerlinguistik . . . 16
2.5.1. Bag-of-Words . . . 16
2.5.2. TF-IDF . . . 17
2.5.3. Part-of-Speech Tagger . . . 18

3. Verwandte Arbeiten **19**
3.1. Textbasierte Ontologieextraktion . . . 19
3.2. Ontologieextraktion aus einer relationalen Datenbank . . . 21
3.3. Natural Language Interfaces to Databases . . . 22
3.3.1. Semantische Grammatiksysteme . . . 23
3.3.2. NLIDB-Systeme mit Zwischenrepräsentationen . . . 23
3.4. Active . . . 24

4. Analyse und Entwurf **27**
4.1. Analyse . . . 27
4.1.1. Manuelle Erzeugung der Aktiven Ontologie . . . 28
4.1.2. Erzeugung aus bestehenden Datenbankanfragen . . . 28
4.1.3. Generelle Aktive Ontologie . . . 29
4.1.4. Struktur der Datenbank in Aktive Ontologie übertragen . . . 29
4.1.5. Gruppieren der Daten in Aktiven Ontologien . . . 29
4.2. Entwurf . . . 31
4.2.1. Datenbanken . . . 31
4.2.2. Intelligente Assistenten . . . 32
4.2.3. Zwischenformat . . . 34
4.2.4. Wörterbücher . . . 34
4.2.5. Erzeugen der Aktiven Ontologie . . . 35

4.2.6. Erzeugen der Datenbankanfrage . . . 37
4.2.7. Bag-of-Words-Mechanismus . . . 38

5. Implementierung **41**
5.1. Übersicht . . . 41
5.2. EASIER-Rahmenarchitektur . . . 42
5.2.1. Synonymknoten . . . 43
5.2.2. Bag-of-Words-Mechanismus . . . 44
5.2.3. Anpassung von bestehenden Knoten . . . 45
5.3. Zwischenformat . . . 46
5.4. Export aus der Datenbank . . . 47
5.5. Export aus Dialogflow . . . 49
5.6. Zusammenführen der Exports . . . 51
5.7. Wörterbuch . . . 53
5.8. Erzeugung der Aktiven Ontologien . . . 54

6. Evaluation **57**
6.1. Datenbasis . . . 57
6.1.1. Datensätze . . . 57
6.1.2. Dialogflow-Agenten . . . 60
6.2. Methodik . . . 61
6.2.1. Testfragen . . . 61
6.2.2. Variation der Sprachinformationen . . . 61
6.2.3. Durchführung . . . 62
6.2.4. Evaluation des Bag-of-Word-Mechanismus . . . 65
6.3. Auswertung der Ergebnisse . . . 65
6.3.1. Vergleich mit Dialogflow . . . 66
6.3.2. Bag-of-Words-Mechanismus . . . 67

7. Zusammenfassung und Ausblick **69**

Literaturverzeichnis **73**

Anhang **77**
A. First Appendix Section . . . 77
A.1. Mögliche weitere Funktionen . . . 77
B. Literaturanalysen . . . 79
B.1. The Ontology Extraction Maintenance Framework Text-To-Onto . . . 79
B.2. Ontology learning for the semantic web . . . 82
B.3. TextOntoEx: Automatic ontology construction from natural English text . . . 85
B.4. Ontology extraction from relational database: Concept hierarchy as background knowledge . . . 87
B.5. Learning Ontology From Relational Database . . . 89
B.6. Active Framework . . . 91
B.7. Easier: An Approacht to Automatically Generate Active Ontologies for Intelligent Assistants . . . 92
B.8. DIALOGIC: A Core Natural-Language Processing System . . . 93
B.9. Natural Language Query Processing Using Semantic Grammar . . . 95

Abbildungsverzeichnis

1.1. Angereicherte Beispielfrage in Dialogflow 1

2.1. Schematische Darstellung einer Aktiven Ontologie nach Guzzoni et al. [GBC07] 4
2.2. Active-Architektur nach Guzzoni et. al [GBC07][S.2] 6
2.3. Beispielhafte Ontologie zur Suche nach Wanderwegen 7
2.4. Beispielfragen eines Intents zu Skipasspreisen aus einem privaten Dialogflow-Agenten. 9
2.5. Aktion und Parameter eines Intents für Sportstätten aus einem privaten Dialogflow-Agenten. 9
2.6. Entity-Typ für verschiedene Stufen der Schwierigkeit aus einem privaten Dialogflow-Agenten. 11
2.7. Flugreservierungsinformationen formatiert nach Schema.org. Quelle: [GBM16] (S.4) 14

3.1. Beispiel für die Umwandlung der Informationen aus der RDB zur OWL Ontologie. (Übernommen aus [SHA11]) 21
3.2. Beispiel für eine Ontologie von Guzzoni et al. (Übernommen aus [GCB06]) 25

4.1. Datenflussgraph zum Ansatz dieser Arbeit 32

5.1. Schnittstellen zwischen den Modulen 42
5.2. Ausschnitt aus einer Aktiven Ontologie 45
5.3. Grafische Oberfläche der *EASIER*-Rahmenarchitektur 56

Tabellenverzeichnis

2.1. Beispiele aus der POS-Tag Tabelle für Deutsch. Quelle: [STS99] 18

3.1. Verbindung zwischen RDB-Komponenten und OWL-Komponenten. (Übernommen aus [SHA11]) 21
3.2. Schema einer relationalen Datenbank. (Auszug aus [LDW05]) 22

6.1. Anzahl an Elementen für die vorhandenen Typen 58
6.2. Anzahl an Fragen und Entitäten für die vorhandenen Themengebiete 60
6.3. Kombinationen der drei Dialogflow-Agenten. 62
6.4. Anzahl an Fragen zur Erzeugung des Bag-of-Words-Mechanismus 65
6.5. Ergebnisse der evaluierten Kombinationen. 66
6.6. Ergebnisse der Evaluation des Bag-of-Words-Mechanismus. 67

1. Einleitung

„Gartner Says 25 Percent of Customer Service Operations Will Use Virtual Customer Assistants by 2020“
[Moo18].

Der aktuelle Trend in der IT-Welt geht in Richtung künstlicher Intelligenz und digitaler Assistenten. Digitale Assistenten, wie Amazon Alexa[1], Google Assistant[2], Apple Siri[3] oder auch Chatbots werden immer populärer [Tra18]. Chatbots sind automatische Systeme die menschliche Kommunikation durch Sprachbefehle, Textnachrichten oder beides imitieren. Diese Systeme können bestimmte Aufgaben ausführen und dem Nutzer automatisiert auf seine Fragen antworten. Die Erstellung benötigt zur Zeit noch viel manuelle Arbeit. Beispielsweise müssen mögliche Frageformulierungen für diese Assistenten erzeugt und gepflegt werden.

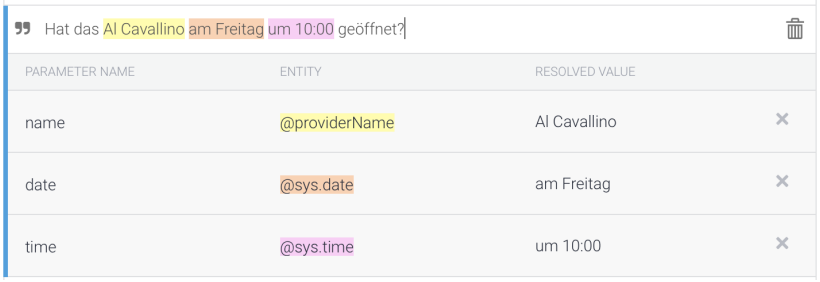


Abbildung 1.1.: Angereicherte Beispielfrage in Dialogflow

In Abbildung 1.1 ist eine solche Frageformulierung dargestellt. Die Abbildung zeigt die Benutzeroberfläche von Dialogflow zur Modellierung von Fragen. Über eine solche Oberfläche müssen alle Beispielfragen eingepflegt werden.
Um möglichst viele Nutzerfragen beantworten zu können, wird eine große Menge an Beispielfragen benötigt. Zum Anreichern der Fragen mit Informationen bieten die meisten Anbieter eigene Sprachverarbeitungswerkzeuge an. Google entwickelt zum Beispiel das

[1] `www.amazon.de/b?ie=UTF8&node=12775495031`
[2] `assistant.google.de/`
[3] `www.apple.com/de/ios/siri/`

Werkzeug Dialogflow[4].
Mit den aus der Frage gewonnenen Informationen wird eine Anfrage an eine Datenbank ausgeführt, um die Frage entsprechend beantworten zu können. Für die Anfrage mittels natürlicher Sprache an eine Datenbank existiert bereits ein Forschungsfeld. Die Verfahren heißen „Natural Language Interfaces to Databases“ (NLIDB) [PRGBAL+13]. Viele bekannte Verfahren weisen jedoch eine schlechte Performanz auf. Diese funktionieren sehr gut für die Domäne, für die sie konfiguriert wurden. Jedoch gibt es Schwierigkeiten bei der Portierung auf Datenbanken in einem anderen Kontext. Sobald diese Systeme ohne erneute Konfigurationen in einem anderen Kontext eingesetzt werden sollen, verschlechtert sich die Genauigkeit der Anfragen drastisch. Des Weiteren funktionieren die meisten Verfahren nahezu ausschließlich mit relationalen Datenbanken, da diese sehr viel häufiger verwendet werden, als deduktive Datenbanken. Im Vergleich zu deduktiven Datenbanken ist es in relationalen Datenbanken nicht möglich transitive Anfragen zu stellen. Dies ist gerade im Bereich der digitalen Assistenten ein Problem. Um beispielsweise benutzerspezifische Vorschläge zu machen, muss transitiv über die Daten iteriert werden können.

1.1. Zielsetzung

Das Ziel dieser Masterarbeit ist die manuelle Arbeit zur Pflege der Beispielfragen oder Frageformulierungen so weit wie möglich zu reduzieren und die existierenden intelligenten Assistenten durch Aktive Ontologien zu ersetzen. Hierfür soll ein generischer Ansatz entwickelt werden, der aus den Daten der Datenbank automatisch Aktive Ontologien erzeugt. Die erzeugten Aktiven Ontologien werden nach Abschluss des Verfahrens zur Extraktion der benötigten Informationen aus natürlichsprachlichen Anfragen verwendet. Ziel ist die Nutzung dieses Ansatzes für einen Chatbot. Dieser beantwortet automatisiert Anfragen von Benutzern zu unterschiedlichen Themengebieten. Dabei ist von besonderer Bedeutung, dass der Ansatz domänenunabhängig funktionieren soll, im Gegensatz zu den bisher bestehenden NLIDB-Verfahren. Sollten neue Daten in die Datenbank hinzugefügt werden, so kann der Ansatz semi-automatisch neue Aktive Ontologien erzeugen.

1.2. Struktur der Arbeit

Die Arbeit besteht aus insgesamt sieben Kapiteln. Zunächst werden die zum Verständnis der Arbeit wichtigen Grundlagen eingeführt (Kapitel 2). Im Anschluss an die Grundlagen folgt die Untersuchung des aktuellen Stands der Technik (Kapitel 3). Hierbei werden drei Themengebiete, welche Ähnlichkeiten zum Verfahren dieser Arbeit besitzen, analysiert. Die drei Themengebiete gliedern sich in Extraktion von Ontologien aus Datenbanken oder Text, natürlichsprachliche Anfragen an Datenbanken und die Rahmenarchitektur Active. Nach der Analyse des aktuellen Standes wird die Problemanalyse erläutert (Kapitel 4), sowie der Entwurf zur Umsetzung des Ansatzes beschrieben. Anschließend folgt eine Darstellung der Implementierung (Kapitel 5). Im vorletzten Kapitel wird die Evaluation (Kapitel 6) und deren Ergebnisse dargestellt. Den Schluss der Arbeit bildet die Zusammenfassung und ein Ausblick auf mögliche weitere Arbeiten (Kapitel 7).

[4] `dialogflow.com`

2. Grundlagen

Dieses Kapitel erläutert die grundlegenden Technologien, die zum Verständnis der Arbeit nötig sind. Der Ansatz dieser Arbeit basiert auf Aktiven Ontologien (Abschnitt 2.1). Diese werden im Rahmen der Arbeit aus der Struktur einer Datenbank erzeugt und anschließend für das Sprachverständnis verwendet. Zur Anreicherung der Aktiven Ontologie werden bereits modellierte Beispielfragen aus *Dialogflow* (Abschnitt 2.2) verwendet. Die Daten in der Datenbank sind konform zu den Schemadefinitionen von *Schema.org*. *Schema.org* (Abschnitt 2.3) beschreibt allgemeine Schemata zur Repräsentation von Daten. Die zur Abfrage benötigten Daten werden in einer deduktiven Datenbank (Abschnitt 2.4) gespeichert.

2.1. Aktive Ontologien

Als konventionelle Ontologie bezeichnet man eine konzeptionelle Datenstruktur zur formalen Repräsentation von Wissen. Sie besteht aus Klassen, Attributen und Relationen. Ein wichtiger Aspekt von Ontologien ist, dass sie lesbar und verständlich sind, sowie von allen implementierungsspezifischen Details abstrahieren. Aktive Ontologien [GBC06, GCB06, GBC07, BL16, BL18] hingegen sind Ausführungsumgebungen. Das bedeutet, dass die Aktionen innerhalb und zwischen Konzepten mittels Regelsätzen definiert werden.
Zur Ausführung von Aktiven Ontologien wird im Rahmen dieser Arbeit die *EASIER*-Rahmenarchitektur verwendet. Diese basiert auf dem in diesem Kapitel vorgestellten Konzept.
Im Folgenden wird zunächst auf den Aufbau von Aktiven Ontologien (Abschnitt 2.1.1) eingegangen. Anschließend werden die einzelnen Komponenten der *Active*-Rahmenarchitektur (Abschnitt 2.1.2) erläutert. Den Abschluss dieses Kapitels bildet die Auswertung von Aktiven Ontologien (Abschnitt 2.1.3).

2.1.1. Aufbau von Aktiven Ontologien

Aktive Ontologien sind von der Struktur ähnlich zu konventionellen Ontologien. Der Unterschied zwischen einer konventionellen Ontologie und einer Aktiven Ontologie sind die Konzepte. Bei Aktiven Ontologien sind die Konzepte Verarbeitungselemente, welche die Ontologie „aktiv“ machen. Jedes Konzept besitzt eine Menge an Regeln die beschreiben, wie Informationen verarbeitet werden. Konzepte kommunizieren Zustandsinformationen, Hypothesen und Anfragen über Relationen. Anhand der ankommenden Nachrichten wird

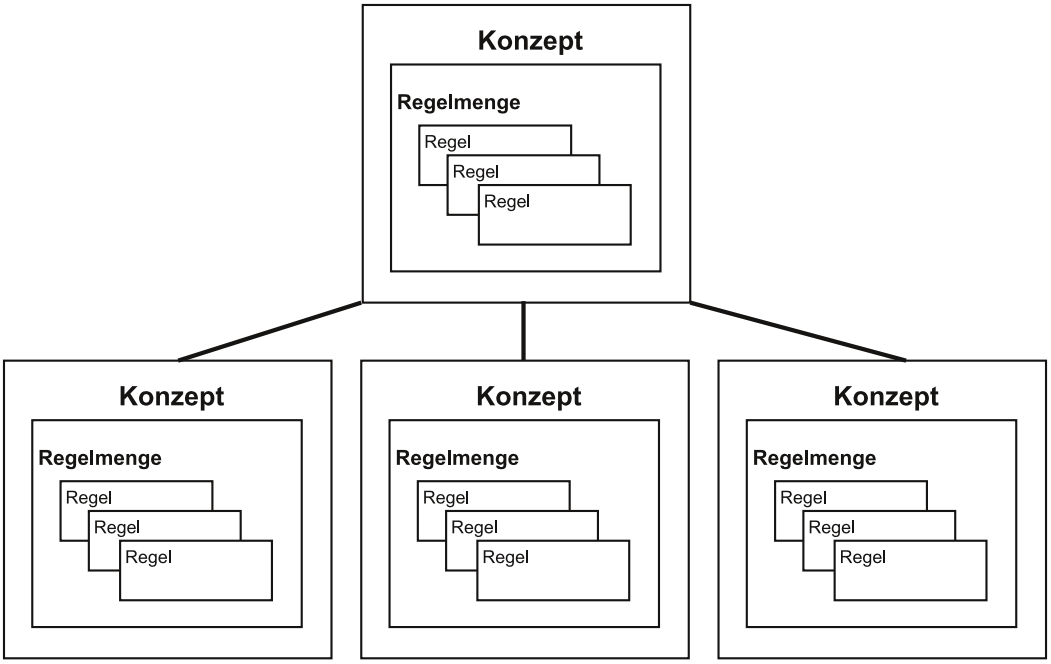


Abbildung 2.1.: Schematische Darstellung einer Aktiven Ontologie nach Guzzoni et al. [GBC07]

entschieden, wie die Informationen weiter verarbeitet werden. Das Verhalten der Verarbeitung ist dabei durch die Regeln der Konzepte beschrieben.
Die Aktive Ontologie ist im Prinzip ein Baum, der von unten nach oben durchlaufen wird. Jedes Konzept speichert den aktuellen Zustand im Faktenspeicher. Dies ist für den Kontext von Konversationen wichtig. Für jedes Konzept ist außerdem angegeben inwiefern es verpflichtend ist. Sollte ein Konzept verpflichtend sein, aber keine Information weitergegeben haben, so kann eine Nachfrage nach dieser Information mittels einer Dialogkomponente erzeugt werden. Regeln der einzelnen Konzepte bestimmen, wie Informationen verarbeitet und weitergegeben werden. Erlaubt eine Relation zwischen zwei Konzepten mehrere Nachrichten, so kann ein Konzept in einer Ausführungsrunde auch mehrere Ergebnisse an den Elternknoten weitergeben.
Abbildung 2.1 zeigt die schematische Darstellung einer Aktiven Ontologie. Diese beispielhafte Ontologie enthält ein Wurzelkonzept und drei Unterknoten. Für jeden Knoten ist die Regelmenge dargestellt. Eine Regel beschreibt zum Beispiel, auf welchen Bezeichner eines Faktes der Knoten reagieren soll. Wenn ein Konzept einen Fakten erkennt, so werden diese entsprechend der Aktion (zusammenfügen, selektieren) verarbeitet und wieder in den Faktenspeicher geschrieben.
Aktive Ontologien besitzen drei Typen von inneren Knoten (beziehungsweise Konzepten). Zum einen die Sammelknoten, diese fassen die Informationen der Kindknoten zusammen und leiten diese an die Elternknoten weiter. Zum anderen Auswahlknoten, die spezielle Informationen der Kindknoten auswählen und diese anschließend weiterleiten. Ein dritter Knotentyp ist ein `Helferknoten`, wenn dieser Knoten feuert wird eine Konfidenz von 100 weitergegeben. Eine Konfidenz von 100 bedeutet, dass der Elternknoten auf jeden Fall feuert. Beziehungen zwischen Konzepten sind definiert durch Relationen, diese definieren nicht nur die Struktur der Ontologie sondern auch den Datenfluss.
Es existieren zwei Typen von Relationen, eine „hat“-Beziehung (zum Beispiel eine Person „hat“ ein Alter) und eine „ist“-Beziehung (zum Beispiel „Max Mustermann“ ist eine „Person“). Die „hat“-Relation enthält zudem eine Kardinalität. Ob ein Unterkonzept zwingend feuern muss, wird mittels der Kardinalität angegeben. Unterschieden wird bei der Kardi-

nalität zwischen obligatorisch und optional. Im obligatorischen Fall, muss der Unterknoten einen Wert liefern. Anderenfalls wird kein Wert erwartet, falls einer existiert, so wird dieser weitergegeben. „Ist“-Relationen zwischen Konzepten haben eine klassifizierende Funktion. Das Active-Semantic-Network, dieses ist für die Ausführung der Aktiven Ontologien zuständig, definiert zudem vier verschiedene Arten von Blattknoten. Blattknoten, auch Sensorknoten genannt, erfassen neue Informationen und leiten diese entsprechend weiter. Es existieren Sensorknoten die Vokabularlisten enthalten. Diese vergleichen ankommende Zeichen mit einer vordefinierten Menge an Wörtern. Wird eines dieser vordefinierten Wörter erkannt, so erhalten die Elternknoten diese Information. Präfix- und Postfixknoten betrachten mehrere Zeichen vor beziehungsweise nach einem oder mehreren gegebenen Schlüsselwörtern. Ein dritter Typ an Sensorknoten verwendet reguläre Ausdrücke, um gültige Zeichenketten, beispielsweise Postleitzahlen oder Telefonnummern, zu identifizieren. Der vierte Knotentyp ist ein Spezialknoten. Dieser kann z.B. Daten oder Uhrzeiten erkennen und in ein explizites Datum umwandeln. Beispielhaft wird das Wort „heute“ in ein Datum der Form „yyyy-MM-dd“ umgewandelt.
Zur Ausführung von Aktiven Ontologien existiert ein Active-Server. Dieser besitzt einen Faktenspeicher für die auszuführenden Aktiven Ontologien. Der Faktenspeicher speichert den aktuellen Zustand des *Active*-Programms. Ein Verarbeitungszyklus wertet die Regelbedingungen auf Grundlage des aktuellen Stands der Faktenbasis aus. Werden die Bedingungen einer Regel erfüllt, so wird die dazugehörige Aktion ausgeführt. Eine Aktion kann beispielsweise das Zusammenführen von Informationen (Sammelknoten) oder das Auswählen der besten Information (Selektknoten) sein. Aktionen können ebenfalls Fakten in den Faktenspeicher schreiben.

2.1.2. Active-Rahmenarchitektur

Die Active-Rahmenarchitektur besteht aus drei Komponenten: Dem Active-Editor, Active-Server und der Active-Console.
Der Editor dient als Entwicklungsumgebung zum Entwurf, der Herausgbae und zum Testen eines Active-Programms. Zur Modellierung der Ontologie besitzt der Editor einen Grapheditor und einen integrierten Code-Text-Editor zur Bearbeitung der Verarbeitungsregeln. Außerdem gibt es einen Defektlokalisierungs(*engl. Debugging*)-Bereich zum Analysieren der Aktiven Ontologie. Der Defektlokalisierer wird dazu mit dem Active-Server verbunden und visualisiert anschließend die Ausführung der Aktiven Ontologie, sodass der Entwickler die Abläufe analysieren kann.
Beim Active-Server handelt es sich um ein skalierbares Laufzeitmodul, welches eine oder mehrere Active-Programme betreibt und ausführt. Dieser Server kann entweder als alleinstehende Anwendung ausgeführt oder auf einem J2EE-konformen Anwendungsserver ausgebracht werden. Zudem bietet er eine SOAP-API, die es externen Sensorkomponenten erlaubt ihre Ergebnisse zu berichten. Dies geschieht indem die Resultate aus der Ferne in den Faktenspeicher eingespeist werden. Das Einspeisen der neuen Fakten setzt einen neuen Auswertungszyklus in Gang.
Die dritte Komponente ist die Active-Konsole; sie ermöglicht die Beobachtung und Wartung eines laufenden Active-Servers.

Die Abbildung 2.2 verdeutlicht auf grafische Weise den Aufbau der Active-Architektur. Der Active-Server ist der Kern der Active-Architektur. Dieser führt die Aktiven Ontologien aus und bietet zudem eine Erweiterung für externe Dienste (beispielsweise an Webdienste, um mit extrahierten Parametern eine Suche auszuführen). Der Active-Editor, sowie die Active-Konsole, können sich mit dem Active-Server verbinden und entsprechend die Aktiven Ontologien überwachen, untersuchen und veröffentlichen.

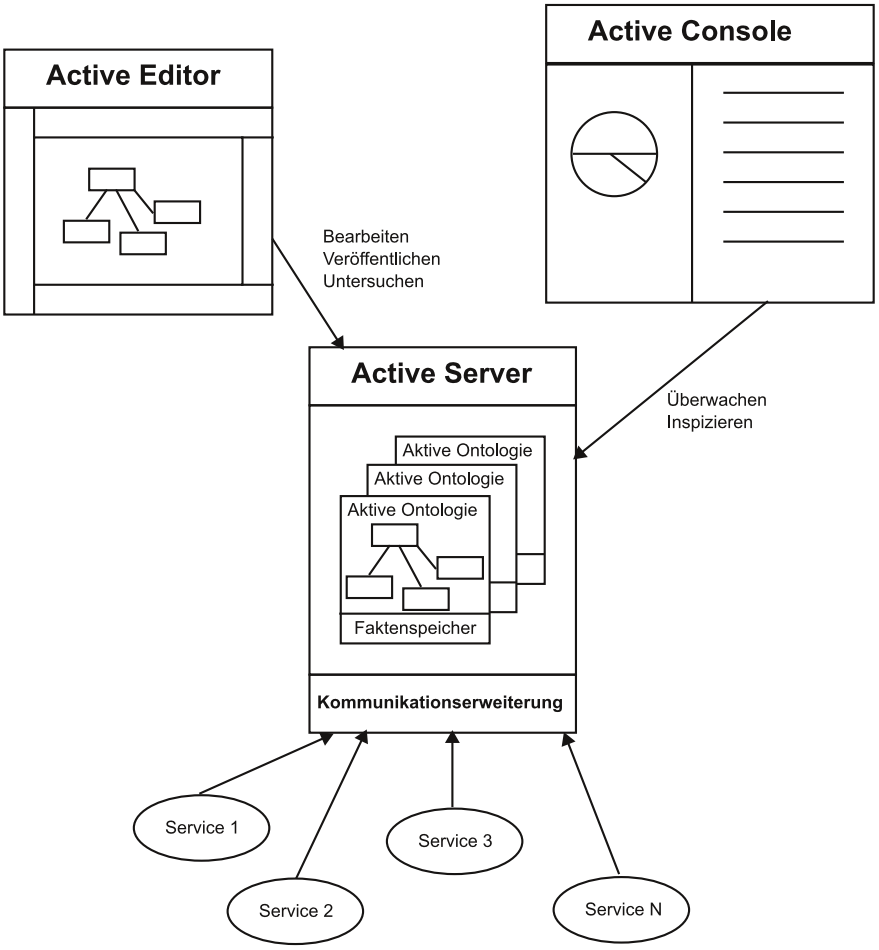


Abbildung 2.2.: Active-Architektur nach Guzzoni et. al [GBC07][S.2]

2.1.3. Auswertung natürlichsprachlicher Eingaben

Der Auswertungsprozess einer Aktiven Ontologie wird durch eine Änderung im Faktenspeicher ausglöst. Im Sinne der Auswertung von natürlicher Sprache, werden die ankommenden Äußerungen im Faktenspeicher hinzugefügt. Die Sensorknoten beginnen daraufhin mit der Verarbeitung der neuen Informationen. Entsprechend dem Typ der Sensorknoten werden Informationen extrahiert und an die Elternknoten weitergegeben. Schlussendlich laufen alle Informationen im Wurzelknoten zusammen.

Beispiel

Abbildung 2.3 zeigt eine beispielhafte Aktive Ontologie zum Thema Wanderwege. Die Konzepte repräsentieren die Informationen, die über Wanderwege abgefragt werden können. In diesem Beispiel kann nach Wanderwegen einer bestimmten **Länge**, **Schwierigkeit** oder an einer bestimmten **Adresse** gefragt werden. Die Adresse unterteilt sich noch einmal in **Stadt**, **Postleitzahl (PLZ)** und **Region**.
Ein Benutzer könnte folgende Frage stellen: „Ich suche einen leichten Wanderweg in Karlsruhe.“
Diese Frage wird im Faktenspeicher der Aktiven Ontologie gespeichert. Eine Veränderung des Faktenspeichers löst automatisch einen Evaluierungsprozess aus. Die Sensorknoten verarbeiten die neuen Informationen. In diesem Beispiel erkennt der Sensorknoten **Schwierigkeit** das Wort „leichten“, da dieses in seiner Wortliste vorkommt. Der Sensorknoten **Stadt** erkennt den Ort „Karlsruhe“. Die anderen Sensor-

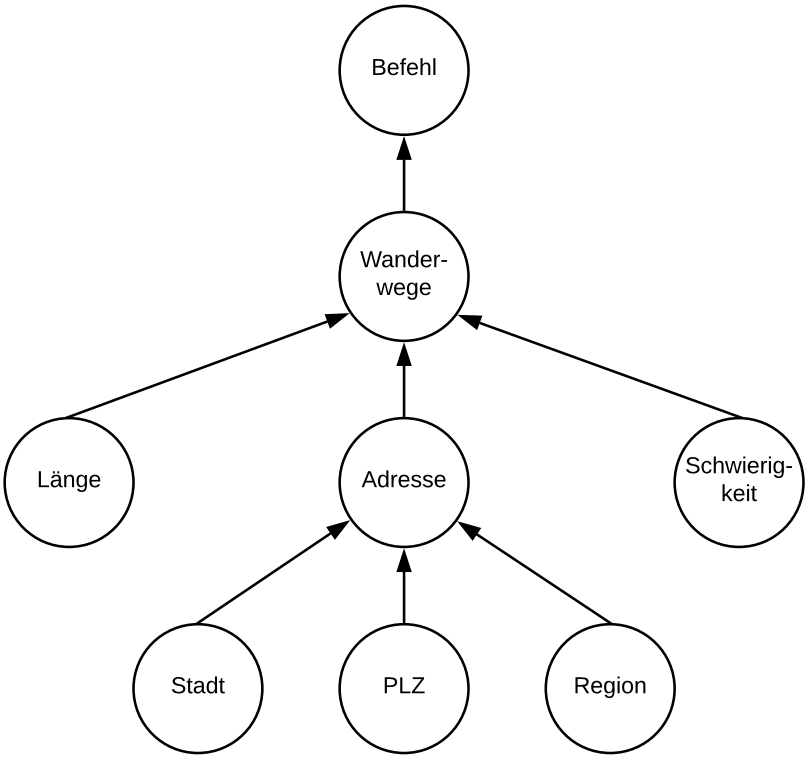


Abbildung 2.3.: Beispielhafte Ontologie zur Suche nach Wanderwegen

knoten erkennen keine neuen Informationen, da die übrig bleibenden Wörter nicht in der jeweiligen Wortliste vorhanden sind. Die erkannten Informationen werden von den Konzepten in den Faktenspeicher geschrieben. Im nächsten Evaluationsschritt wird bei der Auswertung der Regel des Elternknotens, beispielsweise aufgrund einer bestimmten Bezeichnung, dieser Wert durch den Elternknoten erkannt. Entsprechend der definierten Aktion werden entweder alle durch den Elternknoten erkannten Informationen zusammengeführt oder die Information mit der höchsten Konfidenz ausgewählt und auf dieselbe Weise weitergegeben. Der Befehl zur Suche nach einem Wanderweg wird sukzessive zusammengesetzt. Der **Befehl**-Knoten enthält schlussendlich den Befehl zur Anfrage an die Datenbank.
Beispielhafter Befehl:
„Wanderwege(Schwierigkeit(leicht), Adresse(Stadt(Karlsruhe)))“
Der Wurzelknoten (**Befehl**) enthält zudem die Logik zur Umsetzung des Befehls in eine Anfrage an eine Datenbank. Diese Logik würde den Befehl anhand verschiedenster Regeln analysieren und aus den einzelnen Bestandteilen Teile einer SQL-Abfrage generieren. Aus „Wanderwege(...)“ lässt sich beispielsweise ablesen, dass die Tabelle „Wanderwege“ verwendet wird. Innerhalb dieser Tabelle gibt es eine Spalte „Schwierigkeit“ und eine Spalte „Adresse“. Ebenso wird mit den weiteren Informationen verfahren und folgende SQL-Anfrage erzeugt:

```
SELECT *
FROM Wanderwege w
        JOIN Adresse a
                ON w.Adresse_ID = a.ID
WHERE w.Schwierigkeit = 'leicht'
AND a.Stadt = 'Karlsruhe'
```

2.2. Dialogflow

Dieses Kapitel beschäftigt sich mit der Umsetzung von natürlicher Sprache in strukturierte Daten mittels *Dialogflow*[1]. In Dialogflow existieren Agenten, welche vordefinierte Fragen zu bestimmten Themengebieten zusammenfassen. Innerhalb dieser Agenten werden die Fragen nach Absichten gruppiert, beispielsweise nach Fragen zu Geschäften, Sportplätzen und Veranstaltungen. Für jede dieser Absichten müssen Beispielfragen modelliert werden. Neben Dialogflow gibt es weitere Werkzeuge, die im Internet verfügbar sind. Die bekanntesten sind IBM Watson[2], Microsoft LUIS[3], WIT[4] und Amazon Lex[5]. Das Ziel dieser Werkzeuge ist, die relevanten Informationen aus einer natürlichsprachlichen Anfrage zu extrahieren und strukturiert zur Verfügung zu stellen. Zu Beginn werden die wichtigsten Begriffe aus Dialogflow eingeführt (Abschnitt 2.2.1) und anschließend die grobe Funktionsweise erläutert (Abschnitt 2.2.2). Die Erklärung folgen dabei den Definitionen, wie sie in Dialogflow verwendet werden.

2.2.1. Einführung

Zur Definition eines Agenten in Dialogflow müssen Absichten (engl. Intents) und Entitäten (engl. Entities) angelegt werden. Anhand der Intents bestimmt der Agent die Absicht des Benutzers. Für alle vorhandenen Intents eines Agenten existieren Beispielfragen. Stellt der Benutzer nun eine Frage, so wird diese mit den Beispielfragen der Intents verglichen. Der Intent mit der höchsten Übereinstimmung spiegelt die Absicht des Benutzers wieder. Die Entitäten dienen dazu, die entsprechenden Parameter der natürlichsprachlichen Anfrage zu extrahieren. Aufgrund der Annotationen der Beispielfragen, können die entsprechenden Parameter aus der gegebenen natürlichsprachlichen Anfrage extrahiert werden. Wichtig für eine möglichst natürliche Kommunikation mit einem Chatbot ist der Kontext einer Unterhaltung. Mittels des Kontextes können aufeinanderfolgende Sätze zu einem Thema in Verbindung gebracht werden. Den Abschluss der Grundlagen von Dialogflow bilden die Dialoge. Dialogflow bietet zwei unterschiedliche Möglichkeiten, um geführte Dialoge zu definieren.

Absichten (engl. Intents)

Ein Intent ist eine Abbildung von Benutzeranfragen auf die, durch die Software auszuführende, Aktion. Es wird zwischen zwei Typen von Intents unterschieden. Die „Casual Intents“ sind zuständig für den Beginn und das Ende einer Konversation. Sie enthalten Formulierungen für Begrüßungen, Verabschiedungen oder auch für Small Talks. Diese Intents extrahieren keine Parameter aus der natürlichsprachlichen Anfrage und es wird auch keine Anfrage an die Anwendung gestellt. Zur Extraktion der Parameter und für die Anfrage an die Anwendung werden „Business Intents“ verwendet. Wenn im Folgenden die Rede von einem Intent ist, ist der „Business Intent“ gemeint.
Ein Intent enthält Informationen über Trainingsphrasen. Dabei handelt es sich um möglichst viele Formulierungen von Fragen zu genau einer bestimmten Absicht. Je mehr Variationen die Trainingsphrasen aufweisen, desto höher ist die Wahrscheinlichkeit, dass viele unterschiedliche Benutzeranfragen erkannt werden. Ebenso ist in einem Intent der Name der Aktion definiert. Dieser Aktionsname wird in der eigentlichen Softwareanwendung verwendet, um eine bestimmte Aktion auszulösen. Des Weiteren können auch mögliche Antworten definiert werden. Diese Antworten werden nach dem Auslösen eines Intents an

[1] `https://dialogflow.com/docs/getting-started/basics`
[2] `https://www.ibm.com/watson/`
[3] `https://www.luis.ai/home`
[4] `https://wit.ai/`
[5] `https://aws.amazon.com/de/lex/`

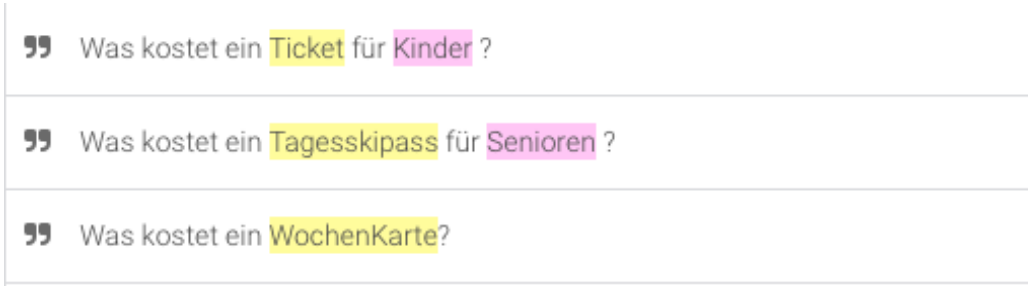


Abbildung 2.4.: Beispielfragen eines Intents zu Skipasspreisen aus einem privaten Dialogflow-Agenten.

