Karlsruhe Institute of Technology

Semi-automatische Generierung von aktiven Ontologien aus Webformularen

Bachelorarbeit
von

Kay Schmitteckert

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. Martin Blersch

Bearbeitungszeit: 10.07.2016 – 10.11.2016

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, 10.11.2016**

. . . . . .

(**Kay Schmitteckert**)

Kurzfassung

Zunehmend sollen intelligente Sprachassistenten den Nutzern die Bedienung des Gerätes erleichtern und einfach Aufgaben wie beispielsweise das Erstellen von Kalendereinträgen übernehmen. Apples bekannter Sprachassistent Siri nutzt für die Verarbeitung natürlicher Sprache aktive Ontologien. Eine aktive Ontologie muss dazu für jede Domäne erzeugt werden. Das manuelle Erzeugen von aktiven Ontologien ist ein zeit- und arbeitsintensiver Prozess. Deshalb befasst sich das Projekt EASIER des Lehrstuhls IPD Tichy mit der Aufgabe, diesen Vorgang möglichst zu automatisieren.

Diese Arbeit wurde als Bestandteil von EASIER erstellt und befasst sich mit der semiautomatischen Generierung aktiver Ontologien aus Webformularen. Dazu wurde zum einen ein Konstruktionsplan zur Zusammenführung von Webformularen einer Domäne erstellt. Außerdem wurden Ableitungsregeln definiert, welche vorgeben wie Webformulare zum Erzeugen aktiver Ontologien verarbeitet werden müssen. Auf der Grundlage dieser Vorarbeit wurde ein Werkzeug entwickelt, welches aus zusammengeführten Webformularen einer Domäne teilautomatisch über die Ableitungsregeln und Benutzerinteraktionen aktive Ontologien erzeugt. Die mit dem Werkzeug generierten aktiven Ontologien wurden evaluiert und weisen einen Fortschritt beim teilautomatischen erzeugen von aktiven Ontologien vor.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

„Hey Siri, erstelle einen neuen Kalendereintrag für morgen um 10 Uhr mit dem Titel Zahnarzt“, mit solchen Befehlen sollen heutzutage intelligente Sprachassistenten dem Nutzer die Bedienung des Gerätes erleichtern und einfache Aufgaben wie beispielsweise das Erstellen von Kalendereinträgen übernehmen. Sprachassistenten - wie Apples Siri, Microsofts Cortana oder Google Now - werden immer wichtiger und unterstützen den Nutzer auch in weniger komfortablen Situationen, unter anderem beim Autofahren. Siri arbeitet mit aktiven Ontologien (AOs), die Konzepte in Verbundenheit mit Regeln und Relationen zum Verarbeiten natürlicher Sprache vorgeben. Für jede Aufgabe muss manuell und aufwändig eine AO erstellt werden. Wenn man viele Dienste einbinden möchte, ist es unerlässlich die Generierung von AOs so weit wie möglich automatisiert durchzuführen.

Das Projekt „EASIER“ [Ble16] am Lehrstuhl IPD Tichy des KIT beschäftigt sich mit der automatischen Generierung von aktiven Ontologien für intelligente Assistenten. Zur Nutzung von formularbasierten Internetdiensten über natürliche Sprache sollen aktive Ontologien teilautomatisiert aus Webformularen erstellt werden. Es soll ermöglicht werden Internetdienste wie Flug-, Bahn- oder Hotelbuchungswebseiten über natürliche Sprache anzusprechen. Um diesem Ziel näher zu kommen, befasst sich diese Arbeit mit der teilautomatisierten Abbildung von Webformularen auf aktive Ontologien.

1.1. Zielsetzung

Um dem Ziel, Webformulare über natürliche Sprache ansprechen zu können, liegt der Fokus dieser Arbeit in der Definition von Ableitungsregeln in Verbindung mit der Erstellung eines Werkzeuges zur semi-automatischen Generierung aktiver Ontologien aus Webformularelementen. Das zu erstellende Werkzeug (AO-Generator) soll aus Webformularelementen eines Webformulars einen Baustein der aktiven Ontologie möglichst automatisiert generieren. Dazu verwendet der AO-Generator Ableitungsregeln, die vorgeben wie aus den Informationen des Webformularelements ein Baustein der aktiven Ontologie, das Konzept, erstellt wird.

Die Informationen der Webformularelemente liegen in Form von Attributen vor. Diese Attribute definieren oft einen Wertebereich. So definiert beispielsweise das Attribut „type“ mit dem Wert „date“ einen Wertebereich über Datumsangaben. Die Informationen über den Wertebereich sind essentiell, um automatisiert ein Konzept der aktiven Ontologie zu erstellen. Deshalb wird in dieser Arbeit analysiert wie aus den vorhandenen Informationen

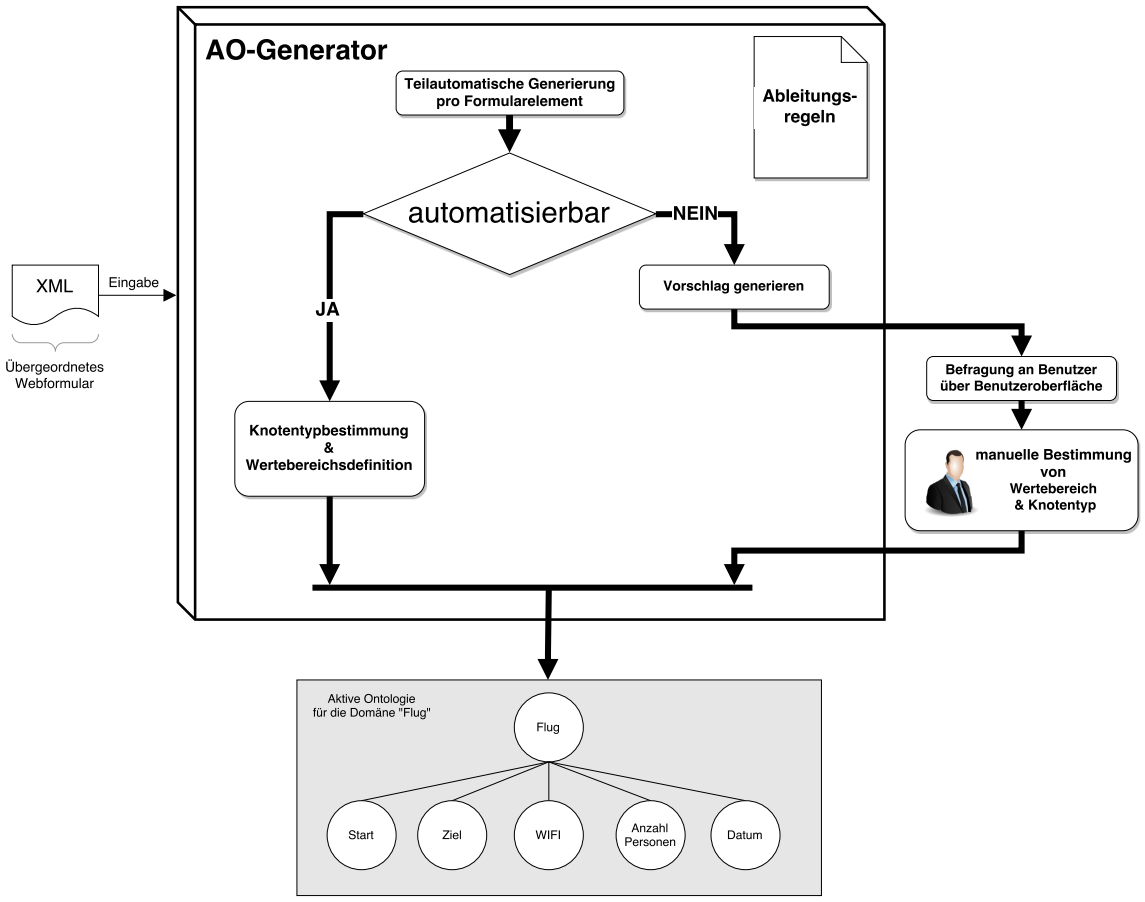

Abbildung 1.1.: Teilautomatische Generierung der aktiven Ontologie „Flug“

der Webformularelemente ein Wertebereich definiert werden kann. Als Beispiel für die vorhandenen Informationen eines Webformulars dient Abbildung 1.2.

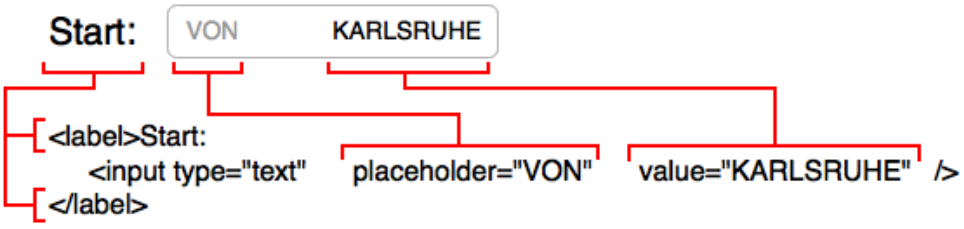

Abbildung 1.2.: Formularelement mit Label, placeholder und value (Zeile 3-5)

In Kooperation mit der parallel laufenden Arbeit **??** wird ein Konstruktionsplan zum Zusammenführen von Webformularen einer Domäne entworfen wie in Abbildung 1.3 für die Domäne „Flug“ skizziert. Außerdem wird ein geeignetes XML Schema für zusammengeführte Webformulare definiert. Aus Webformularen einer Domäne wird in der Arbeit „Extraktion und Konsolidierung von Webformularen zur Erzeugung von aktiven Ontologien“ [May17] von T. Mayer ein übergeordnetes Webformular generiert und daraus ein XML-Dokument nach dem XML Schema erstellt. Dieses XML-Dokument dient dem AO-Generator als Eingabe. Der AO-Generator verarbeitet die Webformularelemente des XML-Dokuments semi-automatisch zu einer aktiven Ontologie. Kann ein Webformularelement nicht automatisiert abgebildet werden, wird ein Vorschlag generiert und dieser dem Benutzer präsentiert. Der Vorschlag kann vom Benutzer erweitert werden.

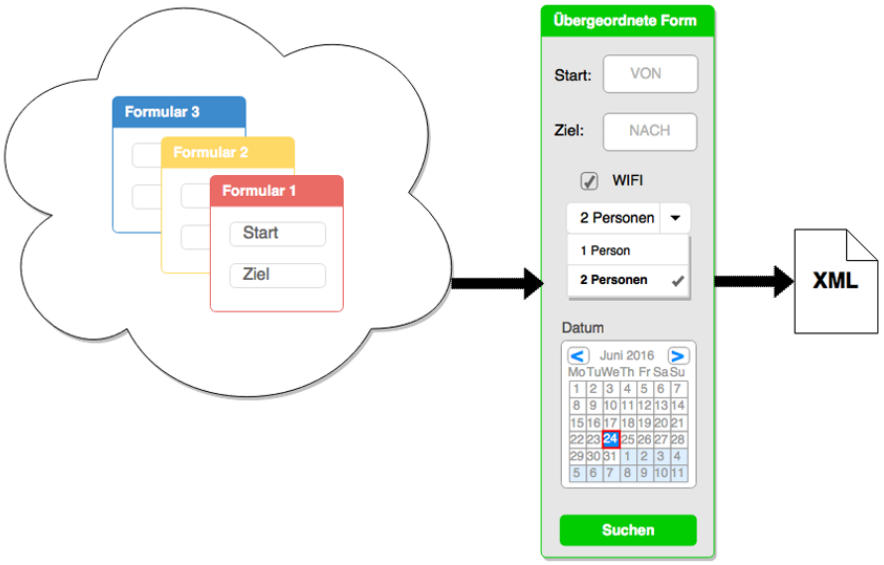

Abbildung 1.3.: Übergeordnetes Formular zu Formularen 1, 2 und 3

1.2. Beispiel

Die Abbildung 1.1 stellt den Ablauf des AO-Generators an folgendem Beispiel dar. Das übergeordnete Webformular aus dem Quelltextausschnitt 1.3 steht als XML-Dokument zu Verfügung. Der Algorithmus des AO-Generators verarbeitet nacheinander die einzelnen Formularelemente des XML-Dokuments und erstellt basierend auf den Ableitungsregeln die Konzepte der aktiven Ontologie. Die Verarbeitung der Elemente mit den Etiketten „WIFI“ und „Datum“ werden in folgendem Beispiel nicht erwähnt.

```
1  <form>
2      <h4>Uebergeordnetes Formular</h4>
3      <label>Start
4          <input type="text" name="von"/>
5      </label>
6      <label>Ziel
7          <input type="text" name="nach"/>
8      </label>
9      <label><input type="checkbox" name="wifi" value="wifi/>WIFI</label>
10     <select>
11         <option value="eine">1 Person</option>
12         <option value="zwei">2 Personen</option>
13     </select>
14     <label>Datum
15     <input type="date" name="datum"/>
16     </label>
17     <button type="submit">Suchen</button>
18 </form>
```

Quelltextausschnitt 1.1: HTML zu übergeordnetem Formular aus Abbildung 1.3

Zuerst wird das Formularelemente mit dem Etikett „Start“ verarbeitet. Sei dieses nach der dazugehörigen Ableitungsregel nicht automatisiert ableitbar. Es wird über die Vorgaben der Ableitungsregel ein Vorschlag anhand der gesetzten Attribute generiert und

dem Benutzer über eine grafische Oberfläche präsentiert. Nach dem manuellen Erweitern des Vorschlags durch den Benutzer wird das Konzept für die aktive Ontologie erstellt. Der Verarbeitung des Formularelements „Ziel“ geschieht analog. Als nächstes wird das Auswahlmenü für die Anzahl der Personen verarbeitet. Die dazugehörige Ableitungsregel kann dieses Formularelement anhand der Informationen automatisiert verarbeiten. Dabei generiert der AO-Generator nach dem Konstruktionsplan der Ableitungsregel ein Konzept. Der Wertebereich wird hier beispielsweise über die „option“-Felder definiert.

1.3. Evaluation

Die Arbeit wird anhand des entwickelten Werkzeuges evaluiert. In einer vorangegangenen Arbeit „Abbildung von Webformularen auf aktive Ontologien“ [Sai16] von W. Said wurden manuell drei aktive Ontologien aus drei Kategorien (Flug, Bahn und Hotel) à 10 Webformularen erstellt. Diese aktiven Ontologien werden für diese Arbeit als Goldstandard definiert. Der Goldstandard wird mit den Ergebnissen des AO-Generators verglichen. Dazu wird eine Fehlerquote ermittelt, indem alle zum Goldstandard unterschiedliche Konzepte gezählt werden. Der Automatisierungsgrad wird als prozentuale Anzahl der automatisch abgebildeten Knoten bestimmt. Die Anzahl der automatisch erstellten Konzepte wird durch die Anzahl aller erstellten Konzepte geteilt.

1.4. Struktur der Arbeit

Im ersten Teil der Arbeit werden die Grundlagen vorgestellt und einen Einblick in den aktuellen Forschungsstand gegeben. Das darauf folgende Kapitel beschäftigt sich mit der Analyse von Webformularen, sowie der Definition der Ableitungsregeln für die Webformularelemente zu Konzepten der aktiven Ontologie. Dabei wird ein Konstruktionsplan für ein übergeordnetes Webformular aus Webformularen einer Domäne entwickelt. Der Konstruktionsplan umfasst die Erstellung übergeordneter Webformularelemente, indem vorgegeben wird wie die Wertebereiche und Attribute semantisch gleicher Webformularelemente zusammengeführt werden. Für das übergeordnete Webformular wird ein XML Schema definiert. Darauf aufbauend wird analysiert wie sich Webformulare möglichst automatisiert auf Konzepte der aktiven Ontologie abbilden lassen. Dazu werden Ableitungsregeln definiert. Im nächsten Kapitel wird auf die Analyse aufbauend der Entwurf für den AO-Generator entwickelt. Außerdem wird in diesem Kapitel die Implementierung des AO-Generators vorgestellt. Auf Basis der manuell erstellten aktiven Ontologien aus der Arbeit „Abbildung von Webformularen auf aktive Ontologien“ [Sai16] von W. Said, werden die Ergebnisse des AO-Generators im darauffolgenden Kapitel evaluiert. Abschließend werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf sich daraus ableitende Herausforderungen gegeben.

2. Grundlagen

Dieses Kapitel ist für das weitere Verständnis der Arbeit essentiell und erläutert die Grundlagen auf welchen diese Arbeit aufbaut. An erster Stelle wird ein Überblick über die Auszeichnungssprache HTML mit den verschiedenen Bausteinen gewährt, da die zu verarbeitenden Webformulare in HTML vorliegen. Weiter die Semantik und Syntax von XML erläutert, da dieses Datenformat für ein einheitliches Eingabeformat verwendet wird, und abschließend die relevanten Komponenten von Ontologien, sowie aktiven Ontologien erklärt, welche aus den Webformularen erzeugt werden sollen.

2.1. Hypertext Markup Language (HTML)

Die Hypertext Markup Language (HTML) nach ISO/IEC 15445[1] ist eine textbasierte Auszeichnungssprache, die von Webbrowsern interpretiert wird und die Grundlage des World Wide Web (WWW) darstellt[2]. HTML strukturiert die logischen Bausteine digitaler Dokumente wie Texte, Hyperlinks, Bilder oder anderen Daten.

Diverse HTML-Elemente stellen meist über einen einleitenden und einen abschließenden Befehl (sog. Tags) die Formatierung der HTML-Dokumente sicher. Spitze Klammern umgeben diese Tags, um Start und Ende des HTML-Elements zu kennzeichnen. Ein abschließender Befehl startet mit dem Präfix „</“, um sie von einleitenden Befehlen zu unterscheiden. Der Inhalt zwischen einleitendem und abschließendem Befehl („<tag>“ bzw. „</tag>“) ist der Gültigkeitsbereich des HTML-Elements und wird entsprechend der Befehle formatiert.

In HTML existieren auch inhaltsleere Elemente, die sogenannten „Standalone-Tags“, welche mit nur einem Tag („<tag>“ oder „<tag/>“) das Element definieren. HTML-Elemente lassen sich durch eine Liste von Attributen im einleitenden Befehl bzw. im Standalone-Tag erweitern. Attribute erweitern das HTML-Element mit zusätzlichen Informationen und bestehen häufig aus einem Name-Wert-Paar („name=‚value‘“).

```
1 <!-- Einleitender und abschließender Befehl -->
2 <tag>formatierter Inhalt</tag>
3
4 <!-- Einleitender und abschließender Befehl mit Attribut -->
```

[2] vgl. http://www.iso.org/iso/catalogue_detail.htm?csnumber=27688, zuletzt besucht am 26.Oktober, 2016.

[2] vgl. http://wirtschaftslexikon.gabler.de/Definition/html.html, zuletzt besucht am 24.Oktober, 2016.

```
5  <tag name=”wert”>formatierter Inhalt</tag>
6
7  <!-- Standalone-Tag -->
8  <tag/>
9
10 <!-- Standalone-Tag mit Attribut -->
11 <tag name=”wert”>
```

Quelltextausschnitt 2.1: Syntax der HTML-Elemente

HTML-Webformulare

Webformulare werden auf Webseiten über das form-Element „<form>“ definiert und ermöglichen eine Interaktion mit dem Besucher der Webseite. Eine solche Interaktion dient zur Informationserfassung vonseiten des Besuchers. Diese Informationen werden zur weiteren Verarbeitung an einen Webserver gesendet.

Ein Webformular setzt sich innerhalb des form-Elements aus HTML-Formularelementen zusammen, welche verschiedene Werte entgegennehmen können.

HTML-Webformularelemente

HTML-Formularelemente spezifizieren das zugehörige Webformular. Es existieren verschiedene Formularelemente wie textbasierte Eingabefelder, Auswahlkästchen, Auswahlmenüs oder Schaltflächen zum Absenden und Verwerfen. Formularelemente werden innerhalb von Webformularen gesetzt und können verschiedene Werte entgegennehmen wie Text, Datumsangaben oder Zahlen. Sie visualisieren dem Nutzer, welche Information benötigt wird, nehmen diese entgegen und leiten sie an den Unterbau weiter. Die Weiterleitung erfolgt nach dem Absenden des Webformulars auf dem Webserver.

Formularelement	**Beschreibung**
<input>	einzeiliges Eingabeelement
<textarea>	mehrzeiliges Eingabeelement
<select>	Auswahlelement
<button>	Schaltflächenelement
<label>	Bezeichnungselement

Tabelle 2.1.: Übersicht der HTML-Formularelemente

<input> Das input-Element bezeichnet ein einzeiliges Eingabefeld, in welches der Benutzer Daten eintragen kann. Das input-Element spezifiziert über Eingabetypen die einzutragenden Daten. Es kann normaler Text eingetragen werden oder auch komplexere Daten wie beispielsweise Datumsangaben, Farbbezeichnungen oder auch Nummern. Attribute spezifizieren diese Eingabetypen noch weiter. Durch Attribute können beispielsweise Nummern nur in vorgeschriebenen Schritten erhöht werden oder ihr Wert auch nur innerhalb eines Bereiches liegen. Die Eingabetypen und die dazugehörigen Attribute sind in Tabelle 2.2 aufgelistet.

<textarea> Das textarea-Element bezeichnet ein mehrzeiliges Eingabefeld, in welches der Benutzer eine definierte Anzahl an Zeichen eintragen und übermitteln kann. Dem Element kann durch die Attribute „col“ und „row“ eine Breite bzw. Höhe zugewiesen werden.

<select> Das select-Element bezeichnet eine Auswahlliste. Die option-Elemente innerhalb des select-Elements definieren die verfügbaren Auswahlmöglichkeiten. Das vorausgewählte option-Element wird durch ein selected-Attribut erweitert. Die option-Elemente werden mit dem value-Attribut angereichert. Das value-Attribut definiert

den zu übertragenen und vordefinierten Wert. Das select-Element kann mit dem Attribut „multiple“ angereichert werden. Dieses Attribut ermöglicht die Mehrfachauswahl von option-Elementen, welche übertragen werden sollen.

<button> Das button-Element bezeichnet eine klickbare Schaltfläche. Das button-Element löst beim Anklicken eine vordefinierte Aktion aus. Eine solche Aktion übermittelt beispielsweise die Daten aller Formularelemente an den Webserver. Text oder Bilder innerhalb des button-Elements definieren wie dieses Element dargestellt wird.

<label> Das label-Element bezeichnet ein Etikett für ein input-Element. Das label-Element kann durch die Attribute „for“ und „form“ erweitert werden. Das for-Attribut bindet das label-Element über das id-Attribut eines Formularelements an dieses, indem die Attributenwerte übereinstimmen. Alternative kann das label-Element auch das dazugehörige HTML-Formularelement umschließen. Der Quelltextausschnitt 2.2 dient als anschauliches Beispiel dafür. Das form-Attribut spezifiziert das dazugehörige Webformular, zu welchem es gehört.

```
1 <!-- Etikett über for-Attribut an HTML-Formularelement gebunden -->
2 <label for="label1">Etikett 2</label>
3 <tag id="label1"/>
4
5 <!-- Etikett umschließt dazugehöriges HTML-Formularelement -->
6 <label>Etikett 1<tag/></label>
```

Quelltextausschnitt 2.2: label-Elelement

2.2. XML

Die Extensible Markup Language (engl. „erweiterbare Auszeichnungssprache“) nach ISO 8879[3], abgekürzt XML, ist eine menschenlesbare Metasprache und dient zum plattform- und implementationsunabhängigen Austausch strukturierter Daten. XML ist nicht an vordefinierte Elemente gebunden, sondern ist erweiterbar. In dieser Metasprache lassen sich Elemente definieren. Diese Elemente dienen nicht nur zur Darstellung der Daten, sondern erfassen oft die Bedeutung des Inhalts.

2.2.1. XML Syntax

XML-Elemente stellen meist über einen einleitenden und abschließenden Befehl (sog. Tags) die Formatierung des XML-Dokumentes in hierarchischer Form sicher. Es existieren „Standalone-Tags“ wie auch bei HTML vorgestellt. Die Tags sind mit spitzen Klammern umgeben. Alle XML-Elemente sind von einem Wurzelelement umgeben. Dies wird durch den XML-Ausschnitt in Quelltextausschnitt 2.3 illustriert.

```
1 <root>
2     <form>
3         <category>FLUG</category>
4     </form>
5 </root>
```

Quelltextausschnitt 2.3: XML-Beispielausschnitt

[3] vgl. http://www.w3.org/XML/, zuletzt besucht am 26.Oktober, 2016.

2.2.2. XSD Syntax

Die XML Schema Definition Language (XSD) ist eine von der W3C empfohlene Sprache zum Definieren von Strukturen für XML Dokumente [4]. XSD wird mit dem XML-Vokabular in Form eines XML-Dokuments dargestellt. XSD unterstützt eine große Menge von atomaren und komplexen Datentypen. Zu den atomaren Datentypen zählen solche wie String, boolean oder integer, wobei komplexe Datentypen ergänzend definiert werden und ein Zusammenschluss von atomaren und komplexen Datentypen darstellen. Zu jedem Element können Attribute definiert werden. Attribute können verwendet werden, um das Element näher zu spezifizieren wie z.B. „minOccurs“ oder „maxOccurs“ definieren wie oft das Element mindestens und maximal innerhalb einer Aufzählung der gleichen Elemente (Sequenz) gesetzt werden darf. In Quelltextausschnitt 2.4 ist ein beispielhaftes Schema dargestellt.

```
1  <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="
       http://www.w3.org/2001/XMLSchema">
2    <xs:element name="forms">
3      <xs:complexType>
4        <xs:sequence>
5          <xs:element name="form" minOccurs="0" maxOccurs="unbounded">
6            <xs:complexType>
7              <xs:sequence>
8                <xs:element type="xs:string" name="category" minOccurs="1" maxOccurs="1"/>
9              </xs:sequence>
10           </xs:complexType>
11         </xs:element>
12       </xs:sequence>
13     </xs:complexType>
14   </xs:element>
15 </xs:schema>
```

Quelltextausschnitt 2.4: XSD zum XML-Beispielausschnitt

2.3. Ontologie

In den letzten Jahren werden Ontologien vermehrt für Wissensmodellierung und Wissensrepräsentation einzelner Domänen genutzt, um bestehendes Wissen global wiederverwenden zu können. So werden für viele Bereiche Ontologien entwickelt, wobei diese immer mehr im Gebiet des „Semantic Webs“ eingesetzt werden. Die Ontologien geben die Semantik für ergänzende Informationen in Webseiten vor. Eine Ontologie stellt eine „explizite, formale Spezifikation einer gemeinsamen Konzeptualisierung“[5] von Wissen dar, also eine universelle, maschinenverständliche Modellierung von Wissen. Eine solche Modellierung beschreibt bestehende Entitäten über sogenannte Konzepte, stellt über Relationen eine Hierarchie dar und versieht diese über Regeln und Axiomen mit einer Logik.

Die Abbildung 2.1 veranschaulicht eine beispielhafte Ontologie der Domäne Person. Die Konzepte stellen eine übergeordnete Entität für individuelle Objekte dar und binden all deren Eigenschaften. Mit der Entität „Person“ sind Frauen, Männer, Studenten oder auch Dozenten beschrieben. Bei den Relationen wird zwischen zwei verschiedenen Relationen unterschieden. Die Entität „Person“ besitzt eine Geburtsurkunde und einen Namen, was sogenannten „hat-ein“-Relationen entspricht, wobei die Entität „Student“ eine Person ist, was eine „ist-ein“-Relation darstellt. Mit den Regeln und Axiomen wird sicher gestellt,

[4] vgl. http://www.w3.org/XML/Schema (Abgerufen am 24.Oktober.2016)

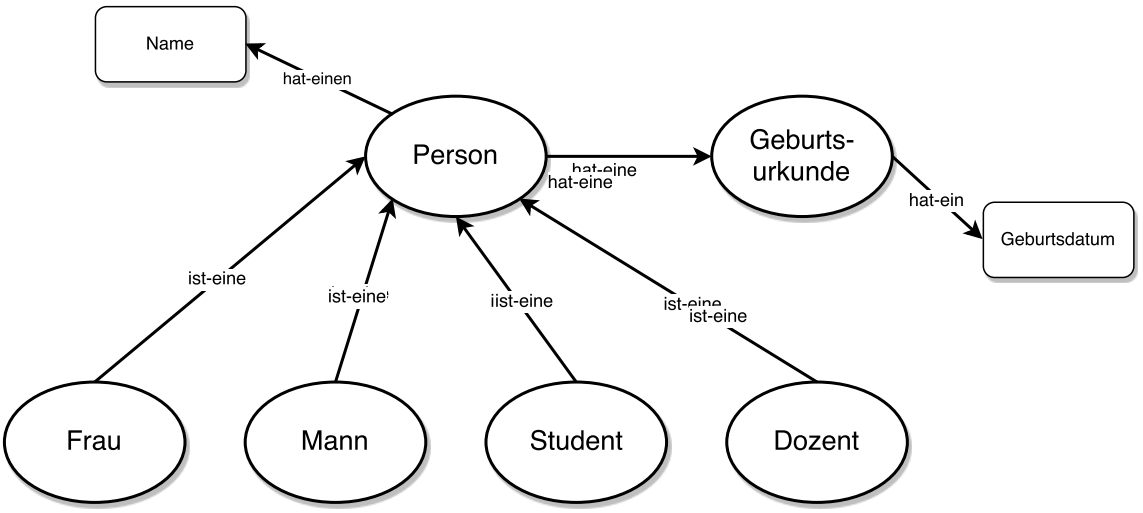

Abbildung 2.1.: Beispiel einer einfachen Ontologie zur Domäne „Person“

dass beispielsweise eine Frau auch gleichzeitig ein Student sein kann, aber die Schnittmenge zwischen Mann und Frau leer sein muss. Regeln und Axiome sind in der genannten Abbildung 2.1 nicht dargestellt.

