



Semantische Datenbanken

Bachelor-Thesis

Theoretische Grundlagen; Aufbau und Nutzung einer semantischen Datenbank

Studiengang: Informatik

Autoren: Sven Osterwalder¹, Mira Günzburger²

Betreuer: Prof. Dr. Jürgen Eckerle³

Experte: Jean-Marie Leclerc

Datum: 14. Januar 2015

¹mira.guenzburger@students.bfh.ch

²sven.osterwalder@students.bfh.ch

³juergen.eckerle@bfh.ch

Versionen

Version	Datum	Status	Bemerkungen
0.1	19.09.2014	Entwurf	Initiale Erstellung des Dokumentes
0.2	05.11.2014	In Bearbeitung	Unterkapitel Reasoner
0.3	22.11.2014	In Bearbeitung	Vorgehen
0.4	05.12.2014	In Bearbeitung	Technologien
0.5	12.12.2014	In Bearbeitung	Vorgehen erweitert
0.6	13.12.2014	In Bearbeitung	Administration
0.7	16.12.2014	In Bearbeitung	Administration erweitert, Aufgabenstellung, Layoutanpassungen, Vorgehen, Lösung
0.8	17.12.2014	In Bearbeitung	Überarbeitung, Korrekturen
0.9	23.12.2014	In Bearbeitung	Lösung, Komponenten
0.10	26.12.2014	In Bearbeitung	Korrekturen
0.11	27.12.2014	In Bearbeitung	Fazit
0.12	30.12.2014	Vorgelegt	Anhänge, Formatierung, Gegenlesen, Quellenangaben
0.13	03.01.2015	Abgestimmt	Fertigstellung des Dokumentes
1.0	14.01.2015	Abgenommen	Feinschliff

Inhaltsverzeichnis

1. Einleitung	1
2. Administratives	2
2.1. Beteiligte Personen	2
2.2. Aufbau der Dokumentation	2
2.3. Ergebnisse (Deliverables)	2
3. Aufgabenstellung	4
3.1. Motivation	4
3.2. Ausgangslage	4
3.3. Ziele und Abgrenzung	4
4. Vorgehen	5
4.1. Arbeitsorganisation	5
4.2. Projektphasen	5
5. Lösung	15
5.1. Von der Wissenserarbeitung zum Tutorial	15
5.2. Modellierung	15
5.3. Erstellung von Abfragen	19
5.4. Benutzeroberfläche	19
6. Komponenten	20
6.1. Stanford Protégé	20
6.2. Clark & Parsia Stardog	21
6.3. Benutzeroberfläche	22
7. Fazit und Erweiterungsmöglichkeiten	25
7.1. Erweiterungsmöglichkeiten	26
Selbständigkeitserklärung	27
Glossar	28
Literaturverzeichnis	28
Abbildungsverzeichnis	30
Tabellenverzeichnis	31
Auflistungsverzeichnis	32
Anhang	34
A. Anforderungsdokument	35
B. Tutorial Wissensmodellierung	47
C. Beispiele	113
C.1. Allgemein	113
C.2. Wie ist Prolog aufgebaut?	114

C.3. Wie funktioniert Unifikation?	115
C.4. Was sind Atome?	118
C.5. Familienbeispiel	123
D. Modell	126
D.1. Grafiken	126
E. Installationhandbuch	132
E.1. Backend	132
E.2. Frontend	133
F. Arbeitsjournal	135

1. Einleitung

In der heutigen Informatik ist die relationale Datenspeicherung de facto Standard. Die Wissensabbildung erfolgt beispielsweise durch UML. Häufig geschieht dies in enger Verbindung mit der objektorientierten Programmierung. Experten aus einer Fachrichtung sind fähig, die Daten zu interpretieren und Schlüsse zu ziehen. Es ist damit aber nicht möglich Fragestellungen zu beantworten, welche über reine Relationsverknüpfungen hinausgehen. Mit dieser Technik sind Objekteigenschaften und -verhalten schwer abbildbar. Semantische Datenbanken sind eine andere Möglichkeit Wissen zu repräsentieren. Sie ermöglichen das Abbilden von Objekteigenschaften und können mit Hilfe von Schlussfolgerungen die Rolle eines Experten einnehmen.

In dieser Bachelor-Thesis soll eine semantische Datenbank aufgebaut und angewendet werden. Die Arbeit enthält zwei Teile: Einen theoretischen und einen praktischen. Der theoretische Teil (das Tutorial) zeigt auf, wie ein Knowledge-Engineer bei der Wissensmodellierung vorgehen kann. Um eine semantische Datenbank aufzubauen nutzt er als Basis Ontologien.

Im praktischen Teil wird eine entsprechende Ontologie aufgebaut und durch eine Benutzerschnittstelle zugänglich gemacht.

Im **theoretischen** Teil gelang es den Autoren, die Theorie der Wissensmodellierung übersichtlich aufzubereiten und wiederzugeben. Als Besonderheit werden fachliche Grundlagen mit praktischen Beispielen verbunden. Aus gemachten Erfahrungen konnten konkrete Hinweise eingeflochten werden. Dieses Dokument ist als Anhang beigefügt.

Im **praktischen** Teil der Arbeit wurde ein Expertensystem für Reisen entwickelt: "In der heutigen Zeit werden Urlaubaufenthalte/Pauschalreisen häufiger per Internet gebucht. Wenn jedoch ein Kunde spezielle Wünsche hat, so muss er sich immer noch in einem Reisebüro beraten lassen."

Für die Automatisierung dieses Prozesses werden Ontologien verwendet. Diese bilden mit Hilfe von Eigenschaften, Kriterien und Regeln die Problemdomäne ab. Mit Hilfe eines Reasoners können unterschiedliche Reiseanfragen beantwortet werden. Für eine individuelle Reiseplanung musste eine Ontologie erstellt werden.

Ohne klaren Rahmen würde diese zu komplex und zu gross. Daher haben die Autoren die Ontologie auf exemplarische Tages- und Wochenendausflüge in der Schweiz eingeschränkt.

Als **Ergebnis** präsentieren die Autoren eine semantischen Datenbank. In dieser wird die gewählte Problemdomäne in einem klar gesteckten Rahmen abgedeckt und die Vorteile der Wissensmodellierung werden veranschaulicht. Die Benutzerschnittstelle wurde durch einen Assistenten umgesetzt, der den Benutzer durch die (Reise-) Möglichkeiten führt ohne Kenntnisse der Ontologie und einer speziellen Abfragesprache zu erfordern.

2. Administratives

Einige administrative Aspekte der Bachelor-Thesis werden angesprochen, obwohl sie für das Verständnis der Resultate nicht notwendig sind.

Im gesamten Dokument wird nur die männliche Form verwendet, womit aber beide Geschlechter gemeint sind.

2.1. Beteiligte Personen

Autoren	Mira Günzburger ¹ Sven Osterwalder ²	
Betreuer	Prof. Dr. Jürgen Eckerle ³	<i>Begleitet Studenten bei der Bachelor-Thesis</i>
Experte	Jean-Marie Leclerc	<i>Zuständig für die Beurteilung der Arbeit</i>

2.2. Aufbau der Dokumentation

Der Aufbau der vorliegenden Arbeit ist wie folgt:

- Einleitung zur Bachelor-Thesis
- Beschreibung der Aufgabenstellung
- Vorgehen der Autoren im Hinblick auf die gestellten Aufgaben
- Lösung der gestellten Aufgaben
- Verwendete Technologien

Die Schwerpunkte dieser Arbeit liegen in:

- der Beschreibung der theoretischen Grundlagen (unter praktischen Aspekten) für ein Expertensystem
- der Entwicklung und Anwendung eines Expertensystems

2.3. Ergebnisse (Deliverables)

Die Bachelor-Thesis besteht aus mehreren Dokumenten. Vorgelegt wird das Abschlussdokument der Arbeit.

Nachfolgend sind die abzugebenden Objekte aufgeführt:

- **Abschlussdokument**
Das Abschlussdokument vereinigt sämtliche Ergebnisse und bietet eine Übersicht des gesamten Arbeitsprozesses
- **Tutorial**
Das Tutorial beinhaltet die theoretischen Grundlagen (unter praktischen Aspekten) für ein Expertensystem

¹mira.guenzburger@students.bfh.ch

²sven.osterwalder@students.bfh.ch

³juergen.eckerle@bfh.ch

- **Modellierung des Expertensystems**

in Form eines RDF/XML-Dokumentes und in Form einer grafischen Abbildung. Die grafische Abbildung ist eine abgewandelte Form eines semantischen Netzes. Der Einfachheit halber wird im gesamten Dokument jedoch der Ausdruck "semantisches Netz" bzw. "semantische Netze" verwendet.

- **Benutzeroberfläche**

Quellcode und Dokumentation der umgesetzten Benutzeroberfläche

- **Installationshandbuch**

Beschreibt Installation und Verwendung des entwickelten Expertensystems einschliesslich der Benutzeroberfläche

3. Aufgabenstellung

3.1. Motivation

Um Wissen aus dem Internet zu beziehen verwendet man hauptsächlich Suchmaschinen. Sie werden von vielen Personen bei der täglichen Arbeit verwendet.

Die meisten Suchmaschinen erfordern die Eingabe von Begrifflichkeiten im Suchfeld. Sie indexieren Inhalte mittels Stich- und Schlagworten. Der Suchende muss daher zumindest eine bestimmte Vorstellung von den erwarteten Suchergebnissen haben.

Fragt man nach Konzepten und Zusammenhängen, so stösst man bei den heutigen Suchmaschinen schnell an deren Grenzen. Sie verfügen über geringes bis kein semantisches Wissen, sie kennen keine Konzepte. Um dennoch die gewünschten Ergebnisse zu erhalten, muss der Anwender entsprechende Stichworte als Eingabe zu einer Suche liefern.

Zwar existieren Mechanismen zur Abbildung von Konzepten und Zusammenhängen. Ihre Nutzung ist aber komplex.

In der vorliegenden Arbeit wurde versucht eine bestimmte Suchmaschine zu entwickeln, welche Wissen über Konzepte und über Zusammenhänge hat. Sie soll eine intuitive, einfach zu bedienende Schnittstelle zwischen Mensch und System bieten.

3.2. Ausgangslage

Grundlagen für die Arbeit waren den Autoren teilweise bekannt. An der Berner Fachhochschule für angewandte Wissenschaften existiert kein Hauptfach für semantische Datenbanken bzw. Semantik.

Für den Erwerb von Vorkenntnissen besuchten die Autoren die Vorlesungen des Wahlpflichtfaches *Künstliche Intelligenz*. Eine vorgängige Projektarbeit beschreibt die Grundlagen für dieses Thema. Damit werden nur einzelne Aspekte dieser Bachelor-Thesis dargestellt.

Ziel dieser früheren Projektarbeit war der Aufbau einer semantischen Suche für Kinder. Ausgegangen wurde von der Annahme, dass die dazu erforderlichen Technologien bereits existieren. Die Annahme erwies sich als nicht korrekt. Nur gewisse theoretische Grundlagen, wie Ontologien oder Sprachen für deren Modellierung konnten übernommen werden.

3.3. Ziele und Abgrenzung

Diese Arbeit besteht aus zwei Teilen. Einerseits dem Tutorial. Dieses zeigt, wie ein Knowledge-Engineer eine Wissensdomäne systematisch modellieren und formalisieren kann. Im zweiten Teil wird eine Ontologie mit Anwendung für ein bestimmtes Problem erstellt. Teil zwei umfasst zusätzlich einen Prototypen einer möglichst benutzerfreundlichen Schnittstelle zur Abfrage der Ontologie.

Eine vollständige Ontologie der Wissensdomäne sollte nicht erstellt werden. Die Ontologie wurde nur soweit entwickelt, dass Abfragen möglich sind, die Mehrwert gegenüber herkömmlichen Suchmaschinen besitzen.

4. Vorgehen

4.1. Arbeitsorganisation

Im Vordergrund dieser Bachelor-Thesis stehen der Prozess der Entstehung eines Expertensystems sowie die Analyse der dazu benötigten Grundlagen. Der Fokus liegt dabei absichtlich nicht in der Softwareentwicklung. Traditionelle Methoden der Projektorganisation wie HERMES oder Scrum wurden daher nicht gewählt. Der Organisation wurden die im Voraus definierten Meilensteine (Teilziele) zu Grunde gelegt. Jeder dieser Meilensteine wurde zeitlich determiniert. Zugleich wurden die Tätigkeiten für jeden Meilenstein durch eine individuelle Liste verteilt. Der benötigte Zeitaufwand sowie die Tätigkeiten wurden durch ein gemeinsames Journal erfasst, siehe F.

Ein wichtiger Teil der Arbeit ist das Dokument, welches gewonnene Erkenntnisse festhält (Dokument Arbeitserkenntnisse). In diesem Dokument wurden Erfolge und Misserfolge beschrieben und analysiert. Es bildet schliesslich die Grundlage für das vorliegende Abschlussdokument (bei Anfrage jederzeit verfügbar).

4.1.1. Regelmässige Treffen

Regelmässige Besprechungen mit dem Betreuer der Arbeit halfen die gesteckten Ziele zu erreichen und Fehlentwicklungen zu vermeiden. Der Betreuer unterstützte die Autoren dabei mit Vorschlägen. Die Treffen fanden mindestens alle zwei Wochen statt, sie wurden in Form eines Protokolles festgehalten (bei Anfrage jederzeit verfügbar).

4.2. Projektphasen

4.2.1. Meilensteine

Um bei der Arbeit ein möglichst strukturiertes Vorgehen einzuhalten, wurden folgende Projektphasen gewählt:

- Projektstart
- Erarbeitung und Festhalten der Anforderungen
- Erarbeitung der formalen und technischen Grundlagen
- Modellierung der Ontologie
- Erstellung der Dokumentation zur Wissensmodellierung (Tutorial)
- Erstellung des Prototypen der Benutzerschnittstelle
- Erstellung der abschliessenden Dokumentation

Die gewonnen theoretischen Grundlagen wurden in das Tutorial eingebaut. Sie konnten direkt zur praktischen Umsetzung des Modelles verwendet werden. Bei der Modellierung gewonnene Erkenntnisse ihrerseits konnten direkt in das Dokument übernommen werden. Die drei Phasen liefen dabei parallel ab.