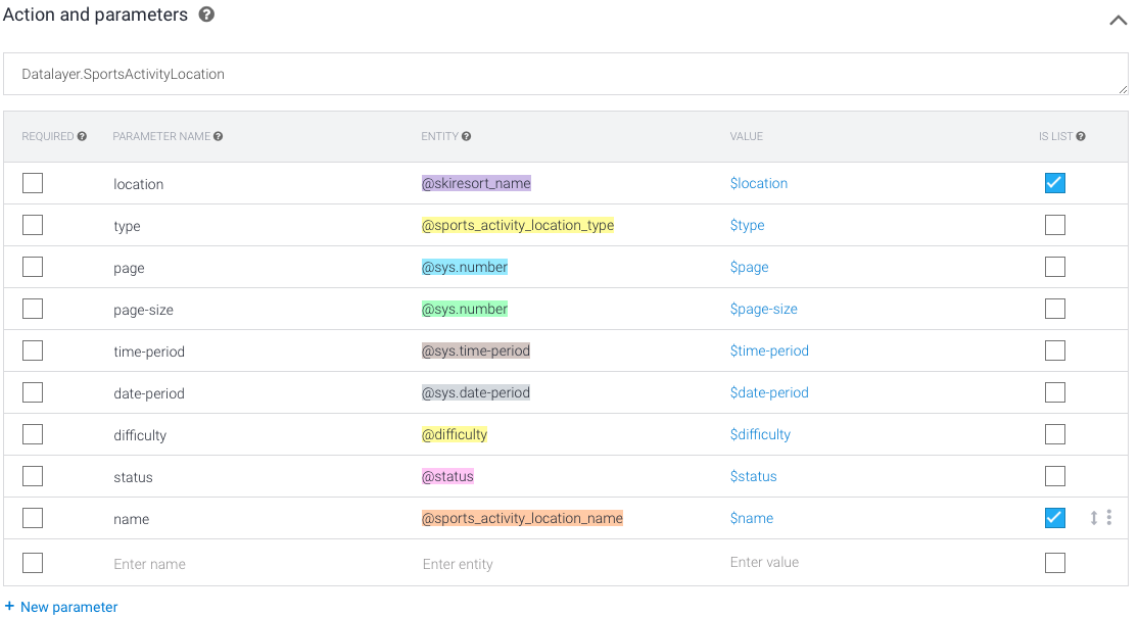


Abbildung 2.5.: Aktion und Parameter eines Intents für Sportstätten aus einem privaten Dialogflow-Agenten.

den Benutzer ausgegeben. Um in längeren Konversationen den Kontext zu behalten, wird dieser ebenfalls im Intent gespeichert. Zudem kann dieser Kontext dazu verwendet werden, den Verlauf der Konversation zu verwalten.

In Abbildung 2.4 ist ein Auszug der Beispielfragen aufgeführt, die für den Intent zu Skipasspreisen definiert wurden. Die farbig-markierten Worte sind die Parameter, die aus den Fragen extrahiert werden. Die Bezeichnung und der Entity-Typ eines Parameters sind in Abbildung 2.5 dargestellt. Das obere Textfeld („Datalayer.SportsActivityLocation“) gibt den Aktionsnamen an. Dieser wird für die eigentliche Anwendung benötigt. In der Tabelle unterhalb der Aktion sind die einzelnen Parameter und ihre Entity-Typen aufgelistet. Die Farben in der Entity-Spalte stimmen mit den Farben in den Beispielfragen überein. Beispielsweise „Mayrhofen“ gehört zum Parameter „location“ und hat den Entity-Typ „@skiresort_name“.

Entitäten (engl. Entities)

Entitäten sind ein sehr mächtiges Element bei der Extraktion von Parameterwerten aus der natürlichsprachlichen Eingabe. In Dialogflow gibt es drei Typen von Entitäten. Die durch Dialogflow vordefinierten System-Entitäten, Entitäten die durch einen Entwickler definiert werden und Entitäten die für jeden Endbenutzer in jeder Anfrage individuell sind. Diese

Typen können klassifiziert werden als Abbildung, Enumeration oder eine Mischform aus Abbildung und Enumeration. Abbildungen sind in diesem Fall definiert als, das erkannte Wort wird durch ein vorgegebenes Wort ersetzt.
Abbildungen besitzen einen Referenzwert, d.h. Eingaben unterschiedlicher Formate werden auf einen vorgegebenen Referenzwert abgebildet.

Beispiel: Abbildungen

Dieses Beispiel verwendet die vordefinierten Entitäten zur Erkennung von Datumsangaben (@sys.date). Diese wandeln gängige Datumsangaben in ein Datum im ISO-8601 Format um. Beispielsweise werden die Datumsreferenzen „Januar 1, 2015“, „Der erste Januar 2015“ und „Erster Januar 2015“ in das Datum „2015-01-01T12:00:00:00-03:00“ umgewandelt.

Enumerationen hingegen besitzen keine Referenzwerte. Worte, die als Entitäten vom Typ Enumeration erkannt werden, werden genau so weiterverwendet, wie sie eingegeben wurden. Es findet also keine Abbildung statt.

Beispiel: Enumeration

Betrachten wir den vordefinierten Entity-Typ @sys.color; dieser erkennt Farben. Gibt ein Benutzer beispielsweise die Farbe „Olivgrün“ ein, so wird nicht auf die Farbe „Grün“ abgebildet. Stattdessen wird genau die Benutzereingabe erkannt und weiterverwendet.

Die Mischform aus beiden zuvor vorgestellten Arten enthält zum einen verschiedene Abbildungen, aber auch Enumerationen. Das Ergebnis der Abbildung ist ein Objekttyp bestehend aus den erkannten Entitäten.

Beispiel: Mischform aus Enumeration und Abbildung

Beispielhaft wird hier der Entity-Typ zur Erkennung von Währungsangaben (@sys.unit-currency) verwendet. Bei einer Eingabe von „50 Euro“ oder auch „fünfzig Euro“ wird ein Objekttyp der Form „{“amount“: 50, “currency“: “EUR“}“ erzeugt.

Abbildung 2.6 stellt die Definition eines Entity-Typs mit Synonymen und Enumerationen zur „Schwierigkeit“, beispielsweise eines Wanderweges, dar. Die erste Spalte definiert den Referenzwert, also der Wert der später als Wert für den Parameter an die Anwendung übergeben wird. In der zweiten Spalte stehen die Synonyme. Wenn eines der Worte der zweiten Spalte erkannt wird, so wird dieses durch das Wort in der ersten Spalte ersetzt und als Wert für den entsprechenden Parameter gespeichert. Bei der Eingabe des Wortes „leicht“ wird „easy“ als Wert für den Parameter „Schwierigkeit“ gespeichert.

Kontext

Der Kontext repräsentiert den aktuellen Zustand einer Unterhaltung. Es werden die bisher erkannten Parameter und die Werte dazu gespeichert. Wie in einem natürlichen Gespräch kann sich der Agent so den aktuellen Zusammenhang der Konversation merken. Das folgende Beispiel erläutert einen Fall, in dem der Kontext der Konversation wichtig ist. Ohne den Kontext in diesem Beispiel könnte der Agent die Frage des Benutzers nicht beantworten, da diese nicht alle nötigen Informationen enthält. Aus dem Kontext heraus, erkennt der Agent die Absichten des Benutzers und kann eine geeignete Antwort zurückgeben. Dazu werden die Informationen, die als Kontext gespeichert wurden, mit den extrahierten Parametern der aktuellen Frage verknüpft.

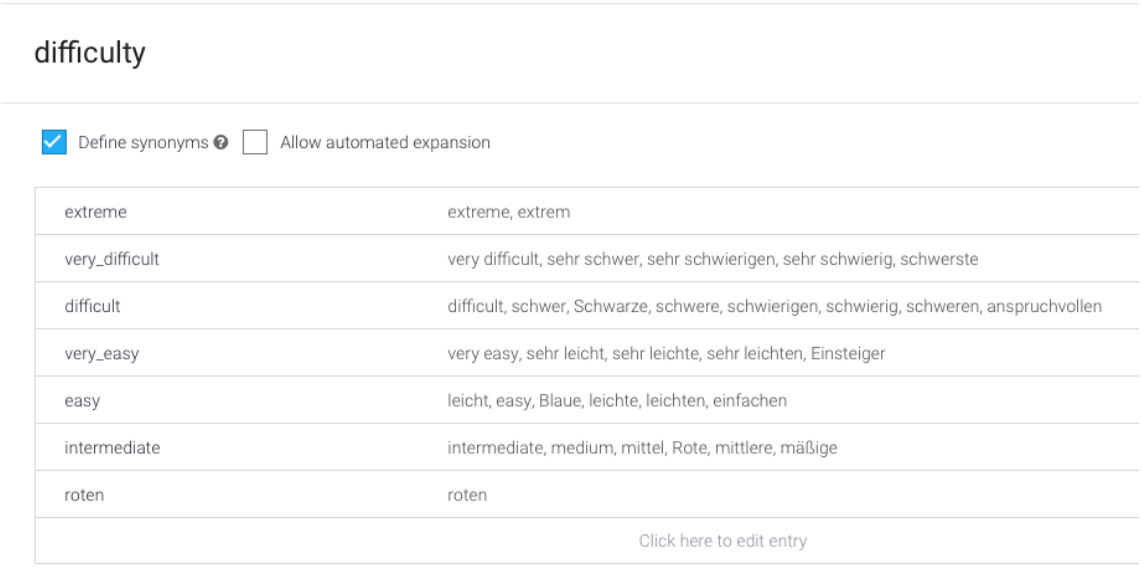


Abbildung 2.6.: Entity-Typ für verschiedene Stufen der Schwierigkeit aus einem privaten Dialogflow-Agenten.

Beispiel

Im folgenden wird eine beispielhafte Konversation dargestellt.
Benutzer: „Ich suche eine Pizzeria, die spezielle italienische Weine anbietet.“
Bot: „Ich kann Ihnen das Restaurant ‚Luigi's Pizzeria‘ empfehlen.“
Benutzer: „Wie sind die Öffnungszeiten?“
Bot: „Mo-Fr 17-22Uhr“
Der Benutzer fragt in diesem Beispiel nach einem Restaurant und bekommt ein ganz bestimmtes vorgeschlagen. Anschließend fragt dieser, ohne die erneute Nennung des Restaurants, nach dessen Öffnungszeiten. Würde der Kontext nicht gespeichert werden, so könnte der Bot nicht bestimmen, welche Öffnungszeiten der Benutzer wissen möchte.

Dialoge

Dialogflow bietet zwei Typen von Dialogen an. Zum einen die linearen Dialoge, diese dienen der einfachen Abfrage von benötigten Informationen zur Beantwortung einer Anfrage. Zum anderen die nicht-linearen Dialoge, mittels derer können je nach Konversationsverlauf andere Fragen gestellt werden.
Lineare Dialoge sind nicht dafür gedacht, kompliziertere Unterhaltungen zu steuern. Stattdessen sollen fehlende Informationen gesammelt werden. Damit diese fehlenden Informationen nachgefragt werden, müssen die einzelnen Parameter als „verpflichtend“ markiert werden. Wird ein Parameter als verpflichtend markiert, so muss eine Frage definiert werden, die gestellt wird.

Beispiel

Betrachten wir einen Intent zur Buchung eines Fluges. Die benötigten Parameter sind **Abflughafen**, **Zielflughafen**, **Datum des Hinflugs**, **Datum des Rückflugs** und die **Anzahl der Personen**. Der Intent beinhaltet als Beispielfragen viele Kombinationen, die alle die benötigten Parameter beinhalten. Allerdings auch Beispielfragen wie „Ich möchte einen Flug buchen.“. Fragt ein Benutzer diese einfache Frage ohne die Angabe von weiteren Informationen, so müssen die fehlenden Parameter erfragt wer-

den. Dazu wurden bei der Modellierung des Dialoges Beispielfragen für die Parameter angegeben. Um die fehlenden Parameterwerte zu füllen, werden diese vordefinierten Fragen nacheinander gestellt. Beispielsweise wurde für den Parameter „Abflughafen“ die Frage „Von wo möchten Sie abfliegen?“ definiert. Da der Parameter in der Beispielfrage fehlt, wird diese Frage an den Benutzer ausgegeben. Dieser Antwortet mit dem entsprechenden Abflughafen, sodass der Parameterwert befüllt werden kann. Analog dazu werden die restlichen fehlenden Parameterwerte erfragt.

Nicht-Lineare Dialoge dienen der Modellierung von komplexeren Konversationen. Dies bedeutet, dass der Bot in Abhängigkeit vom vorherigen Kontext, eine entsprechende Frage stellt.

Beispiel

Es wurde ein Dialog modelliert, der verschiedene Immobilien vorschlagen soll. Die erste Frage wäre also beispielsweise, inwiefern nach einer Wohnung oder einem Haus gesucht wird. Wenn nach einer Wohnung gesucht werden soll, ist die gewünschte Anzahl der Zimmer die nächste Frage. Wird allerdings nach Häusern gefragt, so ist die nächste Frage die Größe der Wohnfläche. Dies wäre ein Beispiel für einen komplexeren Dialog mit Verzweigungen, die auf dem vorherigen Kontext basieren.

2.2.2. Umwandlung natürlicher Sprache in strukturierte Daten

Im Folgenden wird die grobe Funktionsweise der Umwandlung natürlicher Sprache in strukturierte Daten in Dialogflow beschrieben, ohne näher auf interne Details einzugehen. Wird eine natürlichsprachliche Anfrage an Dialogflow geschickt, so wird im ersten Schritt berechnet, zu welchem Intent die Anfrage passt. Dazu wird die eingegebene Frage mit den modellierten Beispielfragen der Intents verglichen. Der Intent mit der höchsten Übereinstimmung wird ausgewählt. Anschließend geht es darum die benötigten Parameter zu extrahieren. Die Informationen in der natürlichsprachlichen Anfrage werden anhand der definierten Entity-Typen erkannt und extrahiert. Wurden alle verpflichtenden Parameter erkannt und es handelt sich nicht um einen Dialog, so werden diese strukturierten Daten an die Anwendung weitergegeben, die diese anschließend verarbeitet. Fehlen Informationen zu verpflichtenden Parametern, so werden diese mittels Fragen vervollständigt und anschließend an die Anwendung gesendet. Die aktuelle Belegung der Parameter wird als Kontext im aktuellen Intent gespeichert. Handelt es sich um einen nicht-linearen Dialog, so werden entsprechend der aktuellen Parameterbelegung weitere Fragen gestellt und nach Abschluss des Dialogs die gesammelten Informationen an die Anwendung weitergegeben.

2.3. Schema.org

Schema.org [GBM16] wurde im Jahr 2011 durch die Anbieter der großen Suchmaschinen (Google, Bing und Yahoo) gegründet. Das Ziel ist ein allgemeines Schema zum Erzeugen und Warten von strukturierten Daten für ein breites Spektrum an Themen (zum Beispiel Personen, Plätze, Veranstaltungen, Produkte, Angebote) zur Verfügung zu stellen. Bei der Veröffentlichung im Jahr 2011 gab es 297 Klassen mit 187 Relationen. 2016 waren es bereits 638 Klassen mit 965 Relationen. Die Klassen sind hierarchisch organisiert, d.h. jede Klasse hat eine oder mehrere Oberklassen. Das besondere an Schema.org ist, dass die Relationen polymorph sind. Dies bedeutet, sie besitzen mehrere Domänen und mehrere Bereiche in denen sie Verwendung finden.
Die erste Anwendung, die auf dieses allgemeine Schema setzte, waren Googles Rich Snippets[6]. Rich Snippets liefert in den Suchergebnissen neben der URL weitere Informationen zur gefundenen Webseite. Mittlerweile wird das Schema auch in E-Mails verwendet.

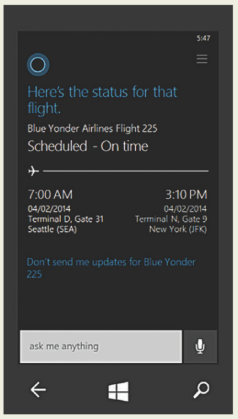
[6] `https://searchenginewatch.com/2018/02/19/the-2018-guide-to-rich-results-in-search/`

Dadurch können Assistenz-Systeme die strukturierten Informationen extrahieren und beispielsweise Kalendereinträge vorschlagen.
Im Folgenden wird auf Design-Entscheidungen von Schema.org eingegangen. Eine wichtige Entscheidung ist die Wahl der richtigen Syntax. Schema.org setzt sowohl auf RDFa [ABMP08] als auch auf JSON-LD [Con14], um den Entwicklern die nötige Freiheit zu bieten. Eine weitere Entscheidung betrifft den Polymorphismus. Viele Wissensrepräsentationen besitzen nur eine einzige Domäne. Dies führt allerdings zu vielen Klassen und es ist schwierig bestehende Relationen wiederzuverwenden ohne die Klassenhierarchie anzupassen. Schema.org hingegen erlaubt mehrere Domänen und auch der Bereich für Relationen ist nicht auf eine Domäne beschränkt. In vielen Modellen hat jedes Objekt einen eindeutigen Bezeichner. Bei diesem Ansatz gibt es nur eine kleine Anzahl an Elementen, die einen festen Bezeichner besitzen. Stattdessen muss derjenige, der ein Element veröffentlicht möglichst viele Informationen zu dem Element hinzufügen, sodass es eindeutig identifiziert werden kann. Beim Aufbau des Schemas wurde darauf geachtet, nicht mit dem komplexesten Fall anzufangen. Es wurde mit einem einfachen Anwendungsfall begonnen und je nach Bedarf, um die entsprechenden Eigenschaften erweitert. Schema.org bietet zwei Arten der Erweiterung, zum einen gibt es eine Gemeinschaft (engl. community) von Freiwilligen, die bestrebt ist, die von ihnen gemachten Erweiterungen in den Kern von Schema.org zu integrieren. Zum anderen gibt es die Erweiterungen, die jeder implementieren und verwenden kann. Diese finden keinen Zugang zum Kern des Schemas, da sie meist spezifisch für den einzelnen Fall angewendet werden.
Schema.org ist stark angelehnt an linked-data[7]. Beide teilen sich dasselbe Datenmodell, dieselbe Schemasprache und verwenden dieselben Syntaxen. Der Unterschied zwischen Schema.org und linked-data sind die Richtlinien zur Veröffentlichung von strukturierten Daten. Bei linked-data sind verschiedene Arten von Aufräumarbeiten, Abgleich und Nachbearbeitung nötig, bevor strukturierte Daten veröffentlicht werden. Linked-data-Daten haben dadurch eine sehr hohe Qualität, aber die Anzahl der Datenquellen ist deutlich geringer.

Beispiel

Abbildung 2.7 zeigt ein Beispiel, wie ein Schema aus Schema.org für eine E-Mail-Bestätigung verwendet werden kann. Durch die Verwendung des Schemas kann Microsofts Cortana die Informationen aus der E-Mail extrahieren und entsprechend anzeigen. Möglich wäre auch diese extrahierten Informationen direkt in den Kalender einzutragen, beispielsweise die Abflugs- und Ankunftszeiten.
Im Folgenden wird kurz auf einige Eigenschaften des Flugreservierungs-Schemas eingegangen. Das Schema definiert, um welche Art von Information es sich handelt. Dies wird durch die **@type**-Eigenschaft definiert. Des Weiteren wird der Reservierungsstatus durch **reservationStatus** angegeben. Die Person, auf die das Ticket läuft, hat als **@type** den Typ **Person** und den entsprechenden Namen (**name**). Außerdem enthält dieses Beispiel Informationen über den genauen Flug (**reservationFor**). Bezüglich des Fluges werden Informationen über die Flugnummer (**flightNumber**), den Abflughafen (**departureAirport**), den Zielflughafen (**arrivalAirport**), die Abflugzeit (**departureTime**), Ankunftszeit (**arrivalTime**) und die Airline (**airline**) angegeben.
Diese Schemas sind in Schema.org vorgegeben, sodass digitale Assistenten anhand dieses fest definierten Schemas genau wissen, welche Informationen zu extrahieren sind. Ebenso weiß der Entwickler, wie die Informationen strukturiert sein müssen, damit diese durch andere Dienste verwendet werden können.

[7] `https://www.w3.org/DesignIssues/LinkedData.html`




```
{ "@context": "http://schema.org/",
  "@type": "FlightReservation",
  "reservationNumber": "QWERT0123456789",
  "reservationStatus":
"http://schema.org/Confirmed",
  "underName":{
    "@type": "Person",
    "name": "Estella Gallagher"
  },
  "reservationFor": {
    "@type": "Flight",
    "flightNumber": "123",
    "departureAirport": {
      "@type": "Airport",
      "name": " Seattle-Tacoma International
Airport",
      "iataCode": "SEA"
    },
    "arrivalAirport": {
      "@type": "Airport",
      "name": " John F Kennedy International
Airport",
      "iataCode": "JFK"
     },
 "departureTime": "2014-04-02T10:32:00Z",
    "arrivalTime": "2014-04-02T11:45:00Z",
    "airline": {
      "@type": "Airline",
      "name": "Blue Yonder Airlines",
      "iataCode": "BY"
    }
  }
}
```

Abbildung 2.7.: Flugreservierungsinformationen formatiert nach Schema.org. Quelle: [GBM16] (S.4)

2.4. Deduktive Datenbanken

Die im Rahmen dieser Arbeit verwendeten Daten liegen in einer deduktiven Datenbank [LHL+98, BF08, LPF+02] vor. In deduktiven Datenbanken werden die Informationen als Ontologien gespeichert und mittels Regeln verarbeitet. Aufgrund von rekursiven Regeln sind deduktive Datenbanken ausdrucksstärker als relationale Datenbanken. Dieses Kapitel erläutert kurz das Grundkonzept von deduktiven Datenbanken (Abschnitt 2.4.1) und geht anschließend auf die verwendete deduktive Datenbank (Abschnitt 2.4.2) ein.

2.4.1. Ursprüngliche Definition deduktiver Datenbanken

Die Verwendung von Logik zur Repräsentation und dem Verändern von Wissen geht auf die Arbeit von Green [Gre69] zurück. Diese war Grundlage für verschiedenste Studien, die zu den sogenannten Frage-Antwort Systemen führten. Ähnliche, auf Resolution basierende, Verfahren wurden an Datenbanken angepasst, um größere Mengen an Daten und weitreichende Anfragen zu unterstützen. Resolution ist ein Verfahren aus der Logik zur Verifizierung einer logischen Formel. Ebenso werden dadurch andere datenbankspezifische Funktionen abgedeckt. Das Ergebnis dieser Anpassungen sind die deduktiven Datenbanken.
Gallaire et al. [GMN84] definieren deduktive Datenbanken wie folgt: Deduktive Datenbanken verwenden eine vereinfachte Version der generellen Logikprogrammierung, genannt Datalog [CGT89, KRS15]. Datalog wurde speziell entwickelt, um mit großen Datenbanken zu interagieren. Im Bereich der Logikprogrammierung wurde versucht Prolog-Systeme mit relationalen Datenbanken zu koppeln. Zur Steigerung der Effizienz dieser Systeme reichen einfache Schnittstellen nicht aus. Es benötigt Schnittstellen die eine stärkere Integration [CGW89] ermöglichen. Datalog ist ein erster Schritt in diese Richtung.
Datalog ist eine vereinfachte Version der generellen Logikprogrammierung. Ein Logikprogramm besteht aus einer endlichen Menge an Fakten und Regeln. Fakten sind Aussagen über relevante Teile der realen Welt (beispielsweise „John ist der Vater von Harry“). Mittels

Regeln können Fakten aus anderen realen Fakten gefolgert werden („Wenn X ein Elternteil von Y ist und wenn Y ein Elternteil von Z ist, dann ist X ein Großelternteil zu Z“). Regeln enthalten normalerweise Variablen, um generisch zu bleiben. In Datalog sind Fakten und Regeln als Hornklauseln [KK06] repräsentiert, allerdings ohne Funktionen. Eine Hornklausel ist eine Klausel der Form $\{L_1, ..., L_r\}$, bei der höchstens ein Literal positiv ist. Ein Beispiel für eine Hornklausel ist: $H = \{\neg A, \neg B, C\}$. Datalog bezieht sich auf den funktionsfreien Teil der Hornlogik. Die generelle Struktur sieht folgendermaßen aus: $L_0 : -L_1, L_2, ..., L_n$. Jedes Literal L_i ist von der Form $p(t_1, ..., t_k)$. p_i ist ein Prädikatsymbol und t_j sind die Terme. Ein Term ist entweder eine Konstante oder eine Variable. Im Gegensatz dazu, können Terme der Hornlogik zusätzlich auch Funktionen enthalten. Die linke Seite einer Regel, alles vor : −, wird als Kopf bezeichnet. Alles was rechts von : − steht wird als Rumpf bezeichnet. Der Rumpf einer Klausel kann leer sein. Eine Klausel mit leerem Rumpf repräsentiert Fakten, die Klausel darf in diesem Fall keine Variablen enthalten. Klauseln mit mindestens einem Literal im Rumpf repräsentieren Regeln.

Beispiel

Fakt: „John ist der Vater von Harry“
Datalog: father(harry, john).
Regel: „Wenn X ein Elternteil von Y ist und wenn Y ein Elternteil von Z ist, dann ist X ein Großelternteil zu Z“
Datalog: grandpar(Z,X) :- par(Y,X), par(Z,Y).
par und **grandpar** sind Prädikatsymbole, **john** und **harry** sind Konstanten und **X,Y,Z** sind Variablen.

Konstanten und Prädikatsymbole beginnen in Datalog mit einem kleinen Buchstaben, Variablen mit einem Großen. Anders als in der Hornlogik, müssen in Datalog alle Literale mit dem selben Prädikatsymbol dieselbe Stelligkeit besitzen, also dieselbe Anzahl an Argumenten. Literale, Fakten, Regeln oder Klauseln, die keine Variable enthalten, werden als „ground“ bezeichnet. Ein Datalog-Programm P muss folgende Sicherheitsbedingungen erfüllen:

- Jeder Fakt von P ist ground
- Jede Variable, die im Kopf einer Regel vorkommt, muss auch im Rumpf der selben Regel vorkommen

Diese Bedingungen garantieren, dass die Menge an Fakten, die von einem Datalogprogramm abgeleitet werden können, endlich sind.

2.4.2. SemReasoner

Der SemReasoner ist eine deduktive Datenbank und wird von der Firma semedy GmbH[8] entwickelt. Es handelt sich um den Nachfolger des OntoBrokers [Ang14]. Auf der Grundlage der Horn-Logik und OO-Logic zur Definition der Regeln, bietet der SemReasoner eine skalierbare und performante deduktive Datenbank. OO-Logic ist der Nachfolger von F-Logic [KLW95, KL89]. Der Vorteil bei der Verwendung von Regeln ist die Trennung von Wissen, Geschäftslogik und der Ausführungslogik. Regeln extrahieren die Logik aus dem eigentlichen Programm. Zusätzlich sind Regeln leichter lesbar und können somit auch von einem Domänenexperten verwaltet werden. SemReasoner kombiniert einen Graphspeicher mit einem deduktiven Schlussfolgerungsmodul. Der Kern enthält eine Speicherschicht in der Ontologien und Ontologieinstanzen gespeichert werden. Die Daten können entweder persistent (Speicherung auf der Festplatte) oder im Arbeitsspeicher gehalten werden. Die persistente

[8] `http://www.semedy.com`

Konfiguration ist als Graphspeicher implementiert. Das deduktive Schlussfolgerungsmodul verarbeitet Logikprogramme. Diese Logikprogramme bestehen aus Horn-Logik, welche durch nicht monotone Negationen erweitert wird. Bottom-Up- und Magic-Set-Auswertung sind die zwei Schlussfolgerungsmethoden des SemReasoners. Die Verarbeitung von unten nach oben (bottom-up) nimmt die in der Datenbank gegebenen Fakten, wendet die Regeln an und erzeugt die abgeleiteten Fakten. Anschließend werden die Regeln erneut angewendet. Dies geschieht so lange, bis keine neuen Fakten mehr abgeleitet werden können. Ein Nachteil bei diesem Verfahren sind die vielen Zwischenergebnisse, teilweise werden diese nicht zur Beantwortung der eigentlichen Anfrage benötigt. Die zweite Ausführungsmethode verwendet die Magic-Set-Auswertung. Bei dieser Methode werden veränderte Regeln erzeugt, die weniger Zwischenergebnisse produzieren. Anschließend wird die Regelmenge von unten nach oben ausgewertet wodurch die nicht benötigten Zwischenergebnisse reduziert werden. Es kann allerdings auch passieren, dass der zusätzliche Aufwand größer ist, als der Performanzgewinn. Es gilt zwischen dem Reduktionseffekt und dem Verlust der Performanz, durch die Erzeugung der zusätzlichen Regeln, abzuwägen.

2.5. Computerlinguistik

Innerhalb dieses Kapitels werden drei Verfahren der Computerlinguistik erläutert. Eine kurze Beschreibung des Bag-of-Words-Mechanismus bildet das erste Unterkapitel. Anschließend wird ein Maß zur Berechnung von Gewichten eines Termes vorgestellt. Das dritte Unterkapitel beschreibt die Funktionsweise eines Part-of-Speech-Taggers.

2.5.1. Bag-of-Words

Der Bag-of-Words-Mechanismus [MRS08] wird in der Verarbeitung natürlicher Sprache und der Informationsgewinnung verwendet. Es geht darum, eine leichtgewichtige Darstellung des Dokuments mit Hilfe der absoluten Häufigkeit des Auftretens von Wörtern zu erstellen. Die exakte Reihenfolge der Terme innerhalb der Dokumente spielt hierbei keine Rolle. Einzig die Anzahl der Vorkommen wird benötigt und auch nur diese gespeichert. Wenn nur die Anzahl der Vorkommen eines Wortes beachtet werden, dann ist zum Beispiel „Mary is quicker than John“ identisch zum Dokument „John is quicker than Mary“.

Beispiel

Satz: „Mary is quicker than John“
Bag-of-Words-Repräsentation:

$$\begin{pmatrix} Mary \\ is \\ quicker \\ than \\ John \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.1}$$

Zweiter Satz: „John is quicker than Mary“
Bag of Word Repräsentation:

$$\begin{pmatrix} John \\ is \\ quicker \\ than \\ Mary \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.2}$$

> Wenn nun die Wortvektoren des ersten und des zweiten Satzes verglichen werden, so sind die beiden Sätze vom Inhalt identisch.

Intuitiv kann daraus gefolgert werden, dass zwei Dokumente mit ähnlichen Bag-of-Words-Repräsentationen auch ähnlich im Inhalt sind. Allerdings sind nicht alle Wörter in einem Dokument gleich wichtig. Es existieren zudem Wörter, welche nicht indiziert werden sollen. Diese müssen während der Extraktion der Bag-of-Words-Informationen herausgefiltert werden. Um die Gewichtung der Wörter anzupassen, wird im nächsten Kapitel ein mögliches Gewichtungsschema (TF-IDF) vorgestellt.

2.5.2. TF-IDF

TF-IDF[MRS08] ist ein Gewichtungsschema für das Auftreten von Termen in einem Dokument. Das Gewichtungsschema besteht aus zwei Teilen, der Termhäufigkeit (tf) und der inversen Dokumentenhäufigkeit (idf).
Für die Termhäufigkeit wird jedem Term in einem Dokument ein Gewicht zugeordnet. Dieses Gewicht ist abhängig von der Anzahl der Vorkommen des Terms im Dokument. Eine einfache Möglichkeit dieses Gewicht zu bestimmen ist, die Anzahl der Vorkommen als Gewicht für den Term zu verwenden. Bezeichnet wird die Termhäufigkeit mit $tf_{t,d}$, wobei t der betrachtete Term in Dokument d ist.
Problem der Termhäufigkeit ist, dass manche Terme in allen Dokumenten häufig vorkommen. Folglich sind diese für das einzelne Dokument wenig aussagekräftig. Eine Sammlung von Dokumenten zur Autoindustrie enthält beispielsweise nahezu in jedem Dokument den Term „Auto“. Die Dokumentenhäufigkeit ist ein Mechanismus, um den Effekt von Begriffen abzuschwächen. Terme, die in einer Sammlung sehr häufig vorkommen, bekommen ein niedrigeres Gewicht. Ein Ansatz ist, das Gewicht eines Terms durch einen Faktor zu reduzieren. Dieser Faktor wächst hierbei mit der Häufigkeit des Auftretens in der Sammlung. Die Dokumentenhäufigkeit ist definiert als Anzahl an Dokumenten in der Sammlung, welche einen Term t enthalten. Intuitiv sollen Terme die in wenigen Dokumenten vorkommen einen höheren Wert erhalten, als Terme die in vielen Dokumenten vorkommen. Für TF-IDF wird die inverse Dokumentenhäufigkeit verwendet:

$$idf_t = log \frac{N}{df_t} \tag{2.3}$$

N ist die Gesamtzahl der Dokumente.
Die Formel für Tf-IDF lautet:

$$\text{tf-idf}_{t,d} = tf_{t,d} * idf_t \tag{2.4}$$

Der Wert von TF-IDF ist am höchsten, wenn t häufig in einem Dokument vorkommt und gleichzeitig in allen anderen Dokumenten wenig bis gar nicht. Niedriger wird der Wert, wenn ein Term weniger häufig in einem Dokument oder in vielen Dokumenten vorkommt. Am niedrigsten ist der Wert, wenn der Term nahezu in allen Dokumenten erscheint. Der TF-IDF-Wert ist nicht nur abhängig vom Wort an sich, sondern auch vom konkreten auftreten eines Wortes in einem Dokument.
Eine Gewichtungsmöglichkeit für die Termfrequenz ist die Augmented-Gewichtung (Gleichung 2.5). Längere Dokumente resultieren in höheren Termhäufigkeiten, da sich dieselben Wörter immer wieder wiederholen. Im Anwendungsfall dieser Arbeit werden die Beispielfragen innerhalb eines Intents zu einem Dokument zusammengefasst. Dadurch ergeben sich Wiederholungen von denselben Wörtern. Um dies auszugleichen wird die Augmented-Gewichtung verwendet.

$$0.5 + \frac{0.5 * tf_{t,d}}{max_t(tf_{t,d})} \tag{2.5}$$

Tag	Beschreibung
ADJA	attributives Adjektiv
ADV	Adverb
ART	bestimmter oder unbestimmter Artikel
NN	normalen Nomen
VVFIN	finites Verb, voll

Tabelle 2.1.: Beispiele aus der POS-Tag Tabelle für Deutsch. Quelle: [STS99]

Für diese Gewichtung wird die maximale Häufigkeit eines Terms $t_{f,d}$ bestimmt. Jeder Term wird anschließend mit 0.5 multipliziert und durch diese maximale Häufigkeit dividiert. Um die resultierende Termfrequenz zu erhalten, wird auf das Ergebnis 0.5 dazu addiert.