2.4. Aktive Ontologie

Eine aktive Ontologie erweitert eine konventionellen Ontologie zu einer Ausführungsumgebung. Diese wird zu einer Ausführungsumgebung, indem die Konzepte Regeln besitzen, welche bei Erfüllung einer Bedingung eine Aktion ausführen [GBC06]. Die Bedingungen werden auf eingehende Informationen (Fakten) geprüft und mit diesen Informationen der Faktenspeicher gefüllt. Die Aktionen können den Faktenspeicher manipulieren und neue Fakten erstellen, vorhandene ändern oder löschen. Strukturell weist eine aktive Ontologie eine hohe Ähnlichkeit zu einem betitelten Graphen auf, also einem Graphen bei welchem die Knoten einen Namen tragen.

2.4.1. Konzepte

Die Konzepte sind für eine aktive Ontologie ein wesentlicher Bestandteil und stellen bezüglich der Graphenanalogie die Knoten eines Graphen dar was die Abbildung 2.2 deutlich machen soll. Ein Konzept besteht aus einem einzigartigen Namen und einem Satz an Regeln.

Regeln

Eine Regel beinhaltet eine oder mehrere Bedingungen und Aktionen. Eine Bedingung gleicht einem Booleschen Ausdruck. Jede Regel wird bei einer Zustandsänderung - wenn Fakten erzeugt werden oder sich ändern - auf ihre Bedingungen geprüft und führt bei positiver Evaluierung die dazugehörigen Aktionen aus. Eine Aktion kann wiederum zu einem Zustandswechsel führen, was zur Folge hat, dass die Fakten erneut geprüft werden.

2.4.2. Relationen

Relationen stellen für die aktiven Ontologien eine wichtige Komponente dar. Eine Relation verbindet ein Kindkonzept mit einem Elternkonzept und bindet einen Satz an Fakten.

[5] Definition zu Ontologien von T.Gruber im Jahre 1993, einem der grundlegenden Forschern in diesem Gebiet [G+93].

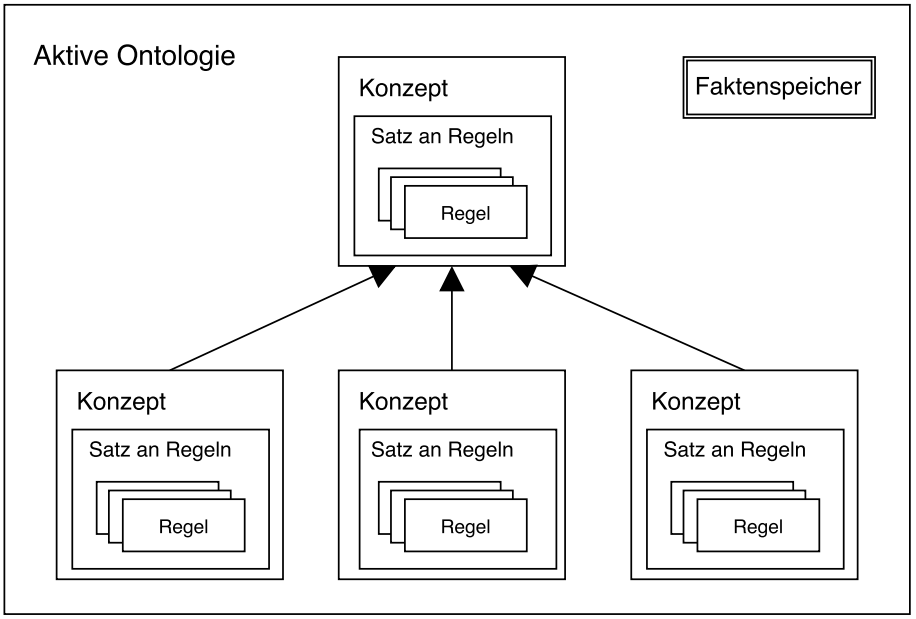

Abbildung 2.2.: Beispiel einer aktiven Ontologie nach Guzzoni [GBC06]

2.4.3. Fakten

Die Fakten entsprechen atomaren Daten, welche in einem gemeinsamen Faktenspeicher gehalten werden. Fakten bilden einen grundlegenden Baustein einer aktiven Ontologie und repräsentieren deren Zustand. Sie werden für die Weiterleitung von Informationen verwendet und stoßen über die Regeln Aktionen an. Die Aktionen können den Zustand der aktiven Ontologie verändern, indem neue Fakten erstellt, geändert oder gelöscht werden. Durch die Veränderung der Fakten entsteht ein Evaluierungszyklus, bei dem die Fakten immer wieder auf die Bedingungen der Regeln geprüft werden.

Eingabetyp	**Beschreibung**	**kompatible Attribute**
Text		
text	einzeiliges Eingabefeld	autocomplete, list, maxlength, name, pattern, placeholder, readonly, required, size, value
email	E-Mail-Adressfeld	autocomplete, list, maxlength, multiple, name, pattern, placeholder, readonly, required, size, value
password	Passwort-Eingabefeld	autocomplete, maxlength, name, pattern, placeholder, readonly, required, size, value
search	Suchfeld	autocomplete, list, maxlength, name, pattern, placeholder, readonly, required, size, value
url	URL-Feld	autocomplete, list, maxlength, name, pattern, placeholder, readonly, required, size, value
Zahlen		
number	Nummern-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
datetime	Datums- & Uhrzeit-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
datetime-local	Datums- & Uhrzeit-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
date	Datums-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
month	Monats- & Jahres-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
week	Wochen- & Jahres-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
time	Uhrzeit-Eingabefeld	autocomplete, list, max, min, name, readonly, required, step, value
tel	Telefonnummer-Eingabefeld	autocomplete, list, maxlength, name, pattern, placeholder, readonly, required, size, value
Schaltflächen		
reset	Zurücksetzen des Formulars	name, value
submit	Absenden des Formulars	formaction, formenctype, formmethod, formnovalidate, formtarget, name, value
button	Aufruf von clientseitiger Aktion	name, value
image	Schaltfläche in Form eines Bildes	alt, formaction, formenctype, formmethod, formnovalidate, formtarget, name, src, value
Sonstige		
checkbox	Checkbox-Element	checked, name, required, value
radio	Radiobutton-Element	checked, name, required, value
file	Dateiupload	accept, multiple, name, required, value
hidden	unsichtbarer Text	name, value
color	Farbauswahl	autocomplete, list, name, value
range	Intervallschieberegler	autocomplete, list, max, min, name, readonly, required, step, value

Tabelle 2.2.: Übersicht der input-Elementtypen und deren kompatiblen Attribute

3. Verwandte Arbeiten

Zur semi-automatischen Generierung von aktiven Ontologien aus Webformularen, sind verwandte Arbeiten bekannt, welche relationale Daten auf Ontologien abbilden. Die Ansätze dieser Arbeiten werden in diesem Kapitel analysiert und daraus Erkenntnisse für den eigenen Entwurf und Konzeption gewonnen. Zur Verarbeitung von Webformularen, müssen diese gründlich analysiert werden. Das bedeutet, dass die vorhanden Informationen eines Webformulars verlustfrei extrahiert werden müssen und durch diverse Verfahren weitere Informationen und Beziehungen zwischen Webformularen in Erfahrung gebracht werden müssen. Es existieren noch keinerlei bekannte Werkzeuge oder Arbeiten, welche aktive Ontologien erzeugen. Deshalb müssen die Ansätze zum Erzeugen von herkömmlichen Ontologien betrachtet werden. Für die Erzeugung von aktiven Ontologien muss der Aufbau und die weitere Verarbeitung dieser ermittelt werden, damit die nötigen Informationen aus den Webformularen extrahiert werden können.

Dieses Kapitel wird in drei Grundbereiche unterteilt.

- Webformularanalyse
- Generierung von Ontologien
- Verarbeitung von Ontologien mithilfe des Active Semantic Networks

3.1. Webformularanalyse zur Erstellung eines übergeordneten Webformulars

Um Webformulare einer Domäne zusammenführen zu können, müssen die einzelnen Webformularelemente auf semantische Gleichheit geprüft werden. Ist diese Analyse abgeschlossen, kann ein globales Webformular aus den Webformularen einer Domäne erstellt werden. Das globale Webformular wird dazu verwendet, um zentral die Webformulare einer Domäne anzusprechen. Unter diesem Punkt werden Arbeiten vorgestellt, die sich unter anderem mit der semantischen Analyse beschäftigen.

Mögliche Wertebereiche für Webformularelemente untersucht die Arbeit „Semantic Deep Web: Automatic Attribute Extraction from the Deep Web Data Sources“ [AGWC07a] von Yoo Jung An et al., indem die vorhandenen Metadaten der Webformularelemente genutzt werden, um weitere Informationen für die Wertebereiche zu generieren. Die Analyse beschränkt sich dabei auf zwei verschiedene Ansichten der Metadaten. Die erste Ansicht wird

als „Programmiereransicht“ („Programmer Viewpoint Attributes (PVA)“) bezeichnet. Die PVA umfasst die Werte der gesetzten Attribute der HTML-Formularelemente. Die zweite Ansicht wird mit „Benutzeransicht“ („User Viewpoint Attributes (UVA)“) beschrieben. Die UVA beinhaltet alle im Vorbau für den Nutzer sichtbaren Werte wie z.B. die Bezeichnungen für Auswahlboxen. Der mögliche Wertebereich setzt sich nun aus den Werten von PVA, UVA und deren über WordNet definierten Synonyme zusammen. Bei diesem Vorgang stößt man auf zwei Probleme:

1. Manche Werte der PVA liegen nur als Abkürzung vor wie beispielsweise „dep“ für „departure“ oder „yr“ für „year“.
2. Manche Werte der UVA liegen inkonsistent im Plural und Singular vor wie beispielsweise „adults“ und „adult“.

Für die Abfrage der Werte über das WordNet sind nun Regeln definiert, um ungeeignete Werte auszusortieren. Dazu werden die Werte der PVA und UVA in atomare Wörter aufgetrennt und über WordNet folgende zwei Regeln geprüft:

1. Wenn diese Wörter als Nomen vorliegen oder sich Nominalisieren lassen, wird das Wort beibehalten. Falls nicht, wird das Wort verworfen.
2. Ist das Wort eine Präposition, wird es behalten. Diese Regel hilft dabei wichtige Wörter wie „von“ oder „nach“ beizubehalten.

Durch dieses Vorgehen wird die „SOPVA“ aus den PVA und die „SOUVA“ aus den UVA gebildet. Die Werte dieser Wertebereiche werden nun paarweise überprüft und bei einem Wortabstand von weniger als $\alpha\%$ beibehalten, ansonsten verworfen. Die beibehaltenen Werte bilden den finalen Satz an Werten bzw. Attributen (FA).

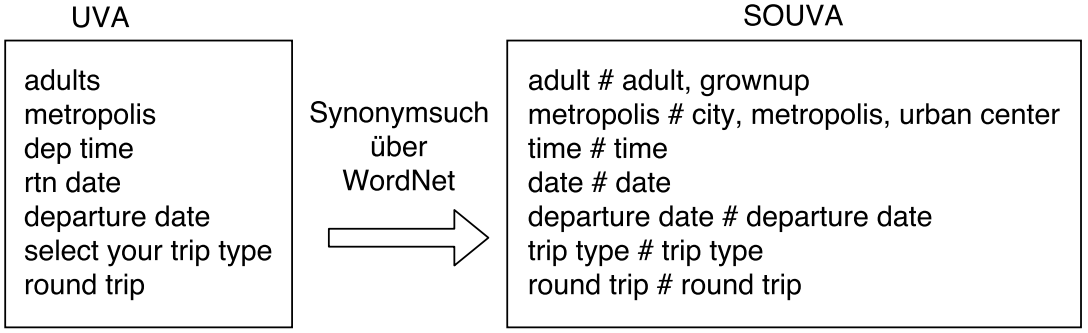

Abbildung 3.1.: Wertebereiche vor und nach dem Anwenden der Regeln aus Arbeit [AGWC07a], Quelle: [AGWC07a]

In Abbildung 3.1 ist beispielhaft dargestellt, wie eine solche Analyse über WordNet aussehen könnte. Auf der linken Seite sind die Werte aus UVA aufgelistet. Nach dem Anwenden der Regeln über das WordNet entstehen die „SOUVA“. Vor der Raute # stehen die Nominalisierten Wörter im Singular, nach der Raute # die gefundenen Synonyme. Dieses Vorgehen geschieht analog für die PVA, woraus die SOPVA entstehen. Die Werte der SOPVA und SOUVA werden nun paarweise auf einen Wortabstand mit $\alpha\%$ verglichen und daraus der finale Satz an Werten „FA“ gebildet wie in Abbildung 3.2 dargestellt.

Die Arbeit „Automatic Generation of Domain-specific Ontology from Deep Web“ [AGWC07b] von K. Chen et al. verwendet für die Ontologiegenerierung einer Domäne das Webformular und die Ergebnisse nach erfolgreicher Anfrage dieses Webformular. Das Erkennen der semantisch gleichen Webformularelemente wird in zwei Schritten durchgeführt. Als erstes werden Attribute aus PVA und UVA zweier Webformularelemente verschiedener Webseiten einer Domäne über WordNet auf semantisch gleiche Bedeutung geprüft. Werden

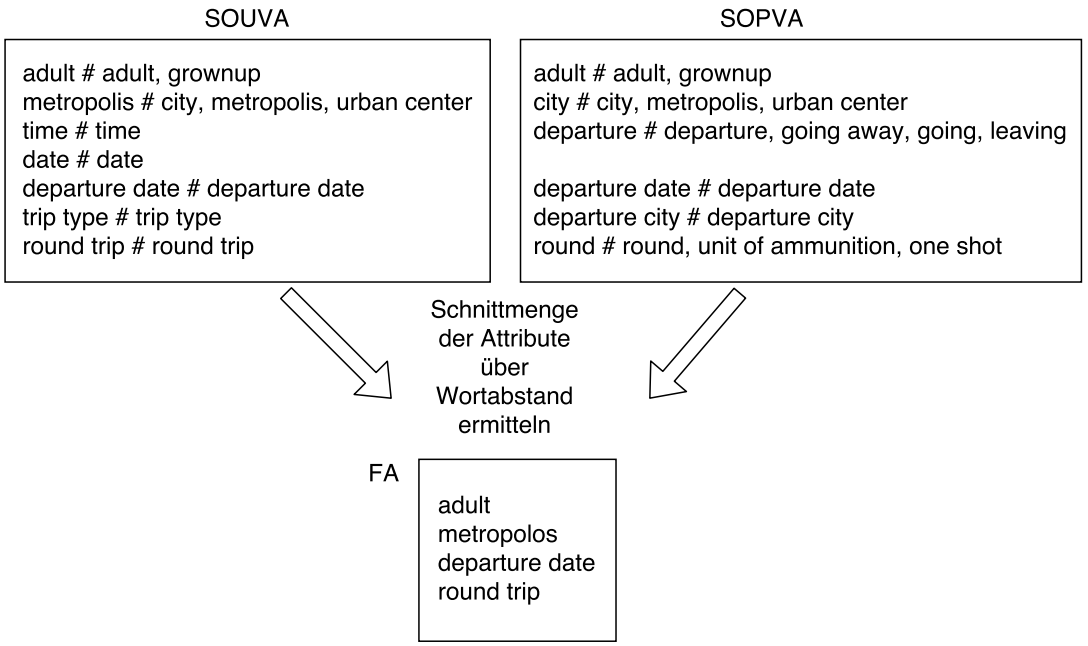

Abbildung 3.2.: Schnittmenge der Wertebereich über Wortabstand „WISE-Integrators“, Quelle: [AGWC07a]

zwei Attribute als Synonyme erkannt, werden daraus zwei Knoten erstellt und verbunden. Werden diese nicht als Synonyme identifiziert, wird der Wortabstand berechnet. Ist der Wortabstand geringer als das zuvor definierte μ, werden daraus ebenfalls zwei Knoten erstellt und verbunden. Durch dieses Vorgehen entsteht ein Geflecht von semantisch gleicher Webformularelemente.

Der Ansatz über WordNet den Wertebereich zu erweitern wird in dieser Arbeit ähnlich gestaltet.

„An interactive clustering-based approach to integrating source query interfaces on the deep web“ [WYDM04] von W. Wu et al. extrahiert aus der hierarchischen Struktur einen geordneten Baum, um semantisch gleiche Webformularelemente - unter anderem auch komplexe (1:m) Abbildungen - verschiedener Webformulare zu finden. Es wird zwischen zwei verschiedenen Arten von komplexen Abbildungen unterschieden. Zwischen einer „aggregierten“ und einer „ist-ein“ komplexen Abbildung. Bei einer „aggregierten“ Abbildung entspricht der Wertebereiche eines Elements der m-Seite einem Teil des Wertebereichs der 1-Seite. Bei einer „ist-ein“ Abbildung bildet der Wertebereich der 1-Seite die Vereinigung der Wertebereiche auf der m-Seite. Die semantisch gleichen Elemente werden über ein transitives Verfahren gruppiert. Sind die Elemente e_1 und e_2 semantisch gleich, aber nicht als solche identifizierbar, allerdings e_1 und e_3, sowie e_2 und e_3, dann kann durch diese Beziehung geschlussfolgert werden, dass auch e_1 und e_2 semantisch gleich sind.

„A hierarchical approach to model web query interfaces for web source integration“ [DKYL09] von E. Dragut et al. stellt einen Algorithmus vor, der die HTML-Formularelemente des Webformulars und dessen geometrische Eigenschaften verarbeitet und so aus einem nichthierarchischen Webformular eine hierarchische Repräsentation dessen extrahiert. Aus den geometrischen Eigenschaften werden zwei Bäume mit vordefinierten Regeln erstellt, die danach zusammengeführt werden. Es wird zwischen Textelementen wie Labels und Feldelementen wie Eingabefelder oder Auswahlmenüs bei den HTML-Formularelementen differenziert. Ein Baum (TT) wird aus den Textelementen erstellt und ein weiterer (FT) aus den Feldelementen. TT wird über Abbildungsregeln auf FT abgebildet und es entsteht

der Schema Baum (ST). ST repräsentiert nun das Webformular in hierarchischer Form, welches sich nun besser weiterverarbeiten lässt.

Das Werkzeug „WISE-Integrator“, welches in den Arbeiten [HMYW05], [HMYW04] und [HML+07] von H. He et al. vorgestellt wird, erstellt automatisch ein vereinigtes Webformular für Webformulare einer Domäne. Dieses globale Webformular sendet bei einer Anfrage Unteranfrage zu jedem einzelnen Webformular und holt die Ergebnisse aller Anfragen ein. WISE besteht aus zwei Hauptbestandteilen: Dem „Schnittstellenmuster-Extraktor“ („interface schema extractor“), sowie dem „Schnittstellenmuster-Integrator“ („interface schema integrator“). Der Ablauf dieses Werkzeuges wird in Abbildung 3.3 verdeutlicht.

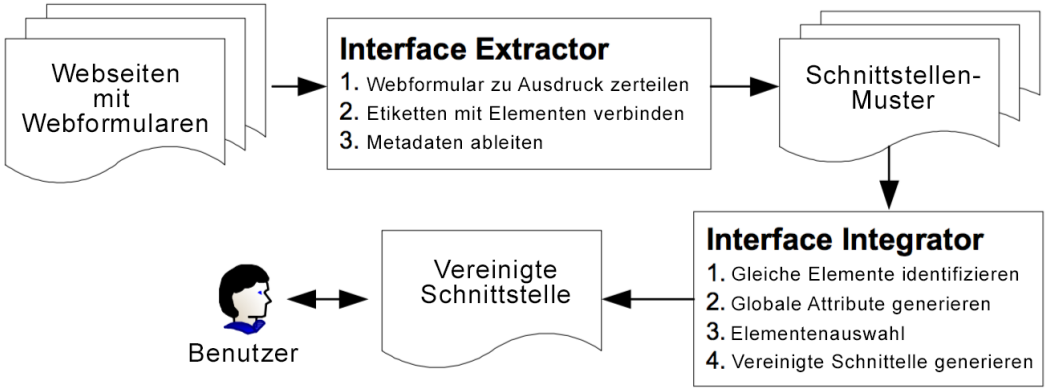

Abbildung 3.3.: Ablaufdiagramm des „WISE-Integrators“, Quelle: [HMYW05]

Der Schnittstellenmuster-Extraktor gruppiert die HTML-Formularelementen der Webformulare zu Tupeln von Etikett und Webformularelement. Diese Gruppierungen werden durch einen Ausdruck (IEXP) festgehalten, welcher die optische Struktur des Webformulars mit Element (e) und Etikett (t) und Trennzeichen (|) repräsentiert. Beispielsweise stellt te|te|t zwei Elemente mit einem Etikett dar, sowie ein Etikett ohne Element (siehte Abbildung 3.4).

Abbildung 3.4.: Verarbeitung der optischen Struktur zu einer geeigneten Repräsentation (IEXP)

Der Ausdruck (IEXP) wird durch eine strukturbasierte Extraktionstechnik (LEX) gebildet, welche die einzelnen Gruppierungen erkennt. Die Extraktionstechnik findet zu einem Webformularelement das dazugehörigen Etikett, indem es die gleiche oder benachbarte Zeilen durchsucht und über verschiedene Heuristiken darauf schließt, dass sie zusammengehören. Dies geschieht z.B. über geringen Wortabstand von Etiketten und „name“-Attribut des Webformularelements, sowie die Positionen von Etikett und Webformularelement. Die entwickelte Extraktionstechnik besitzt eine 94%-ige Genauigkeit bei der Verarbeitung der optischen Struktur. Über diese Gruppierungen und Informationen der Elemente ist es dem Schnittstellenmuster-Extraktor möglich Rückschlüsse auf die Wertebereich, sowie den Datentyp der jeweiligen Webformularelemente zu ziehen. Beispielsweise lässt ein HTML-

Formularelement mit dazugehörigem Etikett „Veröffentlichungsdatum“ auf den Datentyp „Datum“ schließen. Folgende Typinformationen werden bei der Extraktion erkannt:

- Reihenfolge der HTML-Formularelemente (orig. „layout order“)
- Wertebereichseinschränkungen (orig. „domain type“)
  - Bereich („range“)
  - Endlich („finite“)
  - Unendlich („infinite“)
  - Boolean
- Datentyp („data type“)
  - Datum („range“)
  - Datum mit Zeitangabe („finite“)
  - Währung („infinite“)
  - ID
  - Numerischer Wert („number“)
  - Zeichen („char“)
- Standardwert („default value“)
- Einheit („unit“) (z.B. Kilogramm bei Gewicht)
- Beziehungen zwischen HTML-Formularelementen („element relationship“)
  - Bereich („range“)
  - Teil („part“)
  - Gruppe („ group“)
  - Einschränkung („constraint“)

Das Erkennen der Typen erfolgt mithilfe domänenunabhängigem Wissen wie z.B. Erkennung von Mustern von Datums- oder Zeitangaben. Mit dem Erkennen des Datentyps, lassen sich mögliche Wertebereich berechnen. Diese Wertebereiche müssen für ein funktionsfähiges globales Webformular ermittelt werden. Die extrahierten Informationen der einzelnen Webformulare und deren Webformularelemente, speichert der „interface schema extractor“ in einem XML Dokument mit geeignetem Format.

Der Schnittstellenmuster-Integrator nutzt das vom Schnittstellenmuster-Extraktor generierte XML Dokument als Eingabe. Das XML Dokument wird auf semantisch gleiche HTML-Formularelemente geprüft und daraus je ein globales Webformularelement erstellt. Dies geschieht in 3 Schritten.

1. Das Etikett des vereinigten Webformularelements wird über zwei Strategien ermittelt. Zum einen über eine „Mehrheitsstrategie“ („majority strategy“), bei dem das am häufigsten verwendete Etikett der untergeordneten Webformularelemente verwendet wird. Mit der „Generalisierungsstrategie“ („ generality strategy“) wird zu den behandelten Etiketten der allgemeinste Oberbegriff gesucht.

2. Wertebereiche für die globalen Webformularelemente werden mit vier Regeln paarweise aus den untergeordneten Webformularelementen ermitteln.

- endlich + endlich ⇒ endlich
- unendlich + unendlich ⇒ unendlich
- Bereich + Datentyp ⇒ Bereich
- (endlich + unendlich) | (Mischform + endlich) | (Mischform + unendlich) ⇒ Mischform

3. Die Struktur des vereinigten Webformulars wird auch durch die Struktur der ursprünglichen Webformularelemente festgelegt. Es wird der Mittelwert der Positionen der ursprünglichen Webformularelemente ermittelt und darüber die Positionen der globalen Webformularelemente festgelegt. Sind die untergeordneten Webformularelemente beispielsweise eher am Anfang positioniert, wird das abgebildete globale Webformularelement auch am Anfang positioniert sein.

Aus diesen automatisch erzeugten globalen Webformularelementen entsteht nun das übergeordnete Webformular der Domäne, welches die Dienste der zusammengeführten Webformulare ansprechen kann. Als Zusatzfunktion des WISE-Integrators können Webformulare bei einem schon generierten vereinigten Webformular hinzugefügt oder gelöscht werden und das Werkzeug generiert direkt ein neues unifiziertes Webformular.

Die verschiedenen Ansätze zur Wertebereichsextraktion und - zusammenführung, einzelne Internetdienste über Unteranfragen anzusprechen und ein vereinigtes Webformular einer Domäne zu erstellen, werden in dieser Arbeit in ähnlicher Form umgesetzt.

3.2. Generierung von Ontologien

Das manuelle Abbilden von relationalen Daten auf Ontologien ist ein zeitaufwändiger und fehleranfälliger Vorgang [1]. Diese Ontologien lassen sich allerdings ohne weitere Benutzerinteraktionen nicht präzise genug erstellen. Deshalb wird versucht diese Aufgabe mindestens teilautomatisch durchführen zu lassen. Im folgenden werden die Ansätze betrachtet aus Webformularen und aus XML Dokumenten Ontologien zu erstellen, da diese im World Wide Web sehr oft auftreten und viele Informationen mit sich führen.

3.2.1. Aus Webformularen

Zur Generierung von Ontologien aus Webformularen existieren drei verschiedene Ansätze.

- Webformulare zusammenführen und eine globale Ontologie generieren
- Webformulare auf Ontologien abbilden und diese zusammenführen

Diese Ansätze werden unter diesem Punkt mithilfe verwandter Arbeiten erläutert und anschließend die Vorteile erörtert. Die verwandten Arbeiten übernehmen meist nur je einen Teil der Ansätze. So beschäftigen sich manche mit der Generierung von Ontologien aus Webformularen, wohingegen andere sich mit der Ontologiezusammenführung befassen.

Webformulare zusammenführen und eine globale Ontologie generieren

Bei diesem Ansatz werden die Webformulare einer Domäne zuerst zusammengeführt, um daraus eine globale Ontologie generieren zu können. Wie Webformulare zusammengeführt werden können, wurde im vorherigen Abschnitt mit diversen Arbeiten vorgestellt.

[1] „Ontology engineering is a labor-intensive and knowledge-intensive task. As the size, number, and complexity of ontologies grow, ontology engineering and maintenance becomes increasingly difficult.“, Quelle: [Mor13]

Die Arbeit „Extracting Personalised Ontology from Data-Intensive Web Application: an HTML Forms-Based Reverse Engineering Approach“ [BMRB07] von S. M. Benslimane et. al. beschäftigt sich damit, Webformulare über Umwege bzw. über verschiedene Repräsentation eine Ontologie zu generieren. Zunächst werden die HTML-Webformulare zusammengeführt und auf ein XML-Schema abgebildet. Aus dem erstellten XML Dokument wird nun ein UML Klassendiagramm generiert. Über Ableitungsregeln werden die Komponenten des UML Klassendiagramms auf die Komponenten der Ontologie abgebildet und die dazugehörigen Relationen erstellt. Durch das Zwischenergebnis im UML Format lassen sich vorhandene Ableitungsregeln finden, die von UML zu beispielsweise RDFS, OWL oder DAML abbilden. Der Ablauf dieser Arbeit wird in Abbildung 3.5 veranschaulicht.

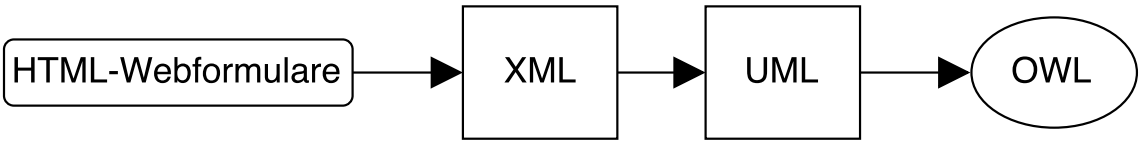

Abbildung 3.5.: Ablauf von [BMRB07]

Der Ansatz die zusammengeführten Webformulare über ein einheitliches Datenformat weiterzuverarbeiten, wird in dieser Arbeit verwendet.

Webformulare auf Ontologien abbilden und diese zusammenführen

Bei diesem Ansatz werden zuerst aus Webformularen Ontologien erstellt und diese danach zu einer globalen Ontologie zusammengeführt. So wird in der Arbeit „Fully Automatic Extraction and Consolidation of Ontologies from Web Sources using Sequence Semantics“ [GMJ04] von H. Roitman und A. Gal das Werkzeug OntoBuilder vorgestellt, das domänenübergreifend Webformulare verschiedener Webseiten analysiert und in einzelne HTML-Bausteine zerteilt. Danach werden die HTML-Formularelemente aus diesen HTML-Bausteinen identifiziert und daraus eine initiale, globale Ontologie (Zielontologie) bzw. lokale Ontologien (Kandidaten) für jedes Webformular entworfen. Durch „Ontology Matching“ bzw. „Alignment“ wird die globale Ontologie über die Kandidaten immer weiter verfeinert. Beim „Ontology Matching“ werden semantisch gleiche Entitäten verschiedener Ontologien gesucht, um diese dann mit „Ontology Alignment“ zusammenzuführen. OntoBuilder unterstützt dafür eine Vielzahl an Matchingalgorithmen, welche auf Konzept- und Wertebereichsübereinstimmung basieren.