4.2.2. Zeitplan/Projektphasen

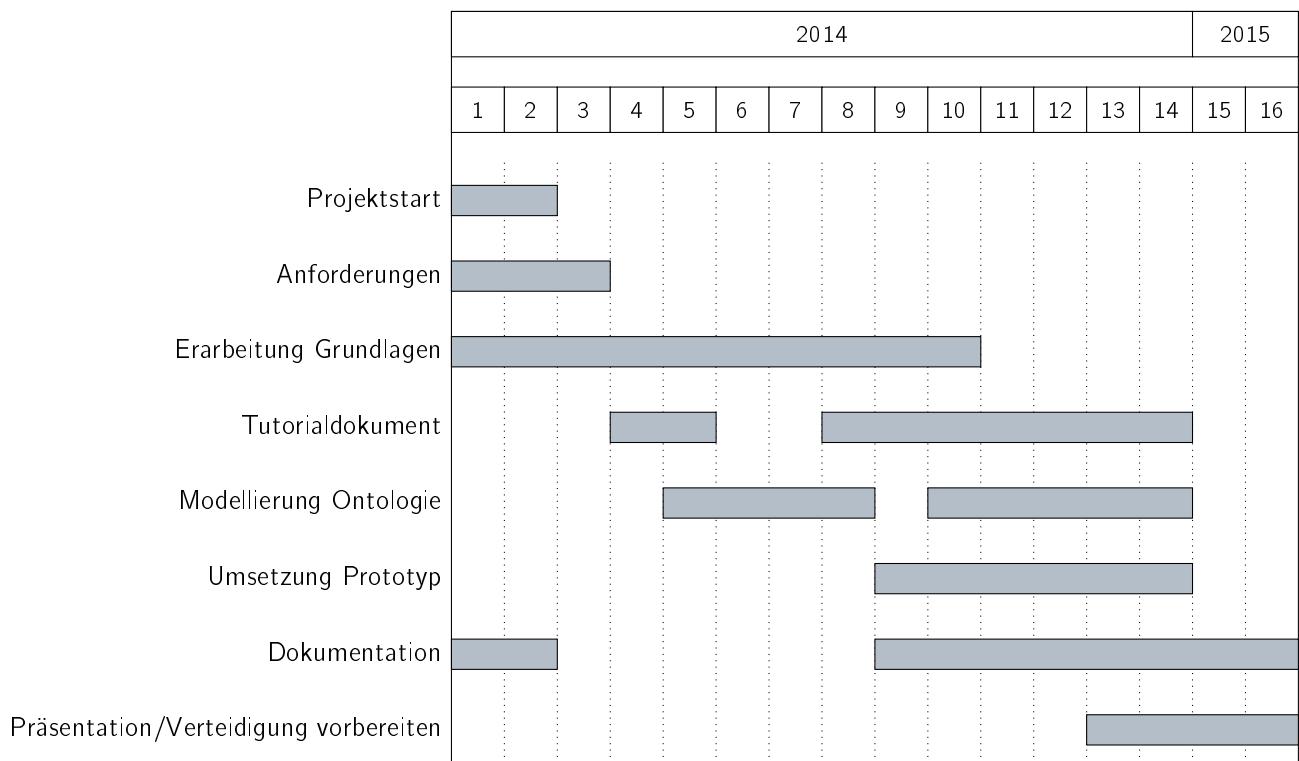


Abbildung 4.1.: Zeitplan; Der Titel stellt Jahreszahlen, der Untertitel Semesterwochen dar

4.2.3. Projektstart

In der ersten Phase wurden die Meilensteine der Arbeit identifiziert und skizziert. Um Details der Aufgabe zu verstehen, wurde das notwendige Vorwissen über semantische Datenbanken erarbeitet. Weiter wurde das Grundgerüst dieser Dokumentation erstellt.

4.2.4. Anforderungen

In dieser Phase wurde das Ziel dieser Bachelor-Thesis festgelegt. Vom Ziel ausgehend wurden die dazu erforderlichen Projektphasen festgelegt. Siehe Dokument in Anhang A.

4.2.5. Formale und technische Grundlagen

Im Vorprojekt zu dieser Bachelor-Thesis (BTI7302, Projekt 2; bei Anfrage jederzeit verfügbar) wurden die Grundlagen für die Erstellung und Modellierung von Ontologien teilweise erarbeitet. Erforderliche Technologien wurden während dieser Bachelor-Thesis vollständig erarbeitet.

Formale Grundlagen

Als formale Grundlagen dienen:

- Aufbau und Modellierung von Ontologien [Busse et al., 2014] sowie [Furrer, 2014]
- RDF-Syntax [Schreiber and Raimond, 2014] und [Beckett, 2013]
- Ontologie-Sprache OWL [Hitzler et al., 2012]

- Regel-Sprache SWRL [Horrocks et al., 2004]
- SPARQL-Abfragesprache [W3C SPARQL Working group, 2013], [group, 2013]

Das Konzept der technischen Umsetzung war durch die oben erwähnte Vorarbeit bereits gegeben.

Es besteht aus zwei Komponenten:

- einem Backend
in Form einer semantischen Datenbank, zur Verarbeitung von (Such-) Anfragen
- einem Frontend
zur einfachen Handhabung von Abfragen mit entsprechenden Ergebnissen für Anwender

Technische Grundlagen

Ursprünglich wurde für das Backend Apache Stanbol¹ gewählt. Während der Arbeit erwies sich, dass Apache Stanbol kein geeignetes Produkt für die geplante Umsetzung des Expertensystems ist.

Hingegen erlaubt die Verwendung der Graphdatenbank Stardog²:

- Ontologien in diversen Formaten zu importieren und exportieren
- Abfragen mittels der SPARQL-Abfragesprache zu stellen
- Ziehung von Schlüssen bei Abfragen

Die Details dazu sind in Kapitel 6 genauer beschrieben.

Bei der Erarbeitung der Grundlagen einigten sich die Autoren, die Modellierung der Ontologie mittels OWL, einer häufig verwendeten Standard-Sprache vorzunehmen. Eine Ontologie-Datei im OWL-Format wächst, gegeben durch ihre Syntax mit zunehmendem Umfang der Ontologie sehr schnell an. Außerdem ist deren Syntax XML-basiert und damit nicht sehr leserlich.

Angesichts dieser beiden Nachteile waren sich die Autoren einig, dass eine Modellierung der Ontologie in der Ontologie-Sprache OWL über ein Hilfsmittel vorgenommen werden sollte.

Die Anwendung Protégé der Universität Stanford erwies sich bei der Recherche am geeignetsten. Protégé erlaubt die Modellierung einer Ontologie in Form von Baumstrukturen und den Export der Ontologie in diverse (Datei-) Formate. Weiter bietet Protégé die Möglichkeit, Informationen einer Ontologie mittels der SPARQL-Abfragesprache abzufragen. Durch einen konfigurierbaren Reasoner erlaubt Protégé das Ziehen von Schlüssen. Eine detaillierte Beschreibung findet sich unter Abschnitt 6.1.

Protégé bietet zum Ziehen von Schlüssen nebst anderen den Pellet-Reasoner an. Die gewählte Graphdatenbank Stardog nutzt ausschließlich Pellet. Daher wurde Pellet als Anwendung zum Ziehen von Schlüssen gewählt. Eine detaillierte Beschreibung von Pellet findet sich unter Unterabschnitt 6.2.1.

Die Modellierung einer Ontologie ist ein schrittweiser Prozess. Die gewählten Werkzeuge (Protégé und Stardog) gestatten dies, bieten allerdings nur bedingt eine grafische Übersicht (Details siehe 6.1.2). Um bei der Modellierung nicht von Beginn an an fixe Strukturen gebunden zu sein, wurde früh im Prozess yEd als geeignetes Werkzeug zur grafischen Modellierung gewählt. Bei yEd handelt sich dabei um ein frei verfügbares Werkzeug zur Erstellung und Bearbeitung von Diagrammen. Innerhalb dieses Werkzeuges entwickelten die Autoren eine Konvention für die grafische Darstellung, um eine konstante Modellierung zu erhalten.

¹<http://stanbol.apache.org>

²<http://www.stardog.com>

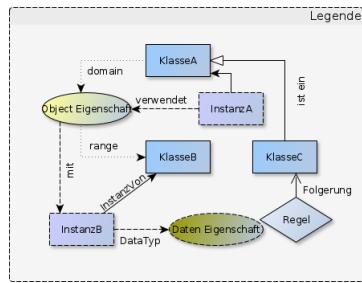
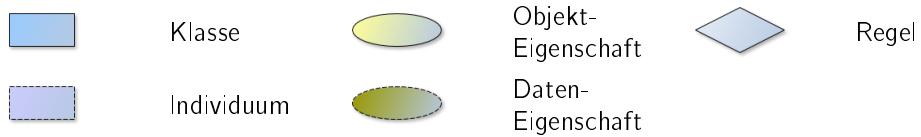


Abbildung 4.2.: Konvention zur grafischen Darstellung der Ontologie.³

Es bedeuten:



Um auch ausserhalb der grafischen Darstellung eine konstante Modellierung zu erhalten, entwickelten die Autoren die folgende Namenskonvention:

Name	Darstellung/Konvention	Beispiel
Klassen	Beginnend mit einem Grossbuchstaben	<i>Ausflug</i>
Individuen	Beginnend mit einem Grossbuchstaben	<i>Seilpark</i>
Objekteigenschaften	Beginnend mit einem Kleinbuchstaben	<i>hatStandort</i>
Dateneigenschaften	Beginnend mit einem Kleinbuchstaben	<i>dauer</i>
Annotationen	Beginnend mit einem Kleinbuchstaben	<i>url</i>

Tabelle 4.1.: Namenskonvention der Modellierung

4.2.6. Modellierung der Ontologie

Vorarbeit für die Modellierung der Ontologie

Ursprünglich wurde versucht ein Modell des Erlernens der Programmierung anhand der (Logik-) Programmiersprache Prolog zu erstellen. Da die Autoren über ein Basiswissen der objektorientierten Programmiersprachen verfügen, wurde UML als Mittel zur Modellierung verwendet. Es zeigte sich rasch, dass es nicht genügt nur Klassen zu definieren; denn eine Ontologie bildet direkte Beziehungen nur zwischen Individuen ab. Die Ontologie muss daher immer über Individuen (Instanzen der Klassen) verfügen.

³Eigene Darstellung mittels yEd.

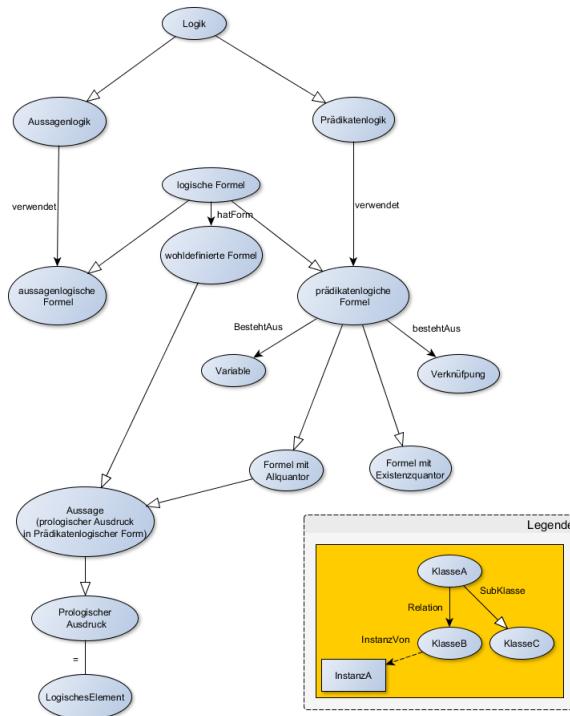


Abbildung 4.3.: Vereinfachte Darstellung von Logik, rein mittels Klassen.⁴

Angesichts dieser Erkenntnis wurde die Modellierung mit Individuen erweitert. Dies ermöglicht die Verwendung von Relationen zwischen Individuen, was richtungsweisend war.

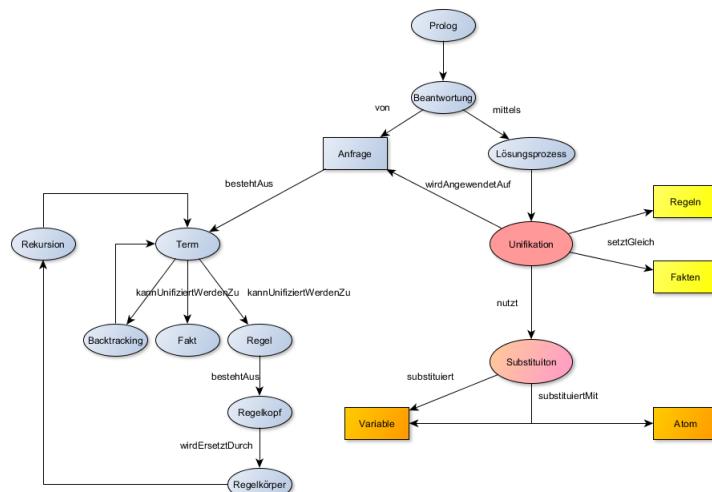


Abbildung 4.4.: Vereinfachte Darstellung eines Teils des Lösungsprozesses von Prolog, mittels Klassen, Individuen und Relationen.⁵

Aufgrund der bisherig erstellten Ontologie konnten konkrete Fragen nicht beantwortet werden. Es zeigten sich Unvollständigkeiten in der Modellierung. Zum Beispiel waren umständlichen Abfragen nötig um einfache Fakten zu erhalten. Details hierzu finden sich im Anhang unter C.