2.5.3. Part-of-Speech Tagger

Das Identifizieren von Wortarten(Part-of-Speech tagging)[Vou03] hat bereits eine lange Geschichte in der Linguistik. Bereits der antike griechische Grammatiker Thrax unterschied zwischen acht Wortklassen. Die Wortklassen gliederten sich in Nomen, Verben, Mittelwörter, Artikel (inklusive Relativpronomen), Pronomen, Präpositionen, Adverben und Konjunktionen. Jede natürliche Sprache besteht aus unterschiedliche Wortarten (Verben, Nomen, usw.). Um einem Wort eine Wortart zuweisen zu können, muss die Grammatik beachtet werden. Beispielsweise syntaktische Verteilungen, syntaktische Funktionen oder morphologische und syntaktische Klassen, denen verschiedene Wortarten zugeordnet werden können. Etikettieren (Tagging) bedeutet automatisches Zuweisen von Etiketten zu Wörtern. Jedes Wort wird hierbei durch einen Tag, je nach Wortart, gekennzeichnet.

Tabelle 2.1 zeigt einige Beispiele aus der deutschen Tabelle für die Wortarten-Tags. Ein Werkzeug was das Etikettieren von Worten erlaubt, ist der Part-of-Speech-Tagger von OpenNLP[9]. Dieser etikettiert ein Wort mit dem entsprechenden Worttyp basierend auf dem Wort selber und dem Kontext des Wortes. Der OpenNLP-Etikettierer verwendet ein Wahrscheinlichkeitsmodell, um das korrekte Etikett vorherzusagen. Um mögliche Etiketten für ein Wort zu begrenzen, kann ein Etikett-Wörterbuch verwendet werden.

[9] `opennlp.apache.org`

3. Verwandte Arbeiten

Dieses Kapitel betrachtet Arbeiten, die ähnlich zur Zielsetzung dieser Arbeit sind. Es werden vier unterschiedliche Bereiche betrachtet. Das erste Unterkapitel(Abschnitt 3.1) beschäftigt sich mit der Extraktion von Ontologien aus Texten. Anschließend werden zwei Extraktionsverfahren für Ontologien aus relationalen Datenbanken (Abschnitt 3.2) vorgestellt. Natürlichsprachliche Anfragen an eine Datenbank bilden ein eigenes Forschungsgebiet (Natural Language Interfaces to Databases (NLIDB)). Zwei ausgewählte Verfahren werden im dritten Unterkapitel (Abschnitt 3.3) vorgestellt. Den Abschluss des Kapitels bildet die Active-Rahmenarchitektur (Abschnitt 3.4). Auf dieser Rahmenarchitektur basiert die in dieser Arbeit verwendete EASIER-Rahmenarchitektur.

3.1. Textbasierte Ontologieextraktion

Ein Ansatz zur Erzeugung einer Ontologie aus bestehendem Wissen ist die Extraktion aus Texten. Maedche et al. [MV01, MS01] definieren eine Rahmenarchitektur zum Extrahieren einer Ontologie. Das Verfahren besteht aus mehreren Schritten, dem Ontologie-Import, -Extraktion, -Reduzierung, -Verfeinerung und -Bewertung. Des Weiteren wird ein Werkzeug zur Ontologiemodellierung bereitgestellt. Eine Ontologie kann aus einem Freitext, aus Wörterbüchern oder vererbten Ontologien extrahiert werden. Zur Extraktion einer Ontologie sind folgende Schritte notwendig: Zunächst werden existierende Strukturen verschmolzen oder Abbildungsregeln definiert, um bestehende Ontologien zu importieren und wieder zu verwenden. Anschließend werden mittels der Ontologieextraktion, das heißt Extraktion der Informationen aus Freitexten, wichtige Teile der Zielontologie modelliert. Nach der Extraktion wird die Ontologie reduziert, das heißt nicht benötigte Elemente werden entfernt, um die Ontologie besser auf den Anwendungsfall anzupassen. Abschließend wird die Ontologie verfeinert. Dieser Zyklus kann wiederholt werden, um neue Domänen zu erschließen.
Zu Beginn des Verfahrens werden bestehende Ontologien importiert und zusammengeführt, um eine einheitliche Basis zu bilden. Auf dieser einheitlichen Basis setzen die nachfolgenden Phasen auf.
Nach dem Import bestehender Ontologien beginnt der eigentliche Extraktionsalgorithmus. Zur Extraktion werden verschiedene Verfahren angewandt. Aus den zur Verfügung stehenden Webdokumenten werden lexikalische Einträge erzeugt und Konzepte extrahiert. Dazu wird der Eingabetext vorverarbeitet und anschließend werden mittels Termextraktionstechniken neue lexikalische Einträge vorgeschlagen. Der Ontologieentwickler muss an-

schließend entscheiden, ob er dem Vorschlag der Rahmenarchitektur folgt. Wird der Vorschlag akzeptiert, so wird ein neuer lexikalischer Eintrag in der Ontologie hinzugefügt. Dieser Eintrag ist nicht mit einem Konzept verknüpft. Der Ontologieentwickler muss daher entscheiden, ob ein neues Konzept eingeführt werden soll, oder ob es mit einem bestehenden Konzept verknüpft wird. Auf die Erzeugung der Konzepte folgt das hierarchische Konzept-Clustering. Beim hierarchischen Clustering werden Ähnlichkeiten von Elementen ausgenutzt, um eine Hierarchie von Elementkategorien vorzuschlagen. Dazu wird ein Ähnlichkeitsmaß aus den Eigenschaften von Objekten berechnet. Term-Adjazenzen oder syntaktische Beziehungen zwischen Termen werden benötigt, um eine semantische Hierarchie von Konzepten einzuführen. Anschließend werden Assoziationsregeln erzeugt. Dies dient dazu, Beziehungen zwischen Konzepten zu finden. Die gegebene Klassenhierarchie dient hierbei als Hintergrundwissen. Ein allgemeines Problem bei der Modellierung von Ontologien in unterschiedlichen Domänen ist, dass häufig nicht genügend Domänenmodelle zur Verfügung stehen, um ein vollständiges Modell erzeugen zu können.
Nach der Extraktion wird die Ontologie reduziert, das heißt nicht benötigte Elemente werden gelöscht. Als erstes müssen die Auswirkungen der Beschneidung einzelner Teile auf den Rest der Ontologie bekannt sein. Zweitens werden Strategien für das Vorschlagen von Elementen, die bestehen bleiben oder entfernt werden sollen, benötigt.
Abschließend muss die Ontologie verfeinert werden. Mittels Daten aus Semantic-Web-Anwendungen, beispielsweise Protokolldateien von Benutzeranfragen, wird die Ontologie verfeinert. Hierfür werden entsprechend Konzepte und Beziehungen, welche aus den Daten extrahiert werden können, zur Ontologie hinzugefügt.

Extraktion von Beziehungen an einem Beispielsatz

Bei der Vorverarbeitung des entsprechenden Textes wird festgestellt, dass das Konzept „TARIFF“ häufig im Zusammenhang mit den Konzepten „MOBILE CALL“, „CITY CALL“ und „INTERNATIONAL CALL“ vorkommt. Auf der Grundlage dieser Vorverarbeitung schlägt der Algorithmus vor, eine neue Beziehung „HAS TARIFF(CALL, TARIFF)“ zwischen den Konzepten „CALL“ und „TARIFF“ einzuführen.
(Beispiel übernommen aus [MV01])

Das Verfahren von Dahab et al. [DHR08] basiert auf semantischen Mustern zur Extraktion von Ontologien aus Text. TextOntoEx ist die Verbindung zwischen linguistischer Analyse und Ontologiekonstruktion. Texte zu Fachgebieten können analysiert werden, um nicht-taxonomische Beziehungen einer bestimmten Domäne zu extrahieren. Beispielsweise kann eine Beziehung zwischen zwei Konzepten durch ein Verb ausgedrückt werden. Semantische Muster sind die generische formale Repräsentation von natürlichen Textfragmenten. Die Struktur setzt sich zusammen aus einer abstrakten Klassen, der Verbgruppe, einem konstanten Textausdruck (Präpositionen und Konjunktionen) und optionalen Elementen. Bei allen Elementen handelt es sich um nicht-terminierende Elemente, bis auf die konstanten Textausdrücken. Mittels dieser semantischen Muster können sowohl nicht-taxonomische, als auch taxonomische Relationen gefunden werden. Allerdings müssen diese Muster zunächst manuell erzeugt werden.
Zur Extraktion der Ontologie aus dem domänenspezifischen natürlichsprachlichen Text wird dieser komplett analysiert. Dazu werden die Absätze einzeln betrachtet und jeder Satz wird mit den semantischen Mustern verglichen. Bei Übereinstimmungen werden die entsprechenden Konzepte und Beziehungen extrahiert und zur Ontologie hinzugefügt. Ein Satz kann mit mehreren semantischen Mustern übereinstimmen. In diesem Fall liefert jedes Muster einen Beitrag zur Ontologie.

RDB component	OWL component
Table	Class
Column	Functional property
Row	OWL individual
Column Metadata:	OWL Property restriction:
Data type	AllValuesFrom Restriction
Mandatory / Not nullable	Cardinality() Restriction
Nullable	maxCardinality() Restriction

Tabelle 3.1.: Verbindung zwischen RDB-Komponenten und OWL-Komponenten. (Übernommen aus [SHA11])

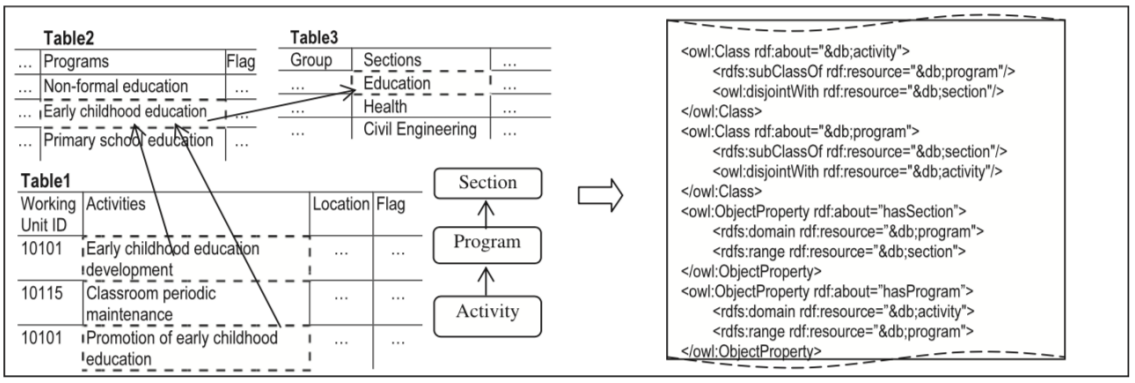


Abbildung 3.1.: Beispiel für die Umwandlung der Informationen aus der RDB zur OWL Ontologie. (Übernommen aus [SHA11])

Semantisches Muster

Ein Beispiel für ein einfaches Muster ist das Muster „<Plant Part> <Becomes.Verb> <Color>“. Der Satz „Leaf turns to yellow“ erfüllt dieses semantische Muster. Das Muster beschreibt die Konzepte „Plant“ und „Color“. Die Beziehung zwischen den beiden Konzepten wird durch das Verb „Becomes“ ausgedrückt.
(Beispiel übernommen aus [DHR08])

3.2. Ontologieextraktion aus einer relationalen Datenbank

Neben der Extraktion von Ontologien aus Text gibt es Ansätze, die aus relationalen Datenbanken Ontologien extrahieren. Das Verfahren von Santoso et al. [SHA11] ist ein konkretes Verfahren. Als Grundlage der Datenextraktion wird eine bereits bestehende Konzepthierarchie verwendet. Konzepthierarchien werden von Domänenexperten bereitgestellt. Mittels dieses Hintergrundwissens ermittelt der Algorithmus über die Fremdschlüsselverbindung Verknüpfungsketten mit anderen Tabellen. Bei schwierigen Entscheidungen ist die Interaktion des Benutzers gefragt. Daher bietet dieses Verfahren nur eine semi-automatische Ontologiekonstruktion.

Tabelle 3.1 zeigt die Beziehung zwischen den Komponenten einer relationalen Datenbank und den Komponenten einer OWL-Ontologie. Diese Definition erleichtert die Erzeugung der OWL-Ontologie erheblich. Anschließend muss der Algorithmus die Beziehungen zwischen den Klassen bestimmen.

Abbildung 3.1 zeigt die beispielhafte Abbildung einer relationalen Datenbank in eine OWL-Ontologie. Die *Tabelle 1* beschreibt in diesem Beispiel die Aktivitäten. Diese sind nach der

Relation	Primary Key	Foreign Key
Staff (LectID int, Room int)	LectID	N
Staff-Default (LectID int, Address string)	LectID	LectID referring to LectID in Staff

Tabelle 3.2.: Schema einer relationalen Datenbank. (Auszug aus [LDW05])

Analyse des Algorithmus eine Unterklasse des Programms. Ein Programm wiederum ist eine Unterklasse einer Sektion. Auf der Grundlage der Analyse des Algorithmus wird aus der vorliegenden Datenbankstruktur die danebenstehende OWL-Ontologie erzeugt.

Ein weiteres Verfahren zur Extraktion einer Ontologie aus einer relationalen Datenbank wurde von Li et al. [LDW05] vorgestellt. Unter Verwendung von Lernregeln wird die Ontologie extrahiert ohne ein Zwischenmodell zu verwenden. Die extrahierte Ontologie enthält Klassen, Eigenschaften, Eigenschaftscharakteristiken, Kardinalitäten und Instanzen einer Klasse. Fünf Gruppen an Lernregeln werden definiert. Für jede der Informationen, die die Ontologie (Klassen, Eigenschaften, Eigenschaftscharakteristiken, Kardinalität, Instanzen) enthält, existiert eine Gruppe an Lernregeln.
Die Idee hinter den Regeln ist, Informationen aus der Datenbank anhand den Informationen des Schemas zu extrahieren. Dazu beschreiben die Regeln gewisse Bedingungen (beispielsweise wenn zwei Relationen denselben Primärschlüssel besitzen, beschreiben sie dasselbe Element), die erfüllt sein müssen, um eine Information zu extrahieren. Informationen, die extrahiert werden können, sind beispielsweise Klassen, Eigenschaften und Kardinalitäten. Anhand von Verlinkungen zwischen Relationen und Informationen innerhalb der Relationen werden die Informationen extrahiert.

Mittels der Tabelle 3.2 wird im Folgenden ein einfaches Beispiel für eine Regel zum Zusammenfassen von Relationen zu einer Ontologieklasse gezeigt. Die erste Spalte des Schemas definiert den Relationsnamen und die Attribute. In der zweiten Spalte stehen die Primärschlüssel und die letzte Spalte gibt die Fremdschlüssel und die Referenzrelationen an.

Beispiel

Die Regel zum Zusammenfassen von Relationen zu einer Ontologieklasse besagt, dass Informationen aus mehreren Relationen zusammengefasst werden können, wenn diese dasselbe Element beschreiben. In Tabelle 3.2 besitzen die Relation „Staff“ und „Staff-Default“ denselben Primärschlüssel. Aufgrund desselben Primärschlüssels wird darauf geschlossen, dass beide Relationen dasselbe Element beschreiben. Somit werden diese beiden Relationen zu einer Ontologieklasse zusammengefasst.

3.3. Natural Language Interfaces to Databases

Ein weiteres verwandtes Forschungsfeld sind natürlichsprachliche Anfragen an Datenbanken (engl. Natural Language Interfaces to Databases). Mittels dieser Systeme ist es möglich mit natürlicher Sprache eine Datenbankanfrage zu erzeugen. Die Zielsetzung dieser Arbeit ist, aus natürlicher Sprache mittels Aktiver Ontologien eine Anfrage an eine Datenbank zu erzeugen. Daher ist dieses Forschungsfeld mit dem Ansatz dieser Arbeit verwandt. Einen Überblick über die unterschiedlichen Arten der NLIDB-Systeme liefert das Papier von Pazos et al. [PRGBAL+13]. Im Folgenden werden zwei Vertreter dieser Systeme betrachtet. Zum einen wird ein semantisches Grammatiksystem (Abschnitt 3.3.1) und zum anderen ein System mit Zwischenrepräsentation (Abschnitt 3.3.2) vorgestellt.

3.3.1. Semantische Grammatiksysteme

Eine Art der NLIDB-Systeme bilden die semantischen Grammatiksysteme. Diese verwenden vordefinierte Grammatiken, um die natürlichsprachlichen Anfragen in eine SQL-Anfrage zu überführen. Rao et al. [RAC+10] beschreiben ein solches Verfahren. Die semantische Grammatik besteht aus zwei Teilen: Das Lexikon speichert alle möglichen Worte, die in der Grammatik vorkommen können. Zum einen enthält es terminierende Symbole, welche später in der Grammatik verwendet werden und zum anderen einzelne englische Wörter. Den zweiten Teil bilden die Regeln. Diese kombinieren die terminierenden Symbole im Lexikon, um Phrasen oder Sätze zu erzeugen.
Um die natürlichsprachliche Eingabe in eine SQL-Anfrage umzuwandeln, wird die englische Eingabe zunächst durch die semantische Grammatik zerteilt. Anschließend gleicht ein Postprozessor Tabellen- und Attributnamen ab und verknüpft Tabellen, wenn die Abfrage mehr als eine Tabelle umfasst. Danach kann der Postprozessor die resultierende SQL-Anfrage erzeugen und ausgeben.
Das folgende Beispiel, aus dem Papier [RAC+10] veranschaulicht die zwei Teile der semantischen Grammatik.

Beispiel

Dieses Beispiel zeigt einen Eintrag im Lexikon und eine Regel zur Extraktion der gewünschten Informationen aus einem natürlichsprachlichen Satz.

Eintrag im Lexikon

Ein einfacher Eintrag eines Lexikons sieht folgendermaßen aus: „(customer → customer patron member)“ und „(customers → customers patrons members)“. Die linke Seite dieser Einträge beschreibt ein Symbol, welches in der Grammatik verwendet werden kann. Wenn das Wort auf der linken Seite in der Grammatik verwendet wird, bezieht es sich auf die Wörter auf der rechten Seite des Eintrages.

Regeln

Mittels „(ATT_TAPE_CUSTOMER → borrowed by customer ATT_NUMBER)“ kann der folgende Satz in der semantischen Grammatik repräsentiert werden: „borrowed by member number 22“. Die linke Seite des Pfeils beschreibt ein nicht-terminales Symbol, welches in anderen Regeln verwendet werden kann. Rechts vom Pfeil steht eine Kombination von nicht-terminalen Symbolen (Großbuchstaben) und Einträgen im Lexikon (Kleinbuchstaben). Die Wörter „borrowed“, „by“ und „customer“ sind im Lexikon vorhanden und „ATT_NUMBER“ bezieht sich auf ein numerisches Attribut. Anschließend wird daraus eine SQL-Anfrage erzeugt.

3.3.2. NLIDB-Systeme mit Zwischenrepräsentationen

C-Phrase [MON08] ist ein Vertreter der Verfahren, welche die natürlichsprachliche Anfrage zunächst in eine Zwischenrepräsentation umwandelt, aus dieser Zwischenrepräsentation wird anschließend eine Anfrage an eine Datenbank erzeugt. Die Zwischenrepräsentation bei diesem Ansatz baut auf der Logik der ersten Ordnung auf. Das folgende Beispiel aus [MON08] zeigt die Zwischenrepräsentation.

Beispiel

Zwischenrepräsentation für den Satz „cities of over 100,000 people in the largest area

mid-western state“:
$\{x|City(x) \wedge x.population > 100000 \wedge$
$(\exists y)(State(y) \wedge x.state = y.name \wedge$
$LargestByArea(y, State(y) \wedge y.name \in \{'Indiana', ..., 'Wisconsin'\})\}$

Logik erster Ordnung unterstützt keine Prädikate höherer Ordnung. Die Autoren ergänzen in ihrem Ansatz die Logik um diese Prädikate. Ein solches Prädikat höherer Ordnung ist „LargestByArea“ aus dem vorherigen Beispiel. Das Ergebnis der Umwandlung der Zwischenrepräsentation in eine SQL-Anfrage zeigt das folgende Beispiel:

Beispiel

Ergebnis der Umwandlungen der Zwischenrepräsentation in eine SQL-Anfrage. (Übernommen aus [MON08].)

```
SELECT *
FROM City AS x
WHERE x.population > 100000 AND
        EXISTS (SELECT * FROM
                (SELECT * FROM State AS z
                 WHERE z.name = 'Indiana' OR ... OR z.name = '
                    Wisconsin'
                 ORDER BY area DESC LIMIT 1) AS y
         where x.state ) y.name);
```

Zum Umwandeln der natürlichen Sprache in die Zwischenrepräsentation wird eine Grammatik definiert. Diese bildet die natürlichsprachlichen Wörter auf die einzelnen Teile der Zwischenrepräsentation ab. Das folgende Beispiel zeigt die Grammatik zur Umwandlung in die Zwischenrepräsentation:

Beispiel

Ein Auszug aus der Grammatik zur Umwandlung von natürlicher Sprache in die Zwischenrepräsentation. $HEAD \rightarrow \langle$"$cities$", $\lambda x.City(x)\rangle$
$PRE \rightarrow \langle$"big", $\lambda f.\lambda x.f(x) \wedge City(x) \wedge x.population > 100,000$
Diese beiden Regeln wandeln die Wörter „big“ und „cities“ in die entsprechende Darstellung in der Zwischenrepräsentation um.
(Übernommen aus [MON08].)

3.4. Active

Guzzoni et al. [GBC06, GCB06, GBC07] haben eine Rahmenarchitektur für Aktive Ontologien vorgestellt. Wie bereits in den Grundlagen erläutert, besteht diese Rahmenarchitektur aus einem Active-Server zur Ausführung der Ontologie, einem Editor und einer Konsole. Bei Guzzoni werden die Aktiven Ontologien händisch erzeugt. In den vorgestellten Beispielen beschränken sich die Aktiven Ontologien auf zwei konkrete Einsatzgebiete. Zum einen auf den klinischen Bereich und zum anderen wurde eine Ontologie zur Abfrage von Restaurants und Filmen gebaut. Da beide Aktiven Ontologien per Hand definiert wurden, müssen neue Funktionen ebenfalls manuell eingepflegt werden. Die Ontologien können nicht ohne weiteres auf andere Bereiche angewendet werden. Soll eine der beiden Ontologien auf eine andere Domäne angewendet werden, so müsste die Ontologie entweder komplett neu erzeugt oder dementsprechend händisch angepasst werden.
Eine weitere Rahmenarchitektur zu Aktiven Ontologien ist EASIER [BL16, BL18]. EASIER basiert auf der von Guzzoni et al. vorgestellten Rahmenarchitektur. Allerdings werden bei EASIER die Aktiven Ontologien nicht manuell erzeugt, sondern semi-automatisch.

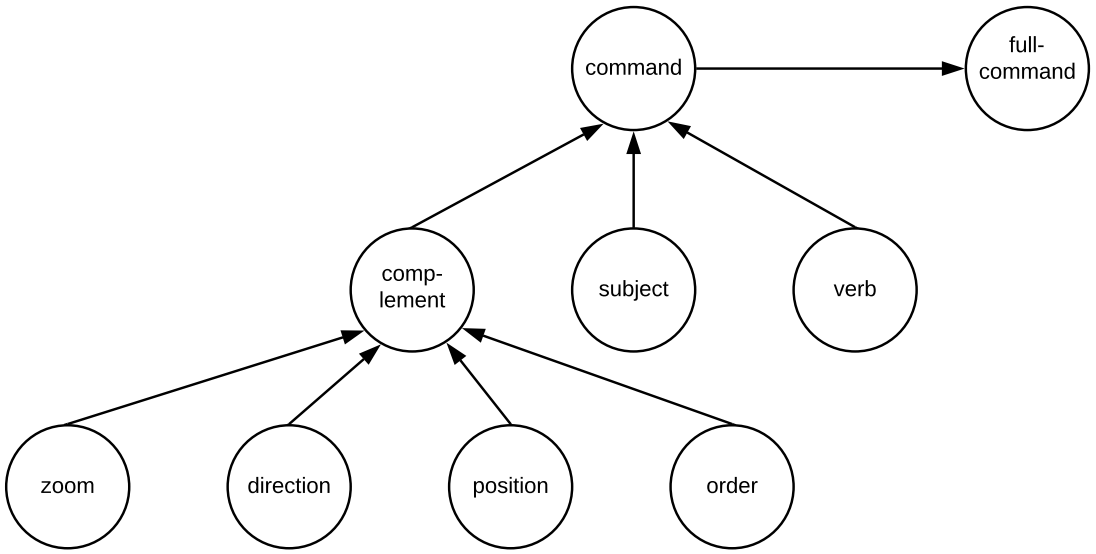


Abbildung 3.2.: Beispiel für eine Ontologie von Guzzoni et al. (Übernommen aus [GCB06])

Dazu identifizieren Web-Crawler formularbasierte Dienste. Anschließend werden die Meta-Informationen der HTML-Formulare dieser Dienste extrahiert. Mittels Spektral-Clustering werden die identifizierten Web-Formulare nach Service-Kategorien kategorisiert. Für jede dieser Servicekategorien wird semi-automatisch eine Aktive Ontologie erzeugt. Gemeinsame Merkmale einer Kategorie resultieren in obligatorischen Knoten. Um diese Ontologie aktiv zu machen, wird für jeden Knoten mittels der Meta-Informationen der Knotentyp bestimmt. Jeder Knotentyp muss mit einer Liste gültiger Werte versehen sein. Die in Formularelementen enthaltenen Informationen werden für Eingabeverifizierungsregeln verwendet. Sollte ein Formularelement keine Informationen zu gültigen Werten oder Datentypen enthalten, so muss der Entwickler die entsprechende Regel manuell formulieren.

In Abbildung 3.2 ist die Ontologie von Guzzoni et al. zur Unterstüzung von Chirurgen dargestellt. Die Ontologie erkennt Befehle um gewisse medizinische Instrumente zu bewegen. Dazu werden aus den natürlichsprachlichen Eingaben Befehle erzeugt, welche die Instrumente ausführen können. Die Struktur der Ontologie ist so aufgebaut, dass diese gültige Befehle modellieren kann. Ein Befehl besteht aus „Subjekt“, „Komplement“ und einem „Verb“. Im Folgenden wird die Funktionsweise der Ontologie mittels der Eingabe „endoscope zoom in“ erläutert. Auf das Wort „endoscope“ reagiert der „subject“-Knoten, „zoom“ ist in diesem Satz das Verb und „in“ wird durch den Komplementknoten „zoom“ erkannt. Jeder Knoten, welcher einen Wert erkannt hat, gibt diesen weiter an den Elternknoten. Im „command“-Knoten wird aus den erkannten Werten der entsprechende Befehl zusammengesetzt.

4. Analyse und Entwurf

Der Titel der Arbeit lautet „Semi-automatische Generierung von Aktiven Ontologien aus Datenbankschemata“. Zunächst werden in diesem Kapitel verschiedene Ansätze zur Erzeugung einer Struktur aus dem Datenbankschema analysiert. Das Ziel einer Aktiven Ontologie ist, aus einer natürlichsprachlichen Eingabe eine Anfrage an eine Datenbank zu erzeugen. Damit eine valide Anfrage erzeugt werden kann, muss eine entsprechende Struktur vorhanden sein. Die Struktur ist schlussendlich für den Aufbau der Anfrage zuständig und muss daher im Voraus sorgfältig modelliert werden. Im Rahmen dieser Arbeit ist das Ziel, die Struktur semi-automatisch aus Datenbankschemata zu erzeugen. Neben der Struktur der Ontologie benötigt eine Aktive Ontologie Sensorknoten, um natürlichsprachliche Konstrukte zu erkennen. Die Befüllung der Sensorknoten geschieht in dieser Arbeit ebenfalls semi-automatisch. Die Analyse geht hierbei darauf ein, wie die Daten aus der Datenbank entsprechend angereichert werden können, um diese anschließend den Sensorknoten zur Verfügung zu stellen. Im ersten Unterkapitel (Abschnitt 4.1) werden verschiedene Ansätze diskutiert und analysiert. Im zweiten Teil dieses Kapitels (Abschnitt 4.2) wird basierend auf der Analyse ein Entwurf zur Lösung des Problem vorgestellt.

4.1. Analyse

Das Ziel der Arbeit ist aus einem Datenbankschema eine Aktive Ontologie (AO) zu erzeugen. Während der Erzeugung der Struktur für die AO müssen diverse Entscheidungen getroffen werden. Abschnitt 2.1.1 beschreibt den Aufbau einer AO und die möglichen Knotentypen. Im Verlaufe des Erzeugungsprozesses muss beispielsweise entschieden werden, ob ein Sammelknoten oder ein Auswahlknoten verwendet werden soll. Eine weitere Entscheidung ist die Wahl der entsprechenden Sensorknoten. Bei der Erzeugung der Sensorknoten gilt es zusätzlich zu bestimmen, ob ein Knoten obligatorisch oder optional ist. Wenn dies nicht aus den Daten bestimmt werden kann, muss dies durch den Entwickler manuell angepasst werden. Des weiteren müssen für die Präfix- und Postfixknoten entsprechende Formulierungen definiert werden. Da diese nicht in der eigentlichen Datenbank gespeichert werden, müssen sie entweder aus gegebenen Texten generiert oder händisch erzeugt werden. Diese Entscheidungen beeinflussen die Zusammensetzung des Befehls an die Datenbank, welcher durch die AO erzeugt wird. Zusätzlich zum Aufbau der Struktur muss entschieden werden, ob viele kleine oder wenige große AOs erzeugt werden sollen. Im Folgenden werden verschiedene Ansätze zur Erzeugung der Struktur skizziert und analysiert. Abschließend werden die Vor- und Nachteile von vielen kleinen gegenüber wenigen großen AOs diskutiert.

4.1.1. Manuelle Erzeugung der Aktiven Ontologie

Zunächst wird ein manueller Ansatz betrachtet. Die AO wird von Hand erzeugt und gewartet, dazu werden die Daten in der Datenbank betrachtet und daraus eine AO erzeugt. Da aus der Struktur die Anfrage an die Datenbank entsteht, muss diese so gewählt werden, dass die entstehende Anfrage zu den Daten in der Datenbank passt. Dazu muss der Entwickler beim modellieren der AO die Struktur der Daten in der Datenbank beachten. Ein Vorteil der manuellen Erzeugung ist, dass der Entwickler für jeden Knoten entscheiden kann, welcher Knotentyp verwendet werden soll. In diesem Fall kann der Entwickler seine Erfahrung dazu nutzen, immer den bestmöglichen Knotentyp zu verwenden. Auch für die Auswahl der Sensorknoten ist der Entwickler verantwortlich. Die erzeugte Aktive Ontologie aus diesem Ansatz ist daher ideal auf die Daten in der Datenbank angepasst. Der Entwickler kann zusätzlich zu den Knotentypen und der Wahl der Sensorknoten definieren, welche Eigenschaften obligatorisch oder optional sind. Genauer bedeutet dies, welche Sensorknoten auf jeden Fall feuern müssen und welche nicht unbedingt feuern müssen.
Wenn eine Anfrage eines Benutzers nicht erkannt oder beantwortet werden kann, so werden die benötigten Informationen händisch zur AO hinzugefügt. Ein Problem des Ansatzes ist, dass dieser sehr viel Vorwissen des Entwicklers benötigt und zudem sehr zeitintensiv ist. Für große Datensätze ist dieser Ansatz aufwändig. Aus dem kompletten Datensatz müssen die Konzepte extrahiert, ihre Verbindungen zueinander definiert und die gültigen Werte für die Sensorknoten eingetragen werden. Das manuelle Erzeugen einer AO ist nur für kleinere Beispieldatensätze mit akzeptablem Aufwand umsetzbar.
Der manuelle Ansatz basiert sehr stark auf der Versuch-und-Irrtum-Methode (engl. trial-and-error approach). Nicht beantwortete Anfragen müssen durch den Entwickler manuell ausgewertet und die entsprechenden Formulierungen zur AO hinzugefügt werden.

4.1.2. Erzeugung aus bestehenden Datenbankanfragen

Eine andere Möglichkeit zur Erzeugung der AO ist eine Variante der manuellen Erzeugung. Hierfür werden die bisherigen Anfragen an eine Datenbank analysiert, um die Formulierungen auf die Anfragen abzustimmen. Auf der Grundlage der bestehenden Anfragen können Daten ausgeschlossen werden, die nie angefragt werden. Dies erleichtert dem Entwickler die Erzeugung der AO, da im ersten Schritt nur die bisher angefragten Daten in der AO modelliert werden müssen. Im folgenden Beispiel wird eine solche Konstellation kurz dargestellt.

Beispiel

Die Datenbank enthält Daten zu den Themen „Auto“, „Flugzeug“, „Fahrrad“ und „Skateboard“. Bei der Analyse der Anfragen ist herausgekommen, dass die Themen „Auto“, „Flugzeug“ und „Fahrrad“ sehr häufig angefragt wurden. In diesem Fall wird die AO für die Daten aus diesen Themen modelliert und die Daten zu „Skateboard“ werden ausgelassen. Diese können in einem späteren Schritt nachgeliefert werden, sobald entsprechende Anfragen vorliegen.

In einem späteren Schritt kann die AO, um die bisher nicht angefragten Daten, erweitert werden. Durch diese vorherige Analyse kann der Arbeitsaufwand reduziert werden.
Die Sensorknoten wurden Anhand der bisherigen Anfragen modelliert. Dies ermöglicht auch die Beantwortung von Anfragen, die ähnlich formuliert sind. Kombinationen aus Präfix, dem eigentlichen Wert und dem Postfix sind bei AOs variabel.
Dieses Verfahren ist eine Vereinfachung der manuellen Variante. Zusätzlich sind die Formulierungen an tatsächlichen Anfragen angelehnt, sodass die Erkennungsrate von Anfragen höher ist, als in der manuellen Variante. Allerdings ist auch bei diesem Ansatz der Modellierungsaufwand durch den Entwickler sehr hoch. Zudem ist problematisch, dass durch

das Aussortieren von nicht angefragten Themen viele Nutzeranfragen nicht beantwortet werden können. Wie auch die manuelle Variante, basiert dieser Ansatz auf der Versuch-und-Irrtum-Methode (engl. trial-and-error approach).

4.1.3. Generelle Aktive Ontologie

Ein weiterer Ansatz basiert auf einer bereitgestellten generellen AO. Im Grundlagenkapitel wurde Schema.org (Abschnitt 2.3) vorgestellt. Schema.org enthält Schemas, mit denen bestimmte Daten beschrieben werden können. Für diese Schemas könnten AOs erzeugt werden. Wenn dann ein solches Schema für die Daten in der Datenbank verwendet wird, kann auf diese AO zurückgegriffen werden. Jeder, der die Schemas von Schema.org verwendet, hätte somit Zugriff auf vordefinierte AOs. Schema.org besitzt Schemas für unterschiedliche Themengebiete, für jedes dieser Themengebiete müsste eine AO angelegt werden. Sind in der Datenbank mehrere unterschiedliche Themengebiete gespeichert, so werden mehrere dieser generellen AOs benötigt. Die Sensorknoten der AO können mit den Daten aus der Datenbank befüllt werden. Präfix und Postfixknoten müssten in diesem Fall händisch erzeugt werden, da die Datenbank keine Informationen zu Formulierungen speichert.
Ein großes Problem bei diesem Ansatz ist, dass aktuell keine generischen AOs existieren. Dies macht es unmöglich diesen Ansatz zu verwenden. Ein weiteres Problem ist, dass die generellen AOs alle Informationen eines Schemas beinhalten. Wird in einer Datenbank nur ein Teilschema benötigt, so muss entweder die entsprechende Teilontologie aus der generellen AO extrahiert werden oder es wird die komplette AO verwendet. Zusätzlich müsste der Entwickler die einzelnen Knoten überprüfen und gegebenenfalls den Knotentyp entsprechend anpassen. Auch müssen die Kardinalitäten entsprechend an die eigenen Bedürfnisse händisch angepasst werden. Ob Eigenschaften obligatorisch oder optional sind, kann nur der Entwickler entscheiden.