Es existieren auch Arbeiten, die für jedes Webformular nur eine lokale Ontologie erstellen und diese nicht zusammenführen. Durch die damit hohe Anzahl an erstellten Ontologien, gibt es eine große Schnittmenge unter diesen. Um aus den vielen Ontologien eine wiederverwendbare, universelle Ontologie zu konstruieren, müssen sie zusammengeführt und angepasst werden. Das manuelle Zusammenführen von Ontologien ist ein zeitaufwändiger Prozess, der möglichst automatisiert werden soll. Dafür wurden verschiedene Werkzeuge entwickelt, die eine Halbautomatisierung der Ontologieverarbeitung ermöglichen sollen. Der Algorithmus „PROMPT“ [NM03] von N. F. Noy und Mark A. Musen wurde als Erweiterung von „Protege-2000“, einem Werkzeug zum Konstruieren von Wissensgrundlagen, entwickelt. Der Fokus von PROMPT liegt auf dem Zusammenführen von Ontologien. Der Algorithmus generiert - basierend auf der Eingabe von zwei Ontologien - Vorschläge, welche dem Nutzer präsentiert werden. Diese initialen Vorschläge werden über Matchingalgorithmen und den Konzeptnamen generiert. Der Nutzer wählt eine Operation für die weitere Verarbeitung aus, was zu neuen, automatisch generierten Vorschlägen führt. Dieser Vorgang wird so lange wiederholt bis die eingegebenen Ontologien erfolgreich zusammengeführt wurden. Durch die ständige Überwachung eines Benutzers werden die Resultate vielversprechender als bei einer vollautomatisierten Lösung. Ein weiterer Vorteil für die

Benutzerinteraktion ist, dass immer nur zwei Ontologien zur Eingabe verwendet werden, sodass die Benutzerentscheidungen aufgrund von wenigen Daten schnell getroffen werden können. Dieser Ablauf wird in der Abbildung 3.6 dargestellt.

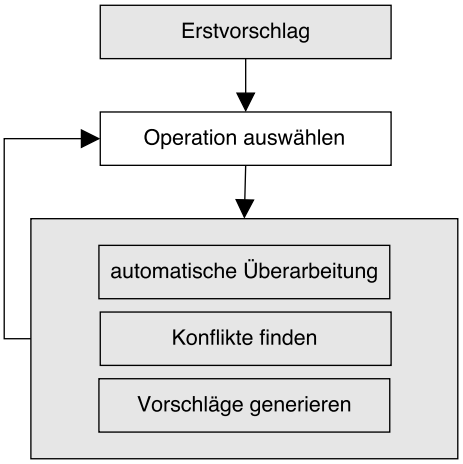

Abbildung 3.6.: Ablaufdiagramm von „PROMPT“, Quelle: [NM03]

Dieses Vorgehen, dem Benutzer einen initialen Vorschlag zu generieren, welcher erweitert werden soll, wird in dieser Arbeit auf ähnliche Weise verwendet.

Das Werkzeug „OntoMiner“, das in der Arbeit „Bootstrapping and Populating Ontologies From Domain Specific Web Sites“ [DVN03] von H. Davulcu et al. vorgestellt wird, nutzt die Struktur der Webformulare von 10-15 Webseiten einer Domäne und verarbeitet dieses unter Verwendung eines Partitionsalgorithmus zu einem hierarchischen Document Object Model (DOM) im XML Format. Dieser DOM-Baum wird nun über den Wortabstand nach mehrmals auftretenden semantisch gleichen Labels untersucht und daraus Konzepte erstellt. Im nächsten Schritt werden hierarchisch unterstehende Konzepte zu Instanzen umgewandelt, welche dann über „ist-ein“-Beziehungen verbunden werden. Der gleiche Ansatz wird auch von dem Werkzeug „DeepMiner“ aus der Arbeit „Bootstrapping Domain Ontology for Semantic Web Services from Source Web Sites“ [WDYM05] von W. Wu et al. verwendet. DeepMiner benutzt einen Satz an Webseiten einer Domäne als Eingabe und erkennt anhand des DOM-Baumes der Webformulare Konzepte, dazugehörige Instanzen und Relationen. Der Ablauf von DeepMiner wird in Abbildung 3.7 dargestellt. Durch die Webformulare wird eine initiale Ontologie erstellt (a). Diese Ontologie wird danach Schritt für Schritt erweitert, indem Anfragen an den Webdienst gesendet (c) und daraufhin Datenseiten erhalten werden (d). Aus den Datenseiten werden weitere Konzepte und Instanzen extrahiert (e). Die Extraktion findet über zwei Klassifikatoren (dem Bezeichner- und dem Instanzklassifikator) statt, die bei jedem Zyklus mit den vorhandenen Daten der Ontologie trainiert werden (b). Durch einen Clusteringalgorithmus werden die neu extrahierten Konzepte bzw. Instanzen mit der bestehenden Ontologie zusammengeführt (f).

Der Ansatz von OntoMiner und DeepMiner die hierarchische Struktur auszunutzen, wird in einer verwandten Form in dieser Arbeit verwendet, um Konzepte der aktiven Ontologie intern hierarchisch zu verwalten. Der maschinell lernende Ansatz von DeepMiner wird nicht umgesetzt, da Benutzerinteraktionen diese Arbeit in verbesserter Form abnehmen.

3.2.2. Aus XML

Im World Wide Web existieren auch massenhaft Daten in Form von XML Dokumenten. Deshalb ist es sinnvoll Werkzeuge zu erstellen, welche direkt XML Dokumente verarbeiten können.

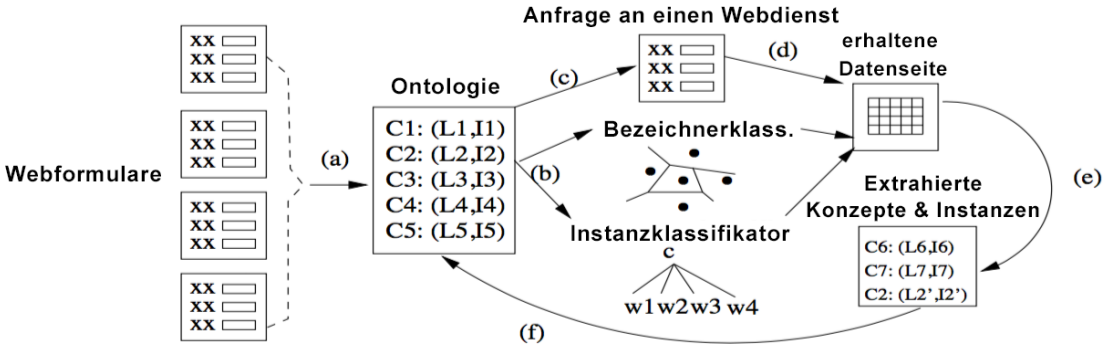

Abbildung 3.7.: Architektur von „DeepMiner“, Quelle: [WDYM05]

So beschäftigen sich die Publikationen „Constructing Complex Semantic Mappings between XML Data and Ontologies“ [ABM05], „Discovering and maintaining semantic mappings between XML schemas and ontologies“ [ABM08], sowie „Translating XML web data into ontologies“ [AM05] von Y. An et al. mit dem Abbilden von XML Dokumenten auf Ontologien. In den Veröffentlichungen wird ein Werkzeug vorgestellt, welches mit drei Eingaben arbeitet: Ein XML-Schema, eine Ontologie und einfach Ableitungsregeln zwischen XML-Schema und Ontologie. Diese drei Eingaben werden durch die Überwachung eines Benutzers und mithilfe eines heuristischen Algorithmus verarbeitet und semantische Abbildungen erzeugt.

Der Ansatz mit einem heuristischen Algorithmus eine aktive Ontologie zu generieren, wird in dieser Arbeit nicht verwendet. Die aktive Ontologie muss die verarbeiteten Internetdienste mit zulässigen Werten ansprechen können. Es reicht nicht aus Näherungswerte von zulässigen Werten als Wertebereich zu besitzen. Deshalb wird auf den heuristischen Algorithmus nicht näher eingegangen.

Die Arbeiten „Mapping XML to OWL Ontologies“ [BA05], sowie „Automated Ontology Creation using XML Schema Elements“ [SCP15], verfolgen den Ansatz bereits auf XML abgebildete relationale Daten auf Ontologien abzubilden. Dazu wird aus dem XML Dokument ein XML-Schema generiert, um danach durch Abbildungsregeln - wie in Tabelle 3.1 aufgelistet - zwischen XML-Schemaelementen und Elementen der „Web Ontology Language“ (OWL) eine Ontologie zu erstellen. Dies hat den Vorteil, dass festgelegt werden kann wie die einzelnen XML-Bausteine abgebildet werden sollen. Nutzt man diesen Umweg nicht und verwendet stattdessen das rohe Webformular als Eingabe, kann es durch fehlerhaften Aufbau der Webformulare zu fehlerhaftem Abbilden kommen.

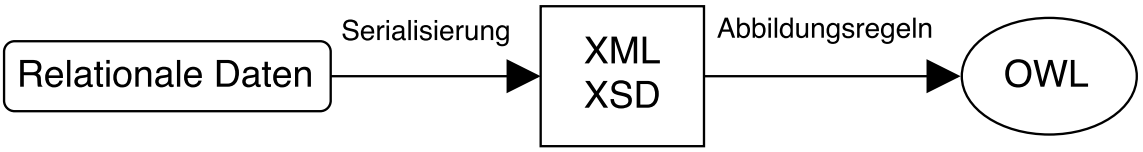

Abbildung 3.8.: Ablaufdiagramm der Arbeit „Mapping XML to OWL Ontologies“

Der Vorgehensweise mit einem bestehenden XML Dokument über Ableitungsregeln eine Ontologie zu erstellen, wird in dieser Arbeit nachgegangen. Anstatt einer Ontologie soll eine aktive Ontologie über Ableitungsregeln generiert werden.

XSD	OWL
xsd:elements, welche Unterlemente oder ein Attribut enthalten	owl:Class, gekoppelt mit owl:ObjectProperties
xsd:elements, welche keine Unterelemente und keine Attribute enthalten,	owl:DatatypeProperties
xsd:complexType, mit einem Namen	owl:Class
xsd:simpleType, mit einem Namen	owl:DatatypeProperties
xsd:minOccurs, xsd:maxOccurs	owl:minCardinality, owl:maxCardinality
xsd:sequence, xsd:all	owl:intersectionOf
xsd:choice	Kombination aus owl:intersectionOf, owl:unionOf und owl:complementOf

Tabelle 3.1.: Ableitungsregeln vom XML-Schema zu OWL, Quelle:

3.3. Verarbeitung von Ontologien mithilfe des Active Semantic Networks

In der Arbeit „Active: A unified platform for building intelligent web interaction assistants“ [Guz08] von D. Guzzoni, wird unter anderem das Active Semantic Network vorgestellt. Das Active Semantic Network verwendet die von Guzzoni entwickelten und implementierten aktiven Ontologien zur Verarbeitung natürlicher Sprache. Die Verarbeitung der natürlichen Sprache basiert auf einer „semantischen Bewertung“, welche von den Konzepten für die einzelnen Wörter erstellt wird. Diese semantische Bewertung erfolgt zwischen den Werten 0 und 100, mit denen die Worterkennung und Konzepteignung eingestuft wird.

3.3.1. Regeln

Guzzoni definiert in seiner Arbeit mehrere Regeln. Durch diese Regeln, gibt eine aktive Ontologie die Verarbeitung für natürliche Sprache in einer Domäne vor, indem die natürliche Sprache zuerst in einzelne Wörter (Fakten) zerlegt wird, um diese dann zu speichern. Dies führt dazu, dass die aktive Ontologie nicht nur Wissen über die natürliche Sprache repräsentiert, sondern auch verarbeitet und damit den Faktenspeicher koordiniert. Die Bedingungen werden über die Wörter (Fakten) geprüft und bei positiver Evaluierung wird die dazugehörige Aktion ausgeführt. Durch diese Evaluierung entsteht ein temporärer Zustand der aktiven Ontologie. Die Aktionen können Fakten zum Faktenspeicher hinzufügen, verändern oder löschen. Durch das Verändern von Fakten entsteht ein ständiger Evaluierungszyklus.

3.3.2. Konzepttypen

Es existieren mehrere Konzepttypen einer aktiven Ontologie, wobei man in erster Linie zwischen Terminalen und Nicht-Terminalen unterscheidet.

Terminale

Terminale sind Sensorknoten und entsprechen nach der Graphenanalogie deren Blattknoten. Sensorknoten agieren als Filter, um Signalwörter aus der natürlichen Sprache zu erkennen. Sie erhalten beim Erkennen der Signalwörter eine semantische Bewertung. Die Sensorknoten, die kein Signalwort erkennen, werden mit der semantischen Bewertung Null bewertet. Die Sensorknoten werden zwischen obligatorischen oder optionalen Knoten differenziert. Die semantische Bewertung der optionalen Knoten wird vernachlässigt, d.h. die Signalwörter dieser Knoten müssen nicht zwingend erkannt werden. Die Signalwörter der obligatorischen Knoten müssen für eine Weiterleitung an die Elternknoten erkannt werden. Man unterscheidet bei Sensorknoten zwischen den folgenden Knotentypen:

Vokabelknoten (VLN) vergleichen die Wörter der Eingabe auf Übereinstimmung mit einer Wortliste.

Präfixknoten/Postfixknoten (PN) sind Erweiterungen der *Vokabelknoten* und prüfen darüber hinaus, ob entsprechende Präfixe oder Postfixe vorhanden sind. Ein Präfix ist ein Wort, das vor einem anderen Wort steht, wobei ein Postfix nach einem anderen Wort steht.

Regexknoten (REN) überprüfen die Eingabewörter mittels regulärem Ausdruck

Spezialknoten (SN) übersetzen mit spezieller Logik zutreffende Eingabewörter in ein gewünschtes Zielformat (z. B. „morgen“ zu einem Datum).

Nicht-Terminale

Es existieren zwei verschiedene Typen von Nicht-Terminalen. Nicht-Terminale werden verwendet, um die Daten an den Elternknoten weiterzuleiten.

Kombinationsknoten leiten die durchschnittliche semantische Bewertung *aller* Kindknoten an den Elternknoten weiter.

Selektionsknoten leiten die höchste semantische Bewertung *eines* Kindknotens an den Elternknoten weiter.

3.3.3. Relationstypen

Nach Guzzoni wird zwischen zwei Typen von Relationen unterschieden, den strukturellen, sowie den klassifizierenden Relationen.

Strukturelle Relationen bauen eine „hat eine“ Beziehung zwischen Kind- und Nicht-Terminalen Elternkonzepten auf. Eine solche Relation beschreibt weitere Eigenschaften der Beziehung wie beispielsweise die Optionalität des Kindkonzeptes.

Klassifizierende Relationen bauen eine „ist ein/e“ Beziehung zwischen Kind- und Elternkonzept auf, wobei das Elternkonzept stets eine SelectNode darstellt. Diese Relation baut eine verallgemeinernde Hierarchie zwischen den Konzepten auf und beschreibt keine weiteren Eigenschaften.

3.3.4. Fakten im Active Semantic Network

Die Fakten entsprechen beim „Active Semantic Network“ den einzelnen Wörtern der natürlichen Sprache, die in einem gemeinsamen Faktenspeicher gehalten werden. Ein Fakt wird angelegt, indem aus der natürlichen Sprache vordefinierte Strukturen von Sensorknoten erkannt und weitergeleitet werden.

3.4. Diskussion und Zusammenfassung

Mit den vorgestellten Arbeiten, wurde verschiedene Ansätze zum Analysieren von Webformularen und Erzeugen von Ontologien aus Webformularen aufgezeigt. Es stellt sich heraus, dass die beiden Ansätze zur Erzeugung von Ontologien aus Webformularen gleichziehen.

Die Lösungen zur Abbildung von Webformularen auf Ontologien erfolgen ohne größere Benutzerinteraktion. Die teilautomatische Generierung einer aktiven Ontologie hingegen, bedarf einer ausgiebigen Rolle der Benutzerinteraktion. Deshalb wird in dieser Arbeit der vielversprechende Ansatz gewählt, zuerst die Webformulare einer Domäne zusammenzuführen [BMRB07], um diese dann über Ableitungsregeln auf eine aktive Ontologie abzubilden [BMRB07][BA05][SCP15]. Damit soll die Anzahl der Benutzerinteraktionen minimiert werden. Das übergeordnete Webformular wird auf ein einheitliches Datenformat

abgebildet. Dies erleichtert die Definition von Ableitungsregeln, da diese nicht generisch sein müssen, sondern für einzelne Bausteine des Datenformats festgelegt werden können. Dieser vielversprechende Ansatz senkt die Fehleranfälligkeit.

Um die Wertebereiche aus den Informationen der Webformulare bestmöglich zu extrahieren und zu erweitern, werden in dieser Arbeit die Ansätze verwendet die Programmierer- und Benutzeransicht der Webformularelemente zu verarbeiten und diese zu erweitern. Die Erweiterung eines Vorschlags der initialen Wertebereiche erfolgt über Benutzerinteraktionen [NM03] und über das WordNet [AGWC07a]. Um Wertebereiche semantisch gleicher Webformularelemente zusammenzuführen und damit die Internetdienste einzeln angefragt werden können, werden die Ansätze aus [HMYW05], [HMYW04] und [HML+07] verfolgt. Dabei werden Regeln zum Zusammenführen verschiedener Wertebereichstypen definiert und für die Anfrage der Internetdienste einzelne Unteranfragen erstellt. Der Ansatz über heuristische Algorithmen die Wertebereiche zu definieren [ABM05][ABM08][AM05] wird für diese Arbeit nicht gewählt, da die Internetdienste mit nur zulässigen Werten angefragt werden sollen.

4. Analyse

Das Ziel der vorliegenden Arbeit ist es, Webformulare einer bestimmten Domäne in eine aktive Ontologie zu überführen, um die verarbeiteten Webformulare über eine zentrale Funktion anzusprechen. Dies ermöglicht den intelligenten Assistenten den Umgang mit Flug-, Bahn und Hotelbuchungen. Der Ablauf gliedert sich in zwei Teilaufgaben, die in zwei Arbeiten bearbeitet werden: Zuerst müssen die Webformulare einer bestimmten Domäne auf ein übergeordnetes Webformular abgebildet werden, wie in Abbildung 4.2 dargestellt. Das übergeordneten Webformular muss wieder auf die ursprünglichen Webformulare abgebildet werden können. Im nächsten Schritt wird aus dem übergeordneten Webformular eine aktive Ontologie teilautomatisch generiert, indem über Ableitungsregeln einzelne Konzepte der aktiven Ontologie aus Webformularelementen erstellt und zusammengeführt werden. Dies ist in Abbildung 4.1 dargestellt.

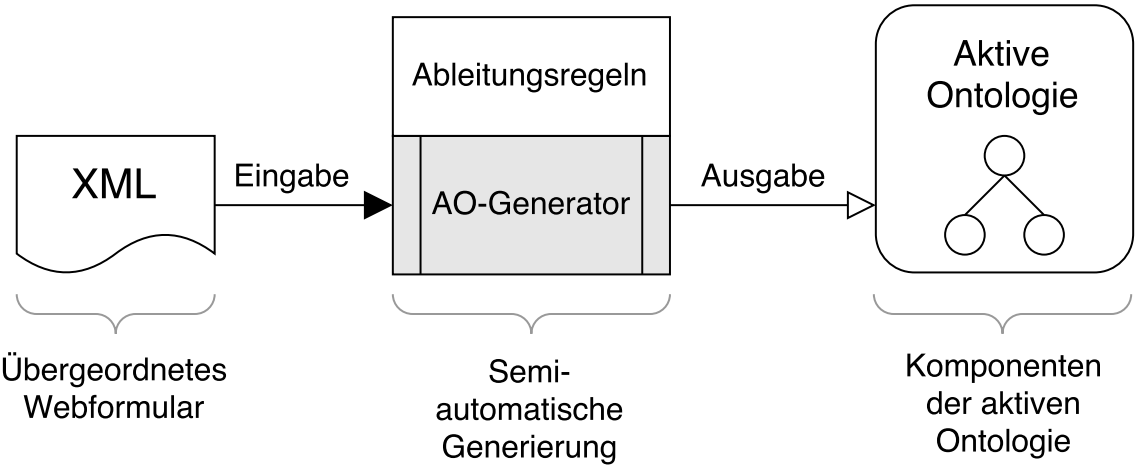

Abbildung 4.1.: Ablaufdiagramm des AO-Generators

Das folgende Kapitel legt den Fokus auf den zweiten Schritt und analysiert im Detail wie die Ableitungsregeln zu definieren sind und welche Webformularelemente automatisierbar abgebildet werden können. Außerdem wird ein Ausschnitt des ersten Schritts behandelt und dafür untersucht was es bei einer automatisierten Abbildung von Webformularelementen auf Komponenten einer aktiven Ontologie bedarf. Zusätzlich wird diskutiert, wie man aus den manuell abzubildenden Webformularelementen bestmöglich einen Vorschlag generiert und diesen vom Nutzer erweitern lässt. Außerdem wird untersucht welche Meta-

daten erforderlich sind, um das übergeordnete Webformular wieder auf die ursprünglichen Webformulare abzubilden. Dafür wird ein geeignetes XML Schema definiert. Das XML Schema wird in Kooperation mit einer parallel laufenden Arbeit „Extraktion und Konsolidierung von Webformularen zur Erzeugung von aktiven Ontologien“ [May17] von T. Mayer konstruiert, um dem zu entwickelnden AO-Generator ein einheitliches Eingabeformat zu gewährleisten. Dabei gibt diese Arbeit vor, wie die Metadaten im XML Schema repräsentiert werden sollen und von der Arbeit [May17] evaluiert wie und ob dies möglich ist.

4.1. Übergeordnetes Webformular einer Domäne

Das übergeordnete Webformular einer Domäne wird benötigt, da ansonsten aus jedem Webformular eine aktive Ontologie erzeugt werden müsste. Diese aktiven Ontologie müssten danach zusammengeführt werden, um eine aktive Ontologie einer Domäne zu erhalten. Da das Erstellen einer aktiven Ontologie ein zeitaufwändiger Prozess ist, werden zuerst die Webformulare zu einem übergeordneten Webformular zusammengeführt. Daraus kann direkt eine aktive Ontologie einer Domäne erzeugt werden. Aus mehreren Webformularen einer Domäne soll ein universelles Webformular erstellt werden, aus welchem sich die einzelnen Webformulare zurück abbilden lassen. In folgendem Punkt wird analysiert, welche Metadaten für eine Rückabbildung der einzelnen Webformulare dieser Domäne erforderlich sind und darauf aufbauend ein geeignetes XML Schema für das universelle Webformular definiert.

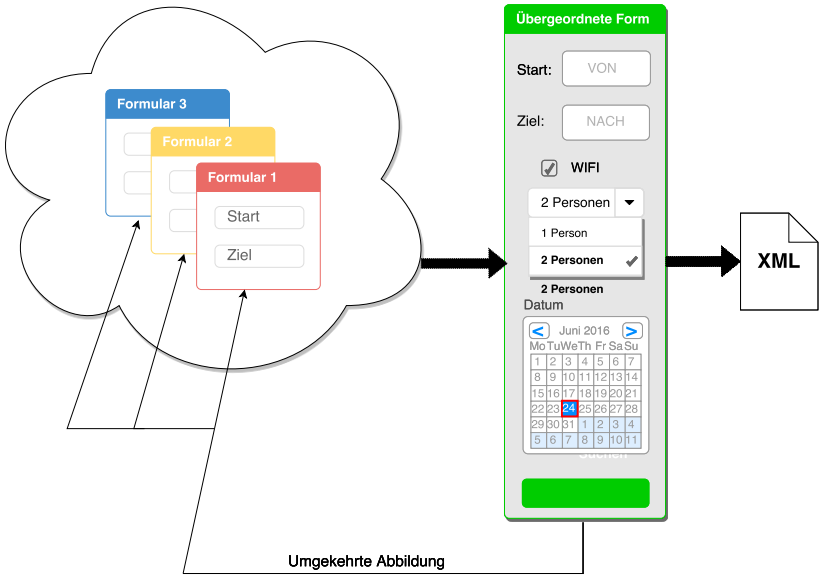

Abbildung 4.2.: Übergeordnetes Webformular zu Webformularen 1, 2 und 3

4.1.1. Repräsentation der Webformulare einer Domäne

Um mehrere Webformulare einer Domäne über ein übergeordnetes Webformular repräsentieren zu können, ist es unumgänglich, die relevanten Informationen aus jedem Webformularelement zu extrahieren und auf das übergeordnete Webformular abzubilden. Eine Voraussetzung für jedes Webformularelement ist, dass es einen Bezeichner besitzt über welchen man Elemente verschiedener Webformulare zusammenführen kann. Dieser Bezeichner

kann in Form eines Etiketts oder über Attribute wie *name* oder *placeholder* vorliegen. Mit diesem Bezeichner werden semantisch gleiche und ähnliche Webformularelemente verschiedener Webformulare erkannt und zu einem übergeordneten Webformularelement zusammengeführt. Diese Zusammenführung beinhaltet neben den Bezeichnern auch die Wertebereiche. Die Wertebereiche der Webformulare werden nach den Ansätzen aus den Arbeiten [HMYW05], [HMYW04] und [HML+07] verarbeitet, indem Regeln für die Zusammenführung zweier Wertebereiche definiert werden. Diese Regeln implizieren für zwei Datentypen von Wertebereichen einen Datentyp für den zusammengeführten Wertebereich. Das Erkennen semantisch gleicher und ähnlicher Webformularelemente wird in der parallel laufenden Arbeit [May17] behandelt und in dieser Arbeit nicht näher darauf eingegangen.

Enthält ein Webformularelement weitere Attribute, sind diese für mögliche Einschränkungen des Wertebereiches relevant und müssen ebenfalls berücksichtigt werden. Im folgenden werden alle Attribute beschrieben, die für die jeweiligen HTML-Formularelemente und für die Eingabetypen eine Einschränkung des Wertebereiches nach sich ziehen können. In Tabelle 4.1 sind diese Attribute zusammengefasst.

Formularelement	**Eingabetyp**	**einschränkende Attribute**
<select>	-	size, multiple
<textarea>	-	maxlength
<input>	text	list, maxlength, pattern
	email	
	search	
	url	
	tel	
	number	list, max, min, step
	range	
	datetime	
	date	
	month	
	week	
	time	
	password	maxlength, pattern
	file	multiple
	color	list

Tabelle 4.1.: Übersicht über Attribute, die den Wertebereich einschränken können

<select>

size gibt an, wie viele *option*-Felder im Auswahlmenü angezeigt werden.

multiple gibt an, ob in einem Auswahlmenü mehrere *option*-Felder ausgewählt werden können.

<textarea>

maxlength gibt an, wie viele Zeichen das Eingabefeld maximal beinhalten kann.

<input>

list gibt eine Liste mit zulässigen Eingabewerten vor.

maxlength gibt an, wie viele Zeichen das Eingabefeld maximal beinhalten kann.

max gibt an, wie groß der maximal zulässige Wert sein darf.

min gibt an, wie klein der minimal zulässige Wert sein darf.

multiple gibt bei Dateifeldern an, wie viele Dateien gleichzeitig ausgewählt werden können.

pattern gibt einen regulären Ausdruck vor, mit dem die Eingabe validiert wird.

step gibt über den Mindestwert vor, in welchem Abstand die Werte zulässig sind.

Wertebereichszusammenführung

Die Formularelemente des übergeordneten Webformulars besitzen die Wertebereiche, die sich aus der Wertebereichsvereinigung semantisch gleicher Formularelemente der Webformulare ergibt. Für die Zusammenführung von den Wertebereichen der verschiedenen HTML-Formularelementen, werden in Kooperation mit der Arbeit **??** Regeln zum Verarbeiten der Wertebereiche definiert. Die HTML-Formularelemente werden dafür zu fünf verschiedenen Wertebereichstypen gruppiert, wie in Tabelle 4.2 aufgelistet. Diese fünf Wertebereichstypen lehnen sich an die der Arbeiten [HMYW05], [HMYW04] und [HML$^+$07] an und untergliedern sich in:

Unendlich beschreibt die Wertebereiche, bei denen jeder Wert zulässig ist.

Endlich beschreibt die Wertebereiche, die abzählbar sind.

Eingeschränkt beschreibt die Wertebereiche, die beispielsweise über einen regulären Ausdruck definiert werden.

Hybrid unendlich beschreibt eine Kombination des Wertebereichs aus *endlich* und *unendlich*.

Hybrid eingeschränkt beschreibt eine Kombination des Wertebereichs aus *endlich* und *eingeschränkt*.

Mit diesen fünf Wertebereichstypen werden folgende Regeln zum Zusammenführen der Wertebereiche definiert. Die Regeln sind in der Tabelle 4.3 definiert.

Attributenzusammenführung

Im vorherigen Abschnitt wurde definiert, wie die Wertebereichstypen abgeleitet werden. Sind zwei Webformularelemente semantisch gleich, müssen auch deren Attribute für das übergeordnete Webformularelement zusammengeführt werden. Dazu werden zwei Strategien, eine Generalisierungsstrategie und eine Mehrheitsstrategie wie auch in den Arbeiten [HMYW05], [HMYW04] und [HML$^+$07] verwendet. Dabei werden numerische Werte mit der Generalisierungsstrategie und alphabetische Werte über die Mehrheitsstrategie zusammengeführt. Die ist in Tabelle 4.4 dargestellt.

In Quelltextausschnitt 4.1 ist beispielhaft dargestellt wie sich zwei semantisch gleiche Webformularelemente auf ein globales Webformularelement abbilden und deren Wertebereichstypen, sowie Attribute zusammengeführt werden.