⁴Eigene Darstellung mittels yEd.

⁵Eigene Darstellung mittels yEd.

Schritt 3: Beziehungen des einfachen Token herausfinden

```

SELECT distinct * WHERE {
    ?i a owl:NamedIndividual.
    ?i a ?C.
    ?C ?XX ?Z.      # Gibt alle Beziehungen aus

    FILTER (regex(str(?i), "atom", "i") && ?C=p1:EinfachesToken)
}

```

i	c	yyy	z
Atom	EinFachesToken	type	Class
Atom	EinFachesToken	subClassOf	SprachElement

Feststellung: Wir sehen, dass einfache Tokens die Prädikate `type` und `subClassOf` mit den Subjekten `Class` bzw. `SprachElement` hat. Dies erlaubt nun die Einschränkung des Queries auf den Teil von Interesse, also `subClassOf`.

Abbildung 4.5.: Beispiel einer Abfrage der Ontologie mittels SPARQL.⁶

Bei den genannten Unvollständigkeiten handelte es sich nicht um Fehler: Die Modellierung drohte ins Uferlose zu entgleiten, da der Detaillierungsgrad der Modellierung nicht abgegrenzt war. Damit zeigte sich: Eine zu feine Granularität kann die Modellierung ins Uferlose führen.

Das Ziel der Arbeit war jedoch die Schaffung eines einfachen Systemes um dadurch Erfahrungen mit semantischen Datenbanken und Expertensystemen zu sammeln.

Herr Prof. Dr. Eckerle empfahl bei der Problembesprechung Inhalte des Buches *Künstliche Intelligenz* von U. Lämmel und J. Cleeve U. Lämmel [2012] als einschränkende Richtlinie zur Modellierung zu verwenden.

Das bedeutet Beschränkung der Wissensdomäne ausschließlich auf die Programmiersprache Prolog und deren Kernkonzepte.

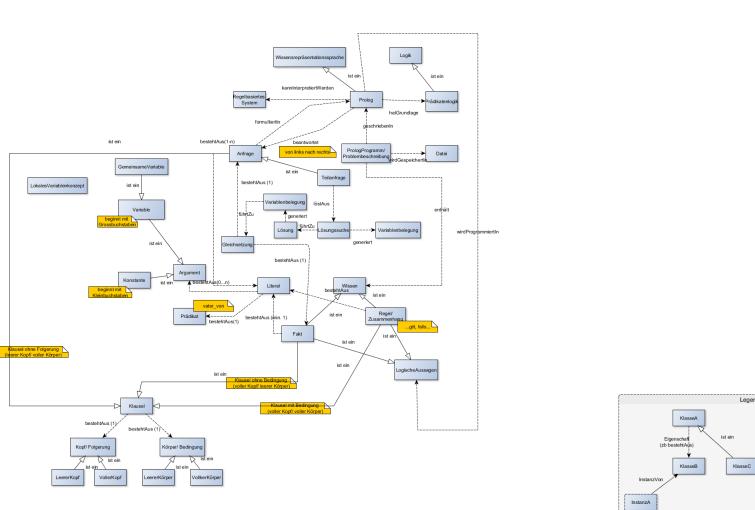


Abbildung 4.6.: Darstellung der Ontologie von Prolog, mit Klassen und Relationen (Individuen wurden der Übersicht halber bewusst weggelassen).⁷

Die verbesserte Modellierung bedeutete jedoch keinen Mehrwert in Form von Inferenz. In mehreren Gesprächen bemängelte Herr Prof. Dr. Eckerle das Fehlen von Regeln in der Ontologie. Um von der objektorientierten Denkweise loszukommen und die Anwendung besser zu verstehen, wurde versucht eine Modellierung der Ontologie direkt in Prolog vorzunehmen. Durch die Struktur von Prolog war die Definition und Anwendung von Regeln einfacher zu verstehen. In Prolog zeigte sich rasch, dass eine Abfrage nach Fakten (ohne Regeln) keinen Mehrwert bringt.

⁶Eigene Darstellung mittels Stanford Protégé und Google Docs.

⁷Eigene Darstellung mittels vEd.

Nach mehreren erfolglosen Versuchen wurden keine Regeln für die Ontologie gefunden. Daraufhin versuchten die Autoren am Beispiel eines Familienstammbaumes eine neue Ontologie mit geeigneten Regeln aufzubauen. In diesem "klassischen" Beispiel [U. Lämmel, 2012, S. 152] wurde der Mehrwert der Wissensmodellierung mittels Ontologien sogleich ersichtlich.

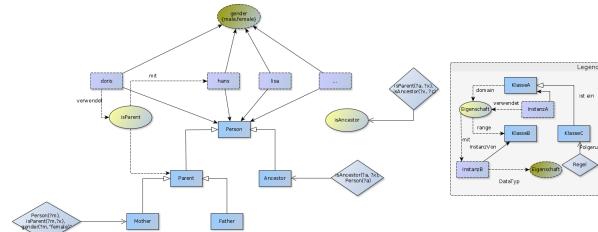


Abbildung 4.7.: *Ontologie eines Familienstammbaumes, mit Klassen, Individuen, Relationen und Regeln.⁸*

Trotz dieser Erfahrung gelang es nicht, Regeln für die eigentliche Ontologie zu finden. Die Abbildung von Prolog in einer Ontologie ist zwar möglich, jedoch nur in lexikalischer Form. Dies macht den Mehrwert der Wissensmodellierung mittels Ontologien zunicht.

Prolog verwendet Unifikation auf Anfragen und setzt dabei Regeln mit Fakten gleich. Bei der Unifikation wird das Konzept Substitution benutzt, welches Variablen durch Variablen und/oder Atome substituiert. Die Abbildung der genannten Konzepte in der Modellierung erfolgte hauptsächlich in Form von Fakten (Klassen und Individuen). Zusätzlich konnten einige wenige (komplexe) Regeln gefunden werden. Diese Regeln führen jedoch nicht zu einem Mehrwert.

Das mit diesen Regeln abgebildete Wissen kann auf eine einfachere und intuitivere Weise mittels Fakten abgebildet werden.

Aufgrund obiger Analyse kamen die Autoren zum Schluss, dass sich die gewählte Wissensdomäne (das Erlernen der Programmierung anhand von Prolog) nicht für eine Ontologie eignet. Der Mehrwert der Wissensmodellierung mittels einer Ontologie entsteht in diesem Falle nicht.

Für die Wissensmodellierung bzw. Expertensysteme eignen sich besser Wissensdomänen, die aufgrund der Ontologie Inferenz und damit Schlussfolgerungen zulassen. Die ursprünglich gewählte Wissensdomäne befindet sich auf einer zu hohen Abstraktionsebene. Daher ist dort der konkrete Nutzen, in Form von Inferenz, nicht gegeben.

Expertensysteme finden entsprechend ihrem Namen dort Anwendung, wo ein Fachexperte erforderlich ist. Durch sein Fachwissen kann der Fachexperte Schlüsse ziehen und damit zusätzliches Wissen beitragen.

Aufgrund der oben ausgeführten Erfahrungen versuchten die Autoren eine neue, geeignete Wissensdomäne zu finden. So wurden zum Beispiel die Planung von Hochzeiten und medizinische sowie pharmazeutische Themengebiete in Betracht gezogen, mangels (Fach-) Wissen aber wieder fallen gelassen.

Als definitive Wissensdomäne wählten die Autoren die Planung von Reisen. Diese Wissensdomäne weist eher in Richtung Expertensysteme. Für Expertensysteme scheinen semantische Netze am geeignetsten zu sein.

⁸Eigene Darstellung mittels yEd.

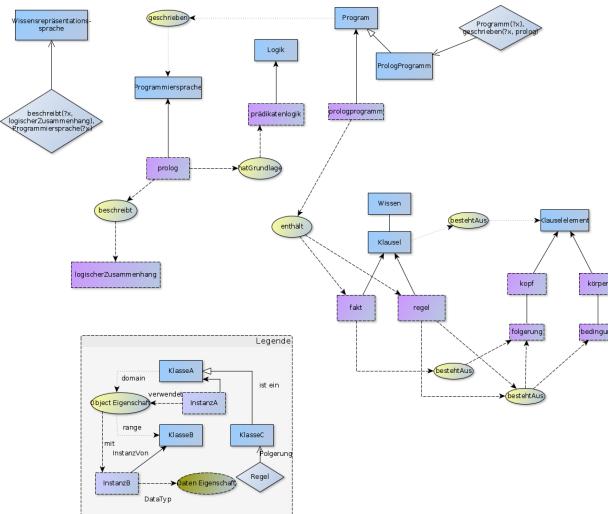


Abbildung 4.8.: Darstellung eines semantischen Netzes zur Abbildung von Prolog, mit Klassen, Individuen, Relationen und Regeln.⁹

Modellierung der tatsächlichen Ontologie

Das heisst, die Modellierung musste neu aufgebaut werden, dabei konnten aber Erkenntnissen und Erfahrungen aus den Vorversuchen verwendet werden.

Die Autoren entschieden sich zunächst Beispiele zu definieren und erst danach die Modellierung der Ontologie vorzunehmen. Auf den Beispielen basierend wurde die Ontologie erzeugt und stetig erweitert.

Zur Veranschaulichung ein konkretes Beispiel:

```
Familie Muster plant einen eintägigen Ausflug .
Die Kinder sind in einem Alter in welchem
sie immer beschäftigt werden müssen .
```

Auflistung 4.1: Konkretes Beispiel einer Reiseplanung.

Die Herleitung des Beispiels findet sich im nachfolgenden Abschnitt. Die zu Grunde liegenden Überlegungen sind im Tutorial aufgeführt.

⁹Eigene Darstellung mittels yEd.

Die Umsetzung des Beispiels führte zu einer sehr einfachen Ontologie, welche in diesem semantischen Netz abgebildet ist:

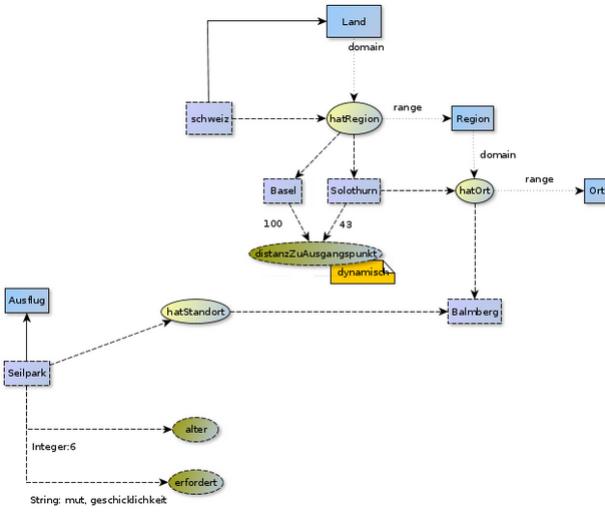


Abbildung 4.9.: Semantisches Netz für den Tagesausflug von Familie Muster.¹⁰

Die Verwendung von Regeln führte in der neu gewählten Wissensdomäne zu einem Mehrwert:

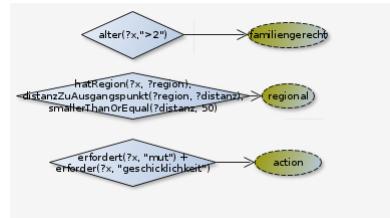


Abbildung 4.10.: Regeln zu dem semantischen Netz für den Tagesausflug von Familie Muster.¹¹

Aus der Modellierung konnten die folgenden Kriterien für die Abfrage abgeleitet werden:

- familiengerecht
- action
- regional

Diese sind zugleich die Schlüsse aus den Regeln (Regelkopf). Mittels einer SPARQL-Abfrage wird der Familie Muster schliesslich der Seilpark in Balmberg vorgeschlagen.

```

SELECT
*
WHERE {
    ?object :familiengerecht true ;
    :regional true ;
    :action true .
}

```

Auflistung 4.2: SPARQL-Abfrage um familiengerechte, regionale und actionreiche Ausflüge zu finden.

¹⁰Eigene Darstellung mittels yEd.

¹¹Eigene Darstellung mittels yEd.

SPARQL Results	
	object
<input checked="" type="checkbox"/>	http://www.semanticweb.org/sosterwalder/ontologies/2014/10/reiseplaner#Seilpark

Abbildung 4.11.: Ergebnis der Suche von Familie Muster.¹²

Damit wird im ausgewählten, bewusst sehr einfach gehaltenen Beispiel, die Herangehensweise der Autoren sichtbar. Bei weiteren, hier nicht aufgeführten Beispielen wurde die Ontologie erweitert.

Zum Beispiel durch Einführung einer Zeiteinheit oder durch eine feinere Rasterung der Eigenschaft familiengerecht. Diese wurde in Untereinheiten aufgeteilt, welche der Altersklasse der Kinder (Kleinkind, Kind oder Teenager) entsprechen.

Die gesamte Ontologie wird im Abschnitt 5.2 genauer erläutert.

4.2.7. Tutorialdokument

Gleichzeitig mit der Modellierung der Ontologie entstand ein Tutorial. Das Tutorial zeigt, wie ein Knowledge-Engineer vorgehen kann um eine Problemdomäne systematisch zu modellieren und formalisieren. Das Tutorial findet sich im Anhang unter B. Da dieses Tutorial, im Gegensatz zu herkömmlichen Tutorials, einen grossen Theorieanteil enthält, wurde es nach drei Aspekte aufgeteilt, als da sind:

- Theoretisches Hintergrundwissen zur Wissensmodellierung;
es wird zum Beispiel erläutert was Expertensysteme sind, wie sie grafisch dargestellt werden können und welche Schreibweisen respektive Sprachen verwendet werden um eine Ontologie abzubilden und darauf Abfragen zu stellen.
- Praktische Hinweise zu den jeweiligen Thematiken
werden im Dokument durch das Symbol Eule gekennzeichnet. Im jeweils danebenstehenden Abschnitt sind praktische Hinweise zur aktuellen Thematik aufgeführt.
- Beispiele, wie ein Thema in der Praxis umgesetzt werden kann
werden im Tutorial durch das Symbol Elefant gekennzeichnet. Sie verkörpern ein Tutorial im klassischen Sinne. Durch Befolgung der aufgeführten Beispiele ist es einem Leser möglich ein simples Beispiel eines Expertensystems aufzubauen.