4.1.4. Struktur der Datenbank in Aktive Ontologie übertragen

Beim letzten Ansatz wird die Struktur der Datenbank direkt in die AO übertragen. Die Daten in der Datenbank liegen in einer bestimmten Struktur vor, diese kann dazu verwendet werden eine AO zu erzeugen. Eine AO benötigt eine Struktur, da diese für die Zusammensetzung des Befehls an die Datenbank verantwortlich ist. In diesem Ansatz enthält der aus der AO resultierende Befehl bereits die Struktur der Daten. Daraus lässt sich einfach die Anfrage an die Datenbank erzeugen. Der komplexe Teil dieses Ansatzes ist die Extraktion der Struktur aus der Datenbank. Bei relationalen Datenbanken beispielsweise setzt sich das Schema möglicherweise aus unterschiedlichen Tabellen zusammen. Des Weiteren kann es vorkommen, dass die Sprache der Bezeichnungen der Eigenschaften oder Tabellen von der Sprache der Daten abweicht. Im Rahmen der automatischen Überführung der Struktur der Datenbank in die AO kann nicht festgestellt werden, welche Eigenschaften obligatorisch oder optional sind. In diesem Fall muss der Entwicklung diese Einstellungen im Nachhinein anpassen. Wenn das Schema der Datenbank direkt in eine AO überführt wird, enthält diese auch Eigenschaften, die nicht in der AO auftauchen sollen (interne IDs, ...). Daher muss der Entwickler vorher festlegen, welche Eigenschaften nicht exportiert werden sollen. Werden Daten zu einem neuen Themengebiet in die Datenbank eingefügt, so wird nach dem selben Muster eine neue AO erzeugt. Dies hat den Vorteil, dass die bestehende AO nicht aktualisiert werden muss.

4.1.5. Gruppieren der Daten in Aktiven Ontologien

Nachdem der entsprechende Ansatz zur Erzeugung von Aktiven Ontologien gewählt wurde, gilt es zu entscheiden, ob viele kleine oder wenige große Ontologien erzeugt werden sollen. In diesem Abschnitt werden die Vor- und Nachteile beider Varianten diskutiert. Zunächst

wird auf die Variante mit vielen kleinen Ontologien eingegangen und anschließend auf die Variante mit wenig großen Ontologien.
Beispielsweise können für jeden Typ (Restaurant, Autowerkstatt, Kiosk) Ontologien erzeugt werden. Die hierbei entstehenden Ontologien enthalten ausschließlich Knoten zu den Eigenschaften dieser Typen. Ebenfalls enthalten die Sensorknoten nur die Werte zu den Elementen des entsprechenden Typs. Dadurch kann jeder Sensorknoten schneller auswerten, ob in der natürlichsprachlichen Anfrage ein Wort aus der Wortliste enthalten ist. Da bei dieser Variante viele kleine Ontologien erzeugt werden, gibt es insgesamt im ActiveServer (Abschnitt 2.1.2) viele Ontologien und somit auch viele Sensorknoten, die ausgewertet müssen. Dies kann unter Umständen dazu führen, dass die Auswertung einer natürlichsprachlichen Anfrage länger dauert. Bei vielen kleinen Ontologien kann bei der Modellierung unterschieden werden, ob jede AO alle Eigenschaften des Typs modelliert oder nur die Eigenschaften, die für diesen Typ charakteristisch sind. Wenn eine AO nur die Eigenschaften modelliert, die für den Typ charakteristisch sind, so muss eine Verbindung zur AO bestehen, welche die weiteren Eigenschaften des Typ besitzt. Im Folgenden werden die beiden Szenarien anhand eines Beispiels erläutert.

Beispiel

Zunächst werden beispielhaft drei Typen mit ihren Eigenschaften aufgezeigt.
LocalBusiness

```
name
adresse
```

Restaurant

```
name
adresse
öffnungszeiten
```

Pizzeria

```
name
adresse
öffnungszeiten
speisekarte
```

Hierbei ist „Pizzeria“ eine Unterklasse von „Restaurant“ und „Restaurant“ eine Unterklasse von „LocalBusiness“. Es ist zu sehen, dass sich einige Eigenschaften bei allen drei Typen überschneiden. Im Fall von vielen kleinen Ontologien, bei denen jede Eigenschaft modelliert wird, würden alle Eigenschaften in den AOs auftauchen. Hierbei würden die Eigenschaften „name“, „adresse“ und „öffnungszeiten“ dupliziert werden. Im zweiten Fall, es werden nur Eigenschaften erzeugt, die für den Typ charakteristisch sind, werden diese Duplikate nicht erzeugt. Es wird eine AO für den Typ „LocalBusiness“, mit den Eigenschaften „name“, „adresse“, erzeugt. Des Weiteren wird eine AO für „Restaurant“ mit der Eigenschaft „öffnungszeiten“ und eine AO „Pizzeria“ mit der Eigenschaft „speisekarte“ erzeugt. Hierbei müssen in den jeweiligen Ontologien Verlinkungen zur übergeordneten Ontologie bestehen. Des Weiteren müssen die entsprechenden Werte der Typen in die Sensorknoten der richtigen AOs geschrieben werden. Beispielsweise müssen für den Typ „Pizzeria“ die Namen in die Liste der gültigen Werte für den Sensorknoten des Typs „LocalBusiness“ geschrieben werden.

Im ersten Fall existieren viele doppelte Sensorknoten. Der Aufwand diese zu erzeugen ist allerdings relativ gering. Im zweiten Fall ist dieser etwas höher, allerdings existieren anschließend keine doppelten Sensorknoten.

Bei wenigen großen AOs werden die einzelnen Typen unter den allgemeinsten Typen zusammengefasst. Beim vorherigen Beispiel würden die Typen „Restaurant“ und „Pizzeria“ unter dem Typ „LocalBusiness“ zusammengefasst. Der Vorteil dieses Ansatzes ist, dass nur wenige, aber dafür sehr große Ontologien entstehen. Die Liste der gültigen Werte in den Sensorknoten enthalten alle Werte der Subtypen, dadurch wird eventuell die Auswertung der einzelnen Sensorknoten langsamer. Duplikate sind in diesem Fall nicht vorhanden, da alle Eigenschaften in die Struktur des Obertyps verschmolzen werden.

4.2. Entwurf

Dieses Kapitel erläutert den Entwurf für die Umsetzung des Ansatzes dieser Arbeit. Bei der Analyse (Abschnitt 4.1) wurden vier Ansätze zur Erzeugung einer Aktiven Ontologie aus Datenbankschemata erläutert. Diese Arbeit verwendet die Struktur der Daten in der Datenbank zur Erzeugung von Aktiven Ontologien. Das Datenbankschema eignet sich zur Extraktion der Struktur. Wenn die Struktur aus der Datenbank verwendet wird, ist die Erzeugung der Anfrage an die Datenbank am Ende des Verfahrens sehr viel einfacher. Die beiden manuellen Ansätze stellen keine Alternative dar, da der manuelle Aufwand in beiden Fällen zu groß ist. Der letzte Ansatz, die Verwendung von generellen Aktiven Ontologien scheitert daran, dass es für Schema.org (Abschnitt 2.3) keine vordefinierten Aktiven Ontologien gibt.
Zur Umsetzung des gewählten Ansatzes ist das Zusammenspiel verschiedener Komponenten nötig. Zum einen wird eine Datenbank verwendet, in der die Daten gespeichert sind und zum anderen wird ein intelligenter Assistent zur Speicherung der Formulierungen und Synonymen verwendet. Es gilt die beiden Datenquellen zusammenzuführen und daraus eine Aktive Ontologie, mit den entsprechenden Sensorknoten, zu erzeugen. Zunächst wird auf die möglichen Datenbanken (Abschnitt 4.2.1) und die intelligenten Assistenten (Abschnitt 4.2.2) eingegangen. Die Daten aus diesen beiden Systemen werden in ein Zwischenformat (Abschnitt 4.2.3) exportiert. Werte im Zwischenformat, für die keine Synonyme vorliegen, werden mittels eines Wörterbuchs (Abschnitt 4.2.4) um entsprechende Synonyme ergänzt. Der letzte Schritt des Ansatzes ist die Erzeugung der Aktiven Ontologien aus dem Zwischenformat. Dazu werden die im Zwischenformat gespeicherten Strukturen eingelesen und die Aktiven Ontologie anhand dieser Struktur aufgebaut. Dies wird im anschließenden Abschnitt 4.2.5 dieses Kapitels erläutert. Das Erzeugen der Datenbankanfrage mittels der Aktiven Ontologie ist der vorletzte Abschnitt 4.2.6 dieses Kapitels. Der letzte Abschnitt 4.2.7 stellt einen Sonderfall in diesem Verfahren dar. Es wird der Entwurf des, in den Grundlagen vorgestellten Bag-of-Word-Mechanismus (Abschnitt 2.5.1), dargestellt. Im Folgenden wird zunächst eine Übersicht über die Module des Ansatzes gezeigt.

Abbildung 4.1 zeigt eine abstrakte Sicht des Lösungsansatzes. Auf der linken Seite ist oben die Datenbank zu sehen und unterhalb davon der intelligente Assistent. In der Datenbank werden die anzufragenden Daten gespeichert. Der intelligente Assistent enthält die Formulierungen und Synonyme für das Anreichern der Daten. Aus beiden Systemen werden die Daten in ein Zwischenformat exportiert. Dieses Zwischenformat wird im nächsten Schritt zu einem gemeinsamen Zwischenformat zusammengeführt (zu sehen in der Mitte der Grafik). Mittels Wörterbüchern werden die Daten des Zwischenformats angereichert und anschließend in die Aktive Ontologie umgewandelt. Dieser Prozess ist auf der rechten Seite der Grafik zu sehen.

4.2.1. Datenbanken

Zur Speicherung der anzufragenden Daten wird eine Datenbank verwendet. Für den allgemeinen Entwurf dieses Verfahrens ist es nicht wichtig, in welcher Art von Datenbank die Daten gespeichert werden. Es können sowohl relationale, als auch deduktive Datenbanken

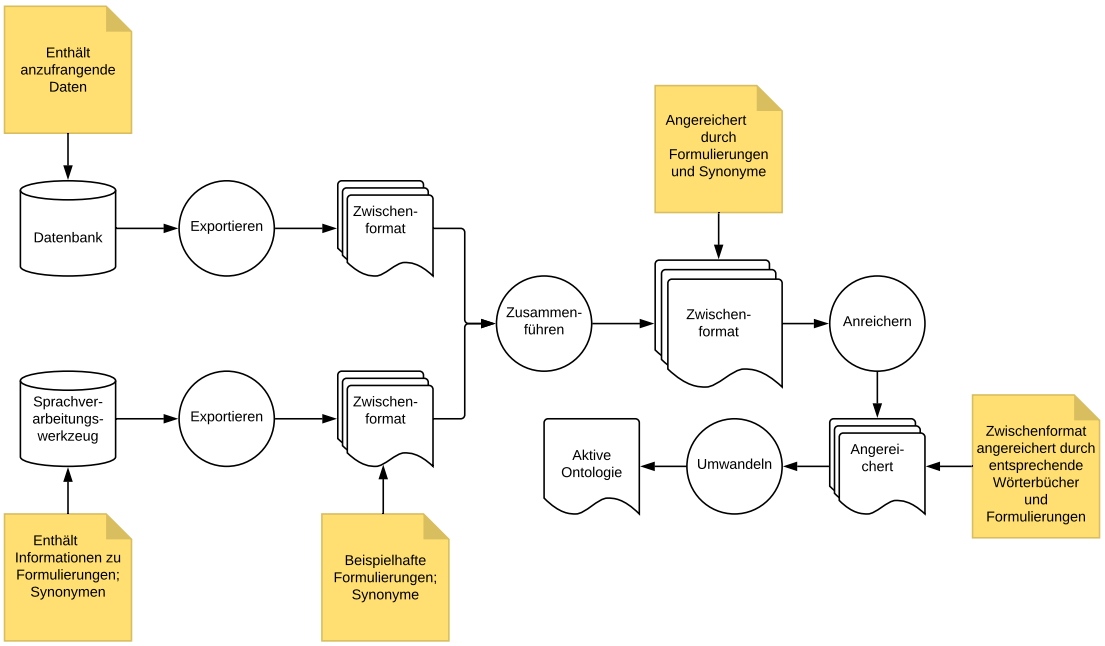


Abbildung 4.1.: Datenflussgraph zum Ansatz dieser Arbeit

verwendet werden. In dieser Arbeit hat die Datenbank zwei Rollen. Zum einen dienen die Daten in der Datenbank zur Erzeugung der Aktiven Ontologien und zum anderen zum Beantworten der Benutzeranfragen. Die aus den Daten erzeugte Aktive Ontologie verarbeitet natürlichsprachliche Anfragen. Eine Anfrage wird durch die Aktive Ontologie in eine Datenbankanfrage umgewandelt. Diese Datenbankabfrage wird anschließend auf der Datenbank ausgeführt, die auch für die Erzeugung der Aktiven Ontologie verwendet wurde. Die Sensorknoten der Aktiven Ontologie reagieren auf Worte aus den Wortlisten der Sensorknoten. Damit Anfragen an die Datenbank erfolgreich umgesetzt werden können, müssen die Sensorknoten mit den Werten aus der Datenbank bestückt sein. Durch die Struktur der Aktiven Ontologie wird eine Anfrage zusammengebaut. In speziellen Fällen ist der Eingriff des Entwicklers nötig, um im Wurzelknoten die erzeugte Anfrage anzupassen.

4.2.2. Intelligente Assistenten

Zusätzlich zu den Daten aus der Datenbank werden Formulierungen und Synonyme zum Anreichern der Sensorknoten der Aktiven Ontologien benötigt. Diese Informationen können aus intelligenten Assistenten, wie Dialogflow oder Watson, exportiert werden. Im Rahmen dieser Arbeit existieren beispielsweise Modellierungen zu den Daten in der Datenbank in Dialogflow. Der Vorteil hierbei ist, dass die Formulierungen und Synonyme auf die Daten in der Datenbank angepasst sind. Daher sind die Informationen für die Anreicherung der Aktiven Ontologie von besonderer Bedeutung. Die Formulierungen werden zur Anreicherung der Sensorknoten mit Prä- und Postfixlisten verwendet. Synonyme werden dazu verwendet, die Werte aus der Datenbank anzureichern.

Beispiel

In der Datenbank gibt es für den Typ eines Elements den Wert „Restaurant“. Dialogflow besitzt für diesen Wert die Synonyme „Lokal“ und „Gaststätte“. Mit den Informationen aus Dialogflow kann der Wert aus der Datenbank angereichert werden, sodass auch andere Worte für den Typ „Restaurant“ verwendet werden können.

Im Folgenden verdeutlicht ein Beispiel die Verbindung zwischen natürlichsprachlicher Eingabe, Informationen aus Dialogflow und den Sensorknoten der Aktiven Ontologie.

Beispiel

Zunächst wird in diesem Beispiel die Benutzeranfrage definiert. Anschließend die Modellierung einer Beispielfrage in Dialogflow. Abschließend werden die Sensorknoten einer Aktiven Ontologie zur Beantwortung der gegebenen Frage dargestellt.

Benutzeranfrage

Die Benutzeranfrage lautet: „Wo gibt es in Karlsruhe eine Post?“

Dialogflow

In Dialogflow sind folgende Fragen modelliert.

```
Wo gibt es in {entityTyp:location} ein {entityTyp:type}.
Wo gibt es in {entityTyp:location} eine {entityTyp:type}.
```

Das Entity zur „location“ enthält die folgenden Einträge:

```
Karlsruhe - Karlsruhe,KA,Karlsruh
```

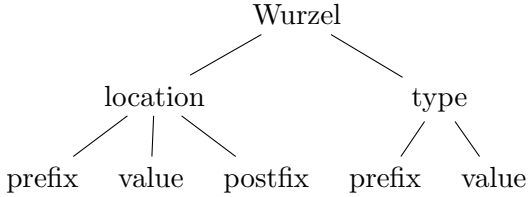
Für das Entity „type“:

```
Post - Postamt,Post
```

Bei der Definition der Entities gibt das Wort vor „-“ den Wert aus der Datenbank an und die Wörter danach definieren die Synonyme.

Aktive Ontologie

Die Aktive Ontologie hat in diesem Fall zwei Sensorknoten für die Datenbankwerte. Zusätzlich für jeden Knoten sowohl einen Prä- als auch einen Postfixknoten.



Für den „value“-Knoten von „location“ besteht die Liste aus Wörtern, die erkannt werden sollen, aus den Wörtern vom Entity-Typ „location“ (siehe Definition oben). Der Präfixknoten enthält beispielsweise die Formulierung, die in Dialogflow vor dem Entity-Typ steht. In diesem Fall enthält der Präfixknoten also „Wo gibt es in“. Als Postfix wird das was, nach dem Entity-Typ steht verwendet, also „ein“, „eine“.

Beim „value“-Knoten von „type“ besteht die Wortliste aus den Werten des Entity-Typs „type“. Der Präfixknoten enthält das Wort „ein“ und „eine“. In diesem Fall gibt es keinen Postfixknoten, da nach dem Entity-Typ „type“, in der Definition in Dialogflow, der Satz endet.

Natürlichsprachliche Anfragen

Wird nun beispielsweise die Anfrage „Wo gibt es in Karlsruhe eine Post?“ gestellt, so reagieren die entsprechenden Sensorknoten auf die Worte im Satz und geben diese weiter. Aufgrund der Informationen aus Dialogflow kann auch folgende Anfrage beantwortet werden: „Wo gibt es in KA ein Postamt?“. Ohne die Informationen aus Dialogflow könnte beispielsweise „Postamt“ nicht auf den Wert „Post“, welcher in der Datenbank steht, abgebildet werden. Der Sensorknoten der Aktiven Ontologie feuert in diesem Fall nur, wenn zusätzlich zum gültigen Wert auch ein Prä- und/oder Postfix gegeben ist.

4.2.3. Zwischenformat

Die Daten aus der Datenbank und dem intelligenten Assistenten werden zunächst in einem Zwischenformat zusammengeführt. Das Zwischenformat besteht aus zwei Teilen. In einem Teil wird die, aus der Datenbank extrahierte, Struktur gespeichert und im anderen die Werte aus der Datenbank, zusammen mit den Formulierungen und Synonymen aus dem intelligenten Assistenten. Aus der exportierten Struktur wird in einem späteren Schritt die Aktive Ontologie erzeugt. Mit den Informationen zu den Formulierungen, Synonymen und Werten werden im selben Schritt die Sensorknoten der Aktiven Ontologie befüllt. In welchem Format die beiden Teile des Zwischenformats gespeichert werden, ist dem Entwickler überlassen. Wichtig ist aber, dass die beiden Informationen voneinander getrennt gespeichert werden. Da die Daten im späteren Schritt für unterschiedliche Zwecke verwendet werden.
Für die Kommunikation der einzelnen Module untereinander ist dieses Zwischenformat erforderlich. Der Zerteiler (Abschnitt 4.2.5) muss aus diesem Format die Aktive Ontologie erzeugen. Zudem liefern nicht alle intelligenten Assistenten dasselbe Format. Um dies auszugleichen wird das Zwischenformat verwendet. Der Export aus den Assistenten muss hierfür entsprechend umgewandelt werden.
Ein weiterer Aspekt für das Zwischenformat ist die Austauschbarkeit von Modulen. Da das definierte Zwischenformat fest ist, kann jedes Modul darauf aufbauen. Soll ein Modul ausgetauscht werden, so kann bei der Implementierung auf das Zwischenformat geachtet werden. Es müssen keine Sonderfälle, wie andere Formate, beachtet werden.

4.2.4. Wörterbücher

Der nächste Schritt (auf der rechten Seite der Grafik Abbildung 4.1) beschreibt das Anreichern des bisherigen Zwischenformats durch Synonyme. Bis zu diesem Schritt haben wir die Struktur und Daten aus der Datenbank, zusammen mit den Informationen aus einem intelligenten Assistenten, in ein Zwischenformat exportiert. Das Zwischenformat besteht aus zwei Teilen, dem Teil für die Struktur und dem Teil für die Werte, Formulierungen und Synonyme. Der erste Teil wird in diesem Modul nicht benötigt. Einzig die Werte, Formulierungen und Synonyme werden verarbeitet.
Dieses Modul wird dazu benötigt, Werte aus der Datenbank, für die keine Synonyme aus dem intelligenten Assistenten extrahiert werden konnten, mit Synonymen zu versehen. Dazu werden bestehende Wörterbücher verwendet. Die bekanntesten sind GermaNet [HF97] im Deutschen oder WordNet [Mil95] im Englischen. Zusätzlich dazu gibt es Wiktionary [1],

[1] `www.wiktionary.org`

ein frei zur Verfügung stehendes Wörterbuch in mehreren Sprachen.
Um nun für ein Wort die Synonyme zu finden, wird in einem Wörterbuch nach dem entsprechenden Wort gesucht. Das Problem hierbei ist, dass wahrscheinlich sehr viele Synonyme für das gesuchte Wort gefunden werden. Dies liegt daran, dass Wörter unterschiedliche Bedeutungen haben können. Daher muss die Bedeutung des aktuellen Wortes möglichst präzise bestimmt werden.
Ein einfacher Ansatz ist, immer den erstbesten Treffer zu verwenden. Hierbei liegt die Hoffnung darauf, dass das erste Suchergebnis möglichst präzise die Bedeutung des gesuchten Wortes wiedergibt. Dieser Ansatz ist risikobehaftet, da es durchaus sein kann, dass der erste Treffer die falsche Bedeutung hat. Daraus werden anschließend falsche Synonyme zum Zwischenformat hinzugefügt. Ein komplexerer Ansatz ist die Suche über die Struktur. Auch Wörterbücher besitzen eine Struktur, beispielsweise Oberwörter, verwandte Wörter oder auch Unterbegriffe. Unter Verwendung der Struktur aus der Datenbank kann versucht werden, diese mit der Struktur aus dem Wörterbuch zu vergleichen. Dazu wird die Struktur der Suchergebnisse für das aktuelle Wort mit der Struktur aus der Datenbank verglichen. Ist die Struktur identisch oder ähnlich genug zu der aus der Datenbank, so wird das entsprechende Ergebnis verwendet. Schlägt die Suche über die Struktur fehl, muss auf den einfachen Ansatz zurückgegriffen werden. Das folgende Beispiel erläutert die Suche über die Struktur.

Beispiel

Es wird ein Wert aus einem Element zu einer Wanderroute betrachtet. Beispielsweise hat eine Wanderroute die folgende Eigenschaft mit einem gegebenen Wert:

```
Schwierigkeitsgrad:einfach
```

Aus dem intelligenten Assistenten konnten für das Wort „einfach“ keine Synonyme extrahiert werden. Also wird mit dem gegebenen Wort im Wörterbuch Wiktionary gesucht. Für das Ergebnis in Wiktionary wird nun die Struktur verglichen. Im Ergebnis ist ein Oberbegriff definiert, dieser enthält das Wort „Schwierigkeitsgrad“. In diesem Fall stimmt die Struktur aus der Datenbank mit der von Wiktionary überein. Es kann also davon ausgegangen werden, dass das Ergebnis der Bedeutung des Wortes aus der Datenbank entspricht. Die Synonyme aus dem Ergebnis können extrahiert und dem Zwischenformat hinzugefügt werden.

Sind die Werte aus der Datenbank in einer anderen Sprache als die, für die Synonyme benötigt werden, so muss zusätzlich zum Wörterbuch ein Übersetzer verwendet werden.

4.2.5. Erzeugen der Aktiven Ontologie

Der letzte Schritt des Verfahrens ist die Erzeugung der Aktiven Ontologie aus dem Zwischenformat. Mittels eines Zerteilers wird aus dem Teil des Zwischenformats, der die Struktur enthält, die Struktur der Aktiven Ontologie erzeugt. Der andere Teil mit Informationen zu Formulierungen, Synonymen und den Werten aus der Datenbank wird zur Erzeugung der Sensorknoten verwendet. Das Beispiel zeigt, wie aus einer beispielhaften, in JSON definierten, Struktur eine Aktive Ontologie erzeugt wird.

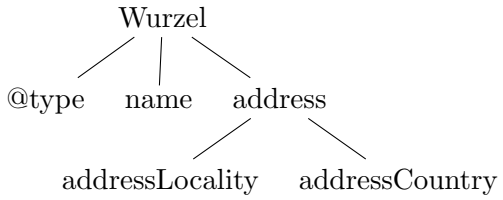
Beispiel

Das JSON für die Struktur aus dem Zwischenformat:

```
{
        "@type":"",
        "name":"",
```

```
        "address":{
                "addressLocality":"",
                "addressCountry":""
        }
}
```

Daraus wird folgende Struktur für die Aktive Ontologie erzeugt.



Die Struktur aus den Daten wird direkt in die Aktive Ontologie übertragen.

Nachdem die Struktur für die Aktive Ontologie erzeugt wurde, werden die Sensorknoten aus dem zweiten Teil des Zwischenformats erzeugt. Um das folgenden Beispiel zu vereinfachen, existieren Formulierungen und Synonyme nur für die Eigenschaft „@type“. Für die anderen Eigenschaften existieren nur Werte aus der Datenbank.

Beispiel

Zunächst werden die Prä- und Postfixinformationen, sowie die Synonym-Informationen zur Eigenschaft „@type“ dargestellt. Anschließend werden kurz die Werte für die restlichen Eigenschaften eingeführt.

@type

Präfixinformationen

```
Wo finde ich ein
Wo gibt es ein
```

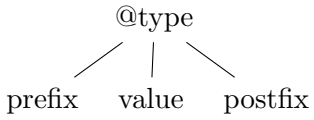
Postfixinformationen

```
mit dem Namen
```

Synonyme

```
Restaurant - Gaststätte, Lokal
```

Aus diesen Informationen werden zur Aktiven Ontologie die entsprechenden Sensorknoten hinzugefügt.



Restliche Eigenschaften

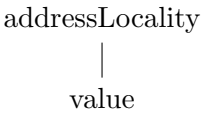
Zunächst werden die Definitionen der Sensorknoten zur Erkennung dargestellt. Anschließend zeigt eine Grafik, welcher Knoten auf Grundlage dieser Informationen zur Aktiven Ontologie hinzugefügt wird. Die „value“-Knoten enthalten die entsprechend definierten Werte.

name

```
Goldener Adler - Goldener Adler, Adler
```

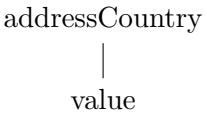
addressLocality

```
Karlsruhe - Karlsruhe, KA
```



addressCountry

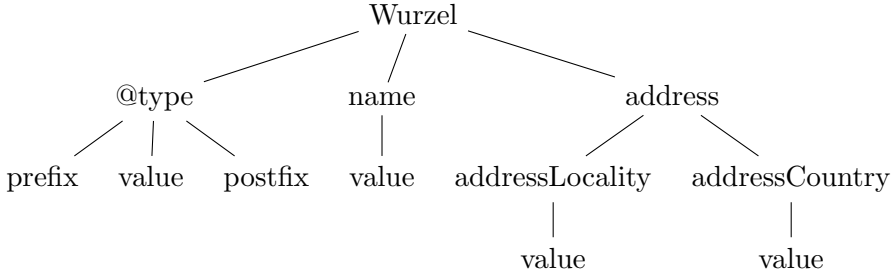
```
Deutschland - Deutschland, DE
```



Nachdem das komplette Zwischenformat verarbeitet wurde, ist die Aktive Ontologie erzeugt. Das Resultat aus beiden Beispielen wird im Folgenden dargestellt.

Beispiel

Ergebnis der beiden Beispiele:



Nach der Erzeugung kann der Entwickler manuell Eigenschaften der Ontologie anpassen. Anschließend ist die Aktive Ontologie bereit für Verarbeitung von natürlichsprachlichen Anfragen.

4.2.6. Erzeugen der Datenbankanfrage

Aus den extrahierten Informationen der Aktiven Ontologie kann eine Anfrage an die Datenbank gestellt werden. In speziellen Fällen ist dazu eine manuelle Veränderung durch

den Entwickler notwendig. Die Struktur der Ontologie sorgt dafür, dass die extrahierten Informationen sich an der Struktur der Daten aus der Datenbank orientieren. In einfachen Fällen kann daraus direkt eine Anfrage erzeugt werden. Das folgende Beispiel zeigt schematisch die Erzeugung einer solchen Anfrage, ohne auf datenbankspezifische Details einzugehen.

Beispiel

Es wird die Ontologie aus dem Beispiel des vorherigen Abschnitt 4.2.5 betrachtet. Für die Eingabe „Wo finde ich ein Restaurant in Karlsruhe“, sehen die durch die Aktive Ontologie extrahierten Informationen wie folgt aus:

```
Parameter1:
        Wurzel.@type:Restaurant

Parameter2:
        Wurzel.address.addressLocality:Karlsruhe
```

Mit diesen Informationen kann nun direkt eine Anfrage an die Datenbank erzeugt werden. Im Fall von Parameter 2 kann beispielsweise folgende Anfrage erzeugt werden: Suche nach allen Elementen, die eine Eigenschaft „address“ haben, welche wiederum eine Eigenschaft „addressLocality“ besitzt. Vergleiche für diese Eigenschaft den Wert aus der Datenbank mit dem gesuchten Wert „Karlsruhe“. Wenn eine Übereinstimmung gefunden wurde, ist das entsprechende Element ein Teil des Ergebnisses.

Solche Anfragen können automatisiert erzeugt werden. Schwieriger wird es, wenn für bestimmte Eigenschaften komplexere Anfragen ausgeführt werden sollen. Beispielsweise soll im Fall von Adressen nicht auf Gleichheit überprüft werden, sondern ob ein gegebenes Wort in der Adresse enthalten ist. Für diese Fälle muss der Entwickler die Erzeugung der Datenbankanfrage entsprechend anpassen.

4.2.7. Bag-of-Words-Mechanismus

Zur Beantwortung ganz spezieller Anfragen wird ein Bag-of-Words-Mechanismus (Abschnitt 2.5.1) verwendet. Dieser Mechanismus beinhaltet die Wichtigkeit von bestimmten Worten in einem gewissen Kontext. In einem Beispiel wird kurz veranschaulicht, wofür dieser Mechanismus benötigt wird.

Beispiel

Es bestehen beispielsweise spezifische Fragen, um nach den Öffnungszeiten von etwas zu fragen.

```
Wann hat das Restaurant geöffnet?
Wie sind die Öffnungszeiten vom Restaurant?
Hat das Restaurant am Donnerstag geöffnet?
```

Aus diesen Fragen werden für den Bag-of-Word-Mechanismus die signifikanten Wörter bestimmt. In diesem Fall zum Beispiel:

```
wann
hat
geöffnet
Öffnungszeiten
```

Wenn nun beispielsweise die Frage gestellt wird „Wann hat das Restaurant geöffnet“,

so erkennt dieser Mechanismus die signifikanten Wörter. Daraus kann geschlossen werden, dass der Benutzer nach den Öffnungszeiten fragen möchte.

Die signifikanten Wörter haben allerdings nicht alle die gleiche Gewichtung. Beispielsweise kann das Wort „wann“ signifikanter als das Wort „hat“ sein. Daher bekommen all diese Wörter ein Gewicht, entsprechend der Signifikanz. Dieses Gewicht wird mittels TF-IDF (Abschnitt 2.5.2) berechnet. TF-IDF verknüpft die Termhäufigkeit mit der inversen Dokumentenhäufigkeit.
Bei den Beispielfragen, die zur Erzeugung des Bag-of-Word-Mechanismus verwendet werden, handelt es sich um sehr kurze Fragen. Für natürlichsprachliche Anfragen werden meist keine langen Texte geschrieben. Daher wird für die Gewichtung der Faktoren der TF-IDF-Formel eine Gewichtung benötigt, die ideal für kurze Dokumente ist. Ein Dokument ist im Rahmen dieser Arbeit eine Gruppe von Fragen zu einem Themengebiet. Beispielsweise bilden alle Fragen zu Unternehmen eine Gruppe. Für diesen Fall eignet sich das „augmented“-Gewicht:

$$0.5 + \frac{0.5 * tf_{t,d}}{max_t(tf_{t,d})} \tag{4.1}$$

Hierbei wird die gegebene Häufigkeit eines Wortes durch die höchste auftretende Häufigkeit eines, möglicherweise anderen Wortes, geteilt. Das Ergebnis ist ein geglättetes Gewicht für das gegebene Wort. Damit der Bag-of-Word-Mechanismus auslöst, müssen eine entsprechende Menge von Worten kombiniert einen definierten Schwellenwert überschreiten. Wird der Schwellenwert überschritten, so geht der Bag-of-Word-Mechanismus beispielsweise von einer Anfrage nach den Öffnungszeiten aus. Auf diese Weise können ganz gezielt Eigenschaften angefragt werden.

5. Implementierung

Dieses Kapitel beschreibt die Implementierung der einzelnen Module (siehe Abbildung 4.1), die zur Umsetzung des Ansatzes dieser Arbeit nötig sind. Insgesamt wurden fünf Module implementiert. Die Datenbank für die Daten und die Informationen des intelligenten Assistenten bilden jeweils ein separates Modul. Beschrieben wird das Modul zum Export der Daten aus der Datenbank in Abschnitt 5.4. Der Export aus Dialogflow wird im Abschnitt 5.5 erläutert. Das nächste Modul in der Abbildung, ist das zum Zusammenführen der Informationen aus der Datenbank und Dialogflow (Abschnitt 5.6). Werte aus der Datenbank für die keine Synonyme vorliegen werden im Modul Wörterbuch (Abschnitt 5.7) angereichert. Abschließend wird aus den Informationen die Aktive Ontologie erzeugt. Die Implementierung dieses Moduls ist in Abschnitt 5.8 beschrieben.
Zu Beginn des Kapitels wird eine Übersicht über die Schnittstellen (Abschnitt 5.1) zwischen den Modulen dargestellt. Darin wird auf die Abhängigkeiten zwischen den Modulen und die Austauschformate eingegangen. Zusätzlich wurden auch einige Anpassungen an der EASIER-Rahmenarchitektur (Abschnitt 5.2) vorgenommen. Die Anpassungen an der Rahmenarchitektur sind für die Umsetzung der einzelnen Module notwendig, daher werden diese zu Beginn des Kapitels erläutert. Anschließend wird auf die Umsetzung der einzelnen Module genauer eingegangen.

5.1. Übersicht

Die Architektur dieser Arbeit besteht aus einzelnen Modulen. Für jeden Bearbeitungsschritt (siehe Abschnitt 4.2) existiert ein eigenes Projekt, welches für genau eine Aufgabe implementiert wurde. Dies ermöglicht es im Nachhinein neue Module zu entwickeln und Bestehende zu ersetzen oder die gesamte Architektur zu erweitern.