Aus den jeweiligen Webformularelementen mit dem Eingabetyp „number“, den Mindestwerten 0 und 10, den Maximalwerten 50 und 70 bzw. den Schrittgrößen 5 und 10 entsteht ein globales Webformularelement vom Eingabetyp „number“ mit dem Mindestwert 0, Maximalwert 70 und der Schrittgröße 5. Die Wertebereichstypen vom Eingabetyp enquotenumber belaufen sich nach Tabelle 4.2 auf „eingeschränkt“. Über die Tabelle 4.3 lässt sich nun ablesen, dass aus diesen Wertebereichstypen wieder der Wertebereichstyp „eingeschränkt“ entsteht.

Wertebereichstyp	Formularelement	Eingabetyp	Attribute
Unendlich	<textarea>	-	-
	<input>	text	-
		search	-
Endlich	<select>	-	-
	<input>	checkbox	-
		radio	-
Eingeschränkt		number	-
		range	-
		email	-
		url	-
		tel	-
		text	pattern
		search	
		date	-
		datetime	-
		datetime-local	-
		month	-
		week	-
		time	-
Hybrid unendlich		gleich wie *unendlich*	list
Hybrid eingeschränkt		gleich wie *eingeschränkt*	

Tabelle 4.2.: Übersicht der Wertebereichstypen und der dazugehörigen HTML-Formularelemente

⊗	**Unendlich**	**Endlich**	**Eingeschränkt**	**Hyb. U.**	**Hyb. E.**
Unendlich	Unendlich	Hyb. U.	Eingeschränkt	Hyb. U.	Hyb. E.
Endlich		Endlich	Hyb. E.	Hyb. U.	Hyb. E.
Eingeschränkt			Eingeschränkt	Hyb. E.	Hyb. E.
Hybrid U				Hyb. U.	Hyb. E.
Unendlich E					Hyb. E.

Tabelle 4.3.: Regeln zur Zusammenführung der Wertebereichstypen

```
1 <!-- Formularelement von Webformular 1 -->
2 <input type="number" min="0" max="50" step="10" name="price" value="0"/>
3
4 <!-- Formularelement von Webformular 2 -->
5 <input type="number" min="10" max="70" step="5" name="price" value="25"/>
6
7 <!-- Formularelement des globalen Webformulars -->
8 <input type="number" min="0" max="70" step="5" name="price" value="0"/>
```

Quelltextausschnitt 4.1: Beispiel zur Vereinigung der Wertebereiche

4.1.2. Rückabbildung des übergeordneten Webformulars auf die ursprünglichen Webformulare

Die Rückabbildung der Webformulare aus dem globalen Webformulare ist unabdingbar, da die einzelnen Dienste der Betreiber angesprochen werden sollen. Deshalb ist es notwendig das globale Webformular mit weiteren Daten anzureichern. Welche Informationen dafür benötigt werden, wird im Folgenden untersucht.

Attribut	Strategie	Gewählter Wert
id	Mehrheitsstrategie	am häufigsten auftretender Wert
label		
name		
placeholder		
min	Generalisierungsstrategie	kleinster Wert
max		größter Wert
step		kleinster Wert, aber größer 0
maxlength		größter Wert
multiple	Wenn das Attribut einmal *true* aufweist, wird *true* gesetzt	*true* oder *false*
checked		
visible		
required		
pattern	Die regulären Ausdrücke werden verodert aneinandergereiht	alle
value	Es werden alle „value“-Attribute aufgelistet	alle

Tabelle 4.4.: Zusammenführung der Attribute

Für die Rückabbildung der einzelnen Webformulare aus dem globalen Webformular, müssen mehrere Aspekte betrachtet werden. Erstens benötigt man Informationen zu dem jeweiligen Anbieter des Webformulars, um den richtigen Dienst anfragen zu können. Zusätzlich sind für jedes Webformularelement die Metadaten erforderlich, um deren Wertebereich wiederherstellen zu können und um das Webformularelement identifizieren zu können.

Um den richtigen Dienst anfragen zu können, wird dessen URL sowie ein Identifizierer für das Webformular benötigt. Der Identifizierer wird für die Anfrage an das richtige Webformular benötigt, da manche Dienste mehrere Webformulare auf der gleichen Seite anbieten. Zur Wiederherstellung der Wertebereiche müssen die ursprünglich verwendeten einschränkenden Attribute jedes Webformularelements (siehe Tabelle 4.1) auf das globale Webformular abgebildet werden. Damit wird sichergestellt, dass bei einer Anfrage nur die Dienste in Anspruch genommen werden, bei denen die Anfrage auch zulässig ist. Eine Anfrage ist dann zulässig, wenn alle übermittelten Werte in den jeweiligen Wertebereichen der einzelnen Webformularelementen liegen. Die Anfragen mit nur zulässigen Werten wird von EASIER nicht unterstützt, ist aber in Zukunft erwünscht. Um zu überprüfen, ob die Werte zulässig sind und um eine Anfrage erfolgreich zu stellen, müssen die einzelnen Webformularelemente identifiziert werden. Dies geschieht über festgelegte Attribute, die dem globalen Webformular für jedes ursprüngliche Webformularelement mitgeliefert werden.

4.1.3. Abbildungstypen

Beim Abbilden von Webformularen einer Domäne auf ein übergeordnetes Webformular, wird zwischen zwei Typen von Abbildungen unterschieden: Einfache Abbildungen und komplexe Abbildungen. In diesem Abschnitt wird erläutert, wie diese Abbildungstypen aufgebaut sind und untersucht, wie man solche auf ein übergeordnetes Webformular abbildet.

Einfache Abbildung

Eine einfache Abbildung ist eine 1:1 Abbildung zwischen zwei semantisch gleichen Webformularelementen. Bei der Erstellung eines übergeordneten Webformulars entspricht dies der Abbildung eines Webformularelements eines untergeordneten Webformulars auf ein

Webformularelement des übergeordneten Webformulars. Die Abbildung 4.3 dient als anschauliches Beispiel für eine einfache Abbildung.

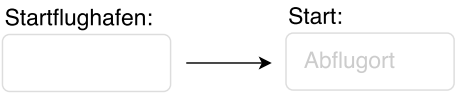

Abbildung 4.3.: Einfache Abbildung am Beispiel eines Textfeldes

Um einfache Abbildungen zurück abzubilden, müssen im übergeordneten Webformularelement die entsprechenden Informationen des untergeordneten Webformularelements beigefügt sein.

Komplexe Abbildung

Eine komplexe Abbildung ist eine 1:m Abbildung zwischen zwei Webformularen. Dabei erfüllen mehrere Webformularelemente eines Webformulars zusammen die Semantik eines einzigen Webformularelements wie z.B. in Abbildung 4.4 die drei Webformularelemente „Tag“, „Monat“ und „Jahr“ zusammen die Semantik einer Datumsangabe besitzen. Bei der Erstellung eines übergeordneten Webformulars entspricht dies der Abbildung zwischen mehreren untergeordneten Webformularelementen eines Webformulars und dem semantisch gleichen übergeordneten Webformularelement.

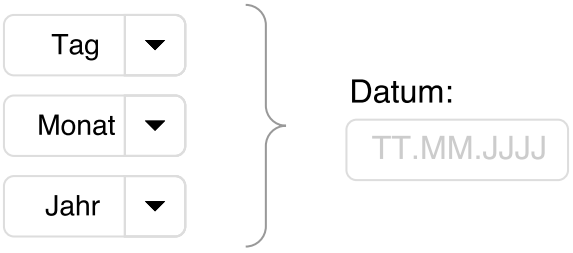

Abbildung 4.4.: Komplexe Abbildung (3:1) am Beispiel Datum

Um komplexe Abbildungen zurück abzubilden, müssen im übergeordneten Webformularelement die entsprechenden Informationen der untergeordneten Webformularelemente gruppiert beigefügt sein. Diese Informationen beinhalten die Daten der untergeordneten Webformularelemente und außerdem für jedes untergeordnete Webformularelement eine Metainformation. Die Metainformation gibt an, welchen Teil der Semantik das untergeordnete Webformularelement im übergeordneten Webformularelement übernimmt. Dadurch können die untergeordneten Webformularelemente bei einer Anfrage an den entsprechenden Dienst mit zulässigen Werten befüllt werden. Bei der Abbildung muss annotiert werden, dass es sich um eine komplexe Abbildung handelt.

4.1.4. XML Schema für das übergeordnete Webformular

Da das übergeordnete Webformular mit all den benötigten Metadaten nicht wie die einzelnen Webformulare über HTML repräsentiert werden kann, wird wie in den Arbeiten [BMRB07], [BA05]und [SCP15] die erweiterte Auszeichnungssprache (XML) verwendet. XML ist geeignet, weil es zum einen plattform- und implementationsunabhängig ist und sich bei XML außerdem eigene Tags definieren lassen. Damit lassen sich die Webformulare vollständig und mit den Informationen für die Rückabbildung auf ein übergeordnetes Webformular abbilden. Die Abbildung 4.5 zeigt den Aufbau des XML Schemas.

Das XML dient dem AO-Generator als Eingabe und wird auf das passende XML Schema (XSD) geprüft, damit der AO-Generator auf einem wohlgeformten XML-Dokument arbeitet. Das XSD (im Anhang zu finden) wurde im Sinne dieser und einer weiteren Arbeit, welche sich mit dem Zusammenführen der Webformulare einer Domäne beschäftigt, manuell entwickelt.

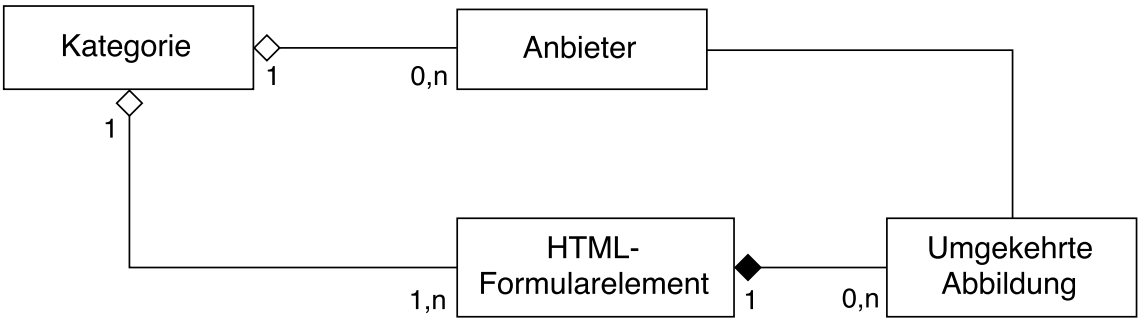

Abbildung 4.5.: Komponenten des XML Schema

Kategorie

Eine Kategorie beschreibt über einen Namen die Domäne des übergeordneten Webformulars. Eine Kategorie besteht aus 0..n Anbietern und Formularelementen.

Anbieter

Ein Anbieter besteht aus dem Namen des Webdienstes, sowie dessen URL. Ein Anbieter besitzt eine Kennung, mit welcher er in der Rückabbildung identifiziert werden kann.

HTML-Formularelement

Ein HTML-Formularelement enthält die Informationen der über den Konstruktionsplan zusammengeführten Webformularelementen der Webformulare einer Domäne. Diese Informationen bestehen aus einem Oberbegriff für das Webformularelement, dessen Attribute und Metainformationen wie des dazugehörigen Etiketts oder der Datenliste. Das HTML-Formularelement enthält außerdem die nötigen Attribute zur Rückabbildung derjenigen Webformularelemente, aus denen es zusammengeführt wurde.

Rückabbildung

Eine Rückabbildung enthält die Informationen des ursprünglichen Webformularelements. Über eine Kennung wird dieses Webformularelement mit dem Anbieter in Verbindung gebracht. Die enthaltenen Informationen bestehen aus Abbildungsrückschlüssen, dessen Attributen und Metainformationen wie des dazugehörigen Etiketts oder der Datenliste. Die Abbildungsrückschlüsse setzen sich aus der Identifikation des untergeordneten Webformularelements auf der Webseite des Webdienstes und dem Abbildungstyp zusammen. Der Abbildungstyp kann „einfach“ oder „komplex“ sein.

4.2. Abbilden von Webformularelementen auf Komponenten der aktiven Ontologie

Um aus einem Webformular eine aktive Ontologie zu erstellen, müssen alle Webformularelemente auf Komponenten einer aktiven Ontologie abgebildet werden. Da das manuelle Abbilden sehr zeitaufwändig ist, wird unter diesem Punkt untersucht welche Webformularelemente automatisch abgebildet werden können und was für eine automatische Abbildung nötig ist. Außerdem wird ein Konzept entworfen, um dem Nutzer für die manuell abzubildenden Webformularelemente einen Vorschlag zu generieren.

4.2.1. Automatisierte Abbildung

Der AO-Generator soll möglichst automatisiert die Webformularelemente auf Komponenten der aktiven Ontologie abbilden. Deshalb wird in diesem Abschnitt der Frage nachgegangen welche HTML-Formularelemente automatisch abgebildet werden können und was dafür benötigt wird. Um diese Frage zu beantworten, werden zunächst die einzelnen Komponenten der aktiven Ontologie untersucht. Ist bekannt aus welche Informationen die Komponenten bestehen, lässt sich definieren welche Daten ein HTML-Formularelement für die automatisierte Abbildung mit sich tragen muss.

Wesentliche Bestandteile der Knotentypen

Unter diesem Abschnitt wird offengelegt, welche wesentliche Bestandteile die einzelnen Knotentypen der aktiven Ontologie nach EASIER auszeichnen.

Vokabelknoten (VLN) definieren über die in der Vokabelliste enthaltenen Wörter einen Wertebereich.

Präfixvokabelknoten (PRE) bestehen aus der Hauptvokabelliste und der Präfix-Vokabelliste, welche je einen Wertebereich definieren.

Postfixvokabelknoten (POS) bestehen aus der Hauptvokabelliste und der Postfix-Vokabelliste, welche je einen Wertebereich definieren.

Präfix-Postfixvokabelknoten (PRE_POS) bestehen aus der Hauptvokabelliste, der Präfix-Vokabelliste und der Postfix-Vokabelliste, welche je einen Wertebereich definieren.

Regex-Knoten (REN) bestehen aus einem regulären Ausdruck, der den Wertebereich bildet. Dieser reguläre Ausdruck gilt als Repräsentation der Vokabelliste.

Attribute zur Wertebereichsextraktion

Der Kern der jeweiligen Komponenten bildet der Wertebereich der Vokabelliste. Für eine automatische Abbildung werden Informationen für die Wertebereichsbildung benötigt. Diese Informationen liegen in Form von Attributen, „option“-Elementen, Etiketten oder Datenlisten vor. Die Informationen sind für manche HTML-Formularelemente nicht zwingend erforderlich. Deshalb lassen sich diese HTML-Formularelemente nicht immer automatisiert abbilden. Aus welchen Informationen der Wertebereich für die jeweiligen HTML-Formularelemente extrahiert werden kann, wird im Folgenden untersucht.

Für die HTML-Formularelemente „checkbox“, „radio“ und „select“ lässt sich der Wertebereich für die Vokabelliste aus den beigefügten Etiketten und option-Elementen extrahieren.

Manche Eingabetypen haben von Grund auf einen vordefinierten Wertebereich. So erwarten beispielsweise „number“ und „range“ einen numerischen Wert. Für diese beiden Elemente sind alle Zahlen zulässige Werte. Die Zahlen definieren den Wertebereich. Weiter hat auch der Eingabetyp „color“ einen vordefinierten Wertebereich. Dieser vordefinierte Wertebereich besteht z.B. aus allen Farbcodes.

Die Wertebereiche der HTML-Formularelemente mit den Eingabetypen „tel“, „url“ und „email“ lassen sich über einen regulären Ausdruck definieren, da sie immer einem bestimmten Muster folgen. So lässt sich beispielsweise der Aufbau einer E-Mail-Adresse folgendermaßen abstrakt beschreiben: *ZEICHENKETTE@ZEICHENKETTE.TLD*
Dabei bildet *ZEICHENKETTE* eine Abfolge von Buchstaben, Zahlen und einigen Sonderzeichen. *TLD* steht für die Toplevel-Domain wie beispielsweise „.de“.

Weiter lassen sich auch die Wertebereiche aller Datumseingabetypen („datetime“, „date“, „month“, „week“ und „time“) über einen regulären Ausdruck definieren. Unter diese Kategorie fallen auch die Eingabetypen „text“ und „search“ insofern sie das Attribut „pattern“ beinhalten.

Haben HTML-Formularelemente das Attribut „list“ definiert, repräsentiert die darunter verlinkte Datenliste eine Teilmenge des Wertebereichs. Dies gilt für alle Eingabetypen, für welche „list“ ein kompatibles Attribut darstellt. Eine Übersicht der kompatiblen Attribute ist in Tabelle 2.2 aufgeführt.

Die in diesem Abschnitt beschriebenen HTML-Formularelemente sind automatisch abbildbar.

Formularelement	**Eingabetyp**	**Automatisch abbildbar**
<select>	-	✓
<input>	email	
	tel	
	url	
	number	
	range	
	datetime	
	date	
	month	
	week	
	time	
	color	
	checkbox	
	radio	
	text	mit „pattern“-Attribut
	search	
	password	
<textarea>	-	×

Tabelle 4.5.: Übersicht über die automatische Abbildbarkeit der HTML-Formularelemente

4.2.2. Manuelle Abbildung

Nicht immer ist eine automatische Abbildung der Webformularelemente auf Komponenten der aktiven Ontologie möglich. Ist dies der Fall, müssen die Komponenten der aktiven Ontologie manuell definiert werden. In diesem Abschnitt wird analysiert wie man diese Webformularelemente trotz der fehlenden Metadaten bestmöglich semi-automatisch abbildet. Dafür soll ein Vorschlag für den Knotentyp und die Wertebereiche präsentiert werden, welcher über den Benutzer ergänzt wird. Diese manuell abzubildenden Webformularelemente beschränkt sich nach Tabelle 4.5 auf das HTML-Formularelement <textarea> und die HTML-Formularelemente <input> mit den Typen „text“, „search“ und „password“. Die fehlenden Informationen für die automatisierte Abbildung der genannten Eingabetypen,

stellt beispielsweise das Attribut „pattern“ dar. Über das Attribut „list“ kann ein Vorschlag präsentiert werden.

Wird bei der Verarbeitung vom AO-Generator ein solch manuell abzubildendes Webformularelement erkannt, soll der Benutzer die fehlenden Informationen ergänzen. Dazu wird ein initialer Vorschlag generiert und danach der Benutzer befragt, wie es auch die Arbeit [NM03] vorsieht. Der Vorschlag umfasst einen Knotentyp und Wertebereiche für die Vokabellisten des Knotentyps. Eine Möglichkeit zur Generierung eines Vorschlags, wäre es als Informationen sämtliche „value“- und „placeholder“-Attribute semantisch gleicher Webformularelemente als Wertebereich zu präsentieren. Der Ansatz, diese Attribute für den Wertebereich zu verwenden, wurde in den Arbeiten [AGWC07a] und [AGWC07b] gewählt. Dies ist möglich, da bei der Konstruktion des globalen Webformulars die Attribute der einzelnen Webformulare erhalten bleiben. Die „value“-Attribute enthalten oft einen Standardwert, der zulässig ist. Die „placeholder“-Attribute enthalten oft einen Wert, der als Prä- oder Postix dient. Betrachtet man die große Menge an Webformularen einer Domäne, kann durch dieses Vorgehen ein großer Wertebereich für den Vorschlag generiert werden.

4.3. Aufbau von Ableitungsregeln

Damit das manuelle und automatisierte Abbilden von Webformularelementen auf Komponenten der aktiven Ontologie einheitlich durchgeführt wird, wird an die Ansätze der Arbeiten [BMRB07], [BA05] und [SCP15] angeknüpft. Diese Arbeiten definieren Ableitungsregeln zwischen einem XML Schema und einer Ontologie. Im Rahmen des AO-Generators werden deshalb Ableitungsregeln zwischen einem Webformular und einer aktiven Ontologie definiert. Dabei wird für jedes HTML-Formularelement eine Ableitungsregel definiert, welche einen Konstruktionsplan für das Konzept der aktiven Ontologie darstellt. Der Konstruktionsplan gibt vor welcher Knotentyp für das Konzept gewählt wird und aus welchen Attributen der Wertebereich definiert oder errechnet wird.

4.3.1. Bedingung

Damit die für das individuelle HTML-Formularelement die richtige Ableitungsregel gewählt wird, müssen bestimmte Bedingungen erfüllt werden. Die Ableitungsregeln werden grob über das HTML-Formularelement und Eingabetypen differenziert. Existieren unterschiedliche Wertebereichsberechnungen, wird über die gesetzten Attribute die Ableitungsregel endgültig bestimmt.

4.3.2. Knotentyp

Für jede Ableitungsregeln muss ein Knotentyp definiert werden. Dieser Knotentyp wird bei einem automatisch abbildbaren Webformularelement für die Komponente der aktiven Ontologie automatisch gewählt. Als Standard wird ein Knotentyp gewählt, der eine Hauptvokabelliste besitzt, da der Knoten dieser Vokabelliste als einziger immer obligatorisch innerhalb des Konzeptes ist. Dies entspricht den Knotentypen VLN, PRE, POS und PRE_POS. PRE, POS oder PRE_POS werden nur gewählt, falls die nötigen Informationen zu Prä- und/oder Postixen in der Ableitungsregel oder im Eingabedokument existieren. Diese Informationen werden aus den Attributen der HTML-Formularelemente gewonnen. Tritt bei einem Webformularelement ein regulärer Ausdruck in Form des Attributs „pattern“ oder eines geeigneten Eingabetyps auf, so wird ein Knotentyp gewählt, bei dem sich der Wertebereich über einen regulären Ausdruck definieren lässt. Dies entspricht dem Regex-Knoten.

4.3.3. Rückabbildung

Um aus der generierten aktiven Ontologie die einzelnen Webdienste wieder ansprechen zu können, ist es essentiell die Webformularelemente der Webdienste zu identifizieren. Deshalb wird in den Ableitungsregeln definiert, welche Attribute für die Identifikation der jeweiligen HTML-Formularelemente erforderlich sind. Im Normalfall reicht hierfür das Attribut „name“ aus. Es gibt allerdings HTML-Formularelemente, für welche mehr Informationen nötig sind. Dies soll der Quelltextausschnitt 4.2 verdeutlichen. In diesem Beispiel besitzen die beiden HTML-Formularelemente denselben Wert im „name“-Attribut. Damit der Unterbau die richtigen Informationen erhält, sind für diese HTML-Formularelemente zur eindeutigen Identifikation die Attribute „name“ und „value“ zusammen vonnöten.

```
1 <input type=”checkbox” name=”extras” value=”wifi” id=”box1”>
2 <label for=”box1”>WLAN</label>
3 <input type=”checkbox” name=”extras” value=”veg” id=”box2”>
4 <label for=”box2”>Vegetarisches Essen</label>
```

Quelltextausschnitt 4.2: Beispiel zur Identifikation eines Webformularelements mit den Attributen „name“ und „value“.

4.3.4. Definition der Ableitungsregeln

Die folgenden Regeln stellen die Ableitungsregeln dar. Die Ableitungsregeln definieren, wie ein Konzept der aktiven Ontologie aus einem Webformularelement generiert wird.

Regel 1

Die Regel 1 behandelt die numerischen Eingabetypen. Es existieren je nach Kombination der gesetzten Attribute verschiedene Wertebereiche. Es wird ein Knotentyp mit einer Vokabelliste benötigt. Zur Identifikation des HTML-Formularelements in der Rückabbildung ist das Attribut „name“ ausreichend.

Regel 1

HTML-Formularelemente

<input> mit Typ $t \in$ {number, range}

HTML-Darstellung

```
<input type="t" name="name" min="a" max="b" step="s">
```

Knotentyp

Vokabel-Knoten

Wertebereich

a. Falls kein Attribut definiert:
$V_a = \mathbb{R}$

b. Falls min, max und step definiert:
$V_b = [a_0, a_1, .., a_n]$ mit $a_0 := a \wedge a_n = a_{n-1} + s \wedge a_n \leq b \wedge a, b, s \in \mathbb{R} \wedge n \in \mathbb{N}_0$

c. Falls max definiert; falls step definiert; falls max und step definiert:
$V_c = (a_{-\infty}, .., a_n]$ mit $a_0 := 0 \wedge a_n = a_{n-1} + s \wedge a_n \leq b \wedge a, b, s \in \mathbb{R} \wedge n \in \mathbb{Z}$

Attribute zur Rückabbildung

name

Regel 2

Die Regel 2 behandelt Auswahlmenüs. Es existiert - abhängig davon ob das „multiple“-Attribut gesetzt ist - eine Einfach- oder Mehrfachauswahl. Dies führt zu zwei verschiedenen Wertebereiche, die allerdings gleich abgebildet werden. Ist das „multiple“-Attribut nicht oder auf *false* gesetzt, bilden die „option“-Elemente den Wertebereich. Ist das „multiple“-Attribut auf *true* gesetzt, wird der Wertebereich aus den „option“-Elementen und den möglichen Untermengen dieser gebildet, indem der dazugehörige Knoten mehrere Werte an seinen Elternknoten weitergeben kann. Es wird ein Knotentyp mit einer Vokabelliste benötigt. Zur Identifikation des HTML-Formularelements in der Rückabbildung werden die Attribute „name“ vom „select“-Element und „value“ bzw. die „value“-Attribute der ausgewählten „option“-Elementen benötigt.

Regel 2

HTML-Formularelemente

<select>

HTML-Darstellung

a.

```
<select name="name" multiple="false">
  <option value="val1">Wert 1</option>
  <option value="val2">Wert 2</option>
  ...
</select>
```

b.

```
<select name="name" multiple="true">
  <option value="val1">Wert 1</option>
  <option value="val2">Wert 2</option>
  ...
</select>
```

Knotentyp

Vokabel-Knoten

Wertebereich

Der Wertebereich wird von allen option-Feldern definiert:
$V =$ {Wert 1, Wert 2, ... }

Attribute zur Rückabbildung

a.
- name
- value

b.
- name
- Liste von value-Elementen

Regel 3

Die Regel 3 behandelt die HTML-Formularelemente, bei die Wertebereiche über einen regulären Ausdruck gebildet werden. Zur Identifikation des HTML-Formularelements in der Rückabbildung ist das Attribut „name“ ausreichend.

Regel 3

HTML-Formularelemente

<input> mit Typ $t \in$ {datetime, date, month, week, time, email, url, tel, color}

HTML-Darstellung

```
<input type="t" name="name">
```

Knotentyp

Regex-Knoten

Wertebereich

Der Wertebereich wird je über eine vordefinierten regulären Ausdruck definiert:
$V = \mathcal{L}(regex_t)$ mit $regex_t := REGEX$ für Typ t

Attribute zur Rückabbildung

name

Regel 4

Die Regel 4 behandelt Auswahlboxen. Die Wertebereiche werden bei diesen HTML-Formularelementen über die zugehörigen Etiketten bzw. deren „Werte“ gebildet. Die Wertebereiche werden gleich abgebildet, allerdings existiert beim Eingabetyp „checkbox“ eine Mehrfachauswahl. Deshalb wird bei diesem Eingabetyp der Wertebereich aus den Etiketten und den möglichen Untermengen dieser gebildet, indem der dazugehörige Knoten mehrere Werte an seinen Elternknoten weitergeben kann. Zur Identifikation des HTML-Formularelements in der Rückabbildung werden die Attribute „name“ und „value“ bzw. die „value“-Attribute der ausgewählten HTML-Formularelementen benötigt.

Regel 4

HTML-Formularelemente

<input> mit Typ $t \in$ {checkbox, radio}

HTML-Darstellung

a.

```
<input type="checkbox" name="name" value="val1" id="label1">
<label for="label1">Wert 1</label>
<input type="checkbox" name="name" value="val2" id="label2">
<label for="label2">Wert 2</label>
...
```

b.

```
<input type="radio" name="name" value="val1" id="label1">
<label for="label1">Wert 1</label>
<input type="radio" name="name" value="val2" id="label2">
<label for="label2">Wert 2</label>
...
```

Knotentyp

Vokabel-Knoten

Wertebereich

Der Wertebereich wird über die angegebenen Etiketten bzw. deren Werte definiert:
$V =$ {Wert 1, Wert 2, ... }

Attribute zur Rückabbildung

a.
- name
- Liste von value-Elementen

b.
- name
- value

Regel 5

Die Regel 5 behandelt Textfelder. Diese Textfelder können mit den Attributen „pattern“ und „list“ einen Wertebereich definieren. Das „pattern“-Attribut gibt einen regulären Ausdruck vor, über welchen der Wertebereich gebildet wird. Das „list“-Attribut gibt mit dem verbundenen „datalist“-Element den Wertebereich vor. Das „datalist“-Element umschließt „option“-Elemente mit dessen „value“-Attribute ein Vorschlag für den Wertebereich definiert wird. Dieser wird vom Benutzer bei Bedarf erweitert. Ist keines der Attribute gesetzt, wird der Wertebereich manuell vom Benutzer definiert. Zur Identifikation des HTML-Formularelements in der Rückabbildung ist das Attribut „name“ ausreichend.

Regel 5

HTML-Formularelemente

<input> mit Typ $t \in$ {text, search}

HTML-Darstellung

a.

```
<input type="t" name="name" pattern="REGEX">
```

b.

```
<input type="t" name="name" list="ident">
<datalist id="ident">
    <option value="Wert 1">
    <option value="Wert 2">
    ...
</datalist>
```

c.

```
<input type="t" name="name">
```

Knotentyp

a. Regex-Knoten b. Vokabel-Knoten c. Vokabel-Knoten

Wertebereich

a. Der Wertebereich wird über den regulären Ausdruck des Attributs „pattern“ definiert:
$V_a = \mathcal{L}(regex_{pattern})$ mit $regex_{pattern} := REGEX$

b. Der Wertebereich wird von allen option-Feldern definiert:
$V_b =$ {Wert 1, Wert 2, ... }

c. Der Wertebereich muss manuell vom Benutzer definiert werden.