4.2.8. Dokumentation

Die vorliegende Arbeit entspricht der zusammenfassenden Dokumentation. Sie wurde während der gesamten Bachelor-Thesis stetig erweitert. Es diente zur Reflexion von fertiggestellten Teilen.

¹²Eigene Darstellung mittels yEd.

5. Lösung

Die Aufgabenstellung beschreibt das Ziel dieser Bachelor-Thesis mit dem Satz "Ziel dieser Arbeit ist die Entwicklung und Anwendung eines Systems zur Speicherung (von Daten) in einer semantischen Datenbank...". Eckerle [2014]. Ein auch in der Informatik eher unbekannter Faktor ist dabei die semantische Datenbank.

Das Nutzen von Semantik bedeutet für die Autoren einerseits Sammlung von neuen Theorien und neuen Technologien, andererseits beinhaltet es eine zunächst ungewohnte Denkweise. Dies ist gerade für Personen aus der Informatik, speziell mit objektorientiertem Hintergrund eine Herausforderung. Die Erfahrungen und Schwierigkeiten der Autoren werden im Unterabschnitt 4.2.6 beschrieben.

Die Autoren stellten sich die Anforderung, innerhalb einer bestimmten Wissensdomäne Fragen beantworten zu können (dies wurde bei der Auflistung der Meilensteine nicht erwähnt). Dies setzt bereits fundiertes Fachwissen der Thematik voraus, da Fragen anhand der Bedeutung der Inhalte (Semantik) beantwortet werden sollen.

Die Organisation sowie Vorgehensweise der Autoren wurden im Kapitel 2 und Kapitel 4 ausführlich beschrieben. Dieses Kapitel beschreibt die tatsächliche Lösung der Aufgabenstellung.

5.1. Von der Wissenserarbeitung zum Tutorial

Zunächst musste ein Tutorial erarbeitet werden, um die Theorien und Technologien zu bestimmen. Dabei war es schwierig, eine einfache, nutzbare Form zu erstellen und dabei dem Leser gleichzeitig genügend Hintergrundwissen zu vermitteln. Der Leser soll wirklich verstehen was Wissensmodellierung respektive ein Expertensystem bedeuten. Es entstand das Tutorialdokument, welches diese Faktoren berücksichtigt.

5.2. Modellierung

Semantische Datenbanken werden auf der Basis von Ontologien gebildet. Eine Ontologie wird für eine Problemdomäne erstellt und bildet deren Wissen ab. Problemdomäne dieser Arbeit ist das Reisen. Es wurde beschlossen sich auf Tages- und Wochenendausflüge in der Schweiz zu begrenzen. Die gesamte Modellierung wurde mit Hilfe von Protégé (siehe Abschnitt 6.1) in OWL formuliert. Bei OWL handelt es sich um eine Ontologieabbildungssprache. Benötigte Regeln werden in der Regelsprache SWRL ausgedrückt. Details dazu können im Tutorial unter Kapitel 9 nachgelesen werden.

Um die Ontologie zu veranschaulichen, wurde diese grafisch in einem semantischen Netz abgebildet. Zur Vereinfachung wurde die Ontologie aufgeteilt und in entsprechenden semantischen Teilnetzen abgebildet. Sämtliche Grafiken finden sich als Anhang unter D.1.

Mit der Einschränkung der Problemdomäne konnten zwei Hauptklassen gebildet werden: Ausflüge und Restaurants.

Die Klassen Restaurant wie auch Ausflug werden durch Eigenschaften von Bedingungen (Bedingungs-Properties) beschrieben. Zum Beispiel könnte dabei ein Restaurantbesitzer die Qualität (Ambiente, Küche, Durchschnittspreise) seines Restaurants angeben.

Ausflüge werden durch zusätzliche Werte, wie der Anzahl Teilnehmer, den Öffnungszeiten des Ausflugzieles oder was ein Ausflug bietet oder erfordert beschrieben.

Mittels Regeln wurden die Folgerungen aus diesen Eigenschaften vordefiniert.

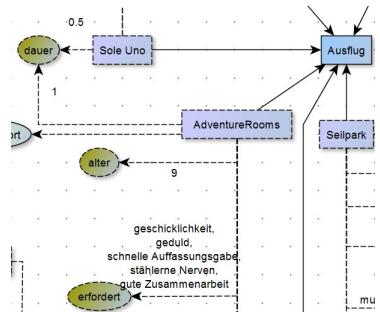


Abbildung 5.1.: Eigenschaften eines Ausflugs.¹

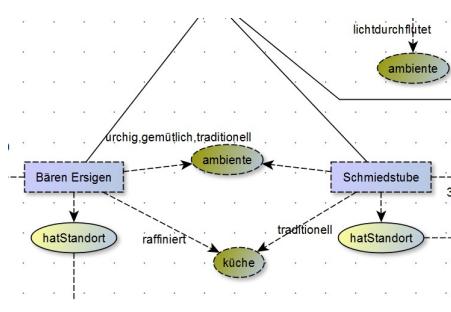


Abbildung 5.2.: Eigenschaften eines Restaurants.¹

Um die verschiedenen Möglichkeiten beim Abbilden einer Ontologie aufzuzeigen, wurde für die Hauptklassen jeweils eine andere Umsetzung gewählt. Die Hauptklasse Restaurant hat verschiedene Unterklassen. Der Benutzer kann dadurch nach verschiedenen Arten von Restaurants suchen. Je nach gewählter Unterklassen (Art des Restaurants) müssen unterschiedliche Kriterien (Bedingungs-Properties) erfüllt sein. Die von den Autoren definierten Regeln lassen zum Beispiel auf ein Gourmetrestaurant schliessen, wenn dieses über eine raffinierte Küche (in Form von Bedingungs-Properties) verfügt.

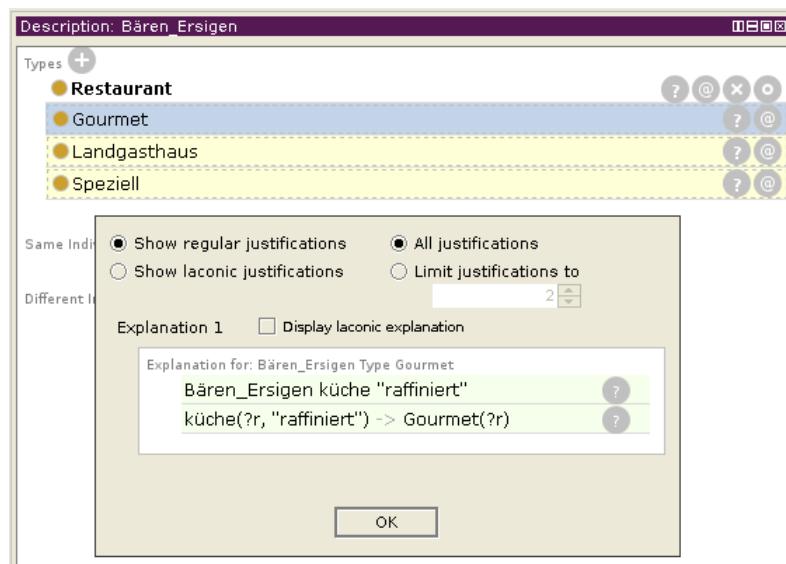


Abbildung 5.3.: Folgerung, dass ein Restaurant ein Gourmetrestaurant ist.²

In semantischen Datenbanken gibt es zwei Arten von Eigenschaften. Erstens Objekteigenschaften (ObjectProperties) und zweitens Dateneigenschaften (DataProperties). Objekteigenschaften beschreiben eine Beziehung zwischen zwei Individuen, dadurch werden sie verknüpft. Dateneigenschaften beschreiben eine konkrete Eigenschaft eines Individuums. Sie haften diesem an.

In der erstellten Ontologie wurde die Eigenschaft *Preissegment* als Beziehung, also als ObjectProperty (*hatPreis*) zwischen zwei Individuen definiert. Daher sind die eigentlichen Preissegmente Individuen der Klasse Preissegment. Jedem Restaurant wird ein Preissegment indirekt zugewiesen. Dies geschieht über die definierte Dateneigenschaft (DataProperty) *Durchschnittspreis*. Liegt der Durchschnittspreis eines Restaurants in einem bestimmten Rahmen, so definieren die Regeln in welchem Preissegment ein Restaurant liegt.

¹Eigene Darstellung mittels yEd.

²Eigene Darstellung mittels Protégé.



Abbildung 5.4.: Regel zur Bestimmung des Preissegmentes „preiswert“.³

Im Gegensatz zu den Restaurants wurden bei Ausflügen Dateneigenschaften zur Schlussfolgerung verwendet. Die gewünschten Dateneigenschaften kann der Benutzer bei der Suche von Ausflügen anwählen. Beispielsweise kann ein Ausflug, der Wellness bietet, entspannend und romantisch sein.

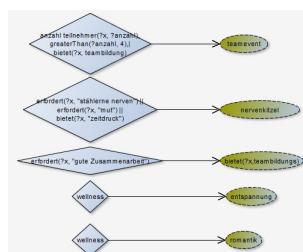


Abbildung 5.5.: Schlussfolgerungen Ausflug.⁴

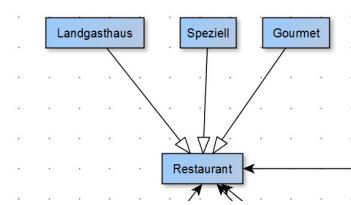


Abbildung 5.6.: Restaurant-Typen.⁴

Dadurch, dass beide Suchziele (Restaurant und Ausflug) einer Region und einem Ort zugewiesen sind, mussten die Autoren berücksichtigen, dass der Benutzer für beide Suchziele entscheiden kann, ob er entweder eine von der Region oder von dem Ort abhängige Beschränkung für die Suche festlegen möchte. Die Beziehung zwischen den Regionen oder Orten der Ausflüge bzw. Restaurants wurde mit Hilfe von Objekteigenschaften (ObjectProperties) umgesetzt.

Um dem Benutzer die Suche nach Restaurants und Ausflügen in derselben Region bzw. demselben Ort zu ermöglichen, haben die Autoren die Objekteigenschaften *gleicherOrt* bzw. *gleicheRegion* eingeführt.

Gewisse Ausflugsziele sind nicht ganzjährig erreichbar (der Seilpark Balmberg ist beispielsweise nur von Frühling bis Herbst geöffnet). Durch die Dateneigenschaften (DataProperties) *offenVon* und *offenBis* eines Ausflugszieles kann festgelegt werden, in welchem Zeitraum ein Ausflug möglich ist. Durch entsprechende Regeln wird festgelegt, in welcher Saison ein Ausflugsziel für Besucher geöffnet ist. Hierfür wurde die Klasse *Jahreszeit* mit den entsprechenden Individuen eingeführt.

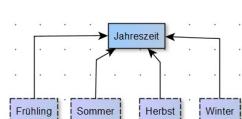


Abbildung 5.7.: Abbildung der Jahreszeiten.⁵

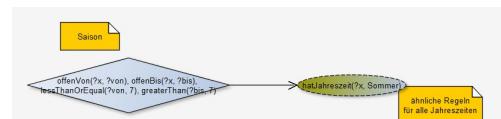


Abbildung 5.8.: Regel zur Bestimmung der Jahreszeiten.⁵

³Eigene Darstellung mittels Protégé.

⁴Eigene Darstellung mittels yEd.

Ein ähnliches Prinzip wurde angewendet, um die Ruhetage eines Ausfluges beziehungsweise eines Restaurants abzubilden. Diese können für jedes Ausflugsziel festgelegt werden. Regeln legen fest, dass der Ausflug an einem bestimmten Tag möglich ist, ausser er sei ein Ruhetag. Diese Festlegung der Ruhetage ist etwas umständlich und nicht sehr intuitiv umgesetzt.

Die Ruhetage bei Ausflugszielen werden durch die Dateneigenschaft *ruhetag* definiert. Dies wird mit einer Zahl von 1 bis 7 (was Montag bis Sonntag entspricht) bezeichnet. Hat ein Ausflug aber keine Ruhetage, so muss diesem dennoch ein Ruhetag zugewiesen werden (eine Zahl die nicht zwischen 1 bis 7, also Montag bis Sonntag liegt). Andernfalls liefern Anfragen keine Ausflüge mehr zurück.

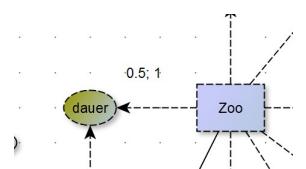
Damit wollen die Autoren eine Einschränkung der Wissensmodellierung mittels Ontologien aufzeigen: Negationen in Regeln sind nicht möglich.

Die zur Umsetzung gewählte Methode zeigt dennoch auch den Vorteil solch einer Modellierung auf. Sie kann beliebig erweitert werden, ohne Notwendigkeit den Programmcode anzupassen.

Ein grösseres Problem bei der Modellierung war die Abbildung von zeitlicher Beschränkung. Ein Benutzer des Reiseplaners soll festlegen können, wieviel Zeit er für eine Reise aufwenden möchte. Im Idealfall würde das System dann alle möglichen Permutationen von Ausflügen berechnen und auflisten. Der Lösung von Planungsproblemen, welche NP-vollständig sind, kann eine ganze Bachelor-Thesis gewidmet werden. Erschwerend kommt hinzu, dass (arithmetische) Berechnungen mittels OWL nur in sehr begrenztem Rahmen möglich sind. Diese Problematik wäre aber mittels Programmlogik zumindest näherungsweise lösbar.