Abbildung 5.1 stellt die Schnittstellen zwischen den einzelnen Modulen dar. Beim Entwurf der Schnittstellen wurde darauf geachtet, dass Standardtypen zum Austausch der Daten verwendet werden. Durch die Verwendung von Standardtypen können die Abhängigkeiten zwischen Projekten minimiert werden. Einzig zwischen dem `DialogflowDownloader` und dem `MergeFormats`-Modul besteht eine Abhängigkeit zum Austausch der Sprachinformationen. Die Informationen, die aus Dialogflow exportiert werden, können nicht mittels eines Standardtyps übertragen werden. Dies liegt daran, dass sowohl die Formulierungen, als auch die Synonyme über diese Schnittstelle übertragen werden müssen. Um die Schnittstelle möglichst einfach zu halten und die Daten nicht miteinander zu vermischen, ist ein spezieller Typ notwendig. Dieser speichert die Informationen zu Formulierungen und

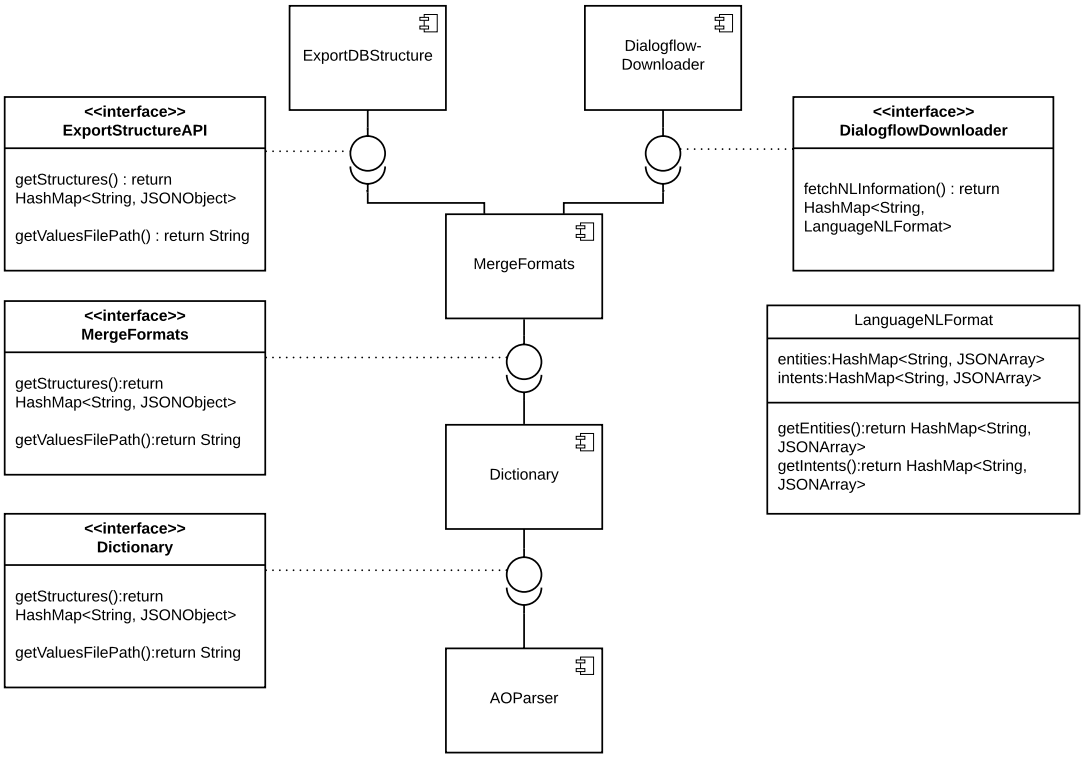


Abbildung 5.1.: Schnittstellen zwischen den Modulen

Synonymen getrennt voneinander innerhalb eines Objektes. Dieses Objekt kann nach Abschluss des Moduls über die Schnittstelle weitergegeben werden. Ansonsten wird zwischen den Modulen eine `HashMap` mit den extrahierten Strukturen ausgetauscht. Die Strukturen liegen als `JSONObject` vor, wobei der Schlüssel der jeweilige Typ ist. Zusätzlich existieren sogenannte Wert-Dateien, diese beinhalten die exportierten Werte aus der Datenbank. Um diese an das nächste Modul weiterzugeben, wird der Pfad zu diesen Dateien übergeben. Auf diese Weise kann das nächste Modul auf die exportierten Werte zugreifen. Zur Reduktion des benötigten Arbeitsspeichers werden diese Werte mittels Dateien übertragen.
Das Modul `ExportDBStructure` stellt eine Schnittstelle für das `MergeFormats`-Modul bereit. Mittels dieser Schnittstelle werden die Struktur und Daten aus der Datenbank extrahiert. Ebenfalls für das `MergeFormats`-Modul stellt der `DialogflowDownloader` eine Schnittstelle für die Formulierungen und Synonyme aus Dialogflow zur Verfügung. Innerhalb des `MergeFormats`-Moduls werden die Informationen zusammengeführt und anschließend kann das `Dictionary` diese Informationen verwenden. Zwischen dem `Dictionary` und dem `AOParser` existiert eine Schnittstelle zum Austausch des angereicherten Zwischenformat (enthält Werte, Formulierungen, Synonyme und die Struktur). In der Abbildung 5.1 ist dargestellt, wie die einzelnen Schnittstellen angesprochen werden können.

5.2. EASIER-Rahmenarchitektur

Im Folgenden werden die Anpassungen der EASIER-Rahmenarchitektur (siehe Abschnitt 2.1) zur Umsetzung des Verfahrens dieser Arbeit beschrieben. Für die Umsetzung waren zum einen die Einführung von zwei neuen Sensorknoten, sowie Anpassungen an bestehenden Knoten notwendig. Bei den bestehenden Sensorknoten wurde eingeführt, dass diese die Werte aus einer Datei auslesen können. Des Weiteren wurde für die inneren Knoten die Weitergabe des Parameternamens angepasst. Zunächst werden in diesem Abschnitt die beiden neuen Knotentypen erklärt und anschließend die Anpassungen genannt.

5.2.1. Synonymknoten

Um die Werte in der Datenbank mit Synonymen anzureichern, bietet Dialogflow die so genannten „Entity-Typen“. Die dazugehörigen Entitäten besitzen zum einen den Wert und zum anderen Synonyme. Der Wert einer Entität wird später für die Anfrage an die Datenbank verwendet. Erkennt Dialogflow eines der gegebenen Synonyme, so wird dies durch den entsprechenden Wert der Entität ersetzt und in einem Intent an die Anwendung gesendet. Um diese Funktionalität aus Dialogflow auf die Aktive Ontologie abzubilden, wird ein Synonymknoten verwendet. Dieser Knoten enthält als Wortliste eine Abbildungsliste, welche Synonyme und ihre Werte auf die das spezielle Synonym abgebildet werden soll. Erkennt der Sensorknoten ein Synonym, so wird nicht der Wert des Synonyms weitergegeben, stattdessen wird das Synonym durch den Wert (aus der Datenbank) ersetzt.
Auch ohne diese Anpassung unterstützen die Aktiven Ontologien eine Synonymersetzung. Diese basiert allerdings auf der Benamung des Knotens. Zum Beispiel existiert ein Sensorknoten mit dem Namen „Restaurant“ und einer Wortliste bestehend aus dem Synonym „Gaststätte“. Wird nun das Wort „Gaststätte“ erkannt, wird der Name des Knotens „Restaurant“ und der erkannte Wert „Gaststätte“ weitergegeben. Bei dieser Art der Implementierung wird für jeden Wert in der Datenbank ein Sensorknoten mit dessen Synonymen benötigt. Für die neu eingeführten Synonymknoten, wird diese Synonymersetzung für alle Werte der Datenbank in einem Knoten durchgeführt. Dadurch muss für jede Eigenschaft nur ein Synonymknoten erzeugt werden, anderenfalls müsste für jeden Wert in der Datenbank ein Sensorknoten erzeugt werden.
Ohne diese Ersetzung liefert eine Suche in der Datenbank kein Ergebnis.
Zur Umsetzung dieses neuen Knotentyps wurde zum einen eine neue Klasse für den speziellen Knotentyp angelegt und zum anderen eine neue Aktion eingeführt. Das besondere an der Klasse des Knotentyps ist das Einlesen der Abbildungslisten. Es existieren bereits Sensorknoten, diese lesen allerdings nur einzelne Wörter aus den Dateien aus und keine Abbildungen.

Abbildungsliste

Dieses Beispiel zeigt eine Abbildungsliste für den Datenbankwert „Restaurant“. Links vom Pfeil stehen die Synonyme und rechts vom Pfeil der Wert auf den das Synonym abgebildet wird.

```
Restaurant
Klassifiziertes Gourmetrestaurant→Restaurant
Gourmetrestaurant→Restaurant
Gaststätte→Restaurant
Gaststätten→Restaurant
Gasthaus→Restaurant
Gasthäuser→Restaurant
Speiselokal→Restaurant
Speiselokale→Restaurant
```

Bis auf das Einlesen der Wortlisten ist die neue Klasse identisch zur Klasse, des bereits bestehenden Sensorknotens.
Besonderheit der neu angelegten Aktion ist die Ersetzung der Synonyme. Der bestehende Sensorknoten bekommt eine Liste an Wörtern und sobald eines dieser Wörter erkannt wird, wird dieses an den Elternknoten weitergegeben. In der neuen Aktion werden die Synonyme als Wortliste verwendet, um zu erkennen, ob der Sensorknoten feuern muss. Erkennt der Sensorknoten ein Synonym, so wird nicht das Synonym weitergegeben, sondern der entsprechende Wert. Dazu verwendet die Aktion die eingelesene Abbildungsliste.

5.2.2. Bag-of-Words-Mechanismus

Zusätzlich zum Abbildungsknoten wurde ein Bag-of-Words-Knoten implementiert. Dieser wird benötigt um zu erkennen, wenn ein Benutzer gezielt eine bestimmte Eigenschaft (beispielsweise Öffnungszeiten) anfragen möchte. Dazu werden aus Beispielfragen, die auf eine Eigenschaft abzielen (beispielsweise Öffnungszeiten) die signifikanten Wörter extrahiert. Wenn der Bag-of-Words-Mechanismus in einer natürlichsprachlichen Anfrage eine gewisse Anzahl der signifikanten Wörter erkennt, so wird die Information weitergegeben, dass nach einer speziellen Eigenschaft gefragt ist.
Der Mechanismus besteht aus einem nicht-terminalen Knoten und mehreren Sensorknoten. Die Sensorknoten enthalten die Wörter und eine Gewichtung, welche die Signifikanz des Wortes ausdrücken. Für jedes Wort in der Bag-of-Words-Wortliste wird ein eigener Sensorknoten angelegt. Konfidenzen der Wörter werden im Konstruktoraufruf übergeben. Sobald ein Sensorknoten ein Wort erkennt, feuert dieser und gibt das Wort inklusive der Konfidenz an den Elternknoten weiter. Der Elternknoten addiert die Konfidenzen der Kindknoten und sobald ein bestimmter Schwellwert überschritten wurde, feuert der nicht-terminale Knoten. Bei diesem nicht-terminalen Knoten handelt es sich um einen Knoten des Typs `Helper-Node` (Abschnitt 2.1.1). Dies bedeutet, sobald dieser Knoten feuert, gibt der Elternknoten eine Konfidenz von 100(höchster Wert) weiter.
Wenn der Bag-of-Words-Mechanismus signifikante Wörter erkannt hat, wird davon ausgegangen, dass der Benutzer nach einer bestimmten Eigenschaft fragen möchte. In diesem Fall gibt dieser Mechanismus einen speziellen Wert an den Elternknoten weiter. Bei diesem Wert handelt es sich um den Namen der Eigenschaft, die angefragt werden soll und einer `Wildcard` als Wert. Auf diese Weise kann bei der Erzeugung der Datenbankanfrage erkannt werden, dass nach der Benutzer nach dieser gegebenen Eigenschaft fragt.
Wenn die Anfrage des Benutzers signifikante Wörter enthält um beispielsweise nach Öffnungszeiten zu fragen, so erkennt der Bag-of-Words-Mechanismus dies. Ist in derselben Anfrage eine Uhrzeit enthalten, welche durch den Sensorknoten zu den Öffnungszeiten erkannt wird, dann wird diese Uhrzeit und nicht die `Wildcard` weitergegeben. In diesem Fall werden also nicht die Öffnungszeiten ausgegeben, sondern Daten mit der gegebenen Uhrzeit abgefragt. Ist in dieser Anfrage keine Uhrzeit enthalten, so wird davon ausgegangen, dass der Benutzer nach Öffnungszeiten fragen möchte.

Beispiel

Beispielhaft existiert in der Aktiven Ontologie ein Bag-of-Words-Mechanismus für die Adresse. Die Wörter auf die der Bag-of-Words Knoten reagiert sind „Wie, ist, Adresse“. Ebenfalls existiert ein Sensorknoten, welcher die Städte(„Karlsruhe“) der Restaurants beinhaltet.
Im Folgenden werden die beiden möglichen Fälle veranschaulicht. Mögliche Anfrage eines Benutzers: „Wie ist die Adresse des Restaurants Goldener Adler“. Der Sensorknoten zur „Stadt“ reagiert in diesem Fall nicht, da „Karlsruhe“ nicht in der Anfrage vorkommt. Die Wörter „Wie, ist, Adresse“ werden vom Bag-of-Words-Knoten erkannt. Aufaddieren der einzelnen Konfidenzen, der erkannten Wörter überschreitet den vorher definierten Schwellenwert. Resultat ist, dass nur der Bag-of-Words Knoten feuert und nicht der „Stadt“-Sensorknoten. In diesem Fall geht die Aktive Ontologie davon aus, dass der Benutzer ausschließlich an der Adresse des gegebenen Restaurants interessiert ist. Dieses Interesse wird mittels einer Wildcard übermittelt. Beim Umwandeln der extrahierten Informationen in eine Anfrage an die Datenbank, muss anschließend beachtet werden, dass der Benutzer in diesem Fall ausschließlich die Informationen zur Adresse erhalten möchte.
Im anderen Fall stellt ein Benutzer möglicherweise folgende Frage: „Ist die Adresse von

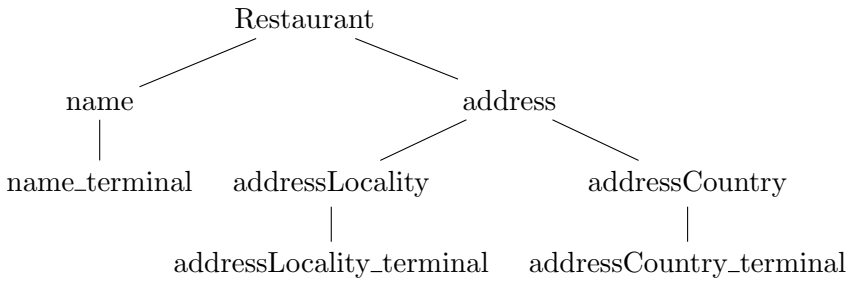


Abbildung 5.2.: Ausschnitt aus einer Aktiven Ontologie

Restaurant Goldener Adler Kaiserstraße 15 in Karlsruhe“. Möglicherweise feuert der Bag-of-Words-Knoten in diesem Fall auch, da die Wörter „ist“ und „Adresse“ erkannt wurden. Zusätzlich hat nun allerdings der Sensorknoten mit den Städten gefeuert, da „Karlsruhe“ erkannt wurde. In diesem Fall wird der Suchparameter „Stadt“ mit dem Wert „Karlsruhe“ weitergegeben. Der Benutzer möchte also nicht die Adresse eines Restaurants wissen. Dies erkennt die Aktive Ontologie daran, dass ein Sensorknoten auf ein Wort in der Anfrage reagiert hat.

5.2.3. Anpassung von bestehenden Knoten

In diesem Abschnitt werden die Anpassungen der bestehenden Knoten dargestellt. Zunächst wird beschrieben, wie die Sensorknoten modifiziert wurden, um die Wortliste aus einer Datei auszulesen. Zudem wurde der Name für die weitergegebenen Parameter dahingehend angepasst, dass diese Pfadinformationen beinhalten.

Wortliste aus Datei einlesen

Wenn die, aus der Datenbank exportierten, Daten in eine Aktive Ontologie umgewandelt werden sollen, werden die Wortlisten der einzelnen Sensorknoten sehr lang. Zu einer Eigenschaft in der Datenbank gibt es meist viele Werte. Bisher war es so, dass die Werte eines Sensorknotens in einer Liste in der Java-Klasse verwaltet wurden. Im Rahmen dieser Arbeit wurde der Sensorknoten dahingehend erweitert, dass die Wortlisten aus einer Datei geladen werden können. Dazu wurde ein neuer Konstruktor mit einem weiteren Parameter hinzugefügt. Als Wert für diesen Parameter wird der Pfad zur Datei angegeben, die Wörter enthält, auf die der Sensorknoten reagieren soll. Beim Initialisieren des Sensorknotens wird die Datei automatisch eingelesen und eine Wortliste im Objekt erzeugt.

Pfad als Parametername

Eine weitere Anpassung betrifft die Namen der erkannten Parameter. Bisher wurde für die Weitergabe eines Wertes nur der Name des eigenen Knotens mitgesendet. Dies wurde angepasst, sodass anhand des Namens eines Wertes der komplette Pfad zum Blattknoten nachvollzogen werden kann.

Die Abbildung 5.2 zeigt einen Ausschnitt aus einer Aktiven Ontologie. Bei einer natürlichsprachlichen Eingabe (bspw. „Karlsruhe“), für die der Sensorknoten „addressLocality_terminal“ anspringt, wird der Name dieses Sensorknotens als Parametername an den Elternknoten weitergegeben. Beim Knoten „addressLocality“ kommt ein Parameter mit dem Namen „addressLocality_terminal“ und dem Wert „Karlsruhe“ an. Dieser Knoten gibt diesen Wert wiederum mit seinem Namen weiter. Der Knoten „address“ erhält somit einen Parameter mit dem Namen „addressLocality“ und dem Wert „Karlsruhe“. Schlussendlich erhält der Knoten „Restaurant“ den Parameter „address“ mit dem Wert „Karlsruhe“. Es ist somit im Wurzelknoten nicht mehr möglich festzustellen, woher der Wert

kommt. Daher wurde diese Weitergabe des Namens in der Rahmenarchitektur angepasst. Die Sensorknoten werden beim Aufbau des Pfades ignoriert. Dies geschieht anhand der Endung des Namens eines Sensorknotens. Anschließend liefert jeder Knoten seinen Anteil am Pfad. Für das obige Beispiel würde der Wurzelknoten den Parameter mit dem Namen „address.addressLocality“ und dem Wert „Karlsruhe“ erhalten. Anhand des Parameternamens lässt sich hierbei eindeutig feststellen, woher der Wert ursprünglich kommt.

5.3. Zwischenformat

Zur Kommunikation zwischen den einzelnen Modulen ist ein vorher definiertes Zwischenformat notwendig. Beim Entwurf des Zwischenformats wurde darauf geachtet standardisierte Formate zu verwenden. Die Struktur der Daten in der Datenbank wird mittels eines JSON-Formats repräsentiert.

Struktur JSON

Dieses Beispiel zeigt eine sehr einfache Struktur eines Elements aus der Datenbank, welches fünf Eigenschaften besitzt. In der Datenbank besitzt dieses Element exakt dieselbe Struktur, allerdings wird es nicht als JSON gespeichert, sondern in eine datenbankspezifische Darstellung umgewandelt. Ein Element besitzt immer einen Typ („@type“), einen Namen („name“) und eine Adresse („adresse“). Die Adresse gliedert sich wiederum in Stadt („Stadt“) und Land („Land“).

```
{
        "@type": "",
        "name":"",
        "addresse": {
                "Stadt":"",
                "Land":""
        }
}
```

Die Werte aus der Datenbank und die natürlichsprachlichen Formulierungen aus Dialogflow werden in separaten Dateien gespeichert. Natürlichsprachliche Formulierungen werden in Prä- und Postfix-Dateien und die Werte aus der Datenbank in Dateien speziell für die Werte gespeichert. Bei diesen Dateien handelt es sich um einfache Textdateien mit einer speziellen Benamung.

Benamung der Dateien

Für die Formulierungen (Prä- und Postfix-Informationen, Werte aus der Datenbank und Bag-of-Words-Informationen) existieren unterschiedliche Dateien. Der Inhalt einer jeden Datei sind Textinformationen. Einzig die Dateiendung unterscheidet die unterschiedlichen Typen.
Die Werte aus der Datenbank werden in Wert-Dateien gespeichert. In einem späteren Schritt werden die Werte aus der Datenbank mit den Informationen aus Dialogflow zusammengeführt. Im Rahmen des Zusammenführens erhalten die Werte aus der Datenbank, die in Dialogflow definierten Synonyme. Diese Synonyme werden ebenfalls in die Wert-Datei geschrieben. Synonyme werden benötigt, um die Werte in der Datenbank mit unterschiedlichen Ausdrücken ansprechen zu können. Zum Beispiel ist in Dialogflow für „Restaurant“ ein Synonym „Gaststätte“ definiert. Mittels dieses Synonyms kann eine natürlichsprachliche Anfrage, welche nach einer „Gaststätte“ fragt entsprechend umgewandelt werden und das Synonym „Gaststätte“ durch den Wert „Restaurant“ ersetzt werden.

```
Wert → .val
```

Die Prä-/ und Postfix Dateien entstehen beim Zusammenführen der Informationen aus Dialogflow mit den Werten aus der Datenbank. Die Formulierungen, die vor einem Wert aus der Datenbank stehen können, werden in Präfix-Dateien gespeichert. Formulierungen nach einem Wert stehen in Postfix-Dateien.

```
Prefix → .pre
Postfix → .pos
```

Der vierte Datentyp enthält Informationen zum Bag-of-Words-Mechanismus und wird ebenfalls in einer separaten Datei gespeichert.

```
Bag−of−Words → .bow
```

Die Daten aus dem Zwischenformat werden in Dateien gespeichert, um den Arbeitsspeicher nicht zu belasten. Zudem kann so garantiert werden, dass dieses Verfahren auch für große Datenbanken funktioniert. Wenn die Daten aus einer Datenbank nicht mehr komplett im Hauptspeicher gehalten werden können, kann auch das Verfahren zur Erzeugung der Aktiven Ontologien nicht komplett im Arbeitsspeicher ablaufen.

5.4. Export aus der Datenbank

Dieser Abschnitt beinhaltet den Export der Daten aus der Datenbank in ein vorher definiertes Zwischenformat. Bei der Datenbank handelt es sich um eine deduktive Datenbank, welche die Daten als Tripel speichert. Ein Tripel besteht aus einem eindeutigen internen Bezeichner, dem Namen der Eigenschaft und dem Wert der Eigenschaft. Der Wert einer Eigenschaft kann wiederum einen internen Bezeichner beinhalten. Auf diese Weise können Tripel miteinander in Verbindung gesetzt werden.

Repräsentation eines Elements in Tripeln

Beispielhaft wird im folgenden die Repräsentation einer JSON-Datei zu einem Restaurant als Tripel dargestellt.

```
{
        "@type": "Restaurant",
        "name":"Goldener␣Adler",
        "addresse": {
                "Stadt":"Karlsruhe",
                "Land":"Deutschland"
        }
}
```

Dieses JSON wird folgendermaßen als Tripel repräsentiert:

```
<interneID1, @type, Restaurant>
<interneID1, name, Goldener Adler>
<interneID1, addresse, interneID1.addresse>
<interneID1.addresse, Stadt, Karlsruhe>
<interneID1.addresse, Land, Deutschland>
```

Beim Exportieren der Daten aus der Datenbank müssen die Tripel wieder zu JSON-Objekten zusammengesetzt werden. Die deduktive Datenbank speichert die eingehenden JSON-Objekte ab, sodass diese theoretisch direkt ausgegeben werden können. Der Grund,

warum die Tripel zusammengesetzt werden müssen ist, dass Elemente in der Datenbank durch Regeln verändert werden können. Beispielsweise können Eigenschaften zu Elementen hinzugefügt werden. Diese zusätzlichen Informationen sind nicht im ursprünglichen JSON-Objekt enthalten. Daher ist es notwendig die Tripel zu exportieren und anschließend das JSON-Objekt aus diesen Tripeln zu erzeugen.
Die Datenbank stellt eine Export-Funktion bereit, welche die Tripel in eine Datei schreibt. Ein Tripel erstreckt sich in der Datei über drei Zeilen. Die erste Zeile enthält die interne ID, die zweite den Namen der Eigenschaft und in der dritten Zeile steht der Wert oder eine interne ID.

Export eines Tripels

Beispiel:

```
<interneID1, @type, Restaurant>
```

Dieses Tripel wird exportiert und folgendermaßen in eine Datei geschrieben:

```
interneID1
@type
Restaurant
```

Des Weiteren sind die Tripel anhand der internen ID sortiert. Dies bedeutet, dass die Tripel eines Elementes immer hintereinander in der Datei liegen.
Zusätzlich zu den Elementen aus der Datenbank wird die Hierarchie exportiert. Ebenfalls wird eine Datei angelegt, welche die Abbildung der IDs der Elemente auf ihren entsprechenden Typ enthält. Diese beiden Informationen werden nach dem Exportieren aus der Datenbank geladen. Sie werden für das Zusammenführen der Strukturen benötigt. Ein eigenes Objekt in diesem Modul verwaltet das Zusammenführen und Extrahieren der Strukturen. Innerhalb dieses Objekts findet die komplette Verarbeitung der zusammengesetzten JSON-Objekte statt.
Nachdem alle Informationen exportiert wurden, müssen die Elemente aus der Datenbank zu JSON-Objekten zusammengesetzt werden. Dazu werden die Dateien, welche die Tripel enthalten, eingelesen und die zusammengehörigen Tripel zu einem JSON-Objekt zusammengesetzt. Das Resultat ist ein JSON-Objekt, wie im ersten Beispiel dieses Kapitels, vor der Umwandlung in Tripel. Dieses resultierende JSON-Objekt wird anschließend verwendet, um die Struktur und die Werte zu extrahieren.
Zu diesem Zweck wird das komplette JSON-Objekt durchlaufen und die Werte in entsprechende Dateien geschrieben. Existieren zu den Eigenschaften aus dem JSON-Objekt bereits Dateien, so werden die Werte hinzugefügt. Die bereits bestehenden Werte werden nicht überschrieben. Ein Dateiname setzt sich zusammen aus dem Typ des Elements und dem entsprechenden Pfad zum Wert.

Beispiel

Der Typ des Elements ist Restaurant und es existieren zwei Eigenschaften, die Werte enthalten. „@type“ und „Stadt“ enthalten einen Wert.

```
{
        "@type": "Restaurant",
        "addresse": {
                "Stadt":"Karlsruhe",
        }
}
```

Aus diesem JSON entstehen zwei Dateien mit Werten.

```
Restaurant.@type.val
Restaurant.addresse.Stadt.val
```

Die erste Datei enthält die Werte aus der Eigenschaft „@type“, also „Restaurant“. „Restaurant.addresse.Stadt.val“ enthält den Wert „Karlsruhe“ aus der Eigenschaft „Stadt“.

Auf diese Weise lässt sich eine Datei eindeutig einer Eigenschaft aus der Struktur zuordnen. Nachdem die Werte aus dem JSON-Objekt in die entsprechende Datei geschrieben wurden, werden diese aus dem JSON-Objekt entfernt. Das Resultat ist ein JSON-Objekt, welches Eigenschaften mit leeren Werten besitzt. Dieses leere JSON-Objekt wird dazu verwendet, um die Struktur der Aktiven Ontologie zu erzeugen. Beim Erstellen der Aktiven Ontologie werden aus den Eigenschaften dieses JSON-Objekts die inneren Knoten erzeugt. Die Namen der inneren Knoten entspricht den Bezeichnungen der Eigenschaften.

Beispiel

Dieses Beispiel zeigt ein JSON-Objekt, welches für die Erzeugung der Struktur der Aktiven Ontologie verwendet wird.

```
{
        "@type": "",
        "addresse": {
                "Stadt":"",
        }
}
```

Anschließend wird überprüft, ob für den aktuellen Typ bereits eine Struktur extrahiert wurde. Wenn bisher keine Struktur für den gegebenen Typ vorliegt, wird die Struktur anhand des Typs abgespeichert. Innerhalb des `Export`-Moduls werden die bereits extrahierten Strukturen als JSON-Objekte (siehe vorheriges Beispiel) gespeichert. Dies geschieht in einer `HashMap`, wobei der Schlüssel den Typ der Struktur enthält und der Wert das exportierte JSON-Objekt. Ist bereits eine Struktur für den aktuellen Typ vorhanden, so müssen die beiden Strukturen verschmolzen werden. In einem Tripel-Speicher muss nicht jedes Element eines Typs dieselben Eigenschaften besitzen. Daher können extrahierte Strukturen von unterschiedlichen Elementen desselben Typs andere Eigenschaften aufweisen. Im Gegensatz zu den Werten aus der Datenbank, werden die Strukturen im Arbeitsspeicher gehalten. Die Anzahl an unterschiedlichen Typen ist deutlich geringer, als die Anzahl an Daten.
Auf der Grundlage des höchsten Super-Typs wurden die Strukturen im Rahmen dieser Arbeit zusammengeführt. Im Fall dieser Arbeit existieren fixe Super-Typen. *LocalBusiness* definiert den Typ für Restaurants, Werkstätten, Betriebe, usw. *SportsActivityLocations* sind beispielsweise Wanderwege, Skipisten, Skilifte, usw. Bei *Events* handelt es sich um Veranstaltungen jeglicher Art. Außerdem existiert ein Typ für Angebote (*Offer*) und ein Typ für öffentliche Plätze (*Place*). Der Vorteil beim Zusammenfassen auf Basis des Super-Typs ist, dass wenige Aktive Ontologien erzeugt werden.
Die Ausgabe dieses Moduls beinhaltet die exportierten Strukturen und ein Pfad zu den Wert-Dateien.

5.5. Export aus Dialogflow

Dialogflow enthält händisch definierte Informationen zu Formulierungen und Synonymen. Diese werden verwendet, um die Prä- und Postfix-Knoten, sowie die Synonymknoten der

Aktiven Ontologie zu erzeugen (siehe Abschnitt 4.2.2). Dadurch wird das Sprachverständnis der Aktiven Ontologie erzeugt. Ohne diese Informationen wäre die Aktive Ontologie nichts anderes als eine Ontologie, die auf Schlüsselwörter aus der Datenbank reagiert. Die Daten in Dialogflow sind auf Basis der Daten in der Datenbank modelliert, das heißt die Definition der Entitäten mit deren Synonymen basiert auf den Werten aus der Datenbank. Dies ist besonders wichtig, da die Daten in der Datenbank auf Englisch abgespeichert werden, die Anfrage allerdings auf Deutsch stattfinden soll. Formulierungen sind wichtig, um herauszufinden, welche Eigenschaften eines Elements ein Benutzer mit seiner Anfrage adressieren möchte.
In Dialogflow können Agenten (Abschnitt 2.2.1) angelegt werden, welche die Intents und Entitäten enthalten. Zur Verarbeitung der modellierten Informationen werden die Agenten aus Dialogflow exportiert. Über die API kann ein Agent direkt als ZIP-Datei exportiert werden. Diese ZIP-Datei enthält alle Informationen zum Agenten, den Intents und Entitäten. Ein Agent kann Intents und Entitäten zu mehreren Sprachen beinhalten. Jeder Dateiname enthält eine Bezeichnung für die Sprache des Inhalts. Das folgende Beispiel erläutert die einzelnen Komponenten eines Dateinamens.

Beispiel

Dieses Beispiel zeigt den Dateinamen eines Intents, aus dem Export eines Agenten, aus Dialogflow. Datalayer_SportsActivityLocation_openingHours_usersays_de.json
In diesem Fall handelt es sich um den Intent zu den Öffnungszeiten einer „SportsActivityLocation“. „usersays“ gibt an, dass diese Datei die annotierten Fragen enthält. „de“ bedeutet, dass die annotierten Fragen in deutscher Sprache vorliegen. Das Präfix „Datalayer“ signalisiert, dass dieser Intent innerhalb der Anwendung direkt an die Datenbank weitergeleitet werden soll.

Analog gibt es ein Namensschema für die Entitäten, aus denen die Sprache extrahiert werden kann.

Beispiel

Im Folgenden wird der Dateiname einer Entität aus Dialogflow erläutert. placeType_entries_de
„placeType“ gibt den Namen der Entität an. Die Datei mit Synonym-Wert-Paaren wird durch „entries“ kenntlich gemacht. „de“ gibt wiederum die Sprache der Synonyme an.

Zum Export der Agenten aus Dialogflow wird die Google REST V2[1]-Schnittstelle verwendet. Eine Anfrage an die REST API liefert eine ZIP-Datei des Agenten. Diese Datei wird nicht direkt gespeichert, sondern entpackt und gefiltert. Nicht alle Dateien werden im späteren Verlauf benötigt. Es werden nur Intent- und Entity-Dateien gespeichert und zwar nur die Dateien, welche die annotierten Fragen bzw. die Entity-Typen mit den Synonymen enthalten. Dafür gibt es entsprechende Filter, die anhand des Dateinamens filtern. Die benötigten Dateien werden in ein temporäres Verzeichnis, welches entsprechend dem Agenten benannt ist, geschrieben.
Im nächsten Schritt werden die Dateien eingelesen und weiter verarbeitet. Zunächst werden die Entity-Dateien verarbeitet. Dazu wird zunächst überprüft, ob für die im Dateinamen gegebene Sprache bereits Entitäten existieren. Das Speichern der Sprachinformationen findet in einer verschachtelten HashMap statt. Die äußere HashMap speichert die Sprache und einen Container mit den Intents und Entitäten. Innerhalb dieses Containers existieren zwei HashMaps, zum einen für die Intents und zum anderen für die Entitäten. Der

[1] `https://dialogflow.com/docs/reference/api-v2/rest`

Schlüssel ist jeweils der Name des Entity-Typs oder der Name des Intents. Wenn für die Sprache noch keine Entitäten existieren, wird ein neuer Eintrag zur Sprache in der HashMap mit einem leeren Container für die Intents und Entitäten angelegt. Existiert für die Sprache bereits ein Eintrag, so wird die HashMap mit den Entitäten zu dieser Sprache als Ausgangspunkt verwendet. Anschließend wird überprüft, ob für den aktuell betrachteten Entity-Typ bereits ein Eintrag in der HashMap vorhanden ist. Ist dies nicht der Fall, werden die neuen Entitäten zur HashMap hinzugefügt. Falls bereits ein Eintrag zum gegebenen Entity-Typ besteht, werden die Informationen miteinander verschmolzen. Für Entitäten, die den selben Datenbankwert beschreiben, werden die bestehenden Synonyme durch die neuen Synonyme erweitert. Duplikate werden in diesem Fall eliminiert.
Analog wird mit der Verarbeitung der Intents vorgegangen. Der Unterschied bei den Intents ist allerdings, dass für die annotierten Fragen keine Duplikateliminierung stattfindet. Im späteren Schritt des Zusammenführens wird beim Erzeugen der Prä- und Postfix-Dateien darauf geachtet, dass keine Duplikate existieren. In diesem Modul werden für gleich benannte Intents, die annotierten Fragen einfach verschmolzen ohne Duplikate zu beachten. Das Resultat dieses Moduls ist eine nach Sprachen sortierte Sammlung der Intents und Entitäten.