Attribute zur Rückabbildung

name

Regel 6

Die Regel 6 behandelt das mehrzeilige Textfeld-Element. Der Wertebereich wird über den Benutzer manuell definiert. Zur Identifikation des HTML-Formularelements in der Rückabbildung ist das Attribut „name“ ausreichend.

Regel 6
HTML-Formularelemente
<textarea>
HTML-Darstellung
`<textarea name="name"></textarea>`
Knotentyp
Vokabel-Knoten
Wertebereich
Der Wertebereich muss manuell vom Benutzer definiert werden.
Attribute zur Rückabbildung
name

Formularelement	**Eingabetyp**	**Auto. abbildbar**	Knotentyp	Attribute für WB	Präfixe/Postfixe
<select>	-	✓	VLN, PRE, POS, PRE_POS	option-Felder	option-Felder, Etiketten, placeholder-Attribut
<input>	number			max, min, step	
	range				
	email		REN	-	
	tel				
	url				
	datetime				
	date				
	month				
	week				
	time				
	checkbox		VLN, PRE, POS, PRE_POS	value	Präfixe: with, without
	radio				
	text & search mit „pattern“-Attribut		REN	pattern	option-Felder, Etiketten, placeholder-Attribut
	text	×	VLN, PRE, POS, PRE_POS	list	
	search				
<textarea>	-			-	

Tabelle 4.6.: Ableitungsregeln zum Abbilden von HTML-Formularelementen auf Komponenten einer aktiven Ontologie

4.4. Erstellung der Komponenten für EASIER zum Verarbeiten einer aktiven Ontologie

Damit EASIER über eine aktive Ontologie natürliche Sprache verarbeiten kann, werden weitere Komponenten benötigt. Die zu der Kategorie einer aktiven Ontologie spezifizierten Komponenten sind die *Dienstkategorie*, sowie die *Dienstanbieter*. Die Dienstkategorie beschreibt abstrakt gesehen das übergeordnete Webformular über die Konzeptnamen der aktiven Ontologie. Ein Dienstanbieter steht für einen konkreten Internetdienst wie beispielsweise „Lufthansa“. Die Dienstanbieter beschreiben die Abbildungen zwischen den Identifikatoren der eigenen Webformularelemente und den Konzeptnamen, welche in der Dienstkategorie gespeichert sind.

4.4.1. Repräsentation der EASIER-Komponenten über eine Datenstruktur

Um die von den EASIER-Komponenten benötigten Daten effizient organisieren, speichern, laden und ändern zu können, muss eine geeignete Datenstruktur gefunden oder entwickelt werden. Im folgenden Abschnitt wird untersucht welche bestehenden Datenstrukturen zur Repräsentation der Struktur und Daten dieser Komponenten in Frage kommen und was zur Verwaltung dieser alles nötig ist.

Datenstruktur

Für die Erstellung der EASIER-Komponenten werden die Informationen des übergeordneten Webformulars benötigt. Diese untergliedern sich in Informationen für die aktive Ontologie und für die Dienstanbieter der Dienstkategorie. Eine aktive Ontologie benötigt die Informationen für Konzepte, Relationen und Wertebereiche. Die Dienstanbieter die Informationen für die Rückabbildung. Um diese Informationen hierarchisch zu ordnen, bedarf es einer eigenen Datenstruktur.

Eine aktive Ontologie weist eine hohe Ähnlichkeit zu einem Baum auf, bei dem die Knoten einen Namen tragen. Deshalb wäre als Grunddatenstruktur ein Baum geeignet. Die Knoten des Baumes entsprechen dann den Konzepten der aktiven Ontologie. Die Informationen für die Relationen, Wertebereiche und die Rückabbildung, können innerhalb der Konzepte organisiert werden.

Bearbeitung einer bestehenden aktiven Ontologie

Eine generierte aktive Ontologie muss für die nachträgliche Bearbeitung gespeichert werden, da die Variablenbelegungen der Objekte nach der Laufzeit verloren gehen. Für eine nachträgliche Bearbeitung müssen auch die Informationen für die Dienstkategorie und die Dienstanbieter gespeichert werden. Dafür wird ein Mechanismus benötigt, der die Variablenbelegung der Datenstruktur für die EASIER-Komponenten sichert (persistiert), um diese zu einem beliebigen Zeitpunkt zu restaurieren. Dies bedarf folgenden Kriterien:

- Objekttypen und Variablenbelegungen müssen gespeichert werden
- Unterobjekte bzw. der Objektbaum muss gespeichert werden

Diese Kriterien werden von unterschiedlichen Datenformaten wie JSON oder XML, sowie von relationalen Datenbanken eingehalten. Da für eine generierte aktive Ontologie nur eine Datenstruktur persistiert werden muss, reichen die Eigenschaften eines kompakten Datenformats aus.

4.4.2. Erstellung von Quelltextdateien für die EASIER-Komponenten

Nach EASIER werden also drei Komponenten benötigt, um die Webdienste erfolgreich über eine aktive Ontologie ansprechen zu können. Diese drei Komponenten werden in diesem Abschnitt nochmal erläutert und es wird untersucht, welche Informationen für jede Komponente benötigt werden. Die einzelnen Komponenten untergliedern sich in:

- Aktive Ontologie
- Dienstkategorie
- Dienstanbieter

Zum Erzeugen dieser Struktur muss der AO-Generator fähig sein aus der Repräsentation der EASIER-Komponenten Quelltextdateien zu generieren, welche in Syntax und Semantik mit denen in EASIER verwendeten Komponenten übereinstimmen.

4.4.3. Aktive Ontologie

Die aktive Ontologie enthält alle Konzepte der dazugehörigen Domäne. Ein Konzept besitzt einen Bezeichner, die Information, ob das Konzept für die Anfrage an den Internetdienst notwendig ist, und Vokabellisten bzw. reguläre Ausdrücke, über welche natürliche Sprache erkannt werden kann.

Um eine solche aktive Ontologie generieren zu können, benötigt man die Konzepte, die Relationen und die Vokabellisten der Konzepte. Die Konzepte werden durch die Ableitungsregeln teilautomatisch generiert, indem aus jedem Webformularelement des übergeordneten Webformulars ein Konzept erstellt wird. Beim Erstellen werden Wertebereiche definiert, aus welchen die Vokabellisten erzeugt werden. Die in den Ableitungsregeln definierten Knotentypen stellen die Informationen der Relationen zur inneren Konzeptkoordinationen. Für die aktive Ontologie muss ein als Wurzelknoten agierender Terminalknoten erstellt werden, welcher die generierten Konzepte zu einem Baum verbindet.

4.4.4. Dienstkategorie

Damit den Dienstanbietern die weitergeleiteten Informationen der Konzepte zugewiesen werden können, enthält eine Dienstkategorie die Konzeptnamen einer aktiven Ontologie und die Information derer Notwendigkeit.

Um die Dienstkategorie für eine aktive Ontologie zu erstellen, werden die generierten Konzepte benötigt und die Information, ob diese für die Anfrage obligatorisch oder optional zu verwenden sind. Diese Informationen werden aus der Datenstruktur für die Repräsentation der EASIER-Komponenten gewonnen.

4.4.5. Dienstanbieter

Von den Konzepten der aktiven Ontologie muss zurück auf die ursprünglichen Webformularelemente geschlossen werden können. Deshalb existiert für jeden Webdienst der Domäne ein eigener Dienstanbieter. Jeder Dienstanbieter enthält für jedes Konzept der aktiven Ontologie die Abbildungsinformationen zwischen den Webformularelementen von sich selbst und den Konzeptrepräsentanten, welche in der Dienstkategorie gespeichert sind. Zu den Abbildungsinformationen zählt außerdem, ob das Konzept für diesen Dienst obligatorisch oder optional zu behandeln ist.

Damit ein Dienstanbieter für die aktive Ontologie generiert werden kann, bedarf es den Informationen zur Identifikation der ursprünglichen Webformularelementen. Diese werden

aus der Datenstruktur der aktiven Ontologie bezogen, welche zuvor aus dem übergeordneten Webformular über die Ableitungsregeln teilautomatisch generiert wurde.

Es ist angedacht die Dienstanbieter nur mit zulässigen Werten anzufragen. Dies unterstützt EASIER momentan noch nicht. Damit dies möglich werden kann, müssen für jeden Dienstanbieter die zu jedem Webformularelement zulässigen Werte gespeichert werden.

4.5. Grafische Benutzeroberfläche zur Benutzerinteraktion

Damit der Nutzer übersichtlich aktive Ontologien semi-automatisch generieren, bearbeiten und löschen kann, muss eine grafische Benutzeroberfläche gewährleistet werden.

Erstellen von aktiven Ontologien soll unkompliziert möglich sein. Deshalb wird eine grafische Benutzeroberfläche geboten, über welche die übergeordneten Webformulare in geeignetem Format eingelesen werden können. Bei fehlerhaftem Einlesen bzw. bei nicht wohldefiniertem Eingabeformat soll eine Fehlermeldung erscheinen. Bei erfolgreichem Einlesen startet der AO-Generator mit der semi-automatischen Generierung der aktiven Ontologien. Kann ein Webformularelement nicht automatisch für ein Konzept der aktiven Ontologie abgeleitet werden, soll ein Dialogfenster erscheinen. Über dieses Dialogfenster soll dem Benutzer ein Vorschlag für den Knotentyp und die Wertebereiche für das manuell abzubildende Webformularelement angezeigt werden. Dabei ist das Dialogfenster in die Bereiche für den Knotentyp und die Wertebereiche unterteilt, welche bedingt durch den Vorschlag bereits Werte enthalten. Dieser Vorschlag kann von dem Benutzer bearbeitet, erweitert und bestätigt werden. Weiter müssen in diesem Dialogfenster Dateien in Form von Wortlisten für die Wertebereiche geladen werden können.

Bearbeiten von zuvor generierten aktiven Ontologien muss ermöglicht werden. Dies kann über ein Dialogfenster für je eine Komponente der aktiven Ontologie geschehen. Es wird eine aktive Ontologie aus dem Hauptdialogfenster ausgewählt. Über ein gesondertes Dialogfenster werden tabellarisch alle Komponenten dieser aktiven Ontologie angezeigt. Es soll eine Komponente ausgewählt werden können, welche sich über ein weiteres Dialogfenster bearbeiten lässt.

Löschen von zuvor generierten aktiven Ontologien kann über das Hauptdialogfenster geschehen. Im Hauptdialogfenster werden dem Benutzer alle zuvor generierten aktiven Ontologien tabellarisch aufgelistet. Es wird eine aktive Ontologie zum Löschen ausgewählt. Es muss eine Bestätigung des Benutzers über ein Dialogfenster erfolgen.

4.6. Zusammenfassung

Der Fokus der Analyse lag auf der Erstellung eines Konstruktionsplans eines übergeordneten Webformulars aus Webformularen einer Domäne sowie dem Aufbau von Ableitungsregeln zwischen Webformularelementen und Konzepten der aktiven Ontologie nach EASIER. Der Konstruktionsplan umfasst die nötigen Informationen, die zu einer Repräsentation der Webformulare benötigt werden, sowie für eine Rückabbildung vom übergeordneten Webformular zurück auf die einzelnen Webformulare. Anhand diesem übergeordneten Webformular wurde untersucht, was für eine automatisierte Abbildung zwischen Webformularelement und Konzepten der aktiven Ontologie nötig ist und wann eine Benutzerinteraktion unabdingbar ist. Dazu wurden sechs Ableitungsregeln definiert. Diese Ableitungsregeln geben vor, wie die Webformularelemente des übergeordneten Webformulars auf die Konzepte der aktiven Ontologie nach EASIER abgebildet werden.

Außerdem wurde untersucht wie die Informationen einer aktiven Ontologie gespeichert und organisiert werden können, um daraus die Komponenten der aktiven Ontologie nach

EASIER generieren zu können. Dabei wurde die Schnittstelle von EASIER eruiert und definiert, wie diese für eine erfolgreiche Verwendung der aktiven Ontologien basierend auf den übergeordneten Webformularen erweitert werden muss, damit die einzelnen Dienstanbieter erfolgreich angefragt werden können.

Mit den Ergebnissen dieser Analyse und der Beschreibung der Benutzerinteraktionen, wird im folgenden Kapitel der AO-Generator entworfen.

5. Entwurf & Implementierung

In diesem Kapitel werden der Entwurf, sowie die Implementierung des AO-Generators erläutert. Der AO-Generator hat die Aufgabe aktive Ontologien aus Webformularen zur Weiterverwendung im Projekt EASIER zu generieren. Mit einer generierten aktiven Ontologie lassen sich die abgebildeten Webformulare der Internetdienste mit EASIER über natürliche Sprache ansprechen.

5.1. Entwurf des AO-Generators

Die Zielsetzung dieses Werkzeuges liegt darin, teilautomatisiert aktive Ontologien aus übergeordneten Webformularen zu generieren. Um dieses Ziel zu realisieren, wurden Regeln für die Ableitung von HTML-Formularelementen zu Konzepten der aktiven Ontologien entwickelt. Dazu benötigt der AO-Generator als Eingabe ein wohldefiniertes XML-Dokument, welches das übergeordnete Webformular repräsentiert. Dieses XML-Dokument wird im Rahmen einer anderen Arbeit erstellt und enthält die in 4.1 analysierten Daten.

Die Abbildung 5.1 stellt den Ablauf des AO-Generators dar. Das eingegebene XML-Dokument wird im ersten Schritt in einzelne Dienstkategorien zerteilt, welche sukzessiv zu aktiven Ontologien verarbeitet werden. Dazu besitzt der AO-Generator einen Satz an Ableitungsregeln. Diese Ableitungsregeln geben einen Konstruktionsplan zur Ableitung der Daten von HTML-Formularelementen zu Konzepten der aktiven Ontologie vor. Diese Ableitung beinhaltet die Knotentypwahl des Konzepts, die Berechnung des Wertebereichs, sowie die Generierung der Relationen. Für jedes HTML-Formularelement des übergeordneten Webformulars der Dienstkategorie wird die passende Ableitungsregel zugeordnet. Über die gewählte Ableitungsregel wird nun entschieden, ob das Formularelement automatisiert zu einem Konzept abgeleitet werden kann. Ist eine automatisierte Ableitung möglich, wird anhand der Daten und der Berechnungsvorschrift für den Wertebereich die Vokabelliste generiert, der definierte Knotentyp gesetzt und die Relationen für die innere Koordination gebildet. Kann ein Formularelement nicht automatisiert abgebildet werden, wird mittels der Ableitungsregel ein bearbeitbarer Vorschlag für das zu generierende Konzept erzeugt und dem Benutzer präsentiert. Sind alle Formularelemente einer Dienstkategorie durchlaufen, wird die aktive Ontologie aus den erstellten Konzepten ausgegeben und die Konzepte für eine weitere Verwendung persistiert.

Im weiteren Verlauf dieses Kapitels wird zunächst der Grobentwurf mittels der grafischen Benutzeroberfläche und der Paketstruktur aufgezeigt und darauf folgend der Feinentwurf der einzelnen Komponenten beschrieben.

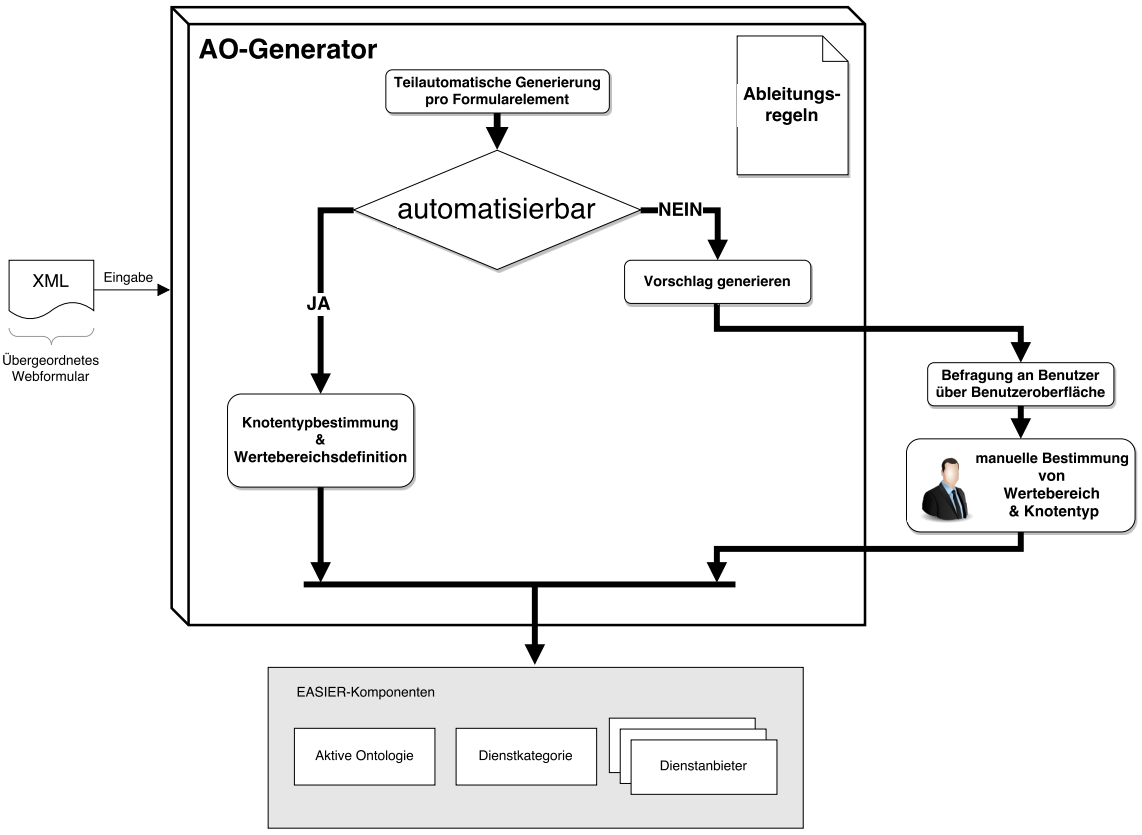

Abbildung 5.1.: Ablaufdiagramm des AO-Generators

5.1.1. Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche des AO-Generators ist in der Abbildung 5.2 dargestellt. Die Oberfläche des Hauptfensters gliedert sich in zwei unterschiedliche Bereiche. Der untere Bereich stellt eine Tabelle dar. Die Tabelle beinhaltet die bereits generierten aktiven Ontologien, um diese bearbeiten, löschen oder die EASIER Komponenten erneut bauen zu können. Diese aktiven Ontologien werden aus den persistierten Konzepten beim Start des AO-Generators geladen. Im oberen Bereich sind die unterschiedlichen Schaltflächen zum Bedienen des AO-Generators eingebettet. Über die *New*-Schaltfläche Dialogfenster zum Auswählen einer XML-Eingabedatei nach definiertem Schema. Die Generierung einer aktiven Ontologie wird nach Bestätigung aus dem ausgewählten XML-Dokument angestoßen. Die *Delete*-Schaltfläche löscht bei einem selektierten Tabellenelement nach Bestätigung die entsprechende aktive Ontologie. Es werden sowohl die persistierten Konzepte, als auch die generierten Quelltextdateien von dem System entfernt. Beim betätigen der *Rebuild*-Schaltfläche werden die Quelltextdateien für ausgewählte aktive Ontologie neu gebaut. Die *Edit*-Schaltfläche ruft bei einem selektierten Tabellenelement ein neues Dialogfenster zum Bearbeiten der ausgewählten aktiven Ontologie auf. Dieses Dialogfenster ist in der Abbildung 5.3 dargestellt.

Das Dialogfenster zum Bearbeiten einer aktiven Ontologie (Abb. 5.3) untergliedert sich in zwei Bereiche. Der erste Bereich stellt eine Tabelle dar, welche die einzelnen Konzepte der aktiven Ontologie beinhaltet. Im zweiten Bereich sind die Schaltflächen zum Bedienen des Dialogfensters eingebettet. Die *Delete*-Schaltfläche löscht das selektierte Konzept nach Bestätigung aus der aktiven Ontologie und generiert erneut deren Quelltextdateien. Die *Close*-Schaltfläche schließt das Dialogfenster. Die *Edit*-Schaltfläche öffnet bei einem ausgewählten Tabellenelement ein weiteres Dialogfenster zum Editieren des entsprechenden Konzepts. Dieses Dialogfenster ist in der Abbildung 5.4 dargestellt.

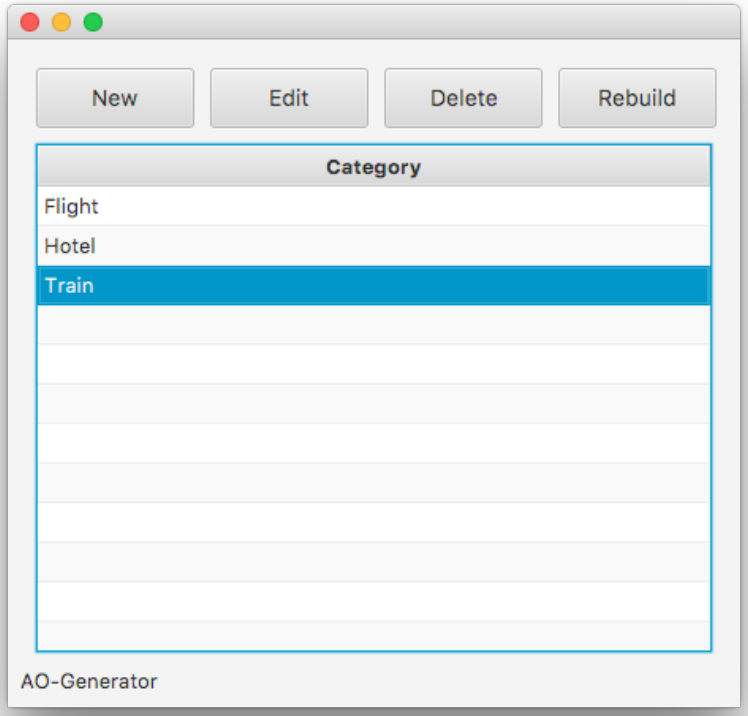

Abbildung 5.2.: Hauptfenster des AO-Generators

Das Dialogfenster zum Bearbeiten eines Konzepts stellt einen interagierenden Bereich dar. Es existiert ein Auswahlmenü, über welches der Knotentyp für das ausgewählte Konzept gesetzt wird. Je nach ausgewähltem Knotentyp, werden die Bereiche zum Definieren der Präfixvokabelliste, Vokabelliste und Postfixvokabelliste angezeigt. Die *Close*-Schaltfläche verwirft die vorgenommenen Änderungen, wohingegen die *Done*-Schaltfläche diese übernimmt.

5.1.2. Paketstruktur

Die Paketstruktur des AO-Generators wird in der Abbildung 5.5 modelliert. Der AO-Generator wird über die *Model-View-Controller*-Architektur koordiniert. Dabei verarbeitet das *Controller*-Paket die Benutzeraktivitäten und koordiniert diese über die Aktionen des *Processor*-Pakets. Dabei werden die Komponenten des *Model*-Pakets erzeugt und verändert. Außerdem verwaltet das *Controller*-Paket die Präsentationen und Dialoge des *View*-Pakets. Das *View*-Paket reagiert basierend auf dem Beobachtermuster auf Änderungen im *Model*-Paket. Das *I/O*-Paket ist für das Einlesen bzw. Ausgeben von Dateien zuständig.

Controller-Paket

Das *Controller*-Paket ist in der Abbildung 5.6 dargestellt und behandelt die Benutzerinteraktionen, welche über die Präsentationen getätigt werden. Dafür existieren im *Controller*-Paket zwei Klassen. Die *IndexController*-Klasse ist für die initiale Hauptpräsentation verantwortlich und leitet für die Generierung einer aktiven Ontologie die Benutzerinteraktionen an das *Processor*-Paket weiter. Atomare Funktionen wie das Bearbeiten oder

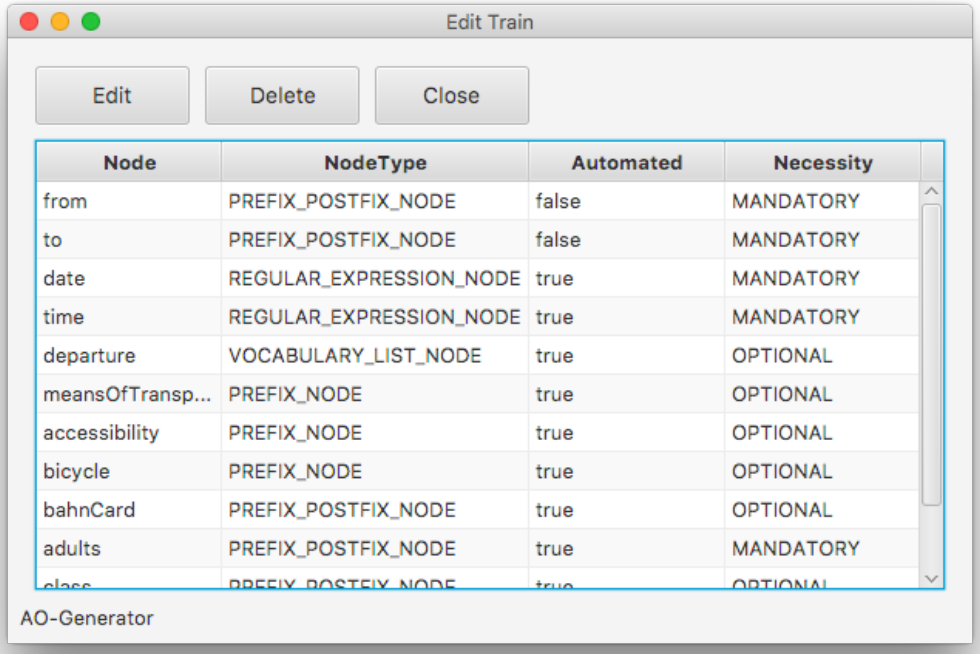

Abbildung 5.3.: Dialogfenster zum Editieren einer selektierten aktiven Ontologie

Löschen einer bestehenden aktiven Ontologie wird direkt von der *IndexController*-Klasse behandelt. Die *EditAOController*-Klasse ist für die Bearbeitung einer aktiven Ontologie und der dazugehörigen Bearbeitungspräsentation zuständig. Um ein Konzept der aktiven Ontologie zu bearbeiten, wird die *UIHelper*-Klasse eingesetzt.

View-Paket

Das *View*-Paket ist in der Abbildung 5.7 dargestellt und ist für die grafische Darstellung des AO-Generators zuständig. Dafür existieren zwei Klassen zur Darstellung der Haupt- und Bearbeitungspräsentation wie in den Abbildungen 5.2 und 5.3 veranschaulicht. Außerdem existiert eine weitere Präsentation zur Konzeptorganisation. Die Präsentationen werden über das *Model*-Paket basierend auf dem Beobachtermuster mit Informationen versorgt.

Model-Paket

Das *Model*-Paket ist in der Abbildung 5.8 dargestellt und ist für die Datenhaltung der Informationen der Webformularelemente zuständig. Die Hauptkomponente des *Model*-Pakets ist die *ActiveOntologyRep*-Klasse. Diese repräsentiert eine aktive Ontologie nach EASIER und enthält die dafür relevanten Informationen. Die *ActiveOntologyRep*-Klasse enthält eine Liste von Relationen und Konzepten. Die Konzepte sind zu je einer Baumstruktur über die *NodeTree*-Klasse angeordnet. Die Relationen werden über die *RelationRep*-Klasse und die Konzepte über die *ConceptNodeRep*-Klasse repräsentiert, welche sich im Unterpaket *datatypes* befinden.

Processor-Paket

Das *Processor*-Paket ist in der Abbildung 5.9 dargestellt und dient als Erweiterung des *Controller*-Pakets. Im *Processor*-Paket enthält die Logik zur Generierung einer Repräsentation der aktiven Ontologie. Dazu besteht das Paket aus drei Hauptkomponenten. Die

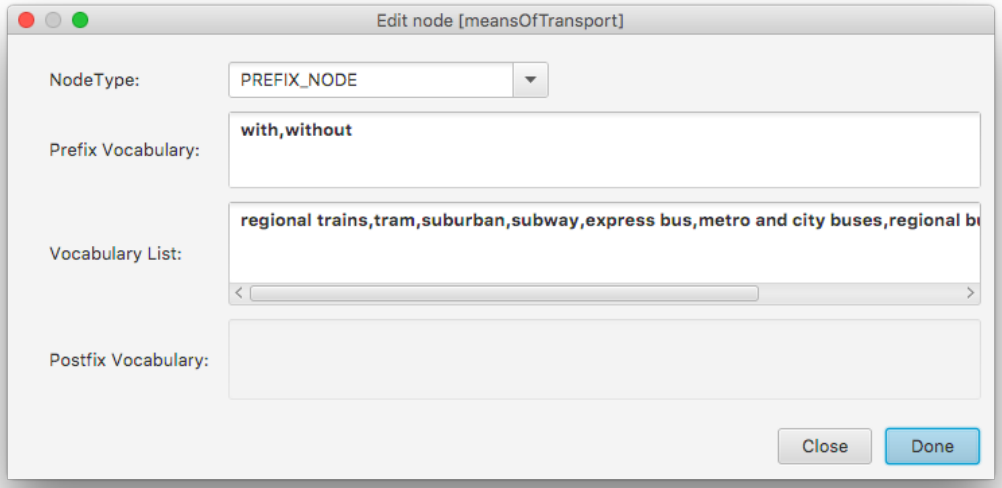

Abbildung 5.4.: Dialogfenster zum Editieren eines Konzepts der selektierten aktiven Ontologie

AOGenerator-Klasse koordiniert den Prozess der Konzept- und Relationserstellung für eine *ActiveOntologyRep*-Instanz. Dabei werden bei der Instanziierung dieser Klasse die Ableitungsregeln aus dem Unterpaket *guidelines* geladen. Für jedes HTML-Formularelement existiert genau eine Ableitungsregel. Die *AOGenerator*-Klasse steuert außerdem die Weiterleitung des *ActiveOntologyRep*-Objekts an die entsprechenden Ausgabeklassen. Die Konzept- und Relationserstellung erfolgt über die Fabriken im Unterpaket *factories*. Dabei wird für jedes Webformularelement des eingelesenen XML-Dokuments über die Ableitungsregeln iteriert und über deren Bedingung bei positiver Evaluierung die entsprechende Fabrik ausgewählt. Die Ableitungsregel, also eine konkrete Instanz der *AbstractGuidelines*-Klasse, dient der Fabrik gleichzeitig als Konstruktionsplan für das zu erstellende Konzept. Eine Ableitungsregel implementiert die Informationen für die Wahl des Knotentyps und für die Zusammensetzung des Wertebereichs.