Um die zeitliche Beschränkung von Ausflügen dennoch zu ermöglichen, haben die Autoren entschieden, diese nur in einer einfachen Form abzubilden (die zeitliche Beschränkung ist nicht ein wesentlicher Teil der Arbeit). Es kann festgelegt werden, wieviel Zeit ein Ausflug in Anspruch nehmen darf. Zur Auswahl stehen die Kriterien halbtags, ganztags und mehrtägig, wobei für die mehrtägigen Ausflüge keine Beispiele eingearbeitet wurden.

In der umgesetzten Version des Reiseplaners werden in der Anfrage immer nur jene Ausflüge ausgegeben, welche der gewünschten Zeiteinheit entsprechen. Um die Dauer eines Ausfluges flexibler zu gestalten haben die Autoren entschieden, dass pro Ausflug mehrere Male das Dauer-Attribut gesetzt werden kann. Mit dieser Erweiterung kann ein Ausflug sowohl als halb-, als auch als ganztägig deklariert werden.



Dauer eines Ausflugs festlegen.⁵



Regel zur Bestimmung des Reiseumfangs.⁵

5.2.1. Mängel in Stardog

Bei der Modellierung stiessen die Autoren auf ein für sie zunächst unerklärliches Phänomen: Dem Reasoner von Stardog war es nicht möglich von einer Dateneigenschaften (DataProperty) auf dieselbe Dateneigenschaft zu schliessen. Konkret wollten die Autoren definieren, dass ein Ausflug, der eine Sauna *bietet* auch Wellness *bietet*. Die Auswertung dieser Regel ist aber in Stardog nicht möglich, hingegen wird in Protégé die Schlussfolgerung richtig angezeigt.

Nach zahlreichen Recherchen befanden die Autoren, dass es sich bei dieser Einschränkung wohl um einen technischen Fehler in Stardog handeln muss, obwohl beide Werkzeuge den gleichen Reasoner (Pellet) verwenden.

Nach dem Melden des Fehlers (Bugreporting) in der Diskussionsgruppe von Stardog, wurde den Autoren mitgeteilt, es handele sich wirklich um einen Systemfehler. Um zirkuläre Referenzen im Vorfeld zu verhindern, wurde in Stardog offenbar ein zu strenges Ausschlussverfahren in der Programmlogik verwendet. Der Fehler wurde kurz vor Abschluss der Arbeit in einem Release korrigiert.

Um die gewünschte Folgerung dennoch modellieren zu können, wurde Wellness als Dateneigenschaft vom Typen Boolean definiert und die Regel entsprechend angepasst. Aus zeitlichen Gründen entschieden die Autoren nach Einspielen der neusten Version von Stardog die Modellierung nicht mehr anzupassen.

⁵Eigene Darstellung mittels yEd.

5.2.2. Ergebnis

Die Ergebnisse der Modellierung sind:

- Ein semantisches Netz
- Ein (durch Protégé) generiertes RDF/XML-Dokument

Das semantische Netz stellt die Modellierung grafisch dar. Zur besseren Übersicht wurde es in Teilnetze aufgeteilt, wie oben bereits beschrieben. Im RDF/XML-Dokument ist die semantische Datenbank in Form von konkreten Daten abgelegt.

5.3. Erstellung von Abfragen

Für die Modellierung, wie auch für die Datenbankabfragen durch die Benutzeroberfläche ("Suche"), mussten wiederholt Abfragen erstellt werden. Bei einer semantischen Datenbank müssen diese mittels SPARQL gestellt werden. SPARQL ist eine Abfragesprache für Ontologien (Details dazu finden sich im Tutorial unter Kapitel 10). Durch direkte Integrierung der Abfragen in die erstellte Benutzeroberfläche konnte vermieden werden, dass der Benutzer weder Sprache noch Ontologie kennen muss.

5.4. Benutzeroberfläche

Weitere Aufgabe: "Besondere Bedeutung kommt dabei der Schnittstelle zwischen Mensch und System zu." Eckerle [2014]

Den Autoren wurde bewusst, Abfragen mittels SPARQL sind in keiner Weise benutzerfreundlich. Daher wurde als Benutzerschnittstelle eine Webapplikation zur Reiseplanung entwickelt.

Da für die Abfrage einer Ontologie zwingend eine Abfragesprache verwendet werden muss, wurde zur Verständlichkeit ein Assistent entwickelt. SPARQL wird weiterhin als Abfragesprache verwendet. In der Webapplikation kann der Benutzer mit Hilfe des Assistenten schrittweise Ausflugskriterien wählen. Dadurch ist sichergestellt, dass nur Abfragen gestellt werden können, welche semantisch eine sinnvolle Kombination darstellen.

In einem ersten Schritt kann der Benutzer entscheiden, ob er nur ein Ausflug planen will oder gleichzeitig nach einem Restaurant sucht.

Im zweiten Schritt werden dem Benutzer sämtliche Eigenschaften und Beziehungen der zuvor gewählten Ausflugsarten als Auswahl angeboten.

Schliesslich erhält der Benutzer eine Liste, die sämtliche den gewählten Kriterien entsprechende Ausflüge enthält. In der Liste sind der Name des Ausflugs, sein Standort und die URL zu seiner Homepage aufgeführt.

Je nach gewählten Kriterien ist es möglich, dass keine Reise gefunden wird. Da nur einige, exemplarische Reisen erfasst wurden, ist bei der Verwendung dieser Webapplikation das Risiko für eine leere Antwort relativ gross.

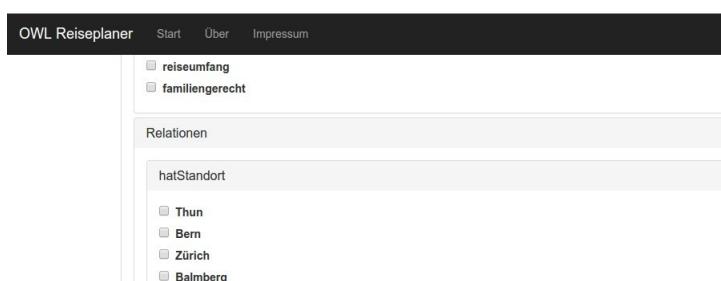


Abbildung 5.9.: Ausschnitt der Benutzeroberfläche. Schritt zwei des Assistenten.⁶

⁶<http://elephantsearch.bfh.ch:5820/>

6. Komponenten

Ein Expertensystem besteht aus drei Komponenten, wie im Tutorial unter Kapitel 2 beschrieben.
Die Komponenten sind:

- Wissensdatenbank: Enthält die Fakten und Regeln des Problembereiches in formaler Sprache
- Inferenz-Maschine: Verarbeitungsmechanismus zum automatischen Ziehen von Schlüssen
- Benutzerschnittstelle

Wie in Abschnitt 4.2.5 erwähnt, ist die Verwendung von Werkzeugen zur Modellierung einer Ontologie sinnvoll. Dabei gibt es Überschneidungen: **Komponenten können auch Werkzeuge sein.**

Das folgende Kapitel beschreibt die während dieser Arbeit verwendeten Komponenten und Werkzeuge.

Die Wissensmodellierung sollte ursprünglich mit Hilfe von Apache Stanbol umgesetzt werden, wie unter Abschnitt 4.2.5 erwähnt. Während der Arbeit wurde erkannt, dass diese Technologie für die vorgesehene Aufgabe nur bedingt sinnvoll ist. In Apache Stanbol ist es möglich die modellierte Wissensdomäne zu importieren. Das importierte Modell wird als Ontologie in Form von Tripeln gespeichert. Die Objekte, ihre Eigenschaften sowie Relationen lassen sich jedoch nicht verwalten.

Für Anfragen an die Wissensdatenbank ist eine entsprechende Schnittstelle notwendig. Von Apache Stanbol wird diese Schnittstelle in Form eines SPARQL-Endpoints zur Verfügung gestellt. Der SPARQL-Endpoint nutzt jedoch die ContentHub-Komponente von Apache Stanbol als Datenbasis. Diese stellt nur Wissen zur Verfügung, welches aus angereicherten Inhalten abgeleitet ist. Mittels Ontologien und Regeln werden diese Inhalte angereichert.

Eine aufwendige Recherche ergab: Anderen Datenquellen können für den SPARQL-Endpoint genutzt werden. Dies bedeutet jedoch einen erheblichen Mehraufwand in Form einer eigenen Implementation.

Damit war Apache Stanbol nicht das geeignete Werkzeug für diese Arbeit. Bei weiteren Recherchen wurden weitere Werkzeuge gefunden, welche als Ersatz für Apache Stanbol genutzt werden konnten: *Protégé* der Universität Stanford sowie *Stardog* der Firma Clark & Parsia.

6.1. Stanford Protégé

Protégé ist eine Entwicklungsumgebung für Ontologien und wurde von den Autoren als *Werkzeug* verwendet. Entwickelt von der Universität Stanford findet es in der Fachwelt häufig Anwendung.

Es unterstützt sowohl die Modellierung von Ontologien, wie auch mittels verschiedener Reasoner das Reasoning.

6.1.1. Merkmale

In Protégé können Ontologien in unterschiedlichen Schreibweisen, wie zum Beispiel OWL/XML, Turtle oder RDF/XML eingelesen und abgespeichert werden. Die Dateien tragen immer die Endung ".owl". Protégé hält eingelesene Ontologien als Graphstruktur im Speicher¹ und ermöglicht Reasonern den direkten Zugriff auf die Graphstruktur².

Die Entwicklungsumgebung zeigt verschiedene Ansichten für dieselbe Ontologie.

Ausser dem Anlegen und Bearbeiten einer Ontologie können in Protégé (ab Version 5.0.0 Beta-16) SWRL-Regeln verwaltet werden. Diese werden von Protégé direkt in der Ontologie abgelegt.

¹<http://protegewiki.stanford.edu/wiki/Protege4Features>

²<http://protege.stanford.edu/products.php>

Durch das SPARQL-Modul bietet Protégé die Möglichkeit Informationen der Ontologie mittels Abfragen zu gewinnen. Ist zusätzlich ein Reasoner-Modul geladen und aktiviert, kann die Umgebung bei Abfragen Inferenzen einbeziehen. Es können verschiedene Reasoner, wie beispielsweise FACT++ oder Pellet genutzt werden (siehe 6.2.1).

(vgl. Stanford University [2014])

6.1.2. Ansichten

Protégé bietet benutzerfreundlich mehrere Sichten auf eine Ontologie an.

Neben der Entitätsansicht (Entity-View, diese enthält sämtliche Elemente einer Ontologie) existiert für alle Elementtypen wie Klassen, Dateneigenschaften, Objekteigenschaften und Individuen eine eigene Ansicht.

Alle Ansichten sind als Baumstruktur organisiert. Dadurch entsteht eine klare Übersicht. Diese wiederum ermöglicht eine Hierarchie-getreue Abbildung des zu Grunde liegenden XML-Dokumentes.

Protégé bietet weitere Ansichten. Beispielsweise kann der Graph einer Ontologie in der OntoGraf-Ansicht betrachtet werden.

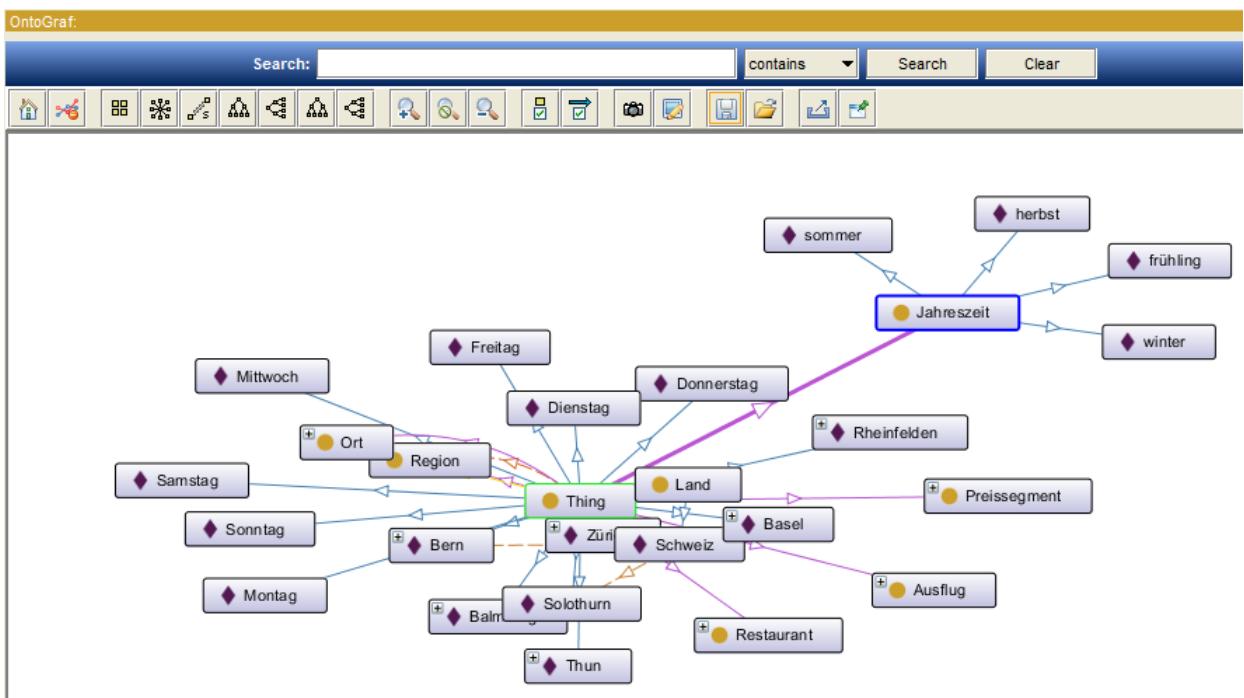


Abbildung 6.1.: Beispiel der OntoGraf-Ansicht.³

Schlussfolgerungen werden in allen Ansichten direkt zur Laufzeit des Reasoners bei allen Elementtypen ergänzt.