5.6. Zusammenführen der Exports

Die Eingabe für das Modul zum Zusammenführen der Informationen besteht zum einen aus den exportierten Strukturen und Werten aus der Datenbank und zum anderen aus den natürlichsprachlichen Informationen. Die Strukturen werden in diesem Schritt nicht verändert, einzig die Dateien werden durch natürlichsprachliche Informationen angereichert. Zunächst werden die natürlichsprachlichen Informationen dahingehend umgewandelt, dass sie zum Anreichern der Daten verwendet werden können. Im Fall der Entitäten bedeutet dies, dass sie entsprechend der Werte, auf den die Synonyme abgebildet sollen, abgespeichert werden. Die Informationen zu den Entitäten werden in einem `Entity`-Objekt gespeichert. Dieses Objekt enthält Informationen, wie den Entity-Typ, den Wert und die Synonyme. Intents, inklusive der Liste der annotierten Fragen, werden anhand der Entity-Typen aus den annotierten Fragen gespeichert. Enthält eine annotierte Frage mehrere Entity-Typen, so taucht diese häufiger unter den entsprechenden Entity-Typen auf.
Im Zuge der Umwandlung der Intents und Entitäten werden anhand der Abbildungen die Bag-of-Words-Informationen berechnet. Zur Berechnung der Bag-of-Words-Informationen existiert eine Abbildungsdatei die angibt, welche Intents für eine spezielle Eigenschaft des JSON-Objekts definiert wurden. Beispielsweise besitzt ein JSON-Objekt die Eigenschaft „Öffnungszeiten“, welche Informationen zu den Öffnungszeiten eines Restaurants enthält. Für diese Eigenschaft wurden in Dialogflow spezielle Intents definiert, welche ausschließlich Fragen zu den Öffnungszeiten beinhalten. In der Abbildungsdatei werden die Namen der Intents angegeben und die Eigenschaft auf die sie sich beziehen.

Beispiel

Es existiert zum Beispiel ein Intent „LocalBusiness_OpeningHours“ und das JSON zu einem Restaurant hat die Eigenschaft „OpeningHoursSpecification“. In der Abbildungsdatei würde folgender Eintrag stehen:

```
LocalBusiness_OpeningHours->OpeningHoursSpecification
```

Ein eigenes Objekt berechnet die benötigten Informationen. Dazu werden alle Intents zu diesem Objekt hinzugefügt. Das Objekt hat die Informationen aus der Abbildungsdatei eingelesen und kann somit entscheiden, welche Intents auf welche Eigenschaften abgebildet

werden sollen. Bei dem Hinzufügen der Intents wird daher unterschieden, ob es sich um einen Intent handelt, der für die Bag-of-Words-Informationen einer Eigenschaft benötigt wird, oder nicht. Diese beiden Typen werden unterschiedlich gespeichert. Die Intents, die auf eine Eigenschaft abgebildet werden, werden anhand dieser Eigenschaft gruppiert. Andere Intents werden anhand des Typs gruppiert. Beispielsweise existieren Intents, die auf Events, Angebote oder Sportlokationen spezialisiert sind. Eine Gruppe von Intents wird als ein Dokument angesehen. Dies ist für die spätere TF-IDF-Berechnung wichtig. Nachdem alle Intents zu diesem Objekt hinzugefügt wurden, wird die eigentliche Berechnung ausgeführt. Zunächst werden die Intents verarbeitet, für die eine Abbildung existiert. Die einzelnen annotierten Fragen dieser Intents werden anhand der Leerzeichen getrennt, um die einzelnen Wörter zu erhalten. Für die einzelnen Wörter wird anschließend berechnet, wie oft sie in allen annotierten Fragen vorkommen. Diese Information wird für die Berechnung des Gewichts (TF-IDF Abschnitt 2.5.2) benötigt. Dazu werden alle annotierten Fragen durchgegangen und es werden die Häufigkeiten aller Wörter gespeichert. Nachdem die annotierten Fragen der Intents mit Abbildung verarbeitet wurden, werden die restlichen gruppierten Intents verarbeitet. Bei den restlichen Intents werden allerdings nicht alle Worthäufigkeiten berechnet, sondern nur die Worthäufigkeiten ergänzt, die im ersten Schritt extrahiert wurden. Zusätzlich wird ermittelt in wie vielen unterschiedlichen Dokumenten die Worte vorkommen. Wie bereits zu Beginn erwähnt, ist eine Gruppe ein Dokument. Taucht ein Wort beispielsweise in den Intents zu *SportsActivityLocations* und *LocalBusinesses* auf, so wird die Anzahl der Dokumente, in der das Wort vorkommt, auf zwei gesetzt. Die Zahl für die Anzahl der Dokumente wird pro Intent-Gruppe nur einmal erhöht. Taucht ein Wort in einer Intent-Gruppe häufiger auf, so wird lediglich die Worthäufigkeit erhöht, aber nicht die Anzahl der Dokumente. Nachdem die restlichen Intents in die Berechnung der Worthäufigkeiten für die Abbildungsintents eingegangen sind, wird bestimmt, welches Wort am häufigsten vorkommt. Dazu werden alle Worthäufigkeiten durchlaufen und die Höchste wird gespeichert. Informationen zu der Worthäufigkeit, Anzahl der Dokumente in der das Wort auftritt, Gesamtzahl der Dokumente und die größte Häufigkeit werden in einem Objekt gespeichert. Jedes Wort besitzt ein eigenes Objekt, welches diese Informationen enthält. Diese Objekte werden in einer Liste gespeichert und können für die Bestimmung der maximalen Häufigkeit durchlaufen werden. Die maximale Häufigkeit wird für jedes dieser Objekte gesetzt. Anschließend werden die Objekte für Wörter herausgefiltert, die hinderlich für den Bag-of-Words-Mechanismus sind. Dies sind die Personalpronomen, Possessivpronomen und die Artikel. Zum Filtern der Wörter wird ein Part-of-Speech-Tagger verwendet. Dieser bestimmt die Worttypen eines jeden einzelnen Wortes, anhand dessen die nicht gewünschten Wörter aussortiert werden. Abschließend wird mittels TF-IDF die inverse Worthäufigkeit berechnet, aus diesem Grund wurden zuerst die Häufigkeiten der einzelnen Wörter bestimmt. Zur Berechnung des TF-IDF-Wert wird für die Termhäufigkeit die gewichtete Formel (Abschnitt 2.5.2) verwendet.

$$tf(t,d) = 0.5 + 0.5 * \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}} \tag{5.1}$$

Für die inverse Dokumentenhäufigkeit wird die ungewichtete Formel verwendet.

$$idf(t,D) = log\frac{N}{d \in D : t \in d} \tag{5.2}$$

N ist die Gesamtzahl der Dokumente und $d \in D : t \in d$ ist die Anzahl der Dokumente in der Term t vorkommt. Nach der Berechnung dieser inversen Worthäufigkeit werden die Wörter mitsamt der Gewichtung (TF-IDF) in eine Datei geschrieben und mit den anderen Dateien an das nächste Modul weitergegeben.
Nachdem die sprachlichen Informationen umgewandelt und die Informationen zu dem Bag-of-Words-Mechanismus erzeugt wurden, werden die Wert-Dateien einzeln verarbeitet. Dazu werden die Dateien nacheinander, zeilenweise, eingelesen und die Werte entsprechend

angereichert. Zunächst wird der ursprüngliche Wert mit Hilfe eines Stemmers in seine Grundform gebracht und anschließend in eine temporäre Datei geschrieben. Diese temporären Dateien werden verwendet, um die Nutzung des Arbeitsspeichers zu reduzieren. Anschließend wird mit dem Wert in den transformierten Entitäten nach dem entsprechenden Entity-Objekt gesucht. Existieren zu dem gegebenen Wert Entity-Informationen, so werden die Synonyme verwendet und in die bereits bestehende temporäre Datei geschrieben. Abbildungszeilen haben eine definierte Struktur. Zunächst wird das Synonym angegeben, darauf folgt ein Pfeil → und abschließend der Wert aus der Datenbank.

Beispiel

Die Einträge in der Wert-Datei für das Wort „Restaurant“ und dessen Synonyme „Gaststätte, Lokal“ sehen wie folgt aus:

```
Restaurant
Gaststätte → Restaurant
Lokal → Restaurant
```

Des Weiteren werden die Formulierungen für die Prä- und Postfix-Dateien verwendet. Nach der Umwandlung sind die annotierten Fragen entsprechend der enthaltenen Entity-Typen gespeichert. Nachdem bereits ein Entity-Typ gefunden wurde, werden mittels dieses Typs die annotierten Fragen extrahiert. Die entsprechenden Formulierungen werden zur Erzeugung der Prä- und Postfix-Knoten verwendet. Dazu wird in der entsprechenden Formulierung der Entity-Typ gesucht und jeweils der Textblock vor und der Textblock nach dem Entity-Typ aus den annotierten Fragen extrahiert. Auf diese Weise entstehen zu jeder Wert-Datei die passenden Prä- und Postfix-Dateien.
Die unveränderten Strukturen und die veränderten Wert-Dateien, inklusive der neu erzeugten Bag-of-Words und Prä- und Postfix-Dateien werden an das nächste Modul weitergegeben.

5.7. Wörterbuch

Nachdem die Sprachinformationen mit den Informationen aus der Datenbank verschmolzen wurden, folgt das Wörterbuch-Modul. In diesem Modul werden Werte aus der Datenbank, für die keine Synonyme in *Dialogflow* modelliert wurden, übersetzt und durch Synonyme angereichert. Dazu werden die einzelnen Wert-Dateien durchgegangen und es wird nach Wörtern gesucht, für die keine Abbildungen (Synonyme) vorliegen. Die Werte in den Wert-Dateien liegen in Englisch vor, sodass sie zunächst ins Deutsche übersetzt werden müssen. Zur Übersetzung wird die Microsoft Translator Text-API[2] verwendet. Diese API ist Teil von Microsofts *Cognitive Services*. Nach der Übersetzung vom Englischen ins Deutsche, muss der übersetzte Wert mit Synonymen angereichert werden.
Zur Anreicherung mit Synonymen wird *Wiktionary*[3] verwendet. Hierbei handelt es sich um ein frei zugängliches mehrsprachiges Wörterbuch. Zur Umsetzung dieses Moduls wurde eine frei zugängliche Backup-Datei heruntergeladen. Diese Datei kann mittels der Bibliothek JWKTL[4] eingelesen werden. Dadurch können die Suchanfragen auf die Wiktionary-Datenbank lokal erfolgen. Um den gewünschten Wert mit Synonymen anzureichern, wird zunächst eine Suche gestartet, bei der nach exakt dem gegebenen Wort gesucht wird. Wird ein Ergebnis gefunden, so werden die entsprechenden Synonyme des Suchergebnisses extrahiert und gespeichert. In der Konfigurationsdatei kann angegeben werden, wie viele

[2] www.microsoft.com/de-de/translator/business/translator-api/
[3] www.wiktionary.org
[4] dkpro.github.io/dkpro-jwktl/

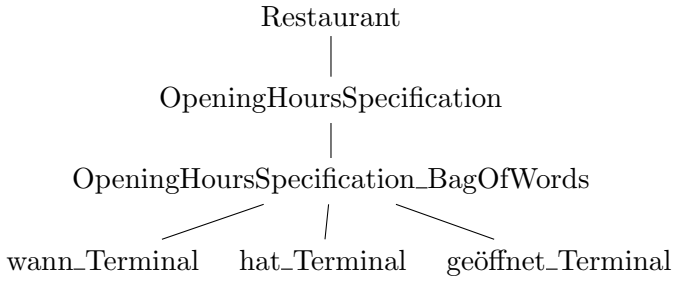
Schritte bei der Synonymsuche gemacht werden sollen. Sollen mehrere Schritte gegangen werden, so werden jeweils die gefundenen Synonyme als Suchwort verwendet, um weitere Synonyme zu finden. Liefert die exakte Suche kein Ergebnis, so wird die Suche etwas abgeschwächt. Es wird nach ähnlichen Ergebnissen gesucht und immer das erste Ergebnis dieser Suche verwendet. Auch hier werden anschließend die Synonyme extrahiert und je nach angegebener Tiefe, werden weitere Synonyme gesucht. Nachdem die Suche in Wiktionary abgeschlossen ist und alle Synonyme extrahiert wurden, werden die Synonyme für den aktuellen Wert in die Wert-Datei geschrieben.

5.8. Erzeugung der Aktiven Ontologien

Im letzten Schritt des Verfahrens werden aus den extrahierten Strukturen und den erzeugten Dateien die Aktiven Ontologien erzeugt. Zunächst werden alle Wert, Prä-, Postfix und Bag-of-Words-Dateien eingelesen und in einer HashMap gespeichert. Als Schlüssel wird der Dateiname verwendet und der Pfad zu der Datei als Wert gespeichert. Der Dateiname spiegelt den Pfad der Eigenschaft wieder, daher eignet sich dieser sehr gut als Schlüssel für die HashMap. Anschließend werden die einzelnen Strukturen verarbeitet und die Aktiven Ontologien daraus erzeugt. Zunächst wird ein Wurzelnoten erzeugt, welcher denselben Namen wie der Typ der aktuell betrachteten Struktur erhält. Dieser Wurzelknoten bekommt das Postfix „_Command“ angehängt. Anschließend wird überprüft, ob für den gegebenen Knoten Bag-of-Words-Informationen vorliegen. Ist das der Fall, so wird ein spezieller Knoten erzeugt. Ein Bag-of-Words-Knoten besteht aus Sensorknoten; für jedes Wort aus der Bag-of-Words-Liste wird ein Sensorknoten erzeugt. Zudem gibt es einen Knoten, der die Konfidenzen für die erkannten Worte addiert. Wird ein definierter Schwellenwert überschritten, so feuert der Bag-of-Words-Knoten.

Beispiel

Ein Restaurant besitzt beispielsweise einen Bag-of-Words-Knoten für die Frage nach den Öffnungszeiten. Signifikante Wörter, um nach den Öffnungszeiten zu fragen, sind „Wann, hat, geöffnet“. Daraus ergibt sich für den Bag-of-Words-Knoten folgende Struktur in der Aktiven Ontologie.



Anschließend werden die Eigenschaften der gegebenen Struktur verarbeitet. Besitzt eine Eigenschaft weitere Kindknoten, so wird die Methode rekursiv aufgerufen. Zunächst wird ein Sammelknoten für die gegebene Eigenschaft erzeugt. Beim Erzeugen der Sammelknoten wird überprüft, ob Prä- und Postfix-Dateien vorhanden sind. Ist dies der Fall, so wird für den Sammelknoten definiert (setzen einer speziellen Aktion), dass zusätzlich zu den Informationen aus dem Synonymknoten auch Informationen aus den Prä- und Postfixknoten überprüft werden müssen. In diesem Fall erwartet der Sammelknoten Informationen der Prä- und Postfixknoten und des Synonymknotens. Wenn keine Prä- und Postfix-Informationen gegeben sind, gibt der Sammelknoten keinen Wert weiter. Daraufhin wird überprüft, ob es zur gegebenen Eigenschaft Bag-of-Words-Informationen gibt. Liegen

Bag-of-Words-Informationen vor, so werden für die gegebenen Worte mit ihren Gewichten Sensorknoten erzeugt. Zusätzlich wird ein übergeordneter Knoten erzeugt, welcher die Gewichte der erkannten Sensorknoten aufaddiert und bei Überschreitung eines definierten Schwellwerts eine entsprechende Information weitergibt (siehe Abschnitt 5.2.2). Anschließend werden die weiteren Eigenschaften verarbeitet. Gelangt der Zerteiler an eine Eigenschaft, die keine weiteren Kindeigenschaften enthält, werden die Sensorknoten erzeugt. Zunächst wird ein Sensorknoten erzeugt, der die Synonym-zu-Datenbankwert-Abbildungen enthält. Anschließend werden die Prä- und Postfix-Knoten erzeugt. Bei der Erzeugung der Knoten wird jeweils der Pfad zur Datei mit den Werten an den Konstruktor übergeben. Aus dieser Datei lesen die Knoten die Information ein, auf die sie reagieren sollen. Der Pfad zur Datei kann aus der HashMap ausgelesen werden, welche die Pfade innerhalb des JSON-Objekts als Schlüssel enthält. Erzeugung der Knoten heißt in diesem Fall, dass der entsprechende Java-Code in eine Datei geschrieben wird.
Nachdem jede Struktur zu einer Aktiven Ontologie umgewandelt wurde, wird eine Datei erzeugt, welche alle Aktiven Ontologien initialisiert und zu einer einzigen Aktiven Ontologie zusammenfasst. In der erzeugten Ontologie werden die Wurzelknoten aller Aktiven Ontologien unter einem Auswahlknoten zusammengefasst. Dieser Auswahlknoten wählt das Ergebnis der Ontologie mit den meisten Informationen. Liefern mehrere Ontologien genau gleich viele Informationen, so wird zufällig eine Ontologie ausgewählt. Der Auswahlknoten gibt die erhaltenen Informationen an einen Sammelknoten weiter, welcher eine spezielle Aktion besitzt. Mittels dieser Aktion wird die Anfrage an den eigentlichen Dienst ausgeführt.
Die Ausgabe dieses Moduls enthält die Java-Dateien zu den Aktiven Ontologien. Ebenfalls enthalten ist eine Java-Datei, die alle Aktiven Ontologien initialisiert und zu einer einzigen zusammenfasst. Diese Dateien können in die *EASIER*-Rahmenarchitektur integriert und ausgeführt werden. Bei der Ausführung der Dateien in der Rahmenarchitektur, wird dem Benutzer eine grafische Oberfläche angezeigt. Mittels dieser Oberfläche können Anfragen an die Aktive Ontologie gestellt werden. Gleichzeitig werden auch verschiedene Informationen zum Ausführungsprozess dargestellt.

Abbildung 5.3 zeigt die Easier-Oberfläche. Das linke Fenster zeigt die Benutzereingaben und Ergebnisse an. Rechts finden sich Informationen zum Ausführungsprozess, sowohl was gerade im Faktenspeicher gespeichert ist, als auch welche Fakten gerade bearbeitet werden. Am unteren Ende des Fensters wird die Konsolenausgabe dargestellt.
Im Anhang (siehe Abschnitt A.1) werden zwei mögliche Funktionen genannt, welche noch in das Verfahren dieser Arbeit integriert werden könnten.

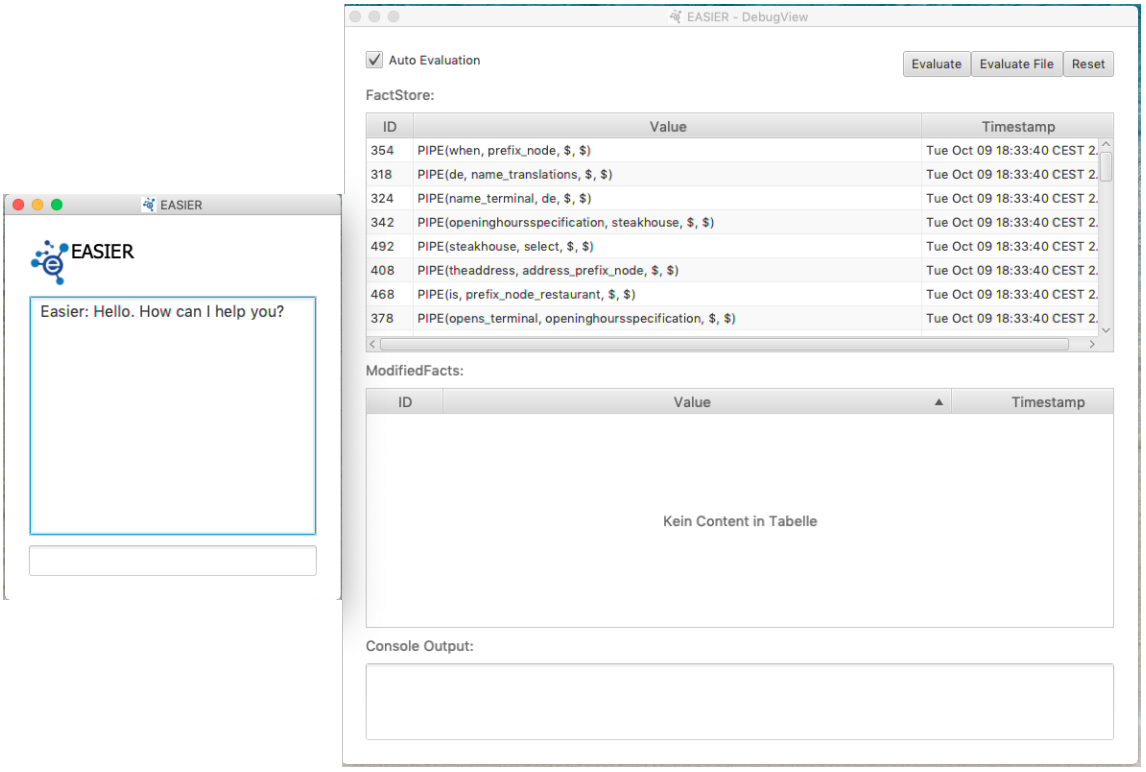


Abbildung 5.3.: Grafische Oberfläche der *EASIER*-Rahmenarchitektur

6. Evaluation

Zur Evaluation des Ansatzes dieser Arbeit werden Daten aus unterschiedlichen Themenbereichen verwendet, um die Aktiven Ontologien zu erzeugen. Auf der Sprachseite werden passend zu den Themengebieten Dialogflow-Agenten verwendet. Diese Agenten existieren bereits und sind auf die Daten abgestimmt. Die Daten aus der Datenbank bleiben für alle Fälle der Evaluation gleich. Das Einzige das variiert, sind die verwendeten Dialogflow-Agenten. Hiermit soll evaluiert werden, ob die Aktiven Ontologien auch Fragen zu Themengebieten korrekt verarbeiten können, für die keine Formulierungen bei der Erzeugung vorliegen. Zum Evaluieren werden allerdings immer alle verfügbaren Fragen verwendet. Auch Fragen aus den Dialogflow-Agenten, welche nicht zur Erzeugung verwendet wurden, werden zur Evaluation verwendet. Die, durch die Aktive Ontologie extrahierten Parameter werden mit den Ergebnissen aus Dialogflow verglichen.
Zu Beginn dieses Kapitels werden die verwendeten Datensätze dargestellt (Abschnitt 6.1). Hierfür werden sowohl die Daten aus der Datenbank vorgestellt, als auch die Informationen aus Dialogflow. Anschließend wird genau erläutert, wie die Evaluation ausgeführt wird (Abschnitt 6.2). Den Abschluss des Kapitels bilden die Ergebnisse der Evaluation (Abschnitt 6.3).

6.1. Datenbasis

Für die Evaluation des Ansatzes dieser Arbeit wird eine deduktive Datenbank verwendet. Die Daten innerhalb dieser Datenbank bestehen aus einem Ausschnitt eines Produktivsystems. Der Ausschnitt enthält Daten zu den Themengebieten *Tourismus*, *Hotels* und *Web-Cams*. Für die Formulierungen und Synonyme werden Agenten aus Dialogflow verwendet. Der erste Teil des Unterkapitels stellt beispielhaft die Daten in der Datenbank vor. Im zweiten Teil werden die Fragen, Formulierungen und Synonyme der Dialogflow-Agenten dargestellt.

6.1.1. Datensätze

Um die Datenbank mit Daten zu befüllen, wurden aus unterschiedlichen Themengebieten Daten aus einem Produktivsystem verwendet. Die Themengebiete umfassen Tourismusdaten, Hoteldaten und Informationen über Webcams.
Toursimusdaten sind beispielsweise Restaurants, Skiverleihe, Schwimmbäder, Veranstaltungen, Angebote und öffentliche Plätze. Das Themengebiet Hotel beinhaltet Informationen zu unterschiedlichen Hotels. Bei Webcams wird beispielsweise der Ort gespeichert und

Themengebiet	Local-Business	Event	Offer	Place	WebCam
Tourismus	1528	5	15	325	0
Hotels	1303	0	0	0	0
WebCams	0	0	0	0	685
Gesamt	**2831**	**5**	**15**	**325**	**685**

Tabelle 6.1.: Anzahl an Elementen für die vorhandenen Typen

ein Link zum Echtzeitbild dargeboten.
Das Produktivsystem unterstützt unterschiedliche Typen an Daten. Es existieren fünf Obertypen („LocalBusiness“, „Event“, „Offer“, „Place“, „WebCam“). Diese Obertypen enthalten diverse Untertypen, beispielsweise „Restaurant“ oder auch „Hotel“ ist jeweils ein Subtyp von „LocalBusiness“. Alle möglichen Arten von Veranstaltungen werden als Subtyp von „Event“ gespeichert. Angebote, welche von einem „LocalBusiness“ angeboten werden, werden unter „Offer“ gespeichert. Die WebCams haben einen eigenen Typ, „WebCam“.

Tabelle 6.1 zeigt eine Übersicht über die Anzahl der Daten, die zu den unterschiedlichen Typen vorliegen. Dies bedeutet beispielsweise, dass die Datenbank 1303 Hotels und 685 WebCams enthält. Die in der Tabelle angegebenen Typen sind Obertypen, diese unterteilen sich innerhalb der Datenbank in weitere Untertypen. Zum Beispiel der Typ `Restaurant` ist ein Untertyp von `LocalBusiness`.
Die folgenden Abschnitte liefern für jeden Obertyp ein Beispiel. In der Datenbank existieren beispielsweise 2831 dieser JSON-Objekte für den Typ `LocalBusiness`. Diese Beispiel-JSON-Dateien beinhalten die möglichen Eigenschaften und mögliche Werte für den gegebenen Typ. Die Werte wurden aus Datenschutzgründen teilweise entfernt oder abgeändert. Sie dienen lediglich zur Veranschaulichung einer möglichen Struktur eines solchen Elements.

LocalBusiness

Beispielhaft wird im Folgenden die JSON-Datei zum Typ `LocalBusiness` vorgestellt.

```
{
        "location_keywords": [],
        "paymentAccepted": null,
        "lodging_informations": [],
        "images": [
                {
                        "remote_url": "",
                        "url": "",
                        "order": 0
                }
        ],
        "address": {
                "addressCountry": "Germany",
                "streetAddress": "Musterstraße 1",
                "@type": "PostalAddress",
                "postalCode": "76130",
                "name": null,
                "external_id": null,
                "telephone": "",
                "addressLocality": "Karlsruhe",
                "addressRegion": null,
                "@context": "http://schema.org"
        },
```

```
        "keywords": [],
        "@type": [
                "MountainHut",
                "Restaurant"
        ],
        "external_id": "",
        "alternateName": "",
        "videos": [],
        "source": "",
        "parentOrganization": null,
        "@context": "http://schema.org",
        "userId": "1",
        "url": "",
        "openingHoursSpecificationSet": [
                {
                        "validThrough": "2018-10-06",
                        "@type": "OpeningHoursSpecificationSet",
                        "name": null,
                        "openingHoursSpecification": [
                                {
                                        "allDay": null,
                                        "dayOfWeek": "Monday",
                                        "validThrough": "2018-10-06T22
                                           :00:00.000Z",
                                        "@type": "
                                           OpeningHoursSpecification",
                                        "opens": "08:30",
                                        "validFrom": "2018-05-25T22
                                           :00:00.000Z",
                                        "@context": "http://schema.org"
                                           ,
                                        "closes": "19:00"
                                }
                        ],
                "validFrom": "2018-05-25",
                "@context": "http://schema.org"
                }
        ],
        "organizationId": "",
        "geo": {
                "elevation": null,
                "@type": "GeoCoordinates",
                "latitude": "0.0",
                "@context": "http://schema.org",
                "longitude": "0.0"
        },
        "name_translations": {
                "de": "Goldene␣Möwe"
        },
        "description_translations": {
                "de": ""
        },
        "ranking": null,
        "@id": "123",
        "starRating": null,
        "email": null
}
```


Diese JSON-Datei beschreibt die möglichen Eigenschaften eines `LocalBusiness`. In die-

Themengebiet	Annotierte Fragen	Entitäten
Tourismus	1140	2430
Hotels	378	727
WebCams	134	1440

Tabelle 6.2.: Anzahl an Fragen und Entitäten für die vorhandenen Themengebiete

sem konkreten Beispiel wird eine JSON-Datei zu einem `Restaurant` dargestellt. Für jeden neuen Eintrag in der Datenbank wird ein solches Objekt eingefügt. Dies bedeutet für die Hotels bestehen 1303 dieser Objekte in der Datenbank. Die Struktur dieser Objekte orientiert sich an der Definition aus *Schema.org*(Abschnitt 2.3). Beispielsweise sind in diesen Objekten Informationen zur *Adresse*, dem *Namen* und *Öffnungszeiten* gespeichert. Alle Informationen, die zu einem Element vorliegen, werden in diesem Objekt gespeichert. Für alle Typen an Daten werden solche JSON-Objekte in die Datenbank eingefügt. Der Unterschied zwischen den Typen sind die vorhandenen Eigenschaften, diese können variieren. Des Weiteren variieren die Werte für die gegebenen Eigenschaften. Die grundlegende Struktur bleibt für die unterschiedlichen Typen bestehen. Dies bedeutet, die Struktur für beispielsweise die Adresse ist für alle Typen gleich.

6.1.2. Dialogflow-Agenten

Zu jedem der Themengebiete aus dem Produktivsystem existiert ein zugehöriger Dialogflow-Agent. Die Fragen in diesen Agenten wurden händisch eingetragen und die entsprechenden Werte annotiert. Annotieren bedeutet, dass die wichtigen Wörter in einem Satz markiert und mit Entity-Typen (siehe Abschnitt 2.2) versehen werden.
Die Agenten werden zum einen dazu benutzt, die Daten aus der Datenbank mit Formulierungen und Synonymen anzureichern (siehe Abschnitt 4.2.3) und zum anderem zur Evaluation. Dazu wurden die Fragen aus den Agenten exportiert.

Tabelle 6.2 zeigt die Anzahl an annotierten Fragen und Entitäten, die in den Agenten zur Verfügung stehen. Diese Fragen werden zum Evaluieren der Aktiven Ontologie durch das Evaluationswerkzeug verwendet.
Im Folgenden werden beispielhaft fünf Fragen eines jeden Themengebiets dargestellt.

Beispiel

Dieses Beispiel zeigt einen Auszug aus den verwendeten Fragen. Die wichtigen Wörter werden hervorgehoben. Bei den Fragen handelt es sich um Fragen die direkt aus den Dialogflow-Agenten extrahiert wurden.

Tourismus

- Wo ist ein **Café**?
- **Essen für Glutenallergiker**
- Zeig mir die **Blaue Pisten** in **Seefeld**
- Ist im Ort ein **Hallenbad**?
- mein **Hörgerät** ist kaputt

Hotel

- **Hotels** mit **Restaurant** und **Sauna** in **Graz**
- Ich möchte in ein **Schlosshotel** mit meiner Freundin für einen **romantischen** Urlaub
- Ich suche ein **Hotel** in **Mayrhofen**, das eine **Sauna**, einen **Whirlpool**, **wlan** und einen **gratis Parkplatz** hat
- Suche **Hotel** mit **gratis Parkplatz**, **Gutschein für Wellnessbereich**, **Wein** im Zimmer und einem inkludierten **Restaurantgutschein** in **Graz**
- **Wellnesshotel** in **Graz** mit **gratis Parkplatz**

WebCam

- Wie sieht die **Wettercam** für **Seefeld** aus?
- Kannst du mir die **Webcams** zeigen?
- ich möchte die **360° bild** über den **Gletscher** sehen
- können wir am **Berg** die **aktuellen Bilder** aufrufen
- Gibt es **Livebilder** vom **Wetter** in **Seefeld**?

6.2. Methodik

Zur Ausführung der Evaluation liegen die, aus dem Produktivsystem exportierten, Daten in einer Datenbank vor. Diese Datenbasis bleibt bei der Evaluation unverändert, einzig die verwendeten Sprachinformationen werden variiert. Dadurch können unterschiedliche Szenarien simuliert und evaluiert werden. Innerhalb dieses Abschnitts werden zunächst die verwendeten Fragen beschrieben. Anschließend wird die eigentliche Methodik erläutert. Um die Evaluation automatisiert durchzuführen, wurde ein Werkzeug entwickelt. Die Funktionsweise wird am Ende dieses Abschnitts erläutert.

6.2.1. Testfragen

Zur Durchführung der Evaluation wurden die Fragen der einzelnen Agenten extrahiert. Ein zufällig ausgewählter Teil dieser extrahierten Fragen wird anschließend als Testfragen verwendet. Da die Ergebnisse der Evaluation händisch ausgewertet werden, wird nur eine Teilmenge der zur Verfügung stehenden Fragen verwendet. Mittels eines Zufallsgenerators werden eine vordefinierte Anzahl an Fragen ausgewählt und für die Evaluation verwendet. Für die Testfälle in denen alle Formulierungen und Synonyme aus Dialogflow, beziehungsweise keine Formulierungen und Synonyme aus Dialogflow verwendet werden, wurden 100 Testfragen aus jedem Themengebiet (insgesamt 300 Fragen) ausgewählt. Für die anderen Fälle jeweils 50 Fragen (insgesamt 150 Fragen) ausgewählt.

6.2.2. Variation der Sprachinformationen

Zur Variation der Sprachinformationen werden die Agenten aus Dialogflow unterschiedlich miteinander kombiniert. Tabelle 6.3 zeigt die acht unterschiedlichen Kombinationen. Es existieren insgesamt drei Dialogflow-Agenten. Zu jedem Themengebiet ein Agent. Diese Agenten werden allesamt miteinander kombiniert. Die zwei Extreme bei dieser Art der

Tourismus	Hotel	WebCam
✓	✓	✓
✓	✓	x
✓	x	✓
✓	x	x
x	✓	✓
x	✓	x
x	x	✓
x	x	x

Tabelle 6.3.: Kombinationen der drei Dialogflow-Agenten.

Evaluation sind, dass ein Fall alle Agenten zur Erzeugung der Aktiven Ontologien und ein anderer Fall gar keinen Agenten verwendet. Durch das Auslassen einzelner Agenten soll evaluiert werden, inwiefern die Aktive Ontologie auch Fragen zu Themengebieten beantworten kann, für die keine Formulierungen vorliegen. In diesen Fällen muss das Wörterbuch die entsprechenden Werte aus der Datenbank anreichern.
Zur Überprüfung werden für alle Kombinationen immer die Fragen von allen Agenten verwendet. Dies bedeutet, auch wenn der Agent „WebCam“ für die Erzeugung der Aktiven Ontologie ausgelassen wurde, so werden die Fragen aus diesem Agenten trotzdem für die Evaluation verwendet.

6.2.3. Durchführung

Zur Durchführung der Evaluation wurde ein Evaluationswerkzeug entwickelt. Zunächst wählt das Werkzeug aus den gegebenen Fragen für die unterschiedlichen Agenten zufällig Fragen aus. In dieser Arbeit wurden zur Evaluation aus jedem Agenten zufällig 100 Fragen verwendet. Die Evaluation dieser Fragen findet einzeln statt. Jede Frage wird als erstes zu Dialogflow, zum Agenten aus dem die Frage exportiert wurde, geschickt. Dies soll mögliche Fehler bei der Erkennung durch Dialogflow ausschließen. Das Ergebnis von Dialogflow wird temporär gespeichert, um es anschließend mit dem Ergebnis der Aktiven Ontologie zu vergleichen. Nachdem die Frage durch Dialogflow verarbeitet wurde, wird dieselbe Frage an die Aktive Ontologie weitergegeben. Das Ergebnis der Aktiven Ontologie wird anschließend mit dem Ergebnis aus Dialogflow verglichen. Stimmen die beiden Ergebnisse überein, so wird das Ergebnis als korrekt gewertet. Im anderen Fall muss das Ergebnis händisch überprüft werden. Wenn in Dialogflow bestimmte Informationen fest definiert sind, so können die extrahierten Parameter in Dialogflow von den gegebenen Informationen in der natürlichsprachlichen Anfrage abweichen. In diesem Fall liefert die Aktive Ontologie ein besseres Ergebnis. Dieser Fall ist vernachlässigbar, da dies in den Testfragen kaum aufgetreten ist. Das folgende Beispiel erläutert diesen Fall.