I/O-Paket

Das *I/O*-Paket ist in der Abbildung 5.10 dargestellt. Dieses Paket behandelt die Ein- und Ausgaben des AO-Generators. Das Eingabedokument im XML-Format wird von der *XMLProcessing*-Klasse verarbeitet. Dabei wird das XML-Dokument ausgelesen und über die *FormsFactory*-Klasse eine Instanz der zum Schema passenden *Categories*-Klasse erstellt. Die *SourceCodeProcessor*-Klasse ist für die Generierung der Quelltextdateien für die EASIER-Komponenten verantwortlich. Dabei wird ein *ActiveOntologyRep*-Objekt ausgelesen und aus den vorhandenen Daten Quelltext erzeugt und als Java-Dateien ausgegeben. Dabei wird auch das *ActiveOntologyRep*-Objekt zur späteren Bearbeitung persistiert.

5.1.3. Laden und Verarbeiten des übergeordneten Webformulars

Der AO-Generator erhält als Eingabe ein XML-Dokument. Das XML-Dokument enthält die Informationen eines oder mehrerer übergeordneter Webformulare inklusive der Daten für die umgekehrte Abbildung. Dieses XML-Dokument ist das Ergebnis einer weiteren Arbeit, welche untergeordnete Webformulare gemäß des Konstruktionsplans für übergeordnete Webformulare zusammenführt und in der entsprechenden Struktur ausgibt. Beim

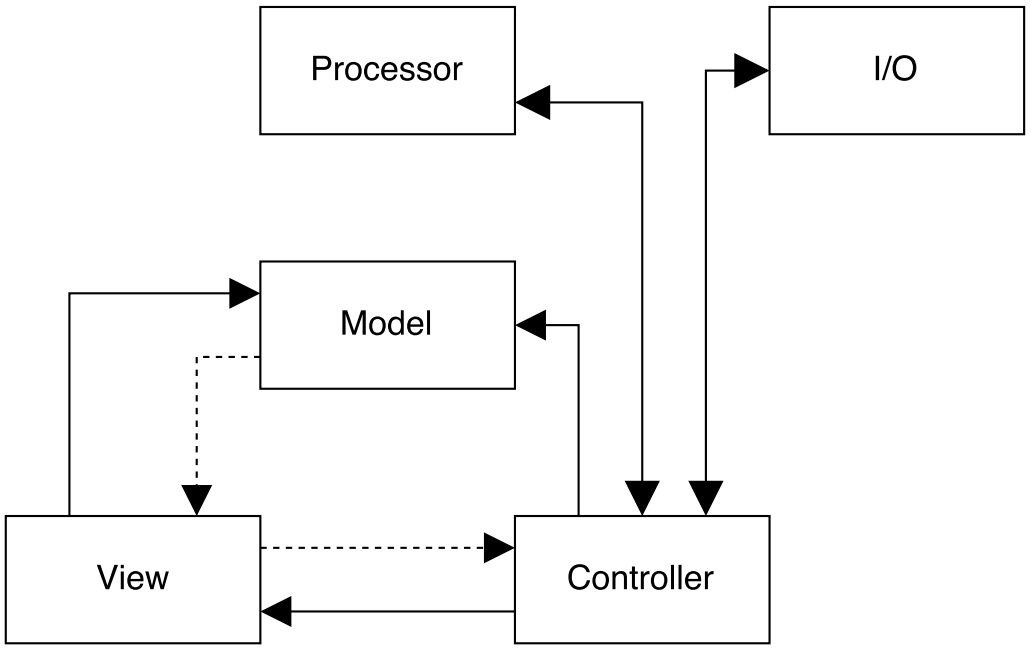

Abbildung 5.5.: Struktur des AO-Generators

Laden wird das XML-Dokument zerteilt und jedes übergeordnete Webformular mit allen Informationen in einem *Model* gespeichert. Das *Model* dient dem AO-Generator zur weiteren Verarbeitung.

5.1.4. Ableitungsregeln für Formularelemente

Das *Guideline*-Paket ist für die Ableitungsregeln, wie sie in 4.3.4 definiert sind, zuständig. Die Ableitungsregeln enthalten die Richtlinien zum Abbilden von Formularelementen auf Konzepte der aktiven Ontologie und bilden die Logik des AO-Generators. Eine Klasse des *Guideline*-Pakets repräsentiert je eine Ableitungsregeln und erbt von einer gemeinsamen, abstrakten Oberklasse. Die abstrakte Oberklasse ist an eine Fabrik zum teilautomatischen Erzeugen von Konzepten gekoppelt, die jede Klasse individuell setzt.

Die Grundidee ist es, für jedes Formularelement über die Ableitungsregeln zu iterieren und zu prüfen ob sich die Anforderungen der Ableitungsregel mit dem Formularelement decken. Die Anforderungen sind beispielsweise der Typ des HTML-Formularelements und gesetzte Attribute. Dies wird mit der *checkMatch*-Methode überprüft. Für jedes Formularelement existiert nur eine zulässige Ableitungsregel. Sobald die zutreffende Ableitungsregel ermittelt wurde, erstellt der AO-Generator ein neues Konzept für das Formularelement nach dem in der Ableitungsregel verankerten Konstruktionsplan.

5.1.5. Generierung einer Repräsentation der EASIER-Komponenten

Für die Generierung einer Repräsentation der aktiven Ontologie bedarf es der Interaktion mehrerer Pakete. Der Algorithmus des AO-Generators bedient sich am *Guideline*-Paket mit den Ableitungsregeln. Die Ableitungsregeln sprechen das *Factory*-Paket an, welche zum Erzeugen der einzelnen Bausteine der aktiven Ontologie auf das *Component*-Paket zugreifen.

Der AO-Generator erstellt für jede Dienstkategorie eine Instanz der hierarchisch aufgebauten *ActiveOntologyRep*-Klasse, welcher initial die Dienstanbieter übermittelt werden und ein Wurzelknoten mit dem Namen der Kategorie erstellt. Die Instanz wird mit den für jedes Formularelement generierten Konzepte und deren Relationen angereichert. Ein Konzept

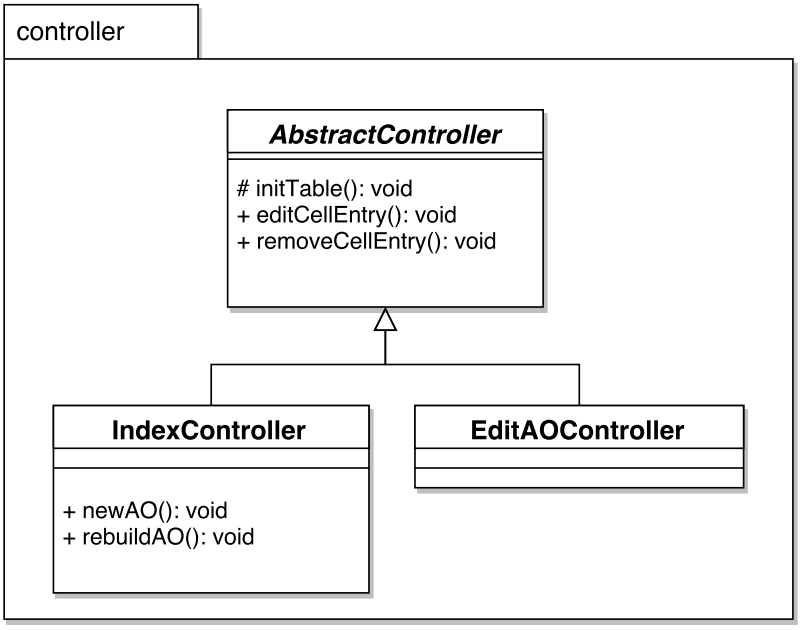

Abbildung 5.6.: *Controller*-Paket

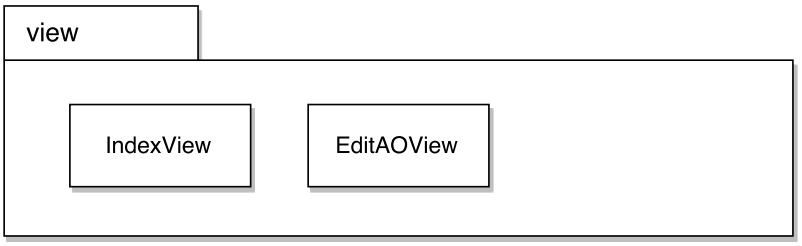

Abbildung 5.7.: *View*-Paket

wird über eine Baumstruktur dargestellt und über den Namen der Wurzel identifiziert. Alle Nicht-Terminale des Konzepts enthalten eine Vokabelliste oder geben die Information für einen regulären Ausdruck vor.

Der Algorithmus iteriert für jedes Formularelement der Kategorie über die Ableitungsregeln und prüft deren Bedingung. Ist die zutreffende Ableitungsregel ermittelt, wird über diese die entsprechenden Fabrik zum Erzeugen des Konzeptes zurückgegeben. Es wird die *createNode*-Methode der zurückgegebenen Fabrik aufgerufen, welche das Konzept Schritt für Schritt und individuell erstellt. Auf abstrakter Ebene erklärt, werden zuerst je nach Konzepttyp der oder die Knoten für das baumartige Konzept erstellt und danach die Ableitungsregel für jeden Knoten nach dem Wertebereich befragt. Die Fabrik gibt das Konzept als Baum zurück, welcher in einer Liste zwischengespeichert wird. Diese Liste wird danach im *ActiveOntologyRep*-Objekt gesetzt.

Für die generierten Konzepte werden anschließend die Relationen generiert und dem *ActiveOntologyRep*-Objekt hinzugefügt. Eine Relation wird mithilfe der dazugehörigen Fabrik erstellt. Dabei wird für jedes generierte Konzept die *createRelation*-Methode der Fabrik aufgerufen und das Quellkonzept, das Zielkonzept, sowie die Notwendigkeit des Quellkonzepts gesetzt. Das Quellkonzept entspricht dem aktuell betrachteten Konzept und das Zielkonzept der Wurzel des *ActiveOntologyRep*-Objekts. Sind alle Relationen erstellt, ist die Repräsentation der aktiven Ontologie vollständig.

5.1.6. Generierung der EASIER-Komponenten als Quelltextdateien

Zur Generierung des Quelltextes der aktiven Ontologie bedarf es einem bereits bestehenden *ActiveOntologyRep*-Objekt. Für diese Generierung ist das *Processor*-Paket zuständig. Das *ActiveOntologyRep*-Objekt wird durchlaufen und eine aktive Ontologie, eine Dienstkategorie und Dienstanbieter nach der Struktur von EASIER (Abschnitt 4.4.2) erstellt.

Dies wird erreicht, indem für jede EASIER-Komponente automatisch eine Java-Quelltextdatei erzeugt wird. Die Java-Quelltextdateien enthalten die nötigen Import-Angaben, Konstruktoren, Felder und Methoden. Außerdem wird ihnen die richtige Paketstruktur zugewiesen und auch unter diesem Pfad gespeichert. Nach dem erfolgreichen Erzeugen der Java-Quelltextdateien, agieren diese unter Benutzung von EASIER gemeinsam als aktive Ontologie. Damit ist es nun möglich die enthaltenen Dienstanbieter dieser Dienstkategorie zentral über natürliche Sprache anzusprechen.

5.1.7. Ausgabe

Nach jeder verarbeiteten Dienstkategorie, erzeugt der AO-Generator eine Ausgabe in Form von Dateien. Die ausgegebenen Dateien können in zwei Blöcke geteilt werden. Der erste Block umfasst die Persistierung der Repräsentation der aktiven Ontologie dieser Dienstkategorie. Der zweite Teil ergibt sich aus den generierten Quelltextdateien der EASIER-Komponenten wie im Abschnitt 5.1.6 beschrieben. Die Ausgabe der Quelltextdateien umfasst eine Quelltextdatei für die aktive Ontologie, eine für die Dienstkategorie, sowie je eine für die Dienstanbieter der Dienstkategorie.

5.2. Implementierung des AO-Generators

Der AO-Generator ist mit der Programmiersprache Java, in der Version 8, entwickelt. Im folgenden Abschnitt wird die Implementierung der entworfenen Bausteinen erläutert.

5.2.1. Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche ist mit der Grafikbibliothek „JavaFX“[1] implementiert. Die JavaFX-Laufzeitumgebung ist seit Java Runtime 7 als Bestandteil dieser verfügbar, wurde jedoch in der neusten Version für den AO-Generator importiert. Die grafische Benutzeroberfläche ist mit FXML (einer XML-Sprache) beschrieben und lässt sich damit in ein Model-View-Controller-Konzept auftrennen. Die Benutzeroberfläche ist durch eine *BorderPane* in einen oberen, mittleren und unteren Bereich aufgeteilt. In diesen Bereichen sind die Tabelle, die Schaltflächen und Etiketten angereiht.

5.2.2. Laden und Verarbeiten des übergeordneten Webformulars

Die Eingabe der Dateien erfolgt über die „Apache Commons IO“-Bibliothek[2]. Das eingelesene XML-Dokument wird von der Bibliothek zu einem String verarbeitet. Dieser String wird über die Programmierschnittstelle „Java Architecture for XML Binding“ (JAXB) zerteilt und aus den enthaltenen Kategorien über die *ObjectFactory*-Klasse eine Instanz der Klasse *Categories* generiert. Das *Categories*-Objekt enthält die aus dem XML-Dokument eingelesenen Kategorien und wird vom AO-Generator verarbeitet.

[1] *JavaFX 8*, http://docs.oracle.com/javase/8/javase-clienttechnologies.htm, zuletzt besucht am 8.November, 2016.

[2] *Apache Commons IO 2.5*, http://commons.apache.org/proper/commons-io/, zuletzt besucht am 8.November, 2016.

5.2.3. Ableitungsregeln für Formularelemente

Es ist für jedes HTML-Formularelement eine eigene Ableitungsregel implementiert. Die Ableitungsregeln erben von der gemeinsamen, abstrakten Oberklasse *AbstractGuideline*, welche den Bauplan für die Ableitungsregel vorgibt. Eine Ableitungsregel ist in drei Bereiche unterteilt: Der erste Teil übernimmt mit der Methode *checkMatch()* die Erkennung, ob die Ableitungsregel das aktuell betrachtete HTML-Formularelement ableitet. Dafür werden für die einzelnen Ableitungsregeln die Typbezeichner gespeichert, welche dann mit dem Typ des HTML-Formularelements vergleichen wird. Der zweite Teil setzt über die Methode *getFactory()* die richtige Fabrik für das HTML-Formularelement, indem die vorhandenen Informationen nach Präfixen, Postfixen oder regulären Ausdrücken durchsucht wird. Diese Fabrik ist für die weitere Verarbeitung verantwortlich. Der letzte Teil gibt je nach gewähltem Knotentyp über die Methoden *getDomain()*, *getPrefixDomain()* und *getPostfixDomain()* die Berechnungen der Wertebereiche und damit die Erstellung der Vokabellisten vor.

5.2.4. Generierung einer Repräsentation der EASIER-Komponenten

Nach dem Einlesen und Verarbeiten des XML-Dokuments, wird eine Instanz der Klasse *AOGenerator* erzeugt, welche als Parameter das zerteilte *Categories*-Objekt erhält. Beim Erzeugen der *AOGenerator*-Instanz werden die Ableitungsregeln initialisiert. Mit der Methode *generateAOForEachCategory()* wird der Prozess zur Erzeugung der Repräsentationen für der EASIER-Komponenten angestoßen. Dabei wird für jede Kategorie die Methode *generateAO()* aufgerufen, welche zuerst die Baumstrukturen für jedes Konzept erstellen und anhand dieser die notwendigen Relationen setzen.

5.2.5. Generierung der aktiven Ontologie nach EASIER

Die Generierung des Quelltextes für die aktiven Ontologien nach EASIER erfolgt über die Programmierschnittstelle „JavaPOET“[3]. JavaPOET ermöglicht die Generierung von Java-Klassen. Es wird eine Instanz der Klasse *SourceCodeProcessor* erzeugt, welche ein vollständiges *ActiveOntologyRep*-Objekt erhält. Es werden für die einzelnen Komponenten der aktiven Ontologie nach EASIER je eine Methode aufgerufen. Die Methode *generateAOCode()* generiert den Quelltext für die aktive Ontologie. Über die Methode *generateServiceProvidersCode()* wird der Quelltext für die einzelnen Dienstanbieter generiert. Die Dienstkategorie wird mit der Methode *generateServiceCategoriesCode()* erzeugt. Jede Methode erzeugt eine Quelltextdatei im Java-Format und speichert diese im Pfad des Paketes der generierten Java-Klasse.

5.2.6. Ausgabe

Die Ausgabe teilt sich in zwei Teile. Der erste Teil umfasst die Persistierung der *ActiveOntologyRep*-Objekte. Die Objekte werden über die „GSON“-Bibliothek[4] persistiert und mit der „Apache Commons IO“-Bibliothek in eine Datei im JSON-Format gespeichert. Der zweite Teil ist für die Ausgabe der Komponenten der aktiven Ontologie nach EASIER verantwortlich. Um Quelltext zu erzeugen, wird die Programmierschnittstelle „JavaPOET“ verwendet. Die Quelltexte für die einzelnen Komponenten werden danach über die „Apache Commons IO“-Bibliothek im nach dem Paket definierten Pfad gespeichert.

[4] *Google-Gson 2.7.0*, https://sites.google.com/site/gson/gson-user-guide, zuletzt besucht am 8.November, 2016.

[4] *JavaPOET 1.3.0*, https://github.com/square/javapoet, zuletzt besucht am 8.November, 2016.

5.3. Zusammenfassung

In diesem Kapitel wurde ein Generator zur semi-automatischen Erstellung aktiver Ontologie nach EASIER aus Webformularen entworfen.

Der in Java implementierte Generator wurde im Grobentwurf in der Model-View-Controller-Architektur entworfen und implementiert. Der Generator nutzt Ableitungsregeln, um aus den übergeordneten Webformularen, welche im XML Format eingelesen werden, die Komponenten einer aktiven Ontologie nach EASIER zu erstellen. Diese Komponenten werden als Java-Klassen ausgegeben. Außerdem lassen sich über eine grafische Benutzeroberfläche neben der Erstellung aktiver Ontologien, diese auch auf Konzeptebene bearbeiten und löschen. Kann ein Webformularelement nicht automatisch abgeleitet werden, wird dem Benutzer ein Vorschlag präsentiert. Dieser Vorschlag kann vom Benutzer erweitert werden.

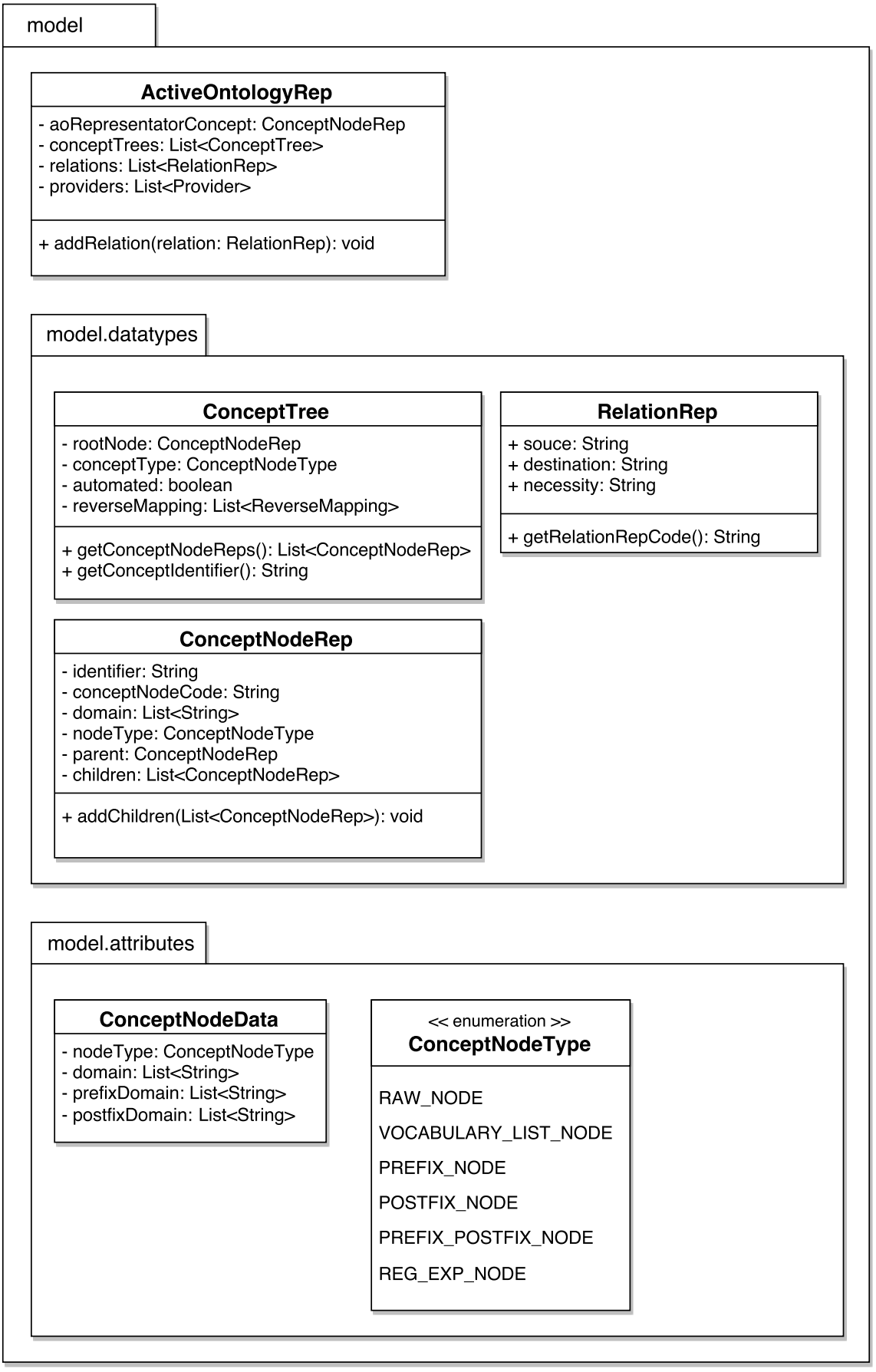

Abbildung 5.8.: *Model*-Paket

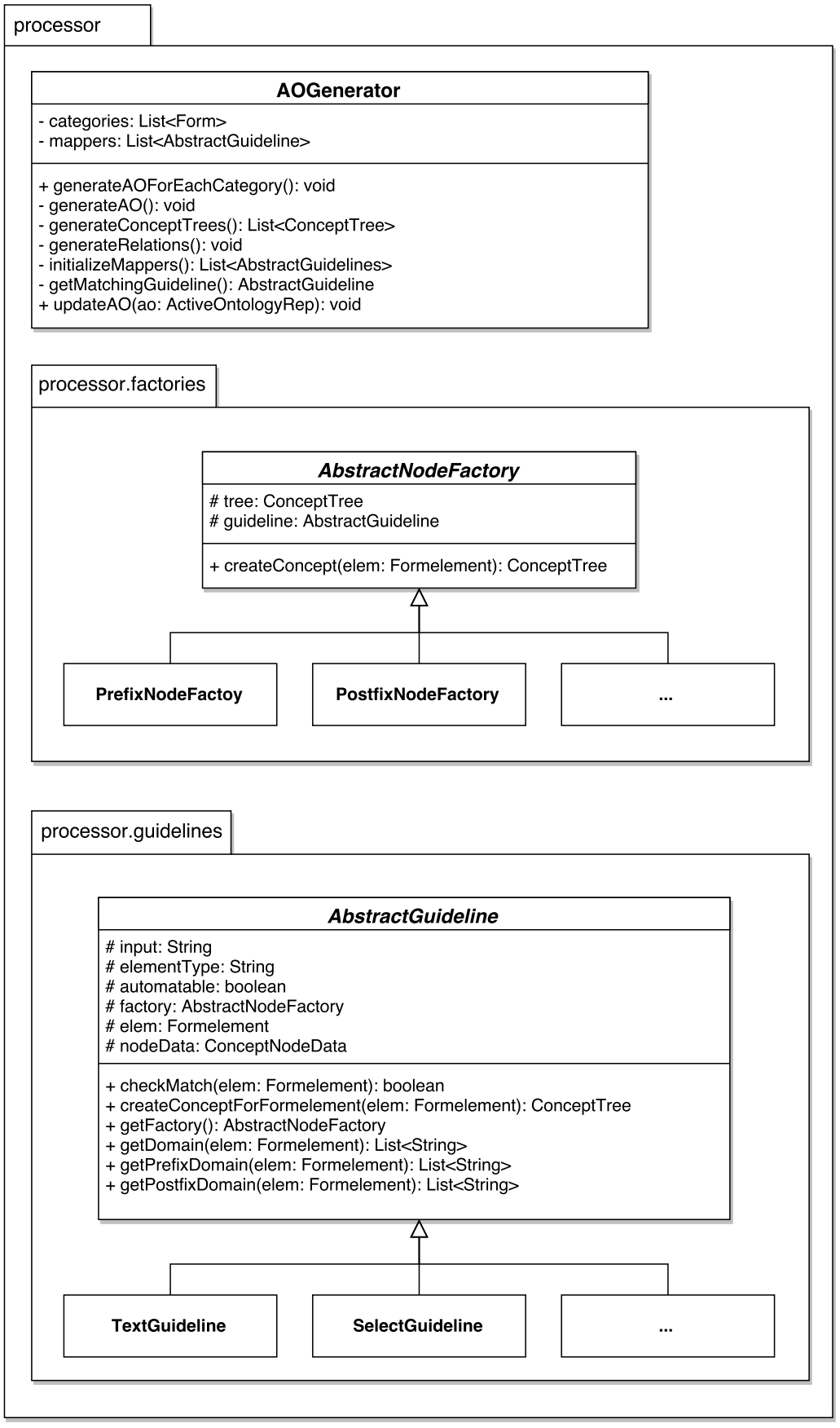

Abbildung 5.9.: *Processor*-Paket

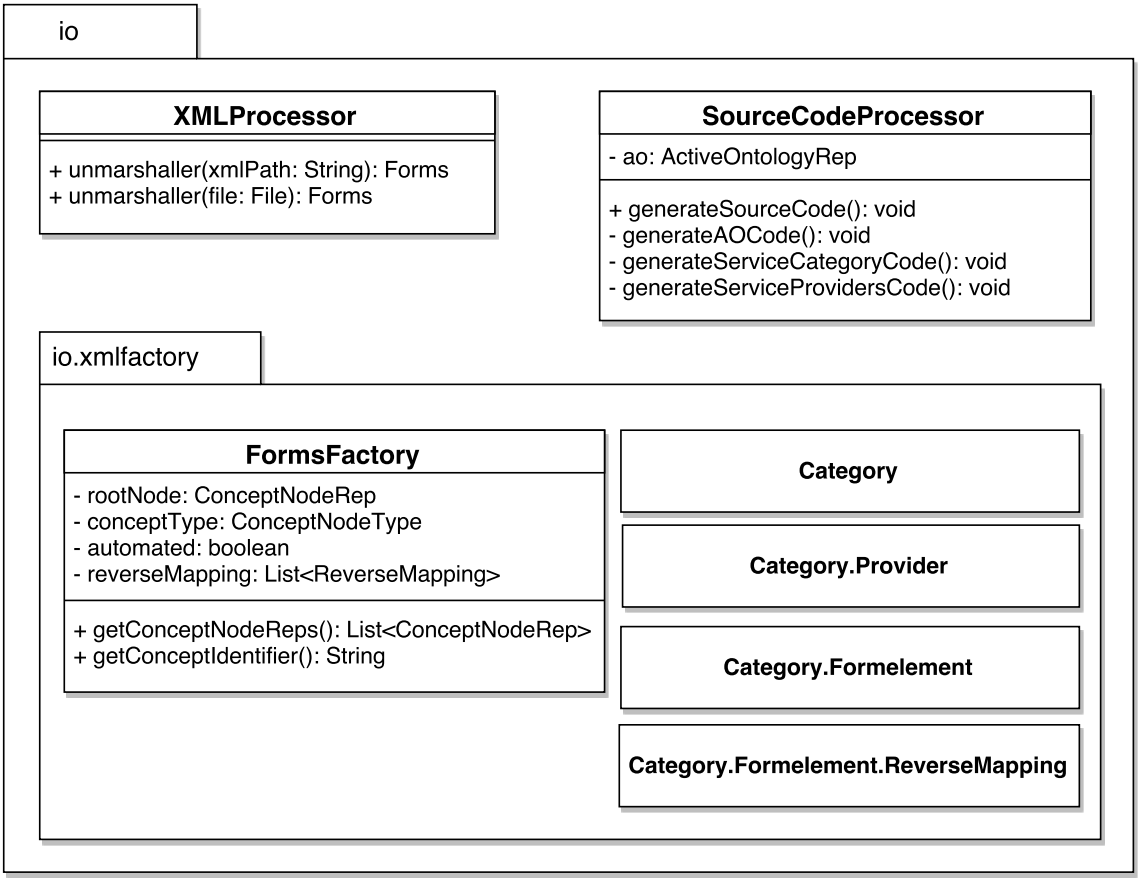

Abbildung 5.10.: *I/O*-Paket

6. Evaluation

Die Bewertung dieser Arbeit teilt sich in die Evaluierung der Präzision bzw. Fehlerquote und des Automatisierungsgrads. In einer vorangegangenen Arbeit „Abbildung von Webformularen auf aktive Ontologien“ [Sai16] von W. Said wurden manuell drei aktive Ontologien aus drei Kategorien (Flug, Bahn und Hotel) à 10 Webformularen erstellt. Die manuell erzeugten aktiven Ontologien definieren den Goldstandard. Für diese Evaluation werden mit dem AO-Generator drei aktive Ontologien aus demselben Datensatz generiert. Bezüglich der Präzision setzt sich diese aus den generierten aktiven Ontologien im Vergleich zum Goldstandard zusammen. Der Automatisierungsgrad wird anhand der generierten aktiven Ontologien ermittelt. Dabei werden die automatisch erzeugten Konzepte durch die Anzahl aller Konzepte geteilt. Dieser Quotient bildet den Automatisierungsgrad.