(vgl. Stanford University [2009])

6.2. Clark & Parsia Stardog

Bei Stardog handelt es sich um eine Graphdatenbank mit Unterstützung des RDF-Graphenmodells. Sie bietet den Import und Export von Ontologien in diversen Format an und die Möglichkeit Informationen einer Ontologie mittels SPARQL abzufragen. Des Weiteren unterstützt Stardog die Sprache OWL sowie SWRL-Regeln. Pellet, in Stardog enthalten, bildet eine zentrale Komponente für die Schlussfolgerungen.

³Eigene Darstellung mittels Protégé.

Stardog bietet viele Kommunikationsschnittstellen, so zum Beispiel HTTP. Programmierschnittstellen sind für zahlreiche Sprachen, so beispielsweise Java, JavaScript oder Python vorhanden.

Stardog unterscheidet drei Versionen der Lizenzierung: Community, Developer und Enterprise. Die Versionen unterscheiden sich durch maximale Anzahl der verwaltbaren Datenbanken und der möglichen Tripel pro Datenbank, Anzahl gleichzeitiger Verbindungen und durch Anzahl Benutzer und Rollen. Je nach gewählter Lizenzierung leistet der Hersteller eine Kundenbetreuung.

(vgl. Parsia [2010])

6.2.1. Reasoner

Reasoner sind Komponenten, die Folgerungen von implizitem Wissen zulassen bzw. anbieten. Es handelt es sich um eine Art "Verstehen" der Maschinen. Ziel ist es, aus explizitem Wissen in Form einer Ontologie implizites Wissen zu gewinnen.

Wie bereits unter Abschnitt 4.2.5 erwähnt, wurde Pellet als Reasoner (Anwendung zum Ziehen von Schlüssen) gewählt. Eine detaillierte Beschreibung von Pellet findet sich im Tutorial unter Abschnitt 6.4.

6.2.2. Konkrete Anwendung der Komponenten zur Modellierung der Ontologie

Im Laufe der Arbeit hat sich herausgestellt, dass das Schlussfolgern in Protégé nicht einwandfrei funktioniert. Gewisse SPARQL-Anfragen wurden nicht erwartungsgemäss beantwortet.

Da sowohl Protégé als auch Stardog dieselbe Komponente für Schlussfolgerungen (den Pellet-Reasoner) verwenden, muss es sich bei dem Fehlverhalten um einen Fehler bei der Aufbereitung der Abfragen handeln.

In Protégé existiert zudem keine HTTP-Schnittstelle. Es existiert zwar eine Programmierschnittstelle mit welcher eine HTTP-Schnittstelle geschaffen werden kann, dies wurde aber aus zeitlichen Gründen verworfen.

Angesichts der Funktionsstörung und mangelnder HTTP-Schnittstelle, wurde die Ontologie in Protégé erstellt und bearbeitet. Die Weiterverarbeitung der Ontologie erfolgte mittels Stardog, da Stardog die SPARQL-Anfragen korrekt beantwortet und ausserdem eine HTTP- und REST-Schnittstelle bietet.

Dafür musste in Stardog eine neue Datenbank erstellt werden. In diese wurde die mittels Protégé vorbereitete Ontologie im RDF/XML-Format eingespielt.

Stardog bietet bei der Verwendung des Reasoners mehr Möglichkeiten als die anderen Werkzeuge (wie zum Beispiel Protégé). Schlussfolgerungen werden mittels einer Kombination von RDFS-, QL-, RL- und EL-Axiomen sowie zusätzlich SWRL-Regeln gezogen. Details finden sich unter der Webseite docs.stardog.com⁴.

Durch die Kombination der Werkzeuge Protégé und Stardog konnte eine zuverlässige Modellierung und Verwendung der Ontologie erreicht werden.

6.3. Benutzeroberfläche

Ein Teil der Aufgabe bestand darin, eine möglichst benutzerfreundliche Schnittstelle zwischen Mensch und System zu entwickeln.

Als Schnittstelle haben die Autoren eine Webapplikation zur Reiseplanung entwickelt, wie im Abschnitt 5.4 ausgeführt. Diese ermöglicht das Planen einer Reise mit Hilfe eines Assistenten. Der Benutzer kann die gewünschten Reisekriterien auswählen und erhält schliesslich eine Auswahl an Reiseobjekten, welche seinen Kriterien entsprechen.

⁴http://docs.stardog.com/#_reasoning_levels

6.3.1. Komponenten der Benutzeroberfläche

Folgende Komponenten wurden zur Umsetzung der Benutzeroberfläche gewählt:

- *Vagrant*⁵
Virtualisierungslösung
- *nginx*⁶
Webserver
- *Ember.js*⁷
Applikations-Framework
- *Bootstrap*⁸
HTML-, CSS- und JavaScript-Framework von Twitter

(Die Auswahl erfolgte aufgrund beruflicher Erfahrungen der Autoren.)

Vagrant

Bei Vagrant handelt es sich um ein Werkzeug zur Erstellung von vollständigen Entwicklungsumgebungen basierend auf Virtualisierung (vgl. HashiCorp [2014]). Details dazu finden sich unter der Webseite [vagrantup.com](http://www.vagrantup.com)⁹.

In der vorliegenden Arbeit wurde Vagrant zur Bereitstellung einer (virtuellen) Entwicklungsumgebung genutzt. Dabei wird die eigentliche Anwendung von der Entwicklungsumgebung getrennt. Dadurch kann die Entwicklungsumgebung jederzeit mittels weniger Befehle wieder neu aufgebaut werden.

nginx

nginx ("engine x" ausgesprochen) ist ein freier open-source Webserver. Seine Stärken liegen unter anderem auf hoher Leistung, hoher Nebenläufigkeit und geringem Speicherverbrauch. Er ist der am zweithäufigsten eingesetzte Webserver (vgl. Alexeev [2012]).

nginx wurde innerhalb der (virtuellen) Entwicklungsumgebung zur Bereitstellung der Applikation genutzt. So kann die Applikation innerhalb eines Rechners wie eine externe Webseite angesprochen werden.

Ember.js

Ember.js ist ein clientseitiges open-source Framework zur Erstellung von Webapplikationen. Es baut auf dem Model-View-Controller (MVC) Softwarearchitekturmuster auf. Es erlaubt die Erstellung von skalierbaren Anwendungen mittels einer einzigen Datei und bietet dabei deklarative Datenbindung, dynamische Eigenschaften, automatisch aktualisierte Templates sowie eine Zustandsverwaltung der Applikation mittels einer Router-Komponente (vgl. Tilde and Inc. [2014]).

Ember.js wurde für die Umsetzung der Applikationslogik der Benutzeroberfläche eingesetzt.

Bootstrap

Bootstrap der Firma Twitter ist eine Sammlung von Werkzeugen um Webseiten und Webapplikationen zu erstellen. Bootstrap bietet unter anderem verschiedene, auf HTML und CSS basierende Vorlagen für Typographie, Formulare, Schaltflächen und Navigation (vgl. Twitter and Inc. [2014]).

Bootstrap wurde zur grafischen Gestaltung der Benutzeroberfläche eingesetzt.

⁵<http://www.vagrantup.com>

⁶<https://nginx.org>

⁷<http://emberjs.com>

⁸<http://getbootstrap.com>

⁹<http://www.vagrantup.com/about.html>

6.3.2. Architektur

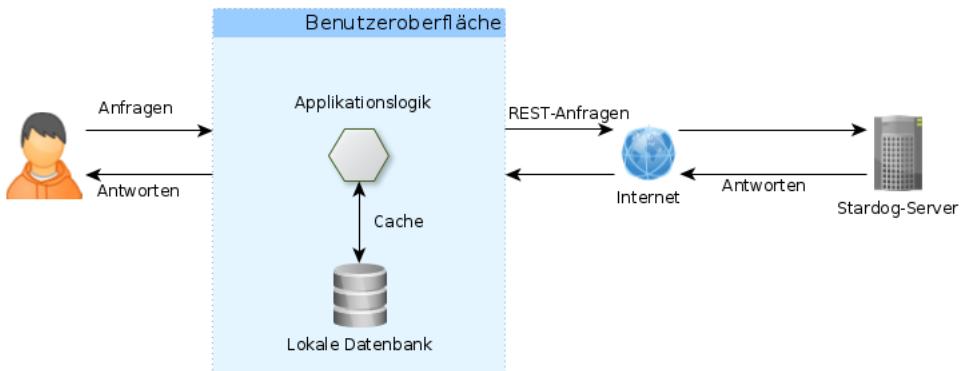


Abbildung 6.2.: Architektur der Benutzeroberfläche¹⁰

Die obenstehende Abbildung 6.2 zeigt die Architektur der Benutzeroberfläche, bestehend aus einem Frontend und einem Backend. Mit Frontend ist die Applikation selbst (Benutzeroberfläche) gemeint, mit Backend der Stardog-Server.

Die Applikation besteht aus *Modellen*, *Ansichten* und *Routen*.

Modelle sind *Reisen*, *Dateneigenschaften* und *Objektrelationen*.

Das *Modell Reise* entspricht *Individuen*, das *Modell Dateneigenschaft DataProperties* und das *Modell Objektrelation* den *ObjectProperties* der Ontologie.

Routen definieren, entsprechend ihrem Namen, den Ablauf der Applikation.

Welcome-Route ist der Einstieg der Applikation. Von der *Welcome*-Route aus werden die *Routen Step1*, *Step2* und *Step3* nacheinander in Form eines Schritt-für-Schritt-Assistenten durchlaufen.

In Schritt 1 wird die Art der Ausflüge gewählt.

In Schritt 2 werden die *Dateneigenschaften* sowie die *Objektrelationen* pro zuvor gewählter Art der Ausflüge gewählt.

Schritt 3 gibt alle Instanzen zurück, welche den gewählten Kriterien entsprechen.

Da in Schritt 1 nur bestimmte Individuen der Ontologie angezeigt werden sollen, wurden in der Ontologie diese Individuen mit der *Dateneigenschaft* "reise" gekennzeichnet. Dadurch beschränkt sich die Abfrage von Schritt 1 nur auf diese Individuen.

Die *Ansichten* kombinieren die Gestaltung (das Layout) der Seite mit der Ansicht der aktuellen Route.

Die gewählte Graphdatenbank Stardog bietet externen Zugriff via REST-Protokoll. Das gewählte Framework Ember.js bietet REST-Unterstützung via REST-Adapter an. Der REST-Adapter implementiert standardmäßig Lesen und Schreiben einer REST-Schnittstelle.

Da das Ziel der Arbeit nicht eine vollständige Umsetzung (Lesen und Schreiben) der Ontologie war, wurde der Zugriff auf die Datenbank nur lesend implementiert. Dadurch kann die umgesetzte Applikation und somit der Benutzer Daten der Ontologie nur lesen (und nicht ändern). Daher musste das Standardverhalten des REST-Adapters von Ember.js durch Nutzung der lokalen Datenbank von Ember.js umgangen werden.

Beim initialen Aufruf der Applikation wird die gesamte Ontologie via REST-Protokoll vom Server abgerufen und nur in der lokalen Datenbank von Ember.js in Form der *Modelle* abgelegt. Damit der Benutzer beim initialen Aufruf der Applikation nicht auf den Datenaustausch warten muss und die Applikation nicht blockiert wird, findet der Datenaustausch asynchron statt. Trotz im Hintergrund laufender Anfragen kann die Applikation benutzt werden. Dabei werden die *Modelle* an die *Ansicht* gebunden. Das heisst, erhält die Applikation Daten eines Modells, wird automatisch dessen *Ansicht* dargestellt bzw. aktualisiert.

¹⁰Eigene Darstellung mittels yEd.

7. Fazit und Erweiterungsmöglichkeiten

Die Ziele der Arbeit konnten erreicht werden, wie im Kapitel 5 dargestellt. Zu erwähnen ist, dass wir im Verlaufe der Arbeit die Problemdomäne ändern mussten. Das Erlernen des Programmierens anhand der Programmiersprache Prolog stellte sich als ungeeignet heraus.

Bei der Auswahl hatten wir noch geringe Kenntnisse über die Wissensmodellierung mittels Ontologien. Uns war nicht bewusst, dass sich diese Form der Wissensmodellierung vor allem für solche Gebiete eignet, die aufgrund der Ontologie Inferenz und damit Schlussfolgerungen zulassen. Die ursprünglich gewählte Problemdomäne, das Erlernen des Programmierens anhand Prolog lag auf einer zu hohen Abstraktionsebene. Im Gegensatz dazu ist die schliesslich gewählte Problemdomäne, das Planen von Reisen viel besser geeignet. Damit war es uns möglich die verschiedenen Spezialitäten eines Expertensystems herauszufinden und die Mächtigkeit von semantischen Datenbanken aufzuzeigen.

Durch diese Arbeit erhielten wir einen Überblick über das Thema semantische Datenbanken. Dabei konnten wir eine mögliche Vorgehensweise bei der Modellierung von Wissen mittels Ontologien analysieren und umsetzen.

Mit Hilfe der eingesetzten Werkzeuge konnte die Modellierung im gewünschten Sinne umgesetzt werden. Hilfreich war beispielsweise die Strukturierung der Daten durch Protégé. Überraschenderweise fanden wir in den beiden Werkzeugen Protégé und Stardog jedoch Fehler (was uns zunächst erhebliche Schwierigkeiten bereitete). Durch die Kombination der beiden Werkzeuge, konnten die Fehler jedoch umgangen werden.

Semantische Netze bieten eine zur Entwicklung freundliche Möglichkeit Wissen zu modellieren. Einerseits bieten sie einen Überblick während der Entwicklung, andererseits ist die Abbildung des Wissens in solchen Netzen auch für Laien gut verständlich.