Beispiel

In diesem Beispiel wird davon ausgegangen, dass die Absicht in Dialogflow die Parameter `Typ` und `Ort` extrahiert. Gegeben sei folgende Anfrage: „Gib mir alle Skipisten“. In Dialogflow ist für die Absicht ein Standardwert für den Ort gegeben, beispielsweise „Ischgl“. Das Ergebnis zu dieser Anfrage in Dialogflow enthält die Parameter „Typ:Skipiste“ und „Ort:Ischgl“. Da in der Anfrage kein Ort genannt wurde, wird der Standardwert verwendet.
Die Aktive Ontologie liefert in diesem Fall nur den Parameter „Typ:Skipiste“. Das Ergebnis der Aktiven Ontologie ist besser, als das Ergebnis aus Dialogflow. Aus der Anfrage kann nur die Information zur Skipiste extrahiert werden und keine Informa-

tion zum Ort.

Zusätzlich zum Komplett-Vergleich (Genauigkeit) mit Dialogflow werden die Parameter einzeln verglichen. Dazu wird überprüft, welche Parameter korrekt, fälschlicherweise oder gar nicht extrahiert wurden. Aus diesen Werten wird die Ausbeute, Präzision und das F1-Maß berechnet.
Zunächst wird auf die Auswertung der Frage durch Dialogflow eingegangen. Anschließend wird die Umwandlung der extrahierten Parameter aus der Aktiven Ontologie in die Form aus Dialogflow vorgestellt. Der letzte Abschnitt dieses Kapitels erläutert, wie die Ergebnisse verglichen und dokumentiert werden.

Dialogflow

Im Folgenden wird auf die Auswertung, beziehungsweise die Ausgabe der extrahierten Parameter in Dialogflow eingegangen. Zunächst wird hierfür ein Beispiel für ein Ergebnis aus Dialogflow mit den extrahierten Parametern dargestellt.

Beispiel

Dieses JSON könnte beispielsweise als Antwort auf die Frage „Ich suche das Restaurant Goldener Löwe“ von Dialogflow ausgegeben werden:

```
{
        "parameters":{
                "type":"Restaurant",
                "name":"Goldener␣Löwe"
        }
}
```

Die Parameter in dieser Frage sind „Restaurant“ und „Goldener Löwe“.

Das Ergebnis aus Dialogflow wird gespeichert, um es anschließend mit dem Ergebnis der Aktiven Ontologie zu vergleichen. Parameternamen, welche Dialogflow zurückgibt, besitzen nicht die Struktur der Daten, wie das obige Beispiel zeigt: Der Parameter „name“ kann nicht direkt auf die Daten in der Datenbank abgebildet werden. Das Feld, welches den Namen eines Elementes speichert, ist unter folgendem Pfad zu finden: „name_translations.de“. Daher müssen für den Vergleich von Dialogflow-Ergebnissen mit den Ergebnissen der Aktiven Ontologien die Parameter auf ein einheitliches Format abgebildet werden. Zur Vereinfachung wird das Ergebnis der Aktiven Ontologie auf das Ergebnis aus Dialogflow abgebildet.

Aktive Ontologie

Nachdem die Frage durch Dialogflow ausgewertet wurde, wird dieselbe Frage an die Aktive Ontologie übergeben. Nach der Verarbeitung durch die Aktive Ontologie werden die erkannten Parameter ausgegeben. Die Namen der Parameter orientieren sich dabei an der Struktur der Daten. Das folgende Beispiel zeigt, wie ein Ergebnis der Aktiven Ontologie aussehen kann. Die Aktive Ontologie gibt allerdings kein JSON aus, sondern eine Liste der Parameter mit ihrem zugehörigen Wert.

Beispiel

Dieses Beispiel zeigt ein mögliches Ergebnis einer Aktiven Ontologie. Als Beispielfrage dient die Frage „Gibt es ein Restaurant in Karlsruhe“. Das Ergebnis kann wie folgt aussehen:

```
LocalBusiness.@type:Restaurant
LocalBusiness.address.addressLocality:Karlsruhe
```

Wie das Beispiel zeigt, spiegeln die Parameternamen die Struktur der Daten in der Datenbank wieder. Für die Evaluation gilt es, diese Parameternamen auf die Parameter aus Dialogflow abzubilden. Hierfür werden Abbildungen definiert. Beispielsweise werden in Dialogflow alle Werte, welche auf die Adresse eines Elementes passen, im Parameter „location“ gespeichert. Um dies mit den Parametern der Aktiven Ontologie umzusetzen, werden reguläre Ausdrücke verwendet.

Beispiel

Um beispielsweise einen Parameter der Aktiven Ontologie auf den Parameter „location“ in Dialogflow abzubilden, wird folgende Abbildungsvorschrift verwendet.

```
.*\.address.*->location
```

Dies bedeutet, alle Parameternamen, welche „.address“ beinhalten, werden auf den Parameternamen „location“ abgebildet.

Nach Abbildung der Parameternamen aus dem ersten Beispiel, sieht das Ergebnis wie folgt aus:

Beispiel

```
{
        "type":"Restaurant",
        "location":"Karlsruhe"
}
```

Die extrahierten Parameter werden zur besseren Handhabung in ein JSON-Objekt eingefügt.

Vergleich der Ergebnisse

Nachdem die Frage an beide Systeme gesendet wurde, müssen die Ergebnisse miteinander verglichen werden. Das Ergebnis aus beiden Systemen sind JSON-Objekte, welche im besten Fall die gleichen Bezeichnungen für die Parameternamen besitzen. Die beiden JSON-Objekte werden anschließend miteinander verglichen. Bei einer vollständigen Übereinstimmung der beiden Objekte wird das Ergebnis der Aktiven Ontologie als korrekt gewertet. Gibt es Unterschiede, muss das Ergebnis der Aktiven Ontologie händisch überprüft werden. Dies ist notwendig, um zu bestimmen ob das Ergebnis der Aktiven Ontologie besser zur Frage passt als das Ergebnis aus Dialogflow. Zusätzlich zum Komplett-Vergleich werden die Parameter einzeln verglichen. Es wird überprüft, wie viele Parameter korrekt, fälschlicherweise oder gar nicht extrahiert wurden. Aus diesen Werten wird die Ausbeute und Präzision, sowie das F1-Maß berechnet.

Ausbeute, Präzision und F1-Maß

Bei der Auswertung des Ergebnisses handelt es sich um eine binäre Klassifikation. Dazu werden die Parameter einzeln betrachtet. Wenn der richtige Sensorknoten reagiert hat und es wurde ein korrekter Wert erkannt, dann handelt es sich um einen korrekt extrahierten Wert (TP). Wurde ein Parameter extrahiert, der nicht in der natürlichsprachlichen Anfrage vorkommt, handelt es sich um einen fälschlicherweise extrahierten Parameter (FP). Der letzte Fall ist, dass ein Parameter nicht extrahiert (FN) wurde, obwohl dieser in der

Eigenschaft	Annotierte Fragen
Öffnungszeiten	239
Adresse	13

Tabelle 6.4.: Anzahl an Fragen zur Erzeugung des Bag-of-Words-Mechanismus

natürlichsprachlichen Anfrage vorkommt. Aus diesen Werten lässt sich anschließend die Ausbeute und Präzision bestimmen. Das F1-Maß wird aus der Ausbeute und Präzision bestimmt. Die Ausbeute (engl. recall) berechnet sich wie folgt:

$$\frac{TP + TN}{TP + FP + TN + FN} \tag{6.1}$$

Für die Präzision (engl. precision) wird folgende Formel verwendet:

$$\frac{TP}{TP + FP} \tag{6.2}$$

Aus der Ausbeute und der Präzision wird das F1-Maß wie folgt berechnet:

$$2 * \frac{precision * recall}{precision + recall} \tag{6.3}$$

6.2.4. Evaluation des Bag-of-Word-Mechanismus

Eine Komponente, welche nicht mit den bisher vorgestellten Methoden der Evaluation getestet werden kann, ist der Bag-of-Word-Mechanismus. Da diese Funktion in Dialogflow nicht unterstützt wird, muss die Evaluation hier händisch durchgeführt werden. Dazu wurden in Dialogflow spezielle Intents modelliert, welche nur eine bestimmte Eigenschaft (beispielsweise „Öffnungszeiten“) anfragen. Im Rahmen der Evaluation wurden zwei Gruppen von Intents angelegt. Eine Gruppe zielt auf die Abfrage der Öffnungszeiten und die zweite Gruppe auf die Frage nach der Adresse ab. Zusätzlich dazu werden auch Kombinationen der beiden Gruppen miteinander evaluiert. Ein Beispiel für eine Kombination aus beidem ist: „Wann hat das Restaurant Goldene Möwe geöffnet und wie lautet die Adresse?“

Die Tabelle 6.4 stellt die für die Erzeugung des Bag-of-Words-Mechanismus verwendeten Fragen dar. Die annotierten Fragen zu den Öffnungszeiten waren bereits ein Bestandteil der verwendeten Dialogflow-Agenten. Zum Simulieren der Kombination von zwei Anfragen, wurden beispielhaft 13 Fragen zu der Eigenschaft „Adresse“ modelliert.
Zum einen werden also die einzelnen Bag-of-Word-Mechanismen der Eigenschaften getestet, aber auch die Kombinationen von beiden miteinander. Die Auswertung muss händisch durchgeführt werden. Das Ergebnis kann nicht automatisiert mit Dialogflow verglichen werden. Dialogflow kann die Fragen den einzelnen Absichten zuordnen, allerdings die kombinierte Variante kann durch Dialogflow nicht erkannt werden. Im Bezug auf diese Funktionalität sind die Aktiven Ontologien besser als Dialogflow.
Als Datengrundlage werden die Tourismusdaten verwendet. Innerhalb dieser Daten finden sich Restaurants, welche sich aufgrund von Öffnungszeiten und Adresse sehr gut zur Evaluation eignen.

6.3. Auswertung der Ergebnisse

Der Komplett-Vergleich mit Dialogflow wurde automatisiert durch das Evaluationswerkzeug ausgewertet. Für den Vergleich der einzelnen Parameter, wurden die Ergebnisse händisch ausgewertet. Aus dem Ergebnis der händischen Auswertung wird das Maß für die

Tourismus	Hotel	WebCam	Komplett	Parameter		
			Genauigkeit	Ausbeute	Präzision	F1-Maß
✓	✓	✓	22%	45%	78%	57%
✓	✓	x	17%	28%	52%	36%
✓	x	✓	24%	46%	79%	58%
✓	x	x	18%	35%	55%	40%
x	✓	✓	16%	33%	51%	43%
x	✓	x	6%	17%	48%	25%
x	x	✓	13%	34%	66%	45%
x	x	x	9%	18%	58%	27%

Tabelle 6.5.: Ergebnisse der evaluierten Kombinationen.

Ausbeute, Präzision und den F1-Wert berechnet.
Zunächst werden die Ergebnisse des Komplett-Vergleichs mit Dialogflow ausgewertet und diskutiert. Anschließend wird auf die Auswertung der einzelnen Parameter eingegangen. Das Ende des Kapitels bildet die Auswertung der Ergebnisse zum Bag-of-Words-Mechanismus.

6.3.1. Vergleich mit Dialogflow

Die Tabelle 6.5 zeigt die Ergebnisse der unterschiedlichen Kombinationen. Es wurden acht Kombinationen evaluiert. Im Folgenden wird zunächst der Komplett-Vergleich ausgewertet. Anschließend werden die Parameter einzeln ausgewertet und analysiert.

Komplett-Vergleich

Die Ergebnisse zeigen, dass der Komplett-Vergleich zwischen Dialogflow und der Aktiven Ontologie zunächst schlecht ausfällt. Allerdings wurden die Aktiven Ontologien für die Evaluation vollständig automatisch erzeugt und es wurden keine manuellen Änderungen vorgenommen. Für den Fall, dass alle Dialogflow-Agenten zur Erzeugung der Aktiven Ontologie verwendet werden, liegt die Genauigkeit bei 45%. Wenn keine Agenten zur Erzeugung verwendet werden, liegt die Genauigkeit bei 9%. Auffällig ist die Genauigkeit, wenn der Hotel-Agent ausgelassen wird. Diese ist höher, als wenn alle Agenten verwendet werden. Dies liegt daran, dass der Hotel-Agent Formulierungen zu speziellen Eigenschaften von Hotels besitzt. Diese Formulierungen überschneiden sich mit Formulierungen aus dem Tourismus-Agenten. Die Folge ist, dass zusätzliche Parameter extrahiert werden, beziehungsweise falsche Parameter die korrekten Parameter überschreiben. Wenn der Hotel-Agent ausgelassen wird, sind die Ergebnisse dementsprechend besser, da die Formulierungen für die speziellen Eigenschaften der Hotels nicht erkannt werden.
Je weniger Agenten bei der Erzeugung der Aktiven Ontologie verwendet werden, desto schlechter wird die Genauigkeit. Dies liegt daran, dass die entsprechenden Formulierungen und Synonyme fehlen. Wenn kein Agent zur Erzeugung der Aktiven Ontologie verwendet wird, gleicht das Extrahieren der Parameter einer Schlüsselwortsuche. Hierbei werden keinerlei Formulierungen beachtet.

Vergleich der Parameter

Neben dem Komplett-Vergleich werden die Parameter einzeln betrachtet. Hierfür wird ausgewertet, wie viele Parameter korrekt extrahiert, fälschlicherweise oder gar nicht extrahiert wurden. Aus diesen Ergebnissen wurde die Ausbeute und die Präzision berechnet. Das F1-Maß stellt den harmonischen Mittelwert der Ausbeute und Präzision dar. Die Ausbeute und Präzision ist in allen Fällen besser, als die Genauigkeit. Der beste Wert (Ausbeute 46%

Öffnungszeiten	Adresse	Genauigkeit
✓	x	65%
x	✓	23%
✓	✓	40%

Tabelle 6.6.: Ergebnisse der Evaluation des Bag-of-Words-Mechanismus.

und Präzision 79%) wird für die Kombination aus Tourismus-Agent und WebCam-Agent erreicht. Ansonsten schwanken die Werte der beiden Metriken zwisch 28% und 35%.

Diskussion der Ergebnisse

Die Ergebnisse der Aktiven Ontologien werden durch zwei Fehlerfälle beeinträchtigt. Im ersten Fehlerfall werden zusätzlich zu den korrekt extrahierten Parameter, falsche Parameter extrahiert. Dies liegt daran, dass für manche Sensorknoten keine Formulierungen definiert wurden. Diese Sensorknoten führen deswegen eine Schlüsselwortsuche durch. Reagiert ein solcher Knoten auf einen gegebenen Wert, so wird dieser Wert dem Ergebnis hinzugefügt. Dieses Verhalten verschlechtert die Ergebnisse.
Beim zweiten Fehlerfall überschreiben falsche Ergebnisse mit einer höheren Konfidenz die richtigen Ergebnisse mit einer niedrigeren Konfidenz. Im Folgenden werden zwei Ontologien betrachtet. Eine kleinere Ontologie mit Sensorknoten ohne Formulierungen und eine größere Ontologie mit Sensorknoten, die Formulierungen enthalten. Für eine natürlichsprachliche Anfrage erkennt die größere Ontologie beispielsweise bestimmte Parameter und extrahiert diese mit einer Konfidenz von 90%. Die kleinere Ontologie erkennt in der natürlichsprachlichen Anfrage ebenfalls Parameter mit einer Konfidenz von 100%. Beide Ontologien geben ihr Ergebnis an einen Sensorknoten weiter, der das beste Ergebnis auswählt. In diesem Fall wird also das falsche Ergebnis der kleinen Ontologie bevorzugt. Dadurch wird ein falsches Ergebnis ausgegeben und die Evaluationswerte verschlechtern sich.
Beim aktuellen Ansatz der Arbeit werden oftmals erwartete Parameter nicht extrahiert oder es werden zu viele extrahiert. Dies spiegelt sich im Verhältnis von der Genauigkeit zur Ausbeute wieder. Im Vergleich zur Genauigkeit ist die Ausbeute sehr viel höher. Allerdings werden oftmals die richtigen Parameter durch falsche Parameter überschrieben.
Bei diesen Evaluationsergebnissen muss bedacht werden, dass die Aktiven Ontologien vollautomatisch erzeugt wurden. Durch kleinere manuelle Änderungen an den Aktiven Ontologien kann das Ergebnis bereits deutlich verbessert werden. Bereits jetzt kann durch diesen Ansatz die manuelle Arbeit für den Entwickler minimiert werden. Auch die Übertragung von Formulierungen aus bekannten Themengebieten auf neue Themengebiete wurde erfolgreich evaluiert.

6.3.2. Bag-of-Words-Mechanismus

Zur Auswertung des Bag-of-Words-Mechanismus wurden die Fragen, die eine bestimmte Eigenschaft anfragen, durch die Aktive Ontologie verarbeitet. Händisch wurde anschließend überprüft, ob die Aktive Ontologie erkannt hat, welche Eigenschaft angefragt werden soll. Tabelle 6.6 zeigt die Ergebnisse der Auswertung. Die Ergebnisse lassen darauf schließen, dass die Auswertung des Bag-of-Words-Mechanismus sehr zuverlässig funktioniert. Sowohl Fragen nach einer speziellen Eigenschaft, als auch nach mehreren Eigenschaften können mit großer Zuverlässigkeit richtig erkannt werden. Die Genauigkeit für Fragen nach der Adresse ist geringer als die Genauigkeit für die Öffnungszeiten, da für die Adresse lediglich 13 Fragen annotiert wurden. Für ein besseres Ergebnis müssten mehr Fragen annotiert werden. Fragen, die beide Eigenschaften anfragen werden mit einer Genauigkeit

von 40% beantwortet. Die Aktive Ontologie erkennt die Frage nach der Eigenschaft an den signifikaten Wörtern (siehe Abschnitt 5.2.2). Wenn für eine Frage nach beispielsweise den Öffnungszeiten die darin enthaltenen Wörter nicht signifikant genug sind, erkennt der Bag-of-Words-Mechanismus nicht, dass nach einer Eigenschaft gefragt werden soll. Dieser Mechanismus reagiert nur, wenn das Gewicht der signifikanten Wörter aufsummiert einen vorher definierten Schwellenwert überschreiten.
Diese Funktion ist ein Alleinstellungsmerkmal und ist in bestehenden intelligenten Assistenten so nicht vorhanden. Aufgrund der Struktur der Aktiven Ontologie, kann das extrahierte Ergebnis direkt in eine Anfrage an die Datenbank umgewandelt werden.

7. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Ansatz zur semi-automatischen Generierung von Aktiven Ontologien aus Datenbankschemata implementiert. Mittels dieser Aktiven Ontologien sollen Parameter aus natürlichsprachlichen Anfragen extrahiert werden. Bei bestehenden intelligenten Assistenten (beispielsweise Dialogflow) werden sehr viele Beispielfragen benötigt. Diese Beispielfragen müssen durch den Entwickler händisch modelliert werden. Dahingehend soll der in dieser Arbeit erstellte Ansatz die manuelle Arbeit reduzieren. Zur Umsetzung dieses Zieles wurden Informationen aus zwei unterschiedlichen Quellen zusammengeführt. Eine Kombination aus Daten einer deduktiven Datenbank bildet mit Formulierungen und Synonymen aus bestehenden Dialogflow-Agenten die Grundlage zur Erzeugung der Aktiven Ontologien.
Die Struktur der Aktiven Ontologie ergibt sich aus der Struktur der Daten in der Datenbank. Dies hat den Vorteil, dass die Namen der extrahierten Parameter die Struktur der anzufragenden Daten beinhalten. Auf diese Weise ist die Erzeugung einer Anfrage an die Datenbank leicht möglich. Zur Unterstützung von Werten, für die keine Synonyme vorliegen, wird ein Wörterbuch verwendet. Dieses reichert Werte aus der Datenbank, die bisher keine Synonyme besitzen, mit Synonymen an. Dadurch können auch Werte für die keine Synonyme in Dialogflow definiert wurden auf unterschiedliche Weise angefragt werden.
Eine Besonderheit der Aktiven Ontologien dieser Arbeit ist der Bag-of-Words-Mechanismus. Hierbei kann die Aktive Ontologie erkennen, wenn ein Benutzer eine bestimmte Eigenschaft erfragen möchte. Dazu muss in Dialogflow eine Absicht mit Fragen zu beispielsweise Öffnungszeiten bestehen. Aus diesen Fragen werden die signifikanten Wörter extrahiert. Erkennt der Bag-of-Words-Mechanismus diese signifikanten Wörter, wird davon ausgegangen, dass der Benutzer nach einer bestimmten Eigenschaft fragt. Diese Information kann dazu verwendet werden, um aus der Datenbank die Öffnungszeiten anzufragen. Dialogflow hingegen kann dies nicht erkennen.
Zur Evaluation wurden unterschiedliche Kombinationen der Dialogflow-Agenten zur Erzeugung der Aktiven Ontologien verwendet. Dies soll simulieren, dass die Aktive Ontologie für Themengebiete ohne Formulierungen, die bereits bestehenden Formulierungen verwendet. Die Datenbasis in der Datenbank ist für alle Kombinationen unverändert. Die Ergebnisse der Aktiven Ontologien wurden mit Dialogflow verglichen. Bei der Verwendung von allen zur Verfügung stehenden Informationen aus Dialogflow lag die Ausbeute bei etwa 45%. Je weniger Informationen aus Dialogflow für die Erzeugung der Aktiven Ontologien verwendet wurden, desto niedriger war die resultierende Ausbeute. Wenn keine Informationen aus Dialogflow verwendet werden, liegt das Ergebnis bei etwa 18%. Dies liegt daran, dass die Aktiven Ontologien entweder mehr Parameter als erwartet oder falsche Parame-

ter extrahiert haben. Für den Vergleich der Ergebnisse wurden drei Varianten verwendet. Eine Variante vergleicht das Ergebnis der Aktiven Ontologie mit Dialogflow. Aus diesem Vergleich wird die Genauigkeit des Ergebnisses bestimmt. Die anderen zwei Varianten betrachten die einzelnen Parameter, welche die Aktive Ontologie extrahiert hat. Aus diesen beiden Varianten wurde die Ausbeute und Präzision bestimmt. Die Rate für die Ausbeute (45%) und Präzision (78%) liegt deutlich über dem Wert der Genauigkeit (22%). Ein Problem der Aktiven Ontologien ist, dass in manchen Fällen falsche Parameter die korrekten Parameter überschreiben, beispielsweise aufgrund einer höheren Konfidenz.
Zur Verbesserung der Ergebnisse, sind im Rahmen dieser Arbeit zwei Konzepte entstanden, die in zukünftigen Arbeiten umgesetzt werden können. In dieser Arbeit wurden die Aktiven Ontologien basierend auf den Obertypen der einzelnen Datenbankelemente erzeugt. Das Resultat sind wenige, dafür sehr große Aktive Ontologien. Eine Möglichkeit wäre, diese Ontologien für jeden Typ zu erzeugen. Dadurch werden viele kleine Ontologien erzeugt, welche die für den Typ spezifischen Eigenschaften erkennen. Die Konfidenzen der erkannten Eigenschaften ist anschließend bei den spezifischen Ontologien größer, als bei einer großen Ontologie. Somit sollten die richtigen Parameter nicht mehr durch falsche Parameter überschrieben werden. Hierbei sollten überschneidende Eigenschaften nicht dupliziert werden. Betrachten wir folgendes Szenario: Eine „Pizzeria“ ist ein Untertyp von „Restaurant“, welches ein Untertyp von „LocalBusiness“ ist. In diesem Fall würde die Ontologie für den Typ „LocalBusiness“ alle allgemeinen Eigenschaften (Beispielsweise „Name“ und „Adresse“) beinhalten. „Restaurant“ und „Pizzeria“ hingegen nur Eigenschaften, welche speziell für diesen Typ (beispielsweise „Angebotene Speisen“) sind. Zusätzlich dazu, existiert eine Verbindung zwischen den Ontologien. Also eine Verlinkung von „Pizzeria“ zu „Restaurant“ und von „Restaurant“ zu „LocalBusiness“. Wird eine Anfrage zu einer Pizzeria gestellt, so erkennen die Auswahlknoten der Pizzeria-AO die entsprechenden speziellen Eigenschaften und die LocalBusiness-AO die allgemeinen Eigenschaften. Auf diese Weise werden viele kleine Ontologien erzeugt, ohne Sensorknoten zu duplizieren. Die Konfidenzen für diese kleinen spezifischen Ontologien sind anschließend höher, als bei großen Ontologien, welche alle Eigenschaften erkennen müssen. Dies könnte verhindern, dass falsch erkannte Eigenschaften mit höherer Konfidenz die richtigen Parameter mit einer geringeren Konfidenz überschreiben.
Eine weitere Möglichkeit zur Optimierung der Aktiven Ontologien ist die Einführung einer weiteren Ebene an Auswahlknoten. Dieser Ansatz kann das Problem lösen, dass zwei Sensorknoten beim gleichen Wert feuern. Wenn zwei Sensorknoten beim gleichen Wert feuern, muss verglichen werden, auf welche Prä- beziehungsweise Postfix-Informationen die Knoten reagiert haben. Es wird davon ausgegangen, dass der Knoten mit dem längeren erkannten Prä- oder Postfix der richtige Knoten ist. Aktuell feuern die Sensorknoten, wenn sie einen Wert erkennen und geben diesen Wert an den Elternknoten weiter. Die Information zu den erkannten Prä- und Postfix-Informationen geht hierbei verloren. Zur Umsetzung dieses Ansatzes geben die Sensorknoten alle erkannten Informationen weiter. Auf oberster Ebene, beispielsweise unterhalb des Wurzelknotens, existiert ein Auswahlknoten, der die entsprechenden Informationen auswertet. Wenn zwei Knoten denselben Wert erkannt haben, so werden die erkannten Prä- und Postfixe der beiden Knoten miteinander verglichen. Der Knoten mit dem längeren Prä- und Postfix ist der richtige Knoten. Dieser Wert wird anschließend weitergereicht.
Diese beiden Ansätze zeigen Optimierungsmöglichkeiten auf, um die Erkennungsrate der Aktiven Ontologien zu verbessern. Zusätzlich werden die falsch erkannten Parameter, gerade durch den zweiten Ansatz, deutlich reduziert.
Dieser Ansatz zeigt, dass es möglich ist die Daten und die Struktur aus der Datenbank mit Formulierungen und Synonymen aus einem Sprachverarbeitungswerkzeug (wie Dialogflow von Google oder Watson von IBM) zu kombinieren, um natürlichsprachliche Anfragen auszuwerten. Um diesen Ansatz produktiv einsetzen zu können, müssen zunächst die

angesprochenen Optimierungen implementiert werden. Anschließend müssen die automatisch erzeugten Ontologien durch einen Entwickler überprüft und eventuelle Anpassungen an den inneren Knoten der Ontologien vorgenommen werden. Zusätzlich müssen für die Sensorknoten ohne Prä- und Postfix-Informationen händisch Formulierungen hinzugefügt werden. Anderenfalls können diese das Ergebnis beeinflussen. Aus der Evaluation kann herausgelesen werden, dass dieser Ansatz bereits ohne Verwendung von Formulierungen eine Ausbeute von ca 20% erreicht. Das bedeutet, mit diesem Ansatz kann mit dem jetzigen Stand ein fünftel der Arbeit eines Entwicklers eingespart werden. Mit den angesprochenen Optimierungen kann dieser Anteil weiter gesteigert werden.

Literaturverzeichnis

[ABMP08] Adida, Ben ; Birbeck, Mark ; McCarron, Shane ; Pemberton, Steven: RDFa in XHTML: Syntax and Processing. In: *Recommendation, W3C* 7 (2008) (zitiert auf Seite 13).

[Ang14] Angele, Jürgen: OntoBroker. In: *Semantic Web* 5 (2014), Nr. 3, S. 221–235 (zitiert auf Seite 15).

[BF08] Bishop, Barry ; Fischer, Florian: Iris-Integrated Rule Inference System. In: *International Workshop on Advancing Reasoning on the Web: Scalability and Commonsense (ARea 2008)*, 2008 (zitiert auf Seite 14).

[BL16] Blersch, Martin ; Landhäusser, Mathias: Easier: An Approach to Automatically Generate Active Ontologies for Intelligent Assistants. In: *Proceedings of the 20th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI 2016)*. Orlando, FL, USA, Juli 2016 (zitiert auf den Seiten 3, 24 und 92).

[BL18] Blersch, Martin ; Landhäusser, Mathias: Semi-Automatic Generation of Active Ontologies from Web Forms for Intelligent Assistants. In: *Proceedings of the IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. Gothenburg, Sweden, Mai 2018 (zitiert auf den Seiten 3, 24 und 92).

[CGT89] Ceri, S. ; Gottlob, G. ; Tanca, L.: What You Always Wanted to Know about Datalog (and Never Dared to Ask). In: *IEEE Transactions on Knowledge and Data Engineering* 1 (1989), März, Nr. 1, S. 146–166. `http://dx.doi.org/10.1109/69.43410`. – DOI 10.1109/69.43410. – ISSN 10414347 (zitiert auf Seite 14).

[CGW89] Ceri, Stefano ; Gottlob, Georg ; Wiederhold, Gio: Efficient Database Access from PROLOG. In: *IEEE Transactions on Software Engineering* 15 (1989), Nr. 2, S. 153–164 (zitiert auf Seite 14).

[Con14] Consortium, World Wide W.: JSON-LD 1.0: A JSON-Based Serialization for Linked Data. (2014) (zitiert auf Seite 13).

[DHR08] Dahab, Mohamed Y. ; Hassan, Hesham A. ; Rafea, Ahmed: TextOntoEx: Automatic Ontology Construction from Natural English Text. In: *Expert Systems with Applications* 34 (2008), Nr. 2, S. 1474–1480 (zitiert auf den Seiten 20, 21 und 85).

[GBC06] Guzzoni, Didier ; Baur, Charles ; Cheyer, Adam: Active: A Unified Platform for Building Intelligent Web Interaction Assistants. In: *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, IEEE Computer Society, 2006, S. 417–420 (zitiert auf den Seiten 3, 24 und 91).

[GBC07] Guzzoni, Didier ; Baur, Charles ; Cheyer, Adam: Modeling Human-Agent Interaction with Active Ontologies. In: *AAAI Spring Symposium: Interaction Challenges for Intelligent Assistants*, 2007, S. 52–59 (zitiert auf den Seiten xi, 3, 4, 6 und 24).

[GBM16] Guha, R. V. ; Brickley, Dan ; Macbeth, Steve: Schema.Org: Evolution of Structured Data on the Web. In: *Communications of the ACM* 59 (2016), Januar, Nr. 2, S. 44–51. http://dx.doi.org/10.1145/2844544. – DOI 10.1145/2844544. – ISSN 00010782 (zitiert auf den Seiten xi, 12 und 14).

[GCB06] Guzzoni, Didier ; Cheyer, Adam ; Baur, Charles: Active, a Platform for Building Intelligent Software. In: *Computational Intelligence* 5 (2006) (zitiert auf den Seiten xi, 3, 24, 25 und 91).

[GHH+82] Grosz, Barbara ; Haas, Norman ; Hendrix, Gary ; Hobbs, Jerry ; Martin, Paul ; Moore, Robert ; Robinson, Jane ; Rosenschein, Stanley: DIALOGIC: A Core Natural-Language Processing System, Association for Computational Linguistics, 1982, S. 95–100 (zitiert auf Seite 93).

[GMN84] GALLAIRE, HERVI ; MINKER, JACK ; NICOLAS, JEAN-MARIE: Logic and Databases: A Deductive Approach. In: *Computing Surveys* 16 (1984), Nr. 2, S. 33 (zitiert auf Seite 14).

[Gre69] Green, Cordell: Theorem Proving by Resolution as a Basis for Question-Answering Systems. In: *Machine intelligence* 4 (1969), S. 183–205 (zitiert auf Seite 14).

[HF97] Hamp, Birgit ; Feldweg, Helmut: Germanet-a Lexical-Semantic Net for German. In: *Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications* (1997) (zitiert auf Seite 34).

[KK06] Kreuzer, Martin ; Kühling, Stefan: *Logik Für Informatiker*. Pearson Studium, 2006 (zitiert auf Seite 15).

[KL89] Kifer, Michael ; Lausen, Georg: F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In: *ACM SIGMOD Record* Bd. 18, ACM, 1989, S. 134–146 (zitiert auf Seite 15).

[KLW95] Kifer, Michael ; Lausen, Georg ; Wu, James: Logical Foundations of Object-Oriented and Frame-Based Languages. In: *Journal of the ACM (JACM)* 42 (1995), Nr. 4, S. 741–843 (zitiert auf Seite 15).

[KRS15] Krötzsch, Markus ; Rudolph, Sebastian ; Schmitt, Peter H.: A Closer Look at the Semantic Relationship between Datalog and Description Logics. In: *Semantic Web* 6 (2015), Nr. 1, S. 63–79 (zitiert auf Seite 14).

[LDW05] Li, Man ; Du, Xiao-Yong ; Wang, Shan: Learning Ontology from Relational Database. In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference On* Bd. 6, IEEE, 2005, S. 3410–3415 (zitiert auf den Seiten xiii, 22 und 89).

[LHL+98] Ludäscher, Bertram ; Himmeröder, Rainer ; Lausen, Georg ; May, Wolfgang ; Schlepphorst, Christian: Managing Semistructured Data with Florid: A Deductive Object-Oriented Perspective. In: *Information systems* 23 (1998), Nr. 8, S. 589–613 (zitiert auf Seite 14).

[LPF+02] Leone, Nicola ; Pfeifer, Gerald ; Faber, Wolfgang ; Calimeri, Francesco ; Dell'Armi, Tina ; Eiter, Thomas ; Gottlob, Georg ; Ianni,

Giovambattista ; Ielpa, Giuseppe ; Koch, Christoph ; Perri, Simona ; Polleres, Axel: The DLV System. Version: 2002. http://dx.doi.org/10.1007/3-540-45757-7_50. In: Goos, G. (Hrsg.) ; Hartmanis, J. (Hrsg.) ; van Leeuwen, J. (Hrsg.) ; Flesca, Sergio (Hrsg.) ; Greco, Sergio (Hrsg.) ; Ianni, Giovambattista (Hrsg.) ; Leone, Nicola (Hrsg.): *Logics in Artificial Intelligence* Bd. 2424. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002. – DOI 10.1007/3–540–45757–7_50. – ISBN 978–3–540–44190–8 978–3–540–45757–2, S. 537–540 (zitiert auf Seite 14).

[Mil95] Miller, George A.: WordNet: A Lexical Database for English. In: *Communications of the ACM* 38 (1995), Nr. 11, S. 39–41 (zitiert auf Seite 34).

[MON08] Minock, Michael ; Olofsson, Peter ; Näslund, Alexander: Towards Building Robust Natural Language Interfaces to Databases. In: Kapetanios, Epaminondas (Hrsg.) ; Sugumaran, Vijayan (Hrsg.) ; Spiliopoulou, Myra (Hrsg.): *Natural Language and Information Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–69858–6, S. 187–198 (zitiert auf den Seiten 23 und 24).

[Moo18] Moore, Susan: *Gartner Says 25 Percent of Customer Service Operations Will Use Virtual Customer Assistants by 2020*. https://www.gartner.com/en/newsroom/press-releases/2018-02-19-gartner-says-25-percent-of-customer-service-operations-will-use-virtual-customer-assistants-by-2020, Februar 2018 (zitiert auf Seite 1).