6.1. Aufbau der Evaluation

Zur Evaluation des AO-Generators muss das Eingabedokument manuell erstellt werden, da das Werkzeug zur einer parallel laufenden Arbeit noch nicht fertiggestellt wurde. Dieses Werkzeug ist für die automatische Erstellung des Eingabedokumentes im definierten XML Schema zuständig. Im Folgenden wird der Datensatz für die drei Kategorien (Flug, Bahn und Hotel) à 10 Webformulare festgelegt, das XML-Dokument manuell erstellt und daraus die aktiven Ontologien nach EASIER semi-automatisch mit dem AO-Generator erzeugt.

6.1.1. Datensatz des Goldstandards

Die folgenden Tabellen stellen die manuell erzeugten aktiven Ontologien aus der Masterarbeit „Abbildung von Webformularen auf aktive Ontologien“ [Sai16] von W. Said dar. Diese aktiven Ontologien definieren den Goldstandard. Am Goldstandard werden die vom AO-Generator erzeugten aktiven Ontologien evaluiert.

Die Tabellen enthalten die Konzepte bzw. die Konzeptnamen, aus welchen sich die Knotentypen ablesen lassen, und den Wertebereich. Der Wertebereich ist jeweils in der Spalte „Semantik (Ausdrücke)“ aufgelistet.

- Goldstandard zur Kategorie „Flug“: Tabelle B.1
- Goldstandard zur Kategorie „Bahn“: Tabelle B.2
- Goldstandard zur Kategorie „Hotel“: Tabelle B.3

6.1.2. Datensatz

Die folgenden Tabellen stellen einen Auszug der Masterarbeit „Abbildung von Webformularen auf aktive Ontologien“ [Sai16] von W. Said dar. Aus diesen Internetdiensten der drei Kategorien wird manuell das XML-Dokumente erzeugt, welches dem AO-Generator zur Eingabe dient. Die Kategorie „Flug“ wurde während der Implementierung als Trainingsmenge verwendet. Die Kategorien „Bahn“ und „Hotel“ stellen die Trainingsmenge dar. Da lediglich drei Kategorien zum Vergleich für die Ergebnisse des AO-Generators vorliegen, wurde nur eine Kategorie als Trainingsmenge gewählt.

Flugkategorie

Die in der Tabelle 6.1 dargestellten Fluggesellschaften wurden für den Goldstandard der Kategorie „Flug“ verwendet.

Fluggesellschaften	**URL**
Lufthansa	lufthansa.com
Air New Zealand	airnewzealand.com
Cathay Pacific Airways	cathaypacific.com
South African Airlines	flysaa.com
British Airways	britishairways.com
Ryanair	ryanair.com
Air Canada	aircanada.com/us/en
Condor	condor.com
Emirates	emirates.com
Swiss Airlines	swiss.com

Tabelle 6.1.: Ausgewählte Fluggesellschaften, Quelle: [Sai16]

Bahnkategorie

Die in der Tabelle 6.2 dargestellten Bahn-Anbieter wurden für den Goldstandard der Kategorie „Bahn“ verwendet.

Bahn-Anbieter	**URL**
Deutsche Bahn	bahn.de
Karlsruher Verkehrsverbund	kvv.de
Münchner Verkehrs- und Tarifverbund	mvv-muenchen.de
Verkehrsverbund Berlin-Brandenburg	vbb.de
Verkehrsver. Bremen-Niedersachsen	vbn.de
Hamburger Verkehrsverbund	hvv.de
Nordhessischer Verkehrsverbund	nvv.de
Aachener Verkehrsverbund	avv.de/de
Mitteldeutscher Verkehrsverbund	mdv.de
Saarländischer Verkehrsverbund	saarvv.de

Tabelle 6.2.: Ausgewählte Bahn-Anbieter, Quelle: [Sai16]

Hotelkategorie

Die in der Tabelle 6.3 dargestellten Hotels wurden für den Goldstandard der Kategorie „Hotel“ verwendet.

Hotels	URL
Ibis	ibis.com/de/hotel-6965-ibis-karlsruhe-hauptbahnhof/index.shtml
Swissotel	swissotel.com/hotels/dresden
Copacabana Rio	copacabanariohotel.com.br/en
Andaz Wall Street	newyork.wallstreet.andaz.hyatt.com/en/hotel/home.html
Hotel Windsor	hotel-windsor.ch
The Toren	thetoren.nl/en
Burj Al Arab	jumeirah.com/en/hotels-resorts/dubai/burj-al-arab
ITC Hotel	itchotels.in/itcgrandbharat
Seven Stars Galleria	sevenstarsgalleria.com
Dar Zemora	darzemora.com

Tabelle 6.3.: Ausgewählte Hotels, Quelle: [Sai16]

6.1.3. Eingabedokument für den AO-Generator

Das übergeordnete Webformular wird manuell aus Webformularen der im Abschnitt 6.1.2 aufgelisteten Internetdiensten nach dem Konstruktionsplan aus Abschnitt 4.1 erstellt. Dabei wird aus dem übergeordnete Webformular nach dem XML Schema im Anhang 7.1 ein XML-Dokument erzeugt. Dieses XML-Dokument dient dem AO-Generator als Eingabe. Das XML-Dokument enthält Informationen für die Dienstanbieter, die Dienstkategorie und für die einzelnen Webformularelemente des übergeordneten Webformulars. Außerdem sind die Informationen für die Rückabbildung, wie in Tabelle 6.4 dargestellt, enthalten.

Übergeordnetes Webformular		**Dienstanbieter „Lufthansa“**	
Webformularelement	**Werte**	**„name“-Attribut**	**„value“-Attribut**
Number of Adults	0 adults	adults	0
	1 adult		1
	2 adults		2
	3 adults		3
	4 adults		4
	5 adults		5
	6 adults		6
	7 adults		-
	8 adults		-
	9 adults		-

Tabelle 6.4.: Rückabbildungsinformationen im XML-Dokument am Beispiel des Webformularelements „Number of Adults“ und dem Dienstanbieter „Lufthansa“

6.2. Ergebnisse des AO-Generators

In diesem Abschnitt werden die Ergebnisse des AO-Generators vorgestellt. Die Ergebnisse umfassen die generierten aktiven Ontologien aus den drei Kategorien „Flug“, „Bahn“ und „Hotel“.

Flug

In der Tabelle 6.5 werden die Ergebnisse des AO-Generators für die Kategorie „Flug“ aufgelistet. Der AO-Generator hat 13 Konzepte erstellt, von welchen zehn automatisiert erzeugt wurden. Die Tabelle zeigt zu jedem Konzept den Namen, den gewählten Knotentyp, ob das Konzept automatisiert erstellt wurde und ob das Konzept obligatorisch oder optional für die Dienstkategorie ist.

Konzeptname	Knotentyp	Automatisiert	Notwendig
from	PRE_POS	×	✓
to	PRE_POS	×	✓
departing	REN	✓	✓
returning	REN	✓	✓
adults	PRE_POS	✓	✓
children	PRE_POS	✓	✓
infants	PRE_POS	✓	✓
class	PRE_POS	✓	×
promoCode	PRE_POS	×	×
teens	PRE_POS	✓	×
flexible	VLN	✓	×
airline	PRE_POS	✓	×

Tabelle 6.5.: Ergebnisse des AO-Generators zur Kategorie „Flug“

Bahn

In der Tabelle 6.6 werden die Ergebnisse des AO-Generators für die Kategorie „Bahn“ aufgelistet. Der AO-Generator hat zwölf Konzepte erstellt, von welchen neun automatisiert erzeugt wurden. Die Tabelle zeigt zu jedem Konzept den Namen, den gewählten Knotentyp, ob das Konzept automatisiert erstellt wurde und ob das Konzept obligatorisch oder optional für die Dienstkategorie ist.

Konzeptname	Knotentyp	Automatisiert	Notwendig
from	PRE_POS	×	✓
to	PRE_POS	×	✓
date	REN	✓	✓
time	REN	✓	✓
departureArrival	VLN	✓	×
meansOfTransport	PRE	✓	×
accessibility	PRE	✓	×
bicycle	PRE	✓	×
bahnCard	PRE_POS	✓	×
adults	PRE_POS	✓	✓
class	PRE_POS	✓	×
viaStation	PRE_POS	×	×

Tabelle 6.6.: Ergebnisse des AO-Generators zur Kategorie „Bahn“

Hotel

In der Tabelle 6.7 werden die Ergebnisse des AO-Generators für die Kategorie „Hotel“ aufgelistet. Der AO-Generator hat neun Konzepte erstellt, von welchen fünf automatisiert erzeugt wurden. Die Tabelle zeigt zu jedem Konzept den Namen, den gewählten Knotentyp, ob das Konzept automatisiert erstellt wurde und ob das Konzept obligatorisch oder optional für die Dienstkategorie ist.

6.3. Metriken der Bewertung

Die folgenden Metriken geben Auskunft über die Qualität der generierten aktiven Ontologien im Vergleich zum Goldstandard. Die typischen Problematiken beim Generieren von

Konzeptname	Knotentyp	Automatisiert	Notwendig
arrival	REN	✓	✓
departure	REN	✓	✓
destination	PRE_POS	×	×
hotel	PRE_POS	×	×
adults	PRE_POS	✓	×
children	PRE_POS	✓	×
rooms	PRE_POS	✓	×
promoCode	PRE_POS	×	×
groupCode	PRE_POS	×	×

Tabelle 6.7.: Ergebnisse des AO-Generators zur Kategorie „Hotel“

aktiven Ontologien sind Konzepte, die automatisiert abgebildet werden. Diese Konzepte können sich von den manuell erzeugten Konzepten des Goldstandards unterscheiden. Die Unterschiede belaufen sich dabei auf dem Wertebereich und dem Knotentyp. Im folgenden werden zwei Metriken zum Evaluieren der Qualitätsunterschiede und eine für den Automatisierungsgrad vorgestellt.

6.3.1. Präzision

Die Präzision P_G gibt Auskunft darüber, wie viel Prozent der generierten Konzepte mit den semantisch gleichen Konzepten des Goldstandards übereinstimmen. P_G setzt sich aus der Präzision des Wertebereichs und der Präzision des Knotentyps zusammen.

$$P_G = \frac{P_W + P_K}{2} \qquad (0 \leq P_G \leq 1) \tag{6.1}$$

Wertebereich

Die Präzision des Wertebereichs P_W gibt Auskunft darüber, zu wie viel Prozent die Wertebereiche der generierten Konzepte mit denen der semantisch gleichen Konzepten des Goldstandards übereinstimmen. Der Wertebereich stimmt nicht mit dem des Goldstandards überein, sobald er nur eine Teilmenge von diesem bildet oder ein Wert abweicht.

Richtig (R_W): Wertebereich stimmt überein.

Falsch (F_W): Wertebereich stimmt nicht überein.

$$P_W = \frac{R_W}{R_W + F_W} \qquad (0 \leq P_W \leq 1) \tag{6.2}$$

Knotentyp

Die Präzision des Knotentyps P_K gibt Auskunft darüber, zu wie viel Prozent die Knotentypen der generierten Konzepte mit denen der semantisch gleichen Konzepten des Goldstandards übereinstimmen.

Richtig (R_K): Knotentyp stimmt überein.

Falsch (F_K): Knotentyp stimmt nicht überein.

$$P_K = \frac{R_K}{R_K + F_K} \qquad (0 \leq P_K \leq 1) \tag{6.3}$$

6.3.2. Automatisierungsgrad

Der Automatisierungsgrad gibt an, wie viel Prozent der Konzepte vom AO-Generator automatisiert generiert wurden.

Automatisiert (A): Konzepte, die automatisiert erstellt wurden.

Nicht automatisiert (NA): Konzepte, die nicht automatisiert erstellt wurden.

$$G_A = \frac{A}{A + NA} \qquad (0 \leq G_A \leq 1) \tag{6.4}$$

6.4. Bewertung der generierten aktiven Ontologien

Für die Evaluation werden die generierten aktiven Ontologien des AO-Generators anhand der Metriken bewertet. Die Metriken messen den Qualitätsunterschied zum Goldstandard, sowie den Automatisierungsgrad der Konzepte.

6.4.1. Präzision

In diesem Abschnitt werden die Konzepte der generierten aktiven Ontologien mit den Metriken zum Evaluieren der Präzision evaluiert. Es werden die Unterschiede der Wertebereiche und der die der Knotentypen gemessen. Anschließend wird der prozentuale Gesamtunterschied berechnet.

Wertebereich

Die Präzision des Wertebereichs wird anhand der Ergebnisse aus den Tabellen C.4, C.5 und C.6 gemessen. Dabei werden die Wertebereiche der Ergebnisse mit den Wertebereichen des Goldstandards verglichen. Stimmen die Knotentypen überein, wird das Konzept der Klasse R_W zugeteilt. Stimmen die Knotentypen nicht überein, wird das Konzept der Klasse F_W zugeteilt. Wird kein semantisch gleiches Konzept gefunden, wird das Konzept ignoriert und in den folgenden Tabellen mit „-“ markiert.

$$P_{W,Flug} = \frac{2}{2+6} = 0.25 \tag{6.5}$$

$$P_{W,Bahn} = \frac{6}{6+6} = 0.5 \tag{6.6}$$

$$P_{W,Hotel} = \frac{2}{2+5} = 0.29 \tag{6.7}$$

Knotentyp

Die Präzision des Knotentyps wird anhand der Ergebnisse aus den Tabellen 6.5, **??** und 6.7 des Abschnitts 6.2 gemessen. Dabei werden die Knotentypen der Ergebnisse mit den Knotentypen des Goldstandards verglichen. Stimmen die Knotentypen überein, wird das Konzept der Klasse R_K zugeteilt. Stimmen die Knotentypen nicht übererin, wird das Konzept der Klasse F_K zugeteilt. Wird kein semantisch gleiches Konzept gefunden, wird das Konzept ignoriert und in den folgenden Tabellen mit „-“ markiert.

$$P_{K,Flug} = \frac{6}{6+2} = 0.75 \tag{6.8}$$

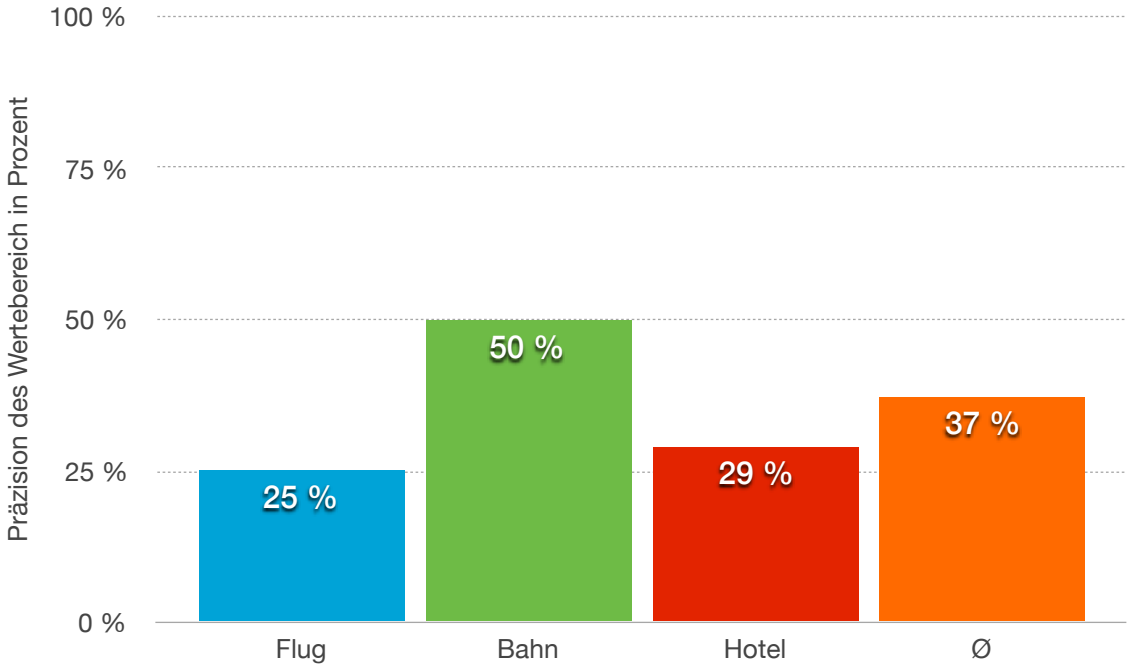

Abbildung 6.1.: Präzision des Wertebereichs in Prozent der generierten aktiven Ontologien im Vergleich zum Goldstandard

$$P_{K,Bahn} = \frac{6}{6+6} = 0.5 \tag{6.9}$$

$$P_{K,Hotel} = \frac{5}{5+3} = 0.625 \tag{6.10}$$

Gesamt

In diesem Abschnitt wird die resultierende Präzision berechnet. Diese setzt sich aus der Präzsion des Wertebereichs und der Präzision des Knotentyps zusammen. Dabei wird der Mittelwert der Präzision des Wertebereichs und der Präzision des Knotentyps berechnet. Dieser Mittelwert stellt die resultierende Präzision einer Kategorie dar.

$$P_{G,Flug} = \frac{0.25+0.75}{2} = 0.5 \tag{6.11}$$

$$P_{G,Bahn} = \frac{0.5+0.5}{2} = 0.5 \tag{6.12}$$

$$P_{G,Hotel} = \frac{0.25+0.625}{2} = 0.438 \tag{6.13}$$

6.4.2. Automatisierungsgrad

In diesem Abschnitt wird der Automatisierungsgrad G_A der generierten aktiven Ontologien nach der Metrik aus dem Abschnitt 6.3 berechnet. Dieser beläuft sich auf den automatisiert erzeugten Konzepten aus den Tabellen 6.5, 6.6 und 6.7. Dabei werden die automatisch

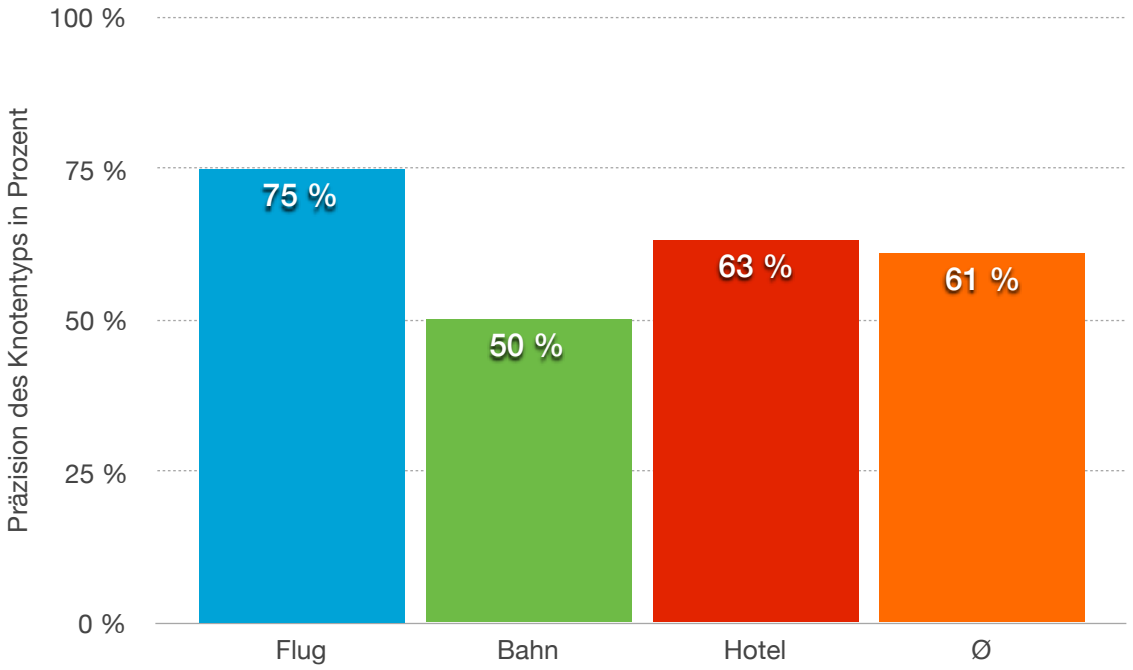

Abbildung 6.2.: Präzision des Knotentyps in Prozent der generierten aktiven Ontologien im Vergleich zum Goldstandard

erzeugten Konzepte der Klasse A, sowie die manuell erzeugten Konzepte der Klasse NA zugewiesen.

$$G_{A,Flug} = \frac{10}{10+3} = 0.77 \tag{6.14}$$

$$G_{A,Bahn} = \frac{9}{9+3} = 0.75 \tag{6.15}$$

$$G_{A,Hotel} = \frac{5}{5+4} = 0.66 \tag{6.16}$$

Die Ergebnisse sind in Tabelle 6.14 zusammengefasst.

6.5. Diskussion

Für die Bewertung des AO-Generators werden die Ergebnisse der generierten aktiven Ontologien betrachtet und über die definierten Metriken aus Abschnitt 6.3 mit dem Goldstandard verglichen.

Betrachtet man die Präzision des Wertebereichs, welche in Abbildung 6.1 dargestellt ist, lässt sich erkennen, dass diese mit durchschnittlich 37% eher gering ausfällt. Dazu kommt, dass alle Konzepte, die nicht automatisiert erstellen werden konnten, mit dem Wertebereich des Goldstandards angereichert wurden. Die daraus gering erscheinende Präzision lässt sich erklären. Zum einen wurde der Wertebereich als falsch eingestuft, sobald er nur eine Teilmenge von dem des Goldstandards bildet. Die Werte wurden außerdem ohne einen Wortabstand betrachtet. Unterscheidet sich ein Wert von dem des Goldstandards, wurde der Wertebereich als falsch eingestuft. Der Wertebereich des Goldstandards wurde zudem

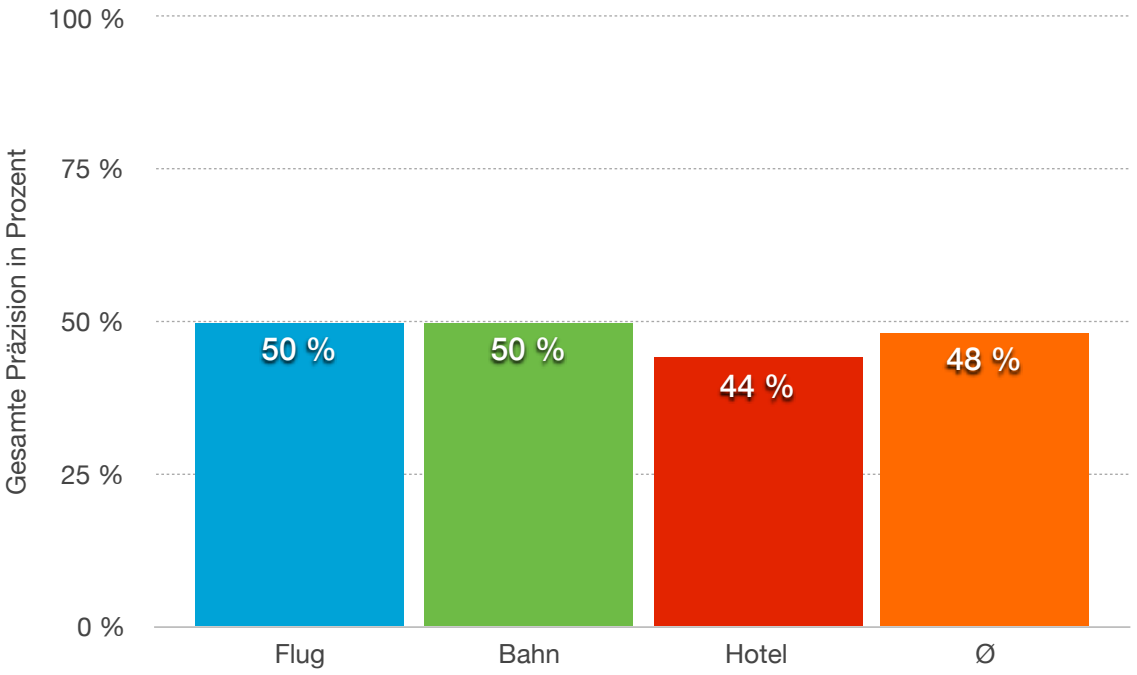

Abbildung 6.3.: Präzision in Prozent der generierten aktiven Ontologien im Vergleich zum Goldstandard

nicht immer über die Informationen der Webformularelemente gebildet, sondern direkt über den Benutzer. Diese Faktoren machen es schwierig über die Präzision des Wertebereichs endgültig zu urteilen.

Die Präzision des Knotentyps aus Abbildung 6.2 weist mit durchschnittlich über 60% einen annehmbaren auf. Außerdem hat sich seit der Erstellung des Goldstandards herausgestellt, dass für einige Konzepte durchaus andere Knotentypen geeignet oder sogar geeigneter sind. Bei einem manuell zu definierenden Konzept wurde allerdings neben dem Wertebereich auch der Knotentyp dem Goldstandard angepasst. Aufgrund des hohen Automatisierungsgrads, wie in Abbildung 6.4 illustriert, fallen die manuell erstellten Konzepte wenig in die Gewichtung.

Der Automatisierungsgrad ist mit durchschnittlich über 70% beachtlich. Dieser Automatisierungsgrad lässt sich damit erklären, dass die Webformulare der Kategorien „Flug“, „Bahn“ und „Hotel“ aus weniger manuell abzubildenden Webformularelementen bestehen, als aus benutzerfreundlichen Auswahlmenüs, Auswahlboxen oder Datumswählern. Diese Webformularelemente lassen sich automatisiert abbilden.

Zusammenfassend betrachtet, ist eine Gesamtpräzision von 48% (Abb. 6.3) in Kombination mit einem Automatisierungsgrad von 71% eine beachtliche Leistung. Somit lassen sich im Schnitt sieben von zehn Konzepten automatisiert erstellen. Diese Konzepte müssen nachträglich vom Benutzer nur leicht geändert und ergänzt werden, um mit dem Goldstandard übereinzustimmen. Dies ist bei der zeit- und arbeitsaufwändigen manuellen Erstellung einer aktiven Ontologie ein erheblicher Erfolg.

6.6. Zusammenfassung

Die Evaluation des AO-Generators wurde mithilfe von Metriken zum Vergleich der Ergebnisse mit dem Goldstandard durchgeführt. Der Goldstandard stammt aus der Arbeit „Abbildung von Webformularen auf aktive Ontologien“ [Sai16] von W. Said, in welcher

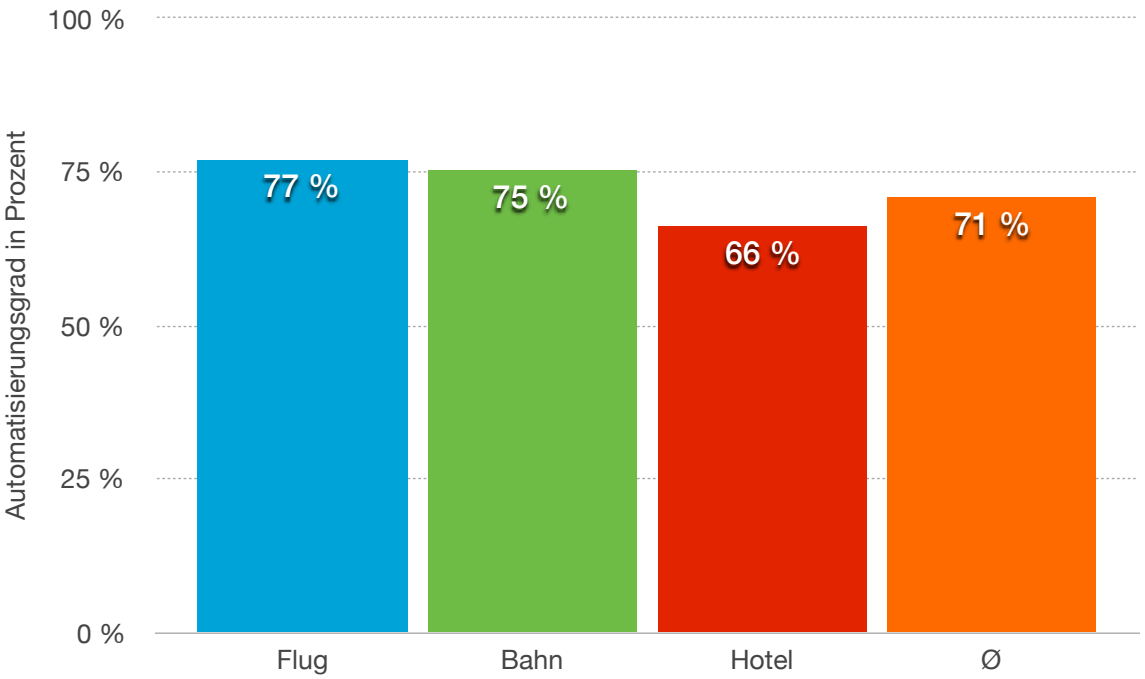

Abbildung 6.4.: Automatisierungsgrad in Prozent der generierten aktiven Ontologien

für drei Kategorien manuell eine aktive Ontologie erzeugt wurde. Die Eingabedatei für den AO-Generator wurde manuell aus denselben Webformularen erstellt, die auch dem Goldstandard die Grundlage bilden.