Nutzt eine Applikation eine semantische Datenbank als Datenmodell, erfordern Anpassungen (Modellierungen) des Datenmodells keine Programmänderungen — bei geschickter Programmierung. Modellierungen sind z.B. das Hinzufügen, Bearbeiten oder Löschen von Entitäten (Klassen, Individuen, Relationen oder Eigenschaften). Im Gegensatz hierzu benötigen Änderungen in relationalen Datenbanken meistens sehr aufwendige Programmänderungen.

Eine eher ungünstige Eigenart dieser Wissensabbildung ist, dass vollständige Prozesse und Abläufe nur schwer abzubilden sind. Dies kann durch andere Sprachen einfacher geschehen.

Bei unserer Arbeit lag der Fokus nicht auf einem definierten Endprodukt. Im Vordergrund stand viel mehr der gesamte Prozess der Wissenserarbeitung. Ein wichtiger Teil der Arbeit bestand in der Modellierung einer Ontologie. Hierbei war es schwierig ein sinnvolles und allgemein gültiges Vorgehen zu finden und zu definieren.

Verschiedenen Schwierigkeiten, auf welche wir während des Prozesses der Wissenserarbeitung gestossen sind, erachten wir als Bereicherung unserer Erfahrung.

Im Rahmen des Prozesses entstand der Prototyp einer Webapplikation zur benutzerfreundlichen Reiseplanung. Diese ermöglicht das Planen einer Reise mit Hilfe eines Assistenten. Der Benutzer kann die gewünschten Reisekriterien auswählen. Er erhält eine Auswahl an Reiseobjekten, welche seinen Kriterien entsprechen.

Weiter entstand ein Tutorial, welches die theoretischen Grundlagen der Wissensmodellierung erklärt und den Leser anhand eines Beispieles durch den gesamten Prozess der Wissensmodellierung führt.

Für die Bachelor-Thesis fiel unsere Entscheidung bewusst auf ein für uns unbekanntes Thema. Wir hofften dadurch, in der Erarbeitung der Bachelor-Thesis in einem in der Geschäftswelt nicht alltäglichen Gebiet forschen und experimentieren zu können. Mit Eintritt in die Berufswelt wird dies wahrscheinlich nur noch begrenzt möglich sein.

Erst beim Reflektieren der Arbeit wurde uns richtig bewusst, ein Themengebiet gewählt zu haben, bei dem wir ausser einigen einfachen Grundlagen alles neu erarbeiten mussten. Dies führte zu einer sehr spannenden und intensiven Arbeit.

7.1. Erweiterungsmöglichkeiten

Bei dem aktuellen Modell der Ontologie sowie der Benutzeroberfläche handelt es sich um einen Nachweis der Machbarkeit in Form von Prototypen. Der Funktionsumfang beschränkt sich daher nur auf das Nötigste und entspricht nicht einer realen Anwendung.

Für eine praxisrelevante Anwendung müsste das Umgesetzte erweitert werden. Um interessante Abfragemöglichkeiten anbieten zu können, müsste die umgesetzte Ontologie stark erweitert werden.

Die Anwendung könnte beispielsweise mit Benutzerprofilen versehen werden, welche Standort des Benutzers speichern und mittels Geoinformationssoftware die Distanz zu den Entitäten (Restaurants und Ausflüge) berechnen.

Bei der Zeitauflösung beschränkt sich das Modell auf halbtägige, ganztägige sowie mehrtägige Ausflüge. Eine feinere Zeitauflösung, zum Beispiel in Form von Gleitkommawerten, würde eine genauere Planung ermöglichen. Da ein Reasoner keine Bedingungserfüllbarkeitsproblem lösen kann, müsste dies in der Applikation umgesetzt werden.

Selbständigkeitserklärung

Wir bestätigen, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt haben. Sämtliche Textstellen, die nicht von uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum: Bern, 14. Januar 2015

Name Vorname: Günzburger Mira Osterwalder Sven

Unterschriften:

Glossar

Abox Assertional Box; Komponente eines Tableau-Reasoners, welche Aussagen zu Individuen enthält, d.h. OWL-Fakten wie Typen, Eigenschaftswerte und logische Äquivalenz. (vgl. Sirin et al. [2005]).

KB Knowledge Base; Eine Kombination einer Abox und Tbox, damit eine komplette OWL-Ontologie. (vgl. Sirin et al. [2005]).

OWL Web Ontology Language; Ontologiesprache für das semantische Web. Mit dieser Sprache können Ontologien beschrieben werden. (vgl. Cambridge Semantics INC (The Smart Data Company) [2014]).

RDF Resource Description Framework; Framework um Informationen aus Ressourcen zu formulieren. Im Web können Informationen mit RDF verarbeitet werden, anstatt diese nur anzuzeigen. RDF bietet ein gemeinsames Framework um die Informationen zwischen Anwendungen auszutauschen ohne dabei die Bedeutung der Informationen zu verändern. (vgl. Schreiber and Raimond [2014]).

RDFS Resource Description Framework Schema; RDF Schema bietet ein Vokabular zur Datenmodellierung von RDF-Daten. Es stellt eine Erweiterung des RDF-Vokabulars dar. (vgl. Brickley and Guha [2014]).

RDQL RDF Data Query Language; Abfragesprache um Informationen von RDF-Graphen zu extrahieren. Die Syntax ist der SPARQL-Syntax sehr ähnlich. (vgl. Seaborne [2004]).

RuleML Rule Markup Language; XML-Sprache zur Beschreibung von Regeln. (vgl. Seaborne [2004]).

SPARQL SPARQL Protocol And RDF Query Language; Bei SPARQL handelt es sich um eine Abfragesprache für RDF. Sie erlaubt es, Abfragen in mehreren Datenquellen vorzunehmen. Dabei werden Anfragen über Graphen vollzogen, auch entlang derer Konjunktionen und Disjunktionen. SPARQL unterstützt weiter Aggregation, Unterabfragen, Negation sowie die Nutzung von Ausdrücken als Werte. Resultate sind entweder eine Menge von Ergebnissen oder RDF-Graphen. (vgl. W3C SPARQL Working group [2013]).

SQL Structured Query Language; "SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen." Wikipedia Foundation [2015].

SWRL Semantic Web Rule Language; Bei SWRL handelt es sich um eine auf OWL und RuleML basierende Regelsprache. Sie erlaubt es Regeln in Form von OWL-Konzepten auszudrücken und bietet dadurch vielfältige Möglichkeiten der Inferenz. (vgl. Horrocks et al. [2004]).

Tbox Terminological Box; Komponente eines Tableau-Reasoners, welche Klassenaxiome enthält, d.h. OWL-Axiome wie z.B. Unterklassen, Gleichheit von Klassen und Klasseneinschränkungen. (vgl. Sirin et al. [2005]).

Literaturverzeichnis

- Johannes Busse, Bernhard Humm, Christoph Lübbert, Frank Moelter, Anatol Reibold, Matthias Rewald, Veronika Schlueter, Bernhard Seiler, Erwin Tegtmeier, and Thomas Zeh. Was bedeutet eigentlich ontologie? *Informatik-Spektrum*, 37(4):286–297, 2014. ISSN 0170-6012. doi: 10.1007/s00287-012-0619-2. URL <http://dx.doi.org/10.1007/s00287-012-0619-2>.
- Frank J. Furrer. Eine kurze geschichte der ontologie. *Informatik-Spektrum*, 37(4):308–317, 2014. ISSN 0170-6012. doi: 10.1007/s00287-012-0642-3. URL <http://dx.doi.org/10.1007/s00287-012-0642-3>.
- Schreiber and Raimond. Rdf 1.1 primer. Website, June 2014. URL <http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/>. W3C Specification; Abgerufen am 10. Oktober 2014.
- Beckett. Rdf/xml syntax specification (revised). Website, March 2013. URL <http://www.w3.org/TR/REC-rdf-syntax/>. W3C Specification; Abgerufen am 19. Oktober 2014.
- Hitzler, Krötzsch, Parsia, Patel-Schneider, and Rudolph. Owl 2 web ontology language. Website, December 2012. URL <http://www.w3.org/TR/2012/REC-owl2-primer-20121211/>. W3C Specification; Abgerufen am 10. Oktober 2014.
- I. Horrocks, Peter F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. Website, May 2004. URL <http://www.w3.org/Submission/SWRL/>. W3C Specification; Abgerufen am 02. November 2014.
- Seaborne W3C SPARQL Working group, Harris. Sparql 1.1 query language. Website, March 2013. URL <http://www.w3.org/TR/sparql11-query/>. W3C Specification; Abgerufen am 10. Oktober 2014.
- W3C SPARQL Working group. Sparql 1.1 overview. Website, March 2013. URL <http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>. W3C Specification; Abgerufen am 18. Oktober 2014.
- J. Cleve U. Lämmel. *Künstliche Intelligenz*. Carl Hanser Verlag München, 4rd edition, 2012. ISBN 978-3-446-42748-7.
- Prof. Dr. Jürgen Eckerle. Aufgabenstellung bachelorthesis. Intranet, September 2014. URL <https://intranet.bfh.ch/TI/de/Studium/Bachelor/Informatik/hochschuljahr20142015/BaThesisHS14/Seiten/default.aspx>. Abgerufen am 03. Januar 2015.
- Stanford University. Protege desktop features. Website, February 2014. URL <http://protegewiki.stanford.edu/wiki/Protege4Features>. Abgerufen am 15. November 2014.
- Stanford University. Protege 4.x views. Website, May 2009. URL http://protegewiki.stanford.edu/wiki/Protege4Views#Individual_views. Abgerufen am 15. November 2014.
- Clark & Parsia. Stardog docs. Website, 2010. URL <http://docs.stardog.com>. Abgerufen am 15. November 2014.
- HashiCorp. About vagrant. Website, 2014. URL <https://www.vagrantup.com/about.html>. Abgerufen am 23. Dezember 2014.
- Andrew Alexeev. *The Architecture of Open Source Applications, Volume II*. May 2012. ISBN 9781105571817.
- Tilde and Inc. Ember.js - a framework for creating ambitious web applications. Website, 2014. URL <http://emberjs.com>. Abgerufen am 23. Dezember 2014.
- Twitter and Inc. Bootstrap - the world's most popular mobile-first and responsive front-end framework. Website, 2014. URL <http://getbootstrap.com>. Abgerufen am 23. Dezember 2014.
- Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner, 2005.

- Cambridge Semantics INC (The Smart Data Company). Owl 101. Website, 2014. URL <http://www.cambridgesemantics.com/semantic-university/owl-101>. Abgerufen am 10. Oktober 2014.
- Dan Brickley and R.V. Guha. Rdf schema 1.1. Website, February 2014. URL <http://www.w3.org/TR/rdf-schema/>. W3C Specification; Abgerufen am 11. Januar 2015.
- Andy Seaborne. Rdql - a query language for rdf. Website, January 2004. URL <http://www.w3.org/Submission/RDQL/>. W3C Specification; Abgerufen am 11. Januar 2015.
- Wikipedia Foundation. Sql. Website, January 2015. URL <http://de.wikipedia.org/wiki/SQL>. Abgerufen am 11. Januar 2015.

Abbildungsverzeichnis

4.1. Zeitplan; Der Titel stellt Jahreszahlen, der Untertitel Semesterwochen dar	6
4.2. Konvention zur grafischen Darstellung der Ontologie. ¹	8
4.3. Vereinfachte Darstellung von Logik, rein mittels Klassen. ²	9
4.4. Vereinfachte Darstellung eines Teils des Lösungsprozesses von Prolog, mittels Klassen, Individuen und Relationen. ³	9
4.5. Beispiel einer Abfrage der Ontologie mittels SPARQL. ⁴	10
4.6. Darstellung der Ontologie von Prolog, mit Klassen und Relationen (Individuen wurden der Übersicht halber bewusst weggelassen). ⁵	10
4.7. Ontologie eines Familienstammbaumes, mit Klassen, Individuen, Relationen und Regeln. ⁶	11
4.8. Darstellung eines semantischen Netzes zur Abbildung von Prolog, mit Klassen, Individuen, Relationen und Regeln. ⁷	12
4.9. Semantisches Netz für den Tagesausflug von Familie Muster. ⁸	13
4.10. Regeln zu dem semantischen Netz für den Tagesausflug von Familie Muster. ⁹	13
4.11. Ergebnis der Suche von Familie Muster. ¹⁰	14
5.1. Eigenschaften eines Ausflugs. ¹¹	16
5.2. Eigenschaften eines Restaurants. ¹	16
5.3. Folgerung, dass ein Restaurant ein Gourmetrestaurant ist. ¹²	16
5.4. Regel zur Bestimmung des Preissegmentes "preiswert". ¹³	17
5.5. Schlussfolgerungen Ausflug. ¹⁴	17
5.6. Restaurant-Typen. ⁴	17
5.7. Abbildung der Jahreszeiten. ¹⁵	17
5.8. Regel zur Bestimmung der Jahreszeiten. ⁵	17
5.9. Ausschnitt der Benutzeroberfläche. Schritt zwei des Assistenten. ¹⁶	19
6.1. Beispiel der OntoGraf-Ansicht. ¹⁷	21
6.2. Architektur der Benutzeroberfläche ¹⁸	24
C.1. Output: Beziehungen der Klasse Unifikation nach oben herausfinden. ¹⁹	116
C.2. Output: Beziehungen der Klasse Unifikation nach unten herausfinden. ²⁰	116
C.3. Output: Details der Beziehungen anzeigen. ²¹	116
C.4. Output: Beziehungen der Klasse Substitution herausfinden. ²²	117
C.5. Output: Details der Beziehungen anzeigen. ²³	117
C.6. Output: Individuen extrahieren. ²⁴	118
C.7. Output: Atom-Objekt selektieren. ²⁵	118
C.8. Output: Klasse/Typ des Individuums herausfinden. ²⁶	119
C.9. Output: Beziehungen des einfachen Token herausfinden. ²⁷	119
C.10. Output: Beziehungen des einfachen Token herausfinden 2. ²⁸	120
C.11. Output: Beziehungen der Klasse SprachElement herausfinden. ²⁹	120
C.12. Output: Beziehungen zu Token. ³⁰	120
C.13. Output: Objekt(e) mit Relation hatSyntaxElement herausfinden. ³¹	121
C.14. Output: Beziehungen zu Prolog herausfinden. ³²	121
C.15. Output: Kommentar(e) der Individuen ausgeben. ³³	122
C.16. Output: Familienbeispiel Anzeige. ³⁴	123
C.17. Semantisches Netz für das Familienbeispiel ³⁵	125