[MRS08] Manning, Christopher D. ; Raghavan, Prabhakar ; Schütze, Hinrich: Scoring, Term Weighting and the Vector Space Model. In: *Introduction to information retrieval* 100 (2008), S. 2–4 (zitiert auf den Seiten 16 und 17).

[MS01] Maedche, A. ; Staab, S.: Ontology Learning for the Semantic Web. In: *IEEE Intelligent Systems* 16 (2001), März, Nr. 2, S. 72–79. http://dx.doi.org/10.1109/5254.920602. – DOI 10.1109/5254.920602. – ISSN 1541–1672 (zitiert auf den Seiten 19 und 82).

[MV01] Maedche, Alexander ; Volz, Raphael: The Ontology Extraction & Maintenance Framework Text-To-Onto. In: *International Conference on Data Mining (ICDM), San Jose, USA, November 29 - December 2, 2001*, IEEE, Los Alamitos (CA), 2001 (zitiert auf den Seiten 19, 20 und 79).

[PRGBAL+13] Pazos R, Rodolfo A. ; González B, Juan J. ; Aguirre L, Marco A. ; Martínez F, José A. ; Fraire H, Héctor J.: Natural Language Interfaces to Databases: An Analysis of the State of the Art. In: *Recent Advances on Hybrid Intelligent Systems* (2013), S. 463–480 (zitiert auf den Seiten 2 und 22).

[RAC+10] Rao, Gauri ; Agarwal, Chanchal ; Chaudhry, Snehal ; Kulkarni, Nikita ; Patil, Dr S H.: NATURAL LANGUAGE QUERY PROCESSING USING SEMANTIC GRAMMAR. 02 (2010), Nr. 02, S. 5 (zitiert auf den Seiten 23 und 95).

[SHA11] Santoso, Heru A. ; Haw, Su-Cheng ; Abdul-Mehdi, Ziyad.T.: Ontology Extraction from Relational Database: Concept Hierarchy as Background Knowledge. In: *Knowledge-Based Systems* 24 (2011), April, Nr. 3, S. 457–464. http://dx.doi.org/10.1016/j.knosys.2010.11.003. – DOI 10.1016/j.knosys.2010.11.003. – ISSN 09507051 (zitiert auf den Seiten xi, xiii, 21 und 87).

[STS99] Schiller, Anne ; Teufel, Simone ; Stockert, Christine: Guidelines fur das Tagging deutscher Textcorpora mit STTS (Kleines und großes Tagset). (1999), S. 87 (zitiert auf den Seiten xiii und 18).

[Tra18] Tractica: *Enterprise Virtual Digital Assistant Users to Surpass 1 Billion by 2025 | Tractica*. Januar 2018 (zitiert auf Seite 1).

[Vou03] Voutilainen, Atro: Part-of-Speech Tagging. In: *The Oxford handbook of computational linguistics* (2003), S. 219–232 (zitiert auf Seite 18).

Anhang

A. First Appendix Section

A.1. Mögliche weitere Funktionen

Im Verlaufe der Implementierung sind zwei Funktionen aufgefallen, die in diesem Verfahren ergänzt werden könnten. Zum einen wird für jeden extrahierten Typ aus der Datenbank der Supertyp gespeichert. Dies könnte zur Optimierung der Aktiven Ontologien genutzt werden. Zum anderen wäre die Einführung eines generellen Formats zur Speicherung von Telefonnummern in der Aktiven Ontologie möglich.

Supertyp

Mittels der Supertyp-Eigenschaft können Duplikate eliminiert werden. Aktuell werden für jeden Typ alle Eigenschaften in die Aktive Ontologie übertragen.

Beispiel

Ein „Restaurant“ ist ein Subtyp von „LocalBusiness“. Der Supertyp von „Restaurant“ ist also „LocalBusiness“. Die Struktur eines LocalBusiness besitzt beispielsweise die Eigenschaften „name“ und „description“. Restaurants besitzen die Eigenschaft „name“, „description“, „address“. Aktuell werden für beide Typen separate Ontologien erzeugt, welche alle Eigenschaften beinhalten.

Optimierung anhand des Supertyps wäre möglich, indem die Aktive Ontologie des Restaurants (aus dem Beispiel) nur die Eigenschaft „address“ besitzt und einen Verweis auf die Aktive Ontologie des „LocalBusiness“. Auf diese Weise würden die doppelten Einträge in unterschiedlichen Ontologien eliminiert werden. Eine Unterontologie enthält nur noch die Eigenschaften, welche die Oberontologie noch nicht besitzt und zusätzlich einen Verweis auf die Oberontologie.

Generelles Format für Telefonnummern

Ein weiterer Punkt betrifft Telefonnummern. Aktuell werden Telefonnummern in die entsprechende Datei zu der Eigenschaft geschrieben. Diese werden allerdings nicht in einem einheitlichen Format gespeichert. Es existieren viele verschieden Formate um Telefonnummern zu speichern. Suchvorgänge nach Telefonnummern sind dadurch nahezu unmöglich,

außer es ist bekannt, wie die Telefonnummer in der Datenbank gespeichert wurde. Daher wäre ein generelles Format zur Beschreibung von Telefonnummern notwendig. Wie bei den Abbildungsknoten dient das generelle Format als Synonym für den Wert in der Datenbank. Ein spezieller Sensorknoten müsste die ankommenden Formate in das generelle Format umwandeln und anschließend wird dieses für die Suche auf die Telefonnummer in der Datenbank abgebildet. Auf diese Weise wäre es möglich Telefonnummern anzufragen.

B. Literaturanalysen

B.1. The Ontology Extraction Maintenance Framework Text-To-Onto

[MV01]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 Zentrales Thema des Papers ist die Extraktion einer Ontologie aus Text
- I 2 Zur Erzeugung der Konzepte wird eine Referenzontologie verwendet
- I 3 Umfangreiche Benutzerunterstützung notwendig
- I 4 Generische Ontologie muss zurückgeschnitten werden, je nachdem welche Konzepte für den aktuellen Anwendungsfall verwendet werden
- I 5 Zusätzlich wird die Ontologie kontinuierlich durch neue Einträge erweitert

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

- D 1 Es existierte keine allgemeine Rahmenarchitektur zur Ontologieextraktion und Wartung aus Text
- D 2 In der Arbeit genannte Ansätze können nicht mit nicht-taxonomischen konzeptuellen Relationen umgehen und beinhalten keine Möglichkeiten der Modellierung durch einen Benutzer

Lösungen

Was sind die eigenen Lösungen der Autoren?

- L 1 Lernprozess berücksichtigt immer die vorhandenen konzeptionellen Strukturen
- L 2 Allerdings auch Neuaufbau möglich, wenn keine konzeptionellen Strukturen verfügbar
- L 3 Ontologieextraktion
  - a) Ergebnis der Ontologieextraktions-Algorithmen sind Elemente der Ontologiestruktur
  - b) Benutzer kann folgende Informationen extrahieren
    - i. Lexikalische Einträge die sich auf Konzepte beziehen
    - ii. Konzepthierarchie
    - iii. Lexikalische Zeichen für Relationen und nicht-taxonomische Relationen
  - c) statische und datamining Ansätze
    - i. Mehrere Algorithmen implementiert
    - ii. Einer davon erfasst Termfrequenzen aus Text
    - iii. Ausgabe dieses Algorithmus kann zur Erzeugung der Konzepte und den zugehörigen lexikalischen Zeichen genutzt werden
    - iv. Zur Ableitung der Konzepthierarchie wurde ein hierarchischer Clustering-Algorithmus verwendet
    - v. Dieser greift auf Hintergrundwissen aus vorhandenen ontologischen Entitäten zugreift, um extrahierte Hierarchie zu kennzeichnen

vi. Um nicht-taxonomische konzeptuelle Beziehungen zu erkennen, wurde ein Algorithmus implementier, der auf häufigen Kopplungen von Konzepten basiert
vii. Dazu werden sprachlich verwandte Wortpaare identifiziert
viii. eigentlicher Algorithmus ist leicht modifizierte Version des Standard-Assoziationsregel-Algorithmus, der auf mehreren Sätzen arbeitet
ix. Hintergrundwissen aus Taxonomie wird verwendet, um Beziehungen auf geeigneter Abstraktionsebene vorzuschlagen
x. Umfangreiche Benutzerunterstützung für die Kontrolle der Algorithmenparameter
xi. Extrahierte Relationen werden dem Benutzer präsentiert
xii. Benennung der Beziehungen wird gegenwärtig durch Benutzer manuell durchgeführt

d) Ontologieextraktion: Musterbasierte Ansätze
i. heuristische Methoden basierend auf regulären Ausdrücken
ii. Muster zur Erkennung von taxonomischen wie nicht-taxonomischen Beziehungen
iii. Nachteile: Muster müssen definiert werden; zeitaufwendig

L 4 Wartung & Pflege der Ontologie

a) Ontologien entwickeln sich weiter
b) Im Rahmen dieser Arbeit besteht die Pflege der Ontologie aus zwei Teilaufgaben
c) Ontologie Zurückschneiden: Entfernen von Elementen aus der Ontologie, die für gegebene Anwendungsdomäne nicht relevant sind
d) Ontologie Verfeinerung: Erlernen der Bedeutung von unbekannten Wörtern, Erkennung wichtiger Wörter, die sich nicht in der Ontologie wiederspiegeln
e) Zurückschneiden der Ontologie
i. Beschneidung notwendig, wenn generische Ontologie zu Domäne verwendet wird
ii. Es wird ein frequenzbasierter Ansatz verwendet um Konzeptfrequenzen in einem Korpus zu bestimmen
iii. Entitäten, die in gegebenem Korpus häufig vorkommen, werden als Bestandteil einer bestimmten Domäne betrachtet
iv. Im Gegensatz zur Ontologieextraktion reicht Häufigkeit ontologischer Entitäten nicht aus
v. Zur Bestimmung der Domänenrelevanz, werden die ontologischen Entitäten mit Häufigkeiten aus einem generischen Korpus verglichen
vi. Benutzer kann mehrere Relevanzmaße für Frequenzberechnung auswählen
vii. Berechnete Häufigkeiten werden verwendet, um relative Relevanz jedes in der Ontologie enthaltenen Konzepts zu bestimmen
viii. Alle bestehenden Konzepte und Beziehungen, die im domänenspezifischen Korpus häufiger vorkommen, bleiben in der Ontologie
f) Verfeinerung der Ontologie
i. Wichtiger Aspekt der Pflege ist inkrementelle Erweiterung einer Ontologie um Konzepte und zugehörige lexikalische Einträge
ii. Extraktionsalgorithmen können auch zur Erweiterung der Ontologie verwendet werden
iii. Algorithmen können jedoch unbekannte Wörter nicht mit bestehenden Konzepten verbinden oder Synonyme desselben Konzepts erkennen

iv. Verfeinerungsansatz basiert auf Annahme: Unbekannte Wörter können ähnliches konzeptionelles Verhalten wie bekannte Wörter haben

v. bsp. „Wochenendausflug“ gleiches konzeptionelles Verhalten wie das Konzept „EXKURSION“

B.2. Ontology learning for the semantic web

[MS01]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

I 1 Rahmenarchitektur zum Extrahieren einer Ontologie
I 2 folgende Schritte: Ontologie-Import, -Extraktion, -Reduzierung, -Verfeinerung und -Bewertung
I 3 Bietet außerdem Werkzeuge zur Ontologiemodellierung
I 4 Ontologie kann aus Freitext, Wörterbüchern und vererbten Ontologien extrahiert werden
I 5 Rahmenarchitektur bietet folgende Schritte:
   a) Verschmelzen existierender Strukturen oder definieren von Abbildungsregeln zwischen diesen Strukturen erlaubt das wiederverwenden und importieren von bestehenden Ontologien
   b) Ontologieextraktion modelliert wichtige Teile der Zielontologie
   c) Ontologiereduzierung um besser auf den eigentlichen Anwendungsfall angepasst zu sein
   d) Verfeinerung der Ontologie
I 6 Dieser Zyklus kann wiederholt werden um neue Domänen zu erschließen

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

D 1 Nachteil einiger Ansätze, die mit Hilfe von maschinen-lernenden Techniken Wissen akquirieren, ist der starke Fokus auf strukturierten Wissen oder Datenbanken

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

P 1 Vollautomatisierte Extraktion einer Ontologie in naher Zukunft nicht möglich
P 2 Daher ist der Ansatz der Autoren eine semi-automatische Lösung, bei der der Mensch manche Entscheidungen selber treffen muss

Lösungen

Was sind die eigenen Lösungen der Autoren?

L 1 Importieren & Wiederverwenden bestehender Ontologien
   a) Im ersten Teil werden die Strukturen des Schemas identifiziert
   b) Genereller Inhalt muss mit Domänenexperten diskutiert werden
   c) Jede Wissensquelle muss separat importiert werden
   d) Im zweiten Teil werden die importierten Strukturen zusammengeführt/aufeinander abgestimmt
   e) Ziel ist gemeinsame Basis zu bilden auf der nachfolgenden Phasen arbeiten können

f) Import-/Wiederverwendung ist am schwierigsten zu verallgemeinern

L 2 Extraktion

a) Modelliert Hauptteile oder große Teile einer neuen Ontologie-Subdomäne

b) Wesentlicher Teil der gesamten Ontologie-Lern-Aufgabe wird in Extraktionsphase abgedeckt

c) Lexikalische Eintrage und Konzeptextrahierung
   i. Text-to-Onto verarbeitet Webdokumente auf morphologischer Ebene
   ii. Inklusive Mehrwortbegriffen wie „database reverse engineering“ durch n-Gramme (einfache statistikbasierte Technik)
   iii. Basierend auf dieser Textvorverarbeitung werden Termextraktionstechniken angewandt, welche auf (gewichteten) statistischen Häufigkeiten basieren um neue lexikalische Einträge vorzuschlagen
   iv. Oft folgt der Ontologieentwickler dem Vorschlag und fügt einen neuen lexikalischen Eintrag in die Ontologie ein
   v. Neuer lexikalischer Eintrag kommt ohne zugehöriges Konzept
   vi. Ontologieentwickler muss entscheiden ob ein neues Konzept eingeführt werden soll, oder ob es mit einem bestehenden Konzept verknüpft wird

d) Hierarchisches Konzeptclustering
   i. Mittels eines Lexikons und einer Menge von Konzepten, ist der nächste Schritt die taxonomische Konzeptklassifizierung
   ii. Anwendbare Methode, hierarchisches Clustering, hier werden Ähnlichkeiten von Elementen ausgenutzt, um eine Hierarchie von Elementkategorien vorzuschlagen
   iii. Ähnlichkeitsmaß für Eigenschaften von Objekten berechnen
   iv. Term-Adjazend oder syntaktische Beziehungen zwischen Termen sind eminent wichtig um semantische Hierarchie von Konzepten zu induzieren, die mit diesen Termen in Beziehung stehen

e) Wörterbücher
   i. Maschinenlesbare Wörterbücher sind für viele Domänen verfügbar
   ii. Obwohl innere Struktur größtenteils freier Text ist, werden wenige Muster verwendet, um Textdefinitionen zu geben
   iii. Wörterbücher weisen hohen Grad an Regelmäßigkeit auf, kann genutzt werden, um Domänenkonzeptualisierung zu extrahieren

f) Assoziationsregeln
   i. Typischerweise werden Assoziations-regel-lern-Algorithmen für prototypische Anwendungen des Data Mining verwendet
   ii. Verallgemeinerter Algorithmus enthält Beschreibungen auf der entsprechenden Taxonomieebene
   iii. Text-to-Onto verwendet modifizierten Algorithmus um Beziehungen zwischen Konzepten zu finden
   iv. Gegebene Klassenhierarchie dient als Hintergrundwissen
   v. Paare von syntaktisch verwandten Konzepten dienen als Eingabe für den Algorithmus
   vi. Algorithmus erzeugt Assoziationsregeln, die die Relevanz verschiedener Regeln beim Auf-/Absieg der Taxonomie vergleichen
   vii. Algorithmus schlägt die relevantesten binären Regeln vor, um Beziehungen in der Ontologie zu modellieren

L 3 Reduzierung der Ontologie

a) Allgemeines Problem der Modellierung in verschiedenen Disziplinen ist das Gleichgewicht zwischen Vollständigkeit und Mangel an Domänenmodellen

b) Vollständigkeit nicht handhabbar und zu rechenintensiv

c) Beschränken auf seltenstes Modell beschränkt Aussagekraft zu sehr

d) Problem des Reduzierung auf zwei Wege

   i. Erstens, Auswirkungen der Beschneidung einzelner Teile auf den Rest der Ontologie

   ii. Zweitens, Strategien für das Vorschlagen von Elementen die behalten oder entfernt werden sollen

   iii. Bei mehreren gegebenen anwendungsspezifischen Dokumenten gibt es mehrere Strategien die Ontologie zu beschneiden

   iv. Basieren auf absoluten oder relativen Anzahl von Termhäufigkeiten kombiniert mit dem Hintergrundwissen der Ontologie

L 4 Verfeinerung der Ontologie

a) Verfeinerung ähnlich zur Extraktion

b) Feinabstimmung der Zielontologie

c) kann Daten aus einer konkreten Semantic Web Applikation verwenden z.B. Protokolldateien von Benutzeranfragen

d) Anpassung und Verfeinerung in Bezug auf Benutzeranforderungen spielt wichtige Rolle bei der Akzeptanz und Weiterentwicklung

e) Prinzipiell können die selben Algorithmen wie für Extraktion verwendet werden

f) Bei Verfeinerung müssen jedoch die bestehende Ontologie und ihre bestehenden Verbindungen im Detail betrachtet werden

B.3. TextOntoEx: Automatic ontology construction from natural English text

[DHR08]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 TextOntoEx erzeugt die Ontologie aus natürlichen Domänentexten mit Hilfe von semantischen Mustern
- I 2 TextOntoEx ist Kette zwischen linguistischer Analyse und Ontologie-Engineering
- I 3 Analysiert natürlichen Domaintext um nicht-taxonomische Beziehungen einer bestimmten Domäne zu extrahieren
- I 4 Beziehung zwischen zwei Konzepten kann bspw. durch ein Verb bestehen
- I 5 Semantische Muster sind eine generische formale Repräsentation von natürlichen Textfragmenten

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

- D 1 Meisten bestehenden Arbeiten unterstützen die Konstruktion von ontologischen Beziehungen (Taxonomie, Gleichheit) aber nicht die Konstruktion von Domänenbeziehungen, nicht-taxonomischen konzeptuellen Beziehungen

Lösungen

Was sind die eigenen Lösungen der Autoren?

- L 1 Struktur der semantischen Muster
  - a) Abstrakte ontologische Klasse
  - b) Verbgruppe
  - c) konstanter Textausdruck (Präpositionen und Konjunktionen)
  - d) Optionale Elemente
- L 2 Alle Elemente sind nicht-terminierende Elemente außer dem konstanten Textausdruck
- L 3 Mittels semantischer Muster können sowohl nicht-taxonomische, als auch taxonomische Relationen gefunden werden
- L 4 Negativ: semantische Muster müssen definiert werden
- L 5 Extrahieren der Ontologie
  - a) Jeder Eingabeabsatz eines Textes einer Domäne wird analysiert
  - b) Es wird sichergestellt, dass der ausgewählte Absatz keine negierten Wörter enthält
  - c) Eingabeabsatz wird in ein oder mehrere musterähnliche Formate konvertiert, das Zwischenformat
  - d) Anschließend wird nach exakten Übereinstimmungen zwischen dem Zwischenformat und den definierten semantischen Mustern gesucht
  - e) Ausdrücke die nicht von Interesse sind werden entfernt
  - f) Sätze können mit mehr als einem semantischen Muster übereinstimmen
  - g) Jedes passende Muster fügt Informationen zur extrahierten Ontologie hinzu.

Nachweise

Welche Nachweise (Evidence) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

N 1 Kleiner Testkorpus wurde zur Evaluierung verwendet (65 Sätze)
N 2 Semantische Relationen wurden manuell extrahiert und mit den Ergebnissen der semantischen TextOntoEx verglichen
N 3 Es gab keine falschen Übereinstimmungen
N 4 Präzision war daher bei 100%
N 5 Rückrufquote bei ca 54%
N 6 Nicht übereinstimmende Muster sind auf Mängel in den gespeicherten semantischen Mustern zurückzuführen

B.4. Ontology extraction from relational database: Concept hierarchy as background knowledge

[SHA11]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 Auf Grundlage einer bestehenden Konzepthierarchie werden die Daten aus der relationalen Datenbank extrahiert
- I 2 Dieses Hintergrundwissen wird von Domänenexperten bereitgestellt
- I 3 Anhand von Ähnlichkeitsmaßen wird nach übereinstimmenden Attributen gesucht
- I 4 Identifizierte Attribute werden verwendet, um Menge der in Frage kommenden Tabellen einzugrenzen

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

- D 1 Meiste Ansätze schlagen nur das Mapping von der Relationalen Datenbank (RDB) zu einem Ontologie-Abbildungs-Ansatz vor, ohne dass sich Wissen in der Datenbank befindet
- D 2 Alle Datenbankkomponenten werden ohne einen Datenauswahlmechanismus konvertiert
- D 3 Da viele Probleme, wie Datenredundanz, immer noch in existierenden Datenbanken auftreten ist die Erstellung einer Ontologie über RDB ohne Datenvorbereitungsphase unproduktiv
- D 4 Umwandlungskosten müssen im Hinblick auf die rasche Zunahme der Datenbankgröße minimiert werden

Lösungen

Was sind die eigenen Lösungen der Autoren?

- L 1 Semi-automatische Ontologiekonstruktion zum Bauen eines initialen Modells einer OWL Ontologie
- L 2 Identifizieren der relevanten Datenmenge benötigt Spezifizierung, auf welche Tabellen, Attribute und Tupel zugegriffen werden soll
- L 3 Distanzbasierte Merkmalsextraktion zur Bestimmung der relevanten Datenmengen (Ähnlichkeit zw. Merkmalswert und Menge an Texteingabe)
- L 4 Konzepthierarchie dient als Grundlage der Datenextraktion
- L 5 Dieses Hintergrundwissen wird von Domänenexperten bereitgestellt; repräsentiert entsprechende Ebenen in der Konzepthierarchie
- L 6 Algorithmus verwendet Zieltabelle und Zielattribute, um über Fremdschlüsselverbindung Verknüpfungsketten mit anderen Tabellen zu erstellen
- L 7 Levenshtein-Distanz als Metrik zur Bestimmung zugehöriger Merkmale
- L 8 Mit Eingabetext wird in einer Menge von Relationen / Tabellen einer gegebenen Datenbank nach übereinstimmenden Attributen gesucht
- L 9 Identifizierte Attribute werden verwendet, um die Menge der möglichen Tabellen zu identifizieren
- L 10 Primärschlüssel ermittelt ob entsprechende Tabelle aus starkem Entitätssatz stammt
- L 11 Fremdschlüssel gibt Beziehung zw. Tabellen an

L 12 Schlüsselbasierte Beziehung manchmal unproduktiv
L 13 Subsumtionsbeziehung in Konzepthierarchie wird in den Prozess der Bestimmung der Konzeptbeziehungen einbezogen

B.5. Learning Ontology From Relational Database

[LDW05]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

I 1 OWL Ontologie aus Daten einer relationalen Datenbank extrahieren
I 2 Unter Verwendung einer Gruppe von Lernregeln anstatt eines Zwischenmodells
I 3 OWL Ontologie enthält Klassen, Eigenschaften, Eigenschaftscharakteristiken, Kardinalität und Instanzen

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

D 1 Andere existierende Verfahren können nicht alle Eigenschaften auf einmal extrahieren:
   a) Klassen, Eigenschaften, Eigenschaftscharakteristiken, Kardinalität und Instanzen
D 2 Andere Verfahren benötigen bspw. ein Zwischenmodell

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

P 1 Das Schema der relationalen Datenbank muss zumindest in der dritten Normalform sein

Lösungen

Was sind die eigenen Lösungen der Autoren?

L 1 Regeln zum erlernen der Ontologie sind in fünf Gruppen aufgeteilt.
L 2 Regeln zum Lernen von
   a) Klassen
   b) Eigenschaften
   c) Hierarchie
   d) Kardinalität
   e) Instanzen
L 3 Regeln zum Lernen von Klassen
   a) Klasse kann aus Informationen von mehreren Beziehungen oder von einer Beziehung abgeleitet werden
   b) Regel 1 integriert Informationen mehrerer Beziehungen in einer Klasse, wenn diese Beziehung ein Element beschreiben
   c) Regel 2 sagt, wenn eine Relation ein Element beschreibt, anstatt eine Beziehung zwischen Relationen, dann kann daraus eine Klasse abgeleitet werden
L 4 Regeln zum Lernen von Eigenschaften und deren Charakteristiken

a) Zwei Arten von Eigenschaften in OWL
   i. Objekteigenschaften
      A. Regeln 3,4,5,6
   ii. Eigenschaften eines Datentyps
      A. Regel 7

b) Regel 3, 4 und 5 lernen Objekteigenschaften mittels der binären Beziehung zwischen Relationen

c) Regel 7 zeigt, dass jedes Attribut in Relationen, welches nicht in eine Objekteigenschaft konvertiert werden kann, in eine Datentypeigenschaft umgewandelt werden kann

L 5 Regeln zum Lernen der Hierarchie

a) Wenn zwei Relationen in der Datenbank eine vererbende Beziehung haben, dann können diese Klassen oder Eigenschaften in einer Hierarchie organisiert werden (Regel 8)

L 6 Regeln zum Lernen der Kardinalität

a) Die Kardinalität der Eigenschaften kann aus den Einschränkungen von Attributen in Relationen gelernt werden (Regeln 9, 10 und 11)

L 7 Regeln zum Lernen der Instanzen

a) Für eine Klasse C_i, bestehen die Instanzen aus Tupel in den Relationen die zu C_i gehören

b) Relationen zwischen Instanzen werden unter Verwendung der Informationen der Fremdschlüssel hergestellt

L 8 Funktionsweise

a) Eingabe der Rahmenarchitektur sind Daten aus einer relationalen Datenbank

b) Mittels eines Datenbankanalyseprogramms werden Schemainformationen extrahiert (z.B. Primärschlüssel, Fremdschlüssel, Abhängigkeiten, usw.)

c) Anschließend generiert der Ontologiegenerator aus den Schemainformationen und den Regeln die Ontologie

B.6. Active Framework

[GCB06, GBC06]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

I 1 A platform for building intelligent software

a) Aktive Ontologie konstruiert strukturierte Befehle aus natürlicher Sprache

b) Besteht aus Konzepten und Relationen die Domäne der Anwendung definieren

c) Sprachbefehl besteht aus Subjekt, Komplement und Verb

d) Wenn Domäne einmal mit Konzepten und Beziehungen definiert wurde, wird mittels Regeln die Sprachverarbeitungsschicht angewendet

e) Die Aktive Ontologie wird von Hand durch einen Programmierer erzeugt

I 2 A unified platform for building intelligent web interaction assistants

a) Entwicklung eines Assistenten der Informationen zu Restaurants und Filmen liefert

b) Geographischer Inhalt wird durch eine Webservice-Schnittstelle durch Yahoo und Google bereitgestellt

c) Aktive Ontologie wird manuell erstellt

d) Befehle bestehen aus einem Verb und einem Event

e) Event: Filmevent, Speiseevent, besitzen Datum und mehr Details

f) Film: Genre, Schauspieler, Bewertung

g) Anschließend hinzufügen der Regeln zu den Konzepten für Verarbeitung der natürlichen Sprache

Nachweise

Welche Nachweise (Evidence) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

N 1 Wird auf regulärer Basis durch Chirurgen bewertet

B.7. Easier: An Approacht to Automatically Generate Active Ontologies for Intelligent Assistants

[BL16, BL18]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 Identifizieren von formularbasierten Diensten
- I 2 Crawlen des Internets nach HTML-Formularen und extrahieren der Metainformationen
- I 3 Clustern der identifizierten Webformulare mit Hilfe von Spectral Clustering nach Service-Kategorien
- I 4 automatische Konstruktion von AOs für jede Servicekategorie
- I 5 Gemeinsame Merkmale einer Kategorie -> obligatorische Knoten
- I 6 Um Ontologie aktiv zu machen -> Bestimmung des Knotentyps für jeden Knoten
- I 7 Jeder Knotentyp muss mit Liste gültiger Werte versehen sein
- I 8 Einige Formularelemente enthalten Informationen zu gültigen Werten/Datentypen
- I 9 EASIER verwendet Typ um Eingabeverifizierungsregeln in generierten Blättern anzugeben
- I 10 Keine Hinweise zur Eingabeüberprüfung -> AO-Entwickler

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

- D 1 Bestehende Projekte beschäftigen sich mit allgegenwärtigen mobilen Anwendungen, die leistungsfähige Middleware-Plattformen bieten, denen aber die Verarbeitung natürlicher Sprache oder multimodale Fusion fehlt
- D 2 Leichtgewichtiger und entwicklerfreundlicher als das sehr ähnliche SOAR Projekt

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

- P 1 Skalierbarkeit und Stabilität des Active Servers gilt es zu verbessern
- P 2 Clustering noch nicht verfügbar
- P 3 Generiert gut AOs, aber Begriffsklärung muss verbessert werden
- P 4 Mehrdeutige Felder ohne zusätzliche Informationen können nicht unterschieden werden

Nachweise

Welche Nachweise (Evidence) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

- N 1 Getestet mit 58 Webformularen aus drei Servicekategorien (Flugkosten (20), Automobile (20), Buchsuche(18))
- N 2 Flug AO deckt 75% der Anfragen ab, allerdings sind nur 36% vollständig abgedeckt
- N 3 Detaillierte Analyse von zehn kompletten Anfragen zeigt, dass EASIER 70% der Anfragen beantworten konnte

B.8. DIALOGIC: A Core Natural-Language Processing System

[GHH+82]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

I 1 DIALOGIC übersetzt englische Sätze in Repräsentationen ihrer wörtlichen Bedeutung im Kontext einer Äußerung
I 2 Diese Repräsentationen (logische Formen) sollen eine rein formale Sprache sein
I 3 Logische Formen sollen der Struktur der natürlichen Sprache so nahe wie möglich kommen
I 4 Ziel war es, DIALOGIC als Kernkomponente der Sprachverarbeitung in einer Vielzahl von Systemen zu verwenden, wobei einige zu neuen Anwendungsbereichen transportierbar sind

Lösungen

Was sind die eigenen Lösungen der Autoren?

L 1 DIAGRAM
- a) Verwendet wird DIAGRAM als generelle Grammatik für die englische Sprache
- b) Beschreibt allgemeine Satztypen, komplexe Hilfs- und Modelltypen, komplexe Nominalphrasen, normalisierte Sätze, alle gängigen Quantifizierer, Relativsätze, Verben mit Satzkomplementen, Vergleichs- und Maßausdrücke, Nebensätze und andere adverbiale Modifikatoren
- c) Lexikon kategorisiert Wörter und Attribute mit denen, die in den Regeln verwendet werden
- d) Jede Regel besitzt einen Konstruktor, der die Einschränkungen für seine Anwendung ausdrückt und ein Übersetzer, der die entsprechende logische Form erzeugt
- e) Phrasen erwerben Attribute aus ihren Bestandteilen und erwerben Attribute von größeren Phrasen die diese enthalten
- f) Attribute werden verwendet um Akzeptanz einer Analyse kontextsensitiv zu beschränken
- g) Vor dem Erzeugen eines Knotens im Parse-Baum ruft die Exekutive den Konstruktor der Regel auf, um auf Zulässigkeit zu prüfen
- h) Konstruktoren können auch Scores zuweisen um alternative Analysen in bevorzugter Reihenfolge aufzulisten
- i) Ergebnis sind ein oder mehrere annotierte Parsebäume

L 2 Übersetzer
- a) Nach syntaktischer Analyse einer Äußerung wird eine Sequenz semantischer Übersetzer aufgerufen
- b) Bauen logische Form auf, die einer wörtlichen Interpretation der Äußerung im Kontext entspricht
- c) Übersetzer für jede Regel legt fest, wie die verschiedenen Bestandteile der Phrase zu einer Interpretation der gesamten Phrase kombiniert werden sollen
- d) Obwohl Übersetzer von oben nach unten arbeiten, wird Übersetzung tatsächlich von unten nach oben aufgebaut

e) Aufbau der logischen Form in zwei Phasen: (1) Fragmente der logischen Form werden an den Parsebaum angehängt; (2) endgültige logische Form wird daraus durch Scoping-Algorithmus zusammengesetzt

B.9. Natural Language Query Processing Using Semantic Grammar

[RAC+10]

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

- I 1 Um Daten aus einer Datenbank abzufragen werden SQL-Kenntnisse benötigt, mit Hilfe des Systems dieses Papers kann mittels natürlicher Sprache eine Datenbank abgefragt werden
- I 2 Generelle natürliche Sprachverarbeitung wird durchgeführt
- I 3 Morphologische Analyse: Einzelne Worte werden analysiert und Nicht-Wort-Zeichen (z.B. Punkte) werden von den Worten separiert
- I 4 Syntaktische Analyse: Lineare Sequenz an Wörtern werden in eine Struktur transformiert, die die Zusammenhänge darstellt
- I 5 Semantische Analyse: Strukturen die durch die syntaktische Analyse erzeugt wurden, werden mit Bedeutungen belegt
- I 6 Diskursintegration: Bedeutung eines Satzes kann von den vorangehenden Sätzen abhängen und Bedeutung der folgenden Sätze beeinflussen
- I 7 Pragmatische Analyse: Struktur die repräsentiert was gesagt wurde, wird neu interpretiert um festzustellen was gemeint war

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

- P 1 Eingabe muss auf Englisch sein
- P 2 Eingabe muss in Form einer Frage sein
- P 3 Limitiertes Daten-Wörterbuch wird verwendet indem alle Wörter passend zum jeweiligen System enthalten sind, dieses muss kontinuierlich erweitert werden
- P 4 Alle Namen der Eingabe müssen in doppelten Hochkommas geschrieben werden
- P 5 Nicht alle Formen von SQL-Anfragen werden unterstützt

Lösungen

Was sind die eigenen Lösungen der Autoren?

- L 1 Semantische Grammatik
  - a) Zwei Teile einer semantischen Grammatik: Lexikon und Regeln
  - b) Lexikon speichert alle möglichen Worte die in der Grammatik vorkommen können
  - c) Nur einzelne englische Worte oder terminierende Symbole können im Lexikon vorkommen
  - d) Regeln kombinieren die terminierenden Symbole im Lexikon um Phrasen oder Sätze auf eine spezielle Weise zu erzeugen
- L 2 Ablauf des Prozesses
  - a) Englische Eingabe wird durch die semantische Grammatik geparst
  - b) Anschließend gleicht ein Postprozessor Tabellen- und Attributnamen ab und verknüpft Tabellen, wenn die Abfrage mehr als eine Tabelle umfasst

c) Danach kann der Postprozessor die resultierende SQL Query erzeugen und ausgeben

L 3 Algorithmus

a) Tokenisierung

b) Aufteilen der Anfrage in Token

c) Jedem identifizierten Token eine Ordnungsnummer zuweisen

d) Aufteilen der Anfrage und extrahieren der Muster

   i. Suche nach Verbindungen zwischen Sätzen / Kriterienwörter

   ii. Aufteilen der Query auf Basis der Konnektor-/Kriterientoken

   iii. Verwende Kriterientoken zur Spezifizierung der Bedingung in der Anfrage

   iv. Finde Attribute und Werte nach dem Kriterientoken

e) Abbilden des Wertes für das identifizierte Attribut und die entsprechende Tabelle

f) Ersetzen der Synonyme durch geeignete Attributnamen

g) Holen der Zwischendarstellung der Anfrage

h) Umwandeln in SQL