Dabei hat die Evaluation gezeigt, dass die Wertebereiche und Knotentypen der generierten Konzepte nur bedingt mit dem Goldstandard übereinstimmen. Diese Übereinstimmung ist mit etwa 50% allerdings eine nicht zu vernachlässigende Zahl. Im Gegensatz dazu konnten über 70% der Konzepte automatisiert generiert werden, was ein enormer Fortschritt zu der manuellen Erzeugung darstellt. Wie die Qualität des AO-Generators möglicherweise verbessert werden kann, wird im nachfolgenden Kapitel behandelt.

Konzeptname	Übereinstimmung des Wertebereichs
from	✓
to	✓
departing	×
returning	×
adults	×
children	×
infants	×
class	×
promoCode	-
teens	-
flexible	-
airline	-

Tabelle 6.8.: Vergleich der Wertebereiche zur aktiven Ontologie „Flug“

Konzeptname	Übereinstimmung des Wertebereichs
from	✓
to	✓
date	×
timme	×
departureArrival	×
meansOfTransport	×
accessibility	✓
bicycle	✓
bahnCard	×
adults	×
class	✓
viaStation	✓

Tabelle 6.9.: Vergleich der Wertebereiche zur aktiven Ontologie „Bahn“

Konzeptname	Übereinstimmung des Wertebereichs
arrival	×
departure	×
destination	✓
hotel	-
adults	×
children	×
rooms	×
promoCode	✓
groupCode	-

Tabelle 6.10.: Vergleich der Wertebereiche zur aktiven Ontologie „Hotel“

Konzeptname	Knotentyp	Knotentyp des Goldstandards	Übereinstimmung
from	PRE_POS	PRE_POS	✓
to	PRE_POS	PRE_POS	✓
departing	REN	PRE_POS	×
returning	REN	PRE_POS	×
adults	PRE_POS	PRE_POS	✓
children	PRE_POS	PRE_POS	✓
infants	PRE_POS	PRE_POS	✓
class	PRE_POS	PRE_POS	✓
promoCode	PRE_POS	-	-
teens	PRE_POS	-	-
flexible	VLN	-	-
airline	PRE_POS	-	-

Tabelle 6.11.: Vergleich der Knotentypen zur aktiven Ontologie „Flug“

Konzeptname	Knotentyp	Knotentyp des Goldstandards	Übereinstimmung
from	PRE_POS	PRE_POS	✓
to	PRE_POS	PRE_POS	✓
date	REN	PRE_POS	×
time	REN	PRE_POS	×
departureArrival	VLN	VLN	✓
meansOfTransport	PRE	VLN	×
accessibility	PRE	VLN	×
bicycle	PRE	VLN	×
bahnCard	PRE_POS	VLN	×
adults	PRE_POS	PRE_POS	✓
class	PRE_POS	PRE_POS	✓
viaStation	PRE_POS	PRE_POS	✓

Tabelle 6.12.: Vergleich der Knotentypen zur aktiven Ontologie „Bahn“

Konzeptname	Knotentyp	Knotentyp des Goldstandards	Übereinstimmung
arrival	REN	PRE_POS	×
departure	REN	PRE_POS	×
destination	PRE_POS	PRE_POS	✓
hotel	PRE_POS	-	×
adults	PRE_POS	PRE_POS	✓
children	PRE_POS	PRE_POS	✓
rooms	PRE_POS	PRE_POS	✓
promoCode	VLN	-	✓
groupCode	PRE_POS	-	-

Tabelle 6.13.: Vergleich der Knotentypen zur aktiven Ontologie „Hotel“

Kategorie / Ergebnisse	**Flug**	**Bahn**	**Hotel**	**Ø**
Automatisiert (A)	10	9	5	24
Nicht automatisiert (NA)	3	3	4	10
Anzahl Konzepte ($A + NA$)	13	12	9	34
Automatisierungsgrad (G_A)	0.77	0.75	0.55	0.706

Tabelle 6.14.: Automatisierungsgrad der erzeugten aktiven Ontologien „Flug“, „Bahn“ und „Hotel“

7. Zusammenfassung und Ausblick

Für jeden Gegenstandsbereich wird eine aktive Ontologie benötigt. Das manuelle Erstellen von aktiven Ontologien ist ein zeit- und arbeitsaufwändiger Prozess. Deshalb war die Zielsetzung der vorliegenden Arbeit die Erzeugung von aktiven Ontologien aus Webformularen weitestgehend zu automatisieren. Diese Arbeit erfolgte im Rahmen von EASIER des Lehrstuhls IPD Tichy.

7.1. Zusammenfassung

Die vorliegende Arbeit behandelte das Vorgehen, um aus den Webformularen einer Domäne semi-automatisch aktive Ontologien zu erzeugen. Dazu wurde ein Werkzeug, der AO-Generator, entwickelt, welches aus einem übergeordneten Webformular einer Domäne semi-automatisch eine aktive Ontologie generiert.

Dabei wurde zuerst analysiert wie sich Webformulare einer Domäne so zu einem übergeordneten Webformulare zusammenführen lassen, dass sich aus dem übergeordneten Webformulare die ursprünglich Webformulare wieder zurück abgebildet werden können. Für diesen Vorgang musste unter anderem untersucht werden wie sich die Wertebereiche von Webformularelementen zusammensetzen und wie man diese zusammenführen kann. Dafür wurden Regeln definiert, welche vorgeben wie Wertebereiche verschiedener Datentypen der Webformularelemente zusammengeführt werden müssen. Außerdem musste für die Rückabbildung auf die ursprünglichen Webformulare das Problem behandelt werden, welche Informationen dazu nötig sind. Aus den Erkenntnissen dieser Untersuchungen erschloss sich ein Konstruktionsplan für das übergeordnete Webformular. Um die übergeordneten Webformulare einheitlich darstellen zu können, wurde ein XML Schema entworfen auf welches die übergeordneten Webformulare abgebildet werden können. Die daraus resultierenden XML Dokumente dienen dem entwickelten AO-Generator als Eingabe.

Für die Verarbeitung des XML Dokuments wurden Ableitungsregeln definiert. Diese Ableitungsregeln geben vor wie aus den Elementen des XML Dokuments der Wertebereich berechnet wird und bestimmt einen Knotentyp. Mit diesen Informationen werden die Komponenten der aktiven Ontologie erzeugt. Der AO-Generator nutzt diese Ableitungsregeln. Kann ein Element automatisiert verarbeitet werden, wird dies getan. Kann ein Element nicht automatisiert abgebildet werden, wird der Benutzer nach einem Knotentyp und Wertebereich befragt. Dabei wird dem Benutzer ein Vorschlag für einen möglichen Wertebereich und Knotentyp präsentiert. Sobald alle Elemente des XML Dokuments verarbeitet

sind, generiert der AO-Generator Quelltextdateien im Java Format, welche zusammen nun von EASIER als aktive Ontologie verwendet werden kann.

7.2. Ausblick: Eingabedokument für den AO-Generator

Für das manuell erstellte Eingabedokument für den AO-Generator wird ein Werkzeug entwickelt, welches den erschlossenen Konstruktionsplan zum Zusammenführen von Webformularen einer Domäne umsetzt. Dabei erhält das Werkzeug die Webformulare einer Domäne und untersucht diese nach semantisch gleichen Webformularelementen. Die semantisch gleichen Webformularelemente werden nach dem Konstruktionsplan zusammengeführt und ein XML Dokument nach dem definierten Schema ausgegeben.

7.3. Ausblick: AO-Generator

Damit der AO-Generator bessere Ergebnisse erzielen kann, existieren zwei Herangehensweisen. Zum einen könnten in Zukunft die Wertebereiche der Webformularelemente durch das Semantik Web innerhalb der Webformulare besser definiert werden. Damit könnte das Eingabedokument für den AO-Generator mit mehr Informationen zum Wertebereich angereichert werden. Die andere Herangehensweise wäre, schon existierendes Wissen wie das „WordNet“ und bestehende Ontologien einzubeziehen. Über diese zwei Wissensbestände könnten die Wertebereiche zu einer bestimmten Semantik gefunden werden und damit die Konzepte der aktiven Ontologie bescheren.

Literaturverzeichnis

[ABM05] An, Yuan ; Borgida, Alex ; Mylopoulos, John: Constructing complex semantic mappings between XML data and ontologies. In: *International Semantic Web Conference* Springer, 2005, S. 6–20 (zitiert auf den Seiten 21 und 24).

[ABM08] An, Yuan ; Borgida, Alex ; Mylopoulos, John: Discovering and maintaining semantic mappings between XML schemas and ontologies. In: *Journal of computing Science and Engineering* 2 (2008), Nr. 1, S. 44–73 (zitiert auf den Seiten 21 und 24).

[AGWC07a] An, Yoo J. ; Geller, James ; Wu, Yi-Ta ; Chun, Soon: Semantic deep web: automatic attribute extraction from the deep web data sources. In: *Proceedings of the 2007 ACM symposium on Applied computing* ACM, 2007, S. 1667–1672 (zitiert auf den Seiten xi, 13, 14, 15, 24 und 35).

[AGWC07b] An, Yoo J. ; Geller, James ; Wu, Yi-Ta ; Chun, Soon A.: Automatic generation of ontology from the deep web. In: *18th International Workshop on Database and Expert Systems Applications (DEXA 2007)* IEEE, 2007, S. 470–474 (zitiert auf den Seiten 14 und 35).

[AM05] An, Yuan ; Mylopoulos, John: Translating XML web data into ontologies. In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* Springer, 2005, S. 967–976 (zitiert auf den Seiten 21 und 24).

[BA05] Bohring, Hannes ; Auer, Sören: Mapping XML to OWL Ontologies. In: *Leipziger Informatik-Tage* 72 (2005), S. 147–156 (zitiert auf den Seiten 21, 23, 31 und 35).

[Ble16] Blersch, Landhäußer: Easier: An Approach to Automatically Generate Active Ontologies for Intelligent Assistants. In: *International Symposium on Information Systems and Software Engineering: ISSE 2016*, 2016 (zitiert auf Seite 1).

[BMRB07] Benslimane, Sidi M. ; Malki, Mimoun ; Rahmouni, Mustapha K. ; Benslimane, Djamal: Extracting personalised ontology from data-intensive web application: an HTML forms-based reverse engineering approach. In: *Informatica* 18 (2007), Nr. 4, S. 511–534 (zitiert auf den Seiten xi, 19, 23, 31 und 35).

[DKYL09] Dragut, Eduard C. ; Kabisch, Thomas ; Yu, Clement ; Leser, Ulf: A hierarchical approach to model web query interfaces for web source integration. In: *Proceedings of the VLDB Endowment* 2 (2009), Nr. 1, S. 325–336 (zitiert auf Seite 15).

[DVN03] Davulcu, Hasan ; Vadrevu, Srinivas ; Nagarajan, Saravanakumar: OntoMiner: bootstrapping and populating ontologies from domain specific web

sites. In: *Proceedings of the First International Conference on Semantic Web and Databases* CEUR-WS. org, 2003, S. 245–262 (zitiert auf Seite 20).

[G+93] Gruber, Thomas R. u. a.: A translation approach to portable ontology specifications. In: *Knowledge acquisition* 5 (1993), Nr. 2, S. 199–220 (zitiert auf Seite 9).

[GBC06] Guzzoni, Didier ; Baur, Charles ; Cheyer, Adam: Active: A unified platform for building intelligent web interaction assistants. In: *Web Intelligence and Intelligent Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on* IEEE, 2006, S. 417–420 (zitiert auf den Seiten xi, 9 und 10).

[GMJ04] Gal, Avigdor ; Modica, Giovanni ; Jamil, Hasan: Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources. In: *Data Engineering, 2004. Proceedings. 20th International Conference on* IEEE, 2004, S. 853 (zitiert auf Seite 19).

[Guz08] Guzzoni, Didier: *Active: a unified platform for building intelligent applications*, École Polytechnique Fédérale De Lausanne, PhD Thesis, Januar 2008 (zitiert auf Seite 22).

[HML+07] He, Hai ; Meng, Weiyi ; Lu, Yiyao ; Yu, Clement ; Wu, Zonghuan: Towards deeper understanding of the search interfaces of the deep web. In: *World Wide Web* 10 (2007), Nr. 2, S. 133–155 (zitiert auf den Seiten 16, 24, 27 und 28).

[HMYW04] He, Hai ; Meng, Weiyi ; Yu, Clement ; Wu, Zonghuan: Automatic integration of Web search interfaces with WISE-Integrator. In: *The VLDB Journal* 13 (2004), Nr. 3, S. 256–273 (zitiert auf den Seiten 16, 24, 27 und 28).

[HMYW05] He, Hai ; Meng, Weiyi ; Yu, Clement ; Wu, Zonghuan: Wise-integrator: A system for extracting and integrating complex web search interfaces of the deep web. In: *Proceedings of the 31st international conference on Very large data bases* VLDB Endowment, 2005, S. 1314–1317 (zitiert auf den Seiten xi, 16, 24, 27 und 28).

[May17] Mayer, Thomas: *Extraktion und Konsolidierung von Webformularen zur Erzeugung von aktiven Ontologien*, KIT, Lehrstuhl IPD Tichy, Bachelorarbeit, 2017 (zitiert auf den Seiten 2, 26 und 27).

[Mor13] Mortensen, Jonathan M.: Crowdsourcing ontology verification. In: *International Semantic Web Conference* Springer, 2013, S. 448–455 (zitiert auf Seite 18).

[NM03] Noy, Natalya F. ; Musen, Mark A.: The PROMPT suite: interactive tools for ontology merging and mapping. In: *International Journal of Human-Computer Studies* 59 (2003), Nr. 6, S. 983–1024 (zitiert auf den Seiten xi, 19, 20, 24 und 35).

[Sai16] Said, Wasim: *Abbildung von Webformularen auf aktive Ontologien*, KIT, Lehrstuhl IPD Tichy, Masterarbeit, 2016 (zitiert auf den Seiten xiii, 4, 63, 64, 65, 71, 86 und 88).

[SCP15] Singapogu, Samuel S. ; Costa, Paulo C. ; Pullen, J M.: Automated Ontology Creation using XML Schema Elements. (2015) (zitiert auf den Seiten 21, 23, 31 und 35).

[WDYM05] Wu, Wensheng ; Doan, AnHai ; Yu, Clement ; Meng, Weiyi: Bootstrapping domain ontology for semantic web services from source web sites. In: *International Workshop on Technologies for E-Services* Springer, 2005, S. 11–22 (zitiert auf den Seiten xi, 20 und 21).

[WYDM04] Wu, Wensheng ; Yu, Clement ; Doan, AnHai ; Meng, Weiyi: An interactive clustering-based approach to integrating source query interfaces on the deep web. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* ACM, 2004, S. 95–106 (zitiert auf Seite 15).

Anhang

A. XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="categories">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="category" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="providers">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="provider" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="name" type="xs:string"></xs:element>
                          <xs:element name="url" type="xs:string"></xs:element>
                          <xs:element name="formIdentAttribute" type="xs:string"></xs:element>
                          <xs:element name="formIdentValue" type="xs:string"></xs:element>
                          <xs:element name="formsteps" type="xs:int"></xs:element>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:int"></xs:attribute>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="formelements">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="formelement" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="hypernym" type="xs:string"></xs:element>
                          <xs:element name="attributes">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="dataType" type="xs:string"></xs:element>
                                <xs:element name="type" type="xs:string"></xs:element>
                                <xs:element name="placeholder" type="xs:string"></xs:element>
                                <xs:element name="possiblePrefixes">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="possiblePrefix"
                                        maxOccurs="unbounded" type="xs:string"></xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                                <xs:element name="values">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="value" maxOccurs="unbounded">
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element name="attribute" type="xs:string"></xs:element>
                                            <xs:element name="mappingId" type="xs:int"></xs:element>
                                            <xs:element name="label" type="xs:string"></xs:element>
                                            <xs:element name="selected" type="xs:string"></xs:element>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
```

```
                    </xs:element>
                    <xs:element name="possiblePostfixes">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="possiblePostfix"
                            maxOccurs="unbounded" type="xs:string"></xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="name" type="xs:string"></xs:element>
                    <xs:element name="min" type="xs:int"></xs:element>
                    <xs:element name="max" type="xs:int"></xs:element>
                    <xs:element name="step" type="xs:int"></xs:element>
                    <xs:element name="maxlength" type="xs:int"></xs:element>
                    <xs:element name="pattern"></xs:element>
                    <xs:element name="required" type="xs:boolean"></xs:element>
                    <xs:element name="multiple" type="xs:boolean"></xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="reverseMappings">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="reverseMapping" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="mappings">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="mapping" maxOccurs="unbounded">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="page" type="xs:int"></xs:element>
                                      <xs:element name="identifier">
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element name="identAttribute"
                                              type="xs:string"></xs:element>
                                            <xs:element name="identValue"
                                              type="xs:string"></xs:element>
                                            <xs:element name="identType" type="xs:string"></xs:element>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
                                      <xs:element name="attributes">
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element name="dataType" type="xs:string"></xs:element>
                                            <xs:element name="type" type="xs:string"></xs:element>
                                            <xs:element name="placeholder"
                                              type="xs:string"></xs:element>
                                            <xs:element name="values">
                                              <xs:complexType>
                                                <xs:sequence>
                                                  <xs:element name="value"
                                                    maxOccurs="unbounded">
                                                    <xs:complexType>
                                                      <xs:sequence>
                                                        <xs:element name="attribute" type="xs:string"></xs:
                                                            element>
                                                        <xs:element name="mappingId" type="xs:int"></xs:element
                                                            >
                                                        <xs:element name="label" type="xs:string"></xs:element>
                                                        <xs:element name="selected" type="xs:boolean"></xs:
                                                            element>
                                                      </xs:sequence>
                                                    </xs:complexType>
                                                  </xs:element>
                                                </xs:sequence>
                                              </xs:complexType>
                                            </xs:element>
                                            <xs:element name="name" type="xs:string"></xs:element>
                                            <xs:element name="min" type="xs:int"></xs:element>
                                            <xs:element name="max" type="xs:int"></xs:element>
                                            <xs:element name="step" type="xs:int"></xs:element>
                                            <xs:element name="maxlength"
                                              type="xs:int"></xs:element>
                                            <xs:element name="pattern" type="xs:string"></xs:element>
                                            <xs:element name="required" type="xs:boolean"></xs:element>
                                            <xs:element name="multiple" type="xs:boolean"></xs:element>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
```

```
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                            <xs:attribute name="mappingType" type="xs:string"></xs:attribute>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="id" type="xs:int"></xs:attribute>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="name" type="xs:string"></xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Quelltextausschnitt 7.1: XML Schema für die übergeordneten Webformulare.

B. Goldstandard

Konzept von „Flight“	**Semantik (Ausdrücke)**
Places	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof,Brandenburger tor
Departure_Prefix	from, departure is
Departure_Postfix	is the departure, is departure
Arrival_Prefix	to, towards, arrival is
Arrival_Postfix	is the arrival, is arrival
Adult_Prefix	number of adults is, adults number is
Adult_Postfix	adult, adults, person, persons, people, passenger, passengers, is the number of adults, is the adults number
Children_Prefix	number of children is, children number is, number of kinder is, kinder number is
Children_Postfix	child, children, kind, kinder, is the number of children, is the children number, is the number of kinder, is the kinder number
Babies_Prefix	number of babies is, babies number is
Babies_Postfix	baby, babies, is the number of babies, is the babies number
Flight_Class_Options	economy, premium economy, business, first
Flight_Class_Prefix	flight class is
Flight_Class_Postfix	class, is the flight class
Departure_Date_Prefix	depart on, departure date is, departure is on
Departure_Date_Postfix	is the departure date
Return_Date_Prefix	return on, back on, return date is, return is on
Return_Date_Postfix	is the return date, is the return
Helper_Flight	flight, flights, fly, airport, airways, airline, plane

Tabelle B.1.: Semantik aller Konzepte der aktiven Ontologie „Flight“, Quelle: [Sai16]

Konzept von „Train“	**Semantik (Ausdrücke)**
Places	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Atlanta, Beijing, Dubai, Chicago, Tokyo, Denver, Madrid, Hong Kong, Delhi, Miami, Damascus, Aleppo, Latakia, Prag, Las Vegas, Beirut, Rome, Milan, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof, Brandenburger tor
Start_Prefix	from, start is, start station is
Start_Postfix	is the start, is start
Destination_Prefix	to, towards,destination is, destination station is
Destination_Postfix	is the destination, is destination

Means_Of_Transport	ice, only local transport, ec, ic, ir, regional, bus, suburban, boat, underground tram, dial a ride, taxi, all excluding ice
Accessibility	no stairs, no escalators, no elevators, low floor vehicles, vehicles with lift platform
Bicycle	bicycle, bike
BahnCard	bahn card
Reservation	reserve a seat, reservation a seat, seat reservation, seat, reservation
Passenger_Prefix	number of passengers is, passenger number is
Passenger_Postfix	passenger, passengers, is the number of passengers, is the passengers number
Travel_Class_Options	first, second
Travel_Class_Prefix	travel class is, class is
Travel_Class_Postfix	class, is the travel class
Journey_Date_Prefix	go on, travel on, ravel date is, date of journey is,date of trip is, leave on, depart on, departure date is,departure is on
Journey_Date_Postfix	is the travel date, is the trip date,is the journey date
Journey_Return_Date_Prefix	return on, back on, return date is, return, is, on
Journey_Return_Date_Postfix	is the return date, is the return
Journey_Time_Prefix	go at, travel at, travel time is, time of journey is, time of trip is, go on, travel on, leave on, travel time is, time of journey is, time of trip is, depart on, departure date is, departure is on
Journey_Time_Postfix	is the travel time, is the trip time, is the journey time
Journey_Return_Time_Prefix	return at, back at, return time is, return is at, return on, back on, return date is, return is on
Journey_Return_Time_Postfix	is the return time, is the back time
Departure_OR_Arrival	at the departure,at the start, at the arrival, at the destination, by the departure, by the arrival, by leaving, by departure, by departuring, by arriving, by arrival, departing before, departing by, leaving before, leaving by, arriving before, arriving by
Walking_Time_Prefix	walking time is, walking time
Walking_Time_Postfix	is the walking time, minutes is the walking time, min is the walking time
Interchange_Time_Prefix	interchange time is, interchange time
Interchange_Time_Postfix	is the interchange time, minutes is the interchange time, min is the interchangetime
Via_Station_Prefix	stopover at, via station, via
Via_Station_Postfix	is the stopover station
Helper_Train	train, trains, ice, bus

Tabelle B.2.: Semantik aller Konzepte der aktiven Ontologie „Train“, Quelle: [Sai16]

Konzept von „Hotel“	**Semantik (Ausdrücke)**
Hotel_Place	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof,Brandenburger tor
Arrival_Date_Prefix	arrive on, from, arrival date is, arrive is on
Arrival_Date_Postfix	is the arrival date, is the arrival
Departure_Date_Prefix	depart on, departure date is, departure is on
Departure_Date_Postfix	is the departure date, is the departure
Adult_Prefix	number of adults is, adults number is
Adult_Postfix	adult, adults, person, persons,people, passenger, passengers, is the number of adults, is the adults number
Children_Prefix	number of children is, children number is, number of kinder is, kinder number is
Children_Postfix	child, children, kind, kinder, is the number of children, is the children number, is the number of kinder, is the kinder number
Room_Prefix	number of rooms is
Room_Postfix	room, rooms, is the number of rooms
Hotel_Duration	day, days, night, nights
Hotel_Promotion_Code	promotion code is, is the promotion code
Helper_Hotel	hotel, room, rooms, suite, suites

Tabelle B.3.: Semantik aller Konzepte der aktiven Ontologie „Hotel“, Quelle: [Sai16]

C. Ergebnisse des AO-Generators

Konzeptnamen von „Train“	**Wertebereich**
From	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Atlanta, Beijing, Dubai, Chicago, Tokyo, Denver, Madrid, Hong Kong, Delhi, Miami, Damascus, Aleppo, Latakia, Prag, Las Vegas, Beirut, Rome, Milan, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof, Brandenburger tor
From_Prefix	from, start is, start station is
From_Postfix	is the start, is start
To	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Atlanta, Beijing, Dubai, Chicago, Tokyo, Denver, Madrid, Hong Kong, Delhi, Miami, Damascus, Aleppo, Latakia, Prag, Las Vegas, Beirut, Rome, Milan, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof, Brandenburger tor
To_Prefix	to, towards,destination is, destination station is
To_Postfix	is the destination, is destination
Date	Regex für Datumsangaben
Date_Prefix	date
Time	Regex für Zeitangaben
Time_Prefix	time
DepartureArrival	departure, arrival
MeansOfTransport	regional trains, tram, suburban, subway, express bus, metro and city buses, regional buses, taxi on demand, prefer boat
MeansOfTransport_Prefix	with, without
Accessibility	no stairs, no escalators, no elevators, low floor vehicles, vehicles with lift platform
Accessibility_Prefix	with, without
Bicycle	bicycle, bike
Bicycle_Prefix	with, without
BahnCard	Bahncard 25 2. Klasse, Bahncard 25 1. Klasse, Bahncard 50 2. Klasse, Bahncard 50 1. Klasse
BahnCard_Prefix	bahncard
BahnCard_Postfix	bahncard
Adults	0 adults, 1 adult, 2 adults, 3 adults, 4 adults, 5 adults, 6 adults, 7 adults, 8 adults, 9 adults
Adults_Prefix	adult, adults, Number of Adults
Adults_Postfix	adult, adults, Number of Adults
Class	first, second
Class_Prefix	class
Class_Postfix	class

ViaStation	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Atlanta, Beijing, Dubai, Chicago, Tokyo, Denver, Madrid, Hong Kong, Delhi, Miami, Damascus, Aleppo, Latakia, Prag, Las Vegas, Beirut, Rome, Milan, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof, Brandenburger tor
ViaStation_Prefix	stopover at, via station, via
ViaStation_Postfix	is the stopover station
Helper_Train	train

Tabelle C.5.: Wertebereiche aller generierten Konzepte des AO-Generators der aktiven Ontologie „Train“

Konzeptnamen von „Flight“	**Wertebereich**
From	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof,Brandenburger tor
From_Prefix	from, departure is
From_Postfix	is the departure, is departure
To	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof,Brandenburger tor
To_Prefix	to, towards, arrival is
To_Postfix	is the arrival, is arrival
Departing	Regex für Datumsangaben
Departing_Prefix	departure date, depart date, departing, departing on, leaving, fly out
Returning	Regex für Datumsangaben
Returning_Prefix	return on, back on, return date is, return is on
Adults	0 adults, 1 adult, 2 adults, 3 adults, 4 adults, 5 adults, 6 adults, 7 adults, 8 adults, 9 adults
Adults_Prefix	adult, adults, Number of Adults
Adults_Postfix	adult, adults, Number of Adults
Children	0 children, 1 child, 2 children, 3 children, 4 children, 5 children, 6 children, 7 children, 8 children
Children_Prefix	child, children, travel with children, Number of Children
Children_Postfix	child, children, travel with children, Number of Children
Infants	0 infants, 1 infants, 2 infants, 3 infants, 4 infants, 5 infants, 6 infants, 7 infants, 8 infants
Infants_Prefix	infant, infants, Number of Babies
Infants_Postfix	infant, infants, Number of Babies
Class	economy, premium economy, business, first
Class_Prefix	class
Class_Postfix	class
TripType	Return,One Way
PromoCode	-
PromoCode_Prefix	promo code, access code
PromoCode_Postfix	promo code, access code
Teens	0 teens, 1 teen, 2 teens, 3 teens, 4 teens, 5 teens, 6 teens, 7 teens, 8 teens
Teens_Prefix	teen, teens, Number of teens
Teens_Postfix	teen, teens, Number of teens
Felxible	flexbile, flexible dates, lowest price
Airline	SWISS, Lufthansa
Airline_Prefix	airline
Helper_Flight	flight

Tabelle C.4.: Wertebereiche aller generierten Konzepte des AO-Generators der aktiven Ontologie „Flight“

Konzeptnamen von „Hotel“	**Wertebereich**
Arrival	Regex für Datumsangaben
Arrival_Prefix	arrival, arr, arrival date, check in date
Departure	Regex für Datumsangaben
Departure_Prefix	departure, dep, departure date, check out date
Destination	Frankfurt, Stuttgart, Berlin, Karlsruhe, New York, Manchester, Amsterdam, London, Los Angeles, Paris, Washington,Durlach, Karlsruhe hbf, Berlin hauptbahnhof,Brandenburger tor
Destination_Prefix	destination
Destination_Postfix	destination
Hotel	-
Hotel_Prefix	hotel
Hotel_Postfix	hotel
Adults	0 adults, 1 adult, 2 adults, 3 adults, 4 adults, 5 adults, 6 adults, 7 adults, 8 adults
Adults_Prefix	adults
Adults_Postfix	adults
Children	0 children, 1 child, 2 children, 3 children, 4 children, 5 children, 6 children, 7 children, 8 children
Children_Prefix	children, kids
Children_Postfix	children, kids
Rooms	0, 1, 2, 3, 4, 5, 6, 7, 8
Rooms_Prefix	rooms
Rooms_Postfix	rooms
PromoCode	-
PromoCode_Prefix	promo code, loyality or membership card number, promo, add promo/corporate code, special code
PromoCode_Postfix	promo code, loyality or membership card number, promo, add promo/corporate code, special code
PromoCode	-
PromoCode_Prefix	group code
PromoCode_Postfix	group code
Helper_Hotel	hotel

Tabelle C.6.: Wertebereiche aller generierten Konzepte des AO-Generators der aktiven Ontologie „Hotel“