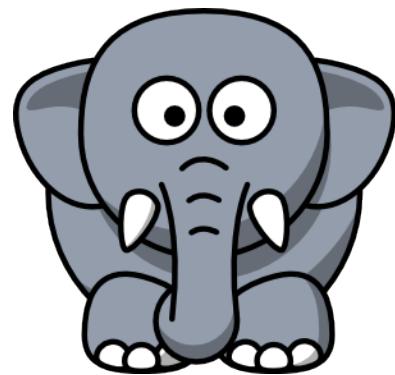
Tabellenverzeichnis

4.1. Namenskonvention der Modellierung	8
--	---

Auflistungsverzeichnis

4.1. Konkretes Beispiel einer Reiseplanung.	12
4.2. SPARQL-Abfrage um familiengerechte, regionale und actionreiche Ausflüge zu finden.	13
C.1. Abfragen auf Relationswerte	113
C.2. Alle Programmiersprachen, welche von einem Programm verwendet werden, welches aus Axiomen besteht	113
C.3. Alle Einsatzgebiete welche in Programmen zum Einsatz kommen, welche aus Axiome bestehen	113
C.4. Alle Einsatzgebiete welche in deklarativen Programmen zum Einsatz kommen	114
C.5. Aus was besteht Prolog?	114
C.6. Aus was bestehen logische Elemente?	115
C.7. Was für sprachliche Elemente verwendet Prolog?	115
C.8. Beziehungen der Klasse Unifikation nach oben herausfinden	115
C.9. Beziehungen der Klasse Unifikation nach unten herausfinden	116
C.10. Details der Beziehungen anzeigen	116
C.11. Beziehungen der Klasse Substitution herausfinden	117
C.12. Details der Beziehungen anzeigen	117
C.13. Individuen extrahieren	117
C.14. Atom-Objekt selektieren	118
C.15. Klasse/Typ des Individuum herausfinden	119
C.16. Beziehungen des einfachen Token herausfinden	119
C.17. Beziehungen des einfachen Token herausfinden 2	119
C.18. Beziehungen der Klasse SprachElement herausfinden	120
C.19. Beziehungen zu Token	120
C.20. Objekt(e) mit Relation hatSyntaxElement herausfinden	121
C.21. Beziehungen zu Prolog herausfinden	121
C.22. Kommentar(e) der Individuen ausgeben	122
C.23. Familienbeispiel in Prolog	123
C.24. Alle Nachfahren von Doris	123
C.25. Ist hans vorfahre von tea?	123
anhang/family.pl	124

Anhang



Elephant Search Anforderungen

Anforderungsdokument der Bachelorthesis

Studiengang: Informatik
Autoren: Sven Osterwalder, Mira Günzburger
Betreuer: Dr. Jürgen Eckerle
Experte: Leclerc Jean-Marie
Datum: 3. Oktober 2014

Versionen

Version	Datum	Status	Bemerkungen
0.1	08.06.2014	Entwurf	Anforderungsdokument erstellen
0.2	08.08.2014	Entwurf	Korrekturen und Ergänzungen
0.3	19.09.2014	Entwurf	Anpassungen an die Aufgabenstellung
0.4	24.09.2014	Entwurf	Anpassung der Milestones nach Besprechung mit Betreuer
1.0	03.10.2014	Entwurf	Fertigstellung des Dokuments nach Absprache mit Betreuer

Inhaltsverzeichnis

1 Einleitung	1
1.1 Aufgabenstellung	1
2 Wissensdomäne	2
3 Komponenten	3
3.1 Architektur	3
3.2 Abbildung der Umwelt mittels Wissensdatenbank	3
3.3 Spracherkennung	4
3.4 Interne und Externe Schnittstellen zur Kommunikation	4
4 Technische Umsetzung	6
5 Ziel der Thesis	7
5.1 Milestones	7
Glossar	8
Literaturverzeichnis	8
Abbildungsverzeichnis	8

1 Einleitung

Das nachfolgende Dokument beschreibt die Anforderungen der Bachelorthesis von Sven Osterwalder und Mira Günzburger. Als Vorarbeit der Bachelorthesis dient die Arbeit über die semantische Suche, welche im Rahmen des Moduls 7302 'Projekt 2' bereits erstellt wurde.

Wie in der Abschlussdokumentation der Projekt 2 Arbeit beschrieben, handelt es sich bei semantischen Suchmaschinen um 'Werkzeuge, die in der Lage sind, auf Fragen mit Hilfe einer Datenbank oder des Internets Antworten zu generieren. Solche Werkzeuge können insbesondere dann eine sehr wertvolle Unterstützung für den menschlichen Experten sein, wenn unter extremer Zeitnot komplexe Entscheidungen getroffen werden müssen, wie beispielsweise in der medizinischen Diagnostik. Die Firma IBM hat vor nicht allzu langer Zeit für eine Überraschung gesorgt, als sie die Leistungsfähigkeit von „Watson“ im Quiz Jeopardy demonstriert hat. In diesem Quiz, wo schwierige, oft zweideutig formulierte Fragen aus beliebigen Bereichen unter Zeitdruck beantwortet werden müssen, konnte sich Watson überlegen gegenüber zwei bisher sehr erfolgreichen menschlichen Champions durchsetzen.' [1]

Wie im Fazit der Projektarbeit beschrieben, muss der Fokus der Thesis verschoben werden. So soll im Rahmen der Bachelorthesis ein Dokument mit Tutorial-Charakter erstellt werden, welches einen leicht verständlichen Einstieg in das Thema der semantischen Datenbanken / Suche bietet. Als Hilfsmittel dafür kommt Apache Stanbol zum Einsatz.

Nachfolgend werden die einzelnen Elemente der Bachelorthesis beschrieben.

1.1 Aufgabenstellung

'Ziel dieser Arbeit ist die Entwicklung und Anwendung eines Systems zur Speicherung in einer semantischen Datenbank auf der Basis von Apache Stanbol. Dies schliesst die Erstellung einer Domänen-Ontologie mittels RDF/OWL ein und die Anwendung dieser Ontologie auf ein Anwendungsproblem, wie beispielsweise der Erlernung einer Programmiersprache ein. Exemplarisch soll aufgezeigt werden, wie dabei ein Knowledge Engineer vorgehen, um eine Problemdomäne systematisch zu modellieren und formalisieren. Besondere Bedeutung kommt dabei der Schnittstelle zwischen Mensch und System zu.' [2]

2 Wissensdomäne

Wie sich in der Vorarbeit herausgestellt hat, ist es notwendig die Domäne, in welcher Anfragen gestellt werden sollen, sehr detailliert abzubilden. Zudem ist die technische Umsetzung der Suche mittels Apache Stanbol weniger weit ausgearbeitet als ursprünglich angenommen.

Um einen leicht verständlichen Einstieg in das Thema der semantischen Datenbanken / Suche bieten zu können, wird die Wissendomäne, mit welcher gearbeitet wird, stark eingeschränkt. Bei der gewählten Domäne handelt es sich um die Programmiersprache Prolog. Anhand der wird an einem exemplarischen Beispiel gezeigt, wie eine Wissendomäne modelliert wird, wie darin enthaltene Daten verknüpft werden und wie schlussendlich die logische Ableitung der Regeln der Domäne erfolgt.

Die Umsetzung der Ontologie, welche die gewählte Wissensdomäne abbildet, wird mittels der Sprache OWL, welche die RDF Syntax nutzt, vorgenommen.

3 Komponenten

Bei den Recherchen der Projektarbeit hat sich die Erkenntnis ergeben, dass eine erfolgreiche Verarbeitung von Anfragen die folgenden Komponenten benötigt:

- Abbildung der Umwelt mittels Wissensdatenbank
- Definieren von Regeln zur Ableitung von Schlüssen mittels Logik
- Interne und externe Schnittstellen zur Kommunikation

Die oben genannten Komponenten werden bereits durch die in der Projektarbeit evaluierten Lösung — Apache Stanbol — zur Verfügung gestellt. Allerdings hat sich gezeigt, dass diese keine dem Menschen leicht verständliche Möglichkeit bietet, um eine Fragestellung in ein für den Menschen logisches Resultat zu überführen.

So kann zum Beispiel die Frage 'Welches ist die Hauptstadt der Schweiz?' nicht einfach so beantwortet werden. Die Lösung bietet Extraktion von Entitäten, deren Eigenschaften und Relationen sowie Abfragen dieser mittels einer speziellen Sprache (SPARQL). Die Überführung der extrahierten Entitäten in eine Abfrage, welche für den Menschen ein sinnvolles Resultat zurückliefert, ist nicht gegeben.

3.1 Architektur

Um die verschiedenen Teile zusammen zu nutzen, bietet Apache Stanbol eine frei konfigurierbare Verkettung dieser an. Dies geschieht mittels einer so genannten Enhancement-Chain. Konkret heißt dies, dass eine beliebige Eingabe dieser Kette übergeben werden kann, worauf dann die erste Softwarekomponente der Kette die Eingabe verarbeitet und das Resultat an die nächste Komponente weiterreicht. Diese Vorgang wird durch sämtliche Komponenten der Kette fortgeführt bis schlussendlich das Endresultat an die anfragende Entität zurückgegeben wird.

3.2 Abbildung der Umwelt mittels Wissensdatenbank

3.2.1 Objekte abbilden

Die Abbildung der Umwelt geschieht in Apache Stanbol mittels dem so genannten Entity Hub. Dieser stellt Informationen zu Entitäten und Objekten einer spezifischen Wissensdomäne zur Verfügung. Die Beziehungen werden in Apache Stanbol in Form von Relationen zwischen den Entitäten abgebildet, analog dazu werden die Eigenschaften als Attribute erfasst.

Die konkrete Arbeit mit dem Entity Hub besteht also darin Objekte, die für die Arbeit gewählten Domäne, als Entitäten abzubilden.

3.2.2 Definieren von Regeln zur Ableitung von Schlüssen mittels Logik

Um nun aus der Wissensdatenbank Schlüsse ziehen zu können, werden Regeln benötigt. Regeln werden verwendet um mittels Bedingungen auf weitere Eigenschaften schliessen zu können. Apache Stanbol unterstützt auf Prädikatenlogik basierende Regeln, welche innerhalb der Rule Store Komponente als Rezepte gespeichert werden. Diese sind nichts anderes als eine Zusammenfassung von Regeln, welche eine ähnliche Objektkategorie betreffen.

3.3 Spracherkennung

Um einen Eingabesatz auszuwerten, muss dieser erst als solcher erkannt werden. Konkret müssen also, die einzelnen Wörter als Tokens identifiziert und einer Sprachkategorie zugeordnet werden. Dies geschieht mittels der Spracherkennungskomponente OpenNLP (open natural language processing). Diese Spracherkennung wird für die Englische Sprache von Stanbol schon weitgehend angeboten.

3.4 Interne und Externe Schnittstellen zur Kommunikation

Um die oben genannten Komponenten ansprechen und nutzen zu können, sind Schnittstellen zur Kommunikation unumgänglich.

3.4.1 Interne Kommunikation

Intern nutzt Apache Stanbol, wie bereits beschrieben, die so genannte Enhancement Chain um von einem gegebenen Input, mittels verschiedenen Komponenten, zu einem Output zu gelangen [3]. Die Enhancement Chain basiert auf einer Graph-Struktur, welche sie zur Kommunikation nutzt.

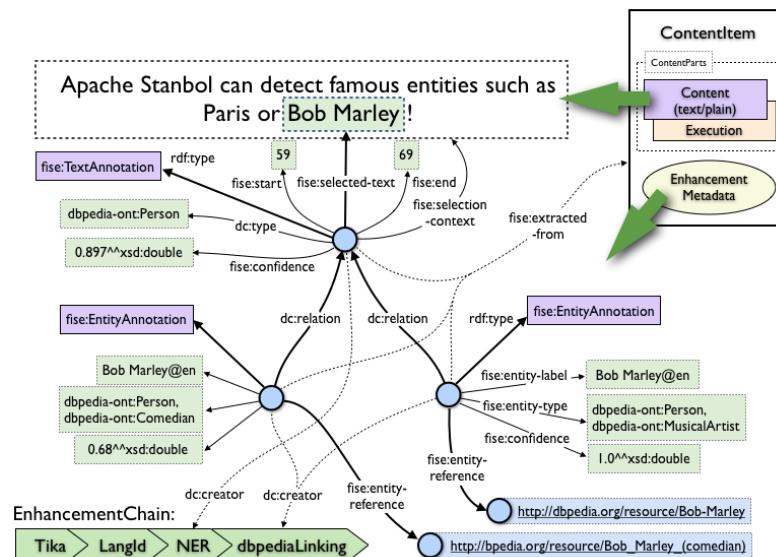


Abbildung 3.1: Apache Stanbol Enhancement Chain¹

¹[4]

3.4.2 Externe Kommunikation

Jede Komponente von Apache Stanbol, so z.B. auch die Enhancement Chain sowie deren Einzelkomponenten, verfügt über ein REST-Interface. Dies dient zur Kommunikation gegen aussen. Ein schematischer Ablauf der Kommunikation wird in Abbildung 3.2 grob dargestellt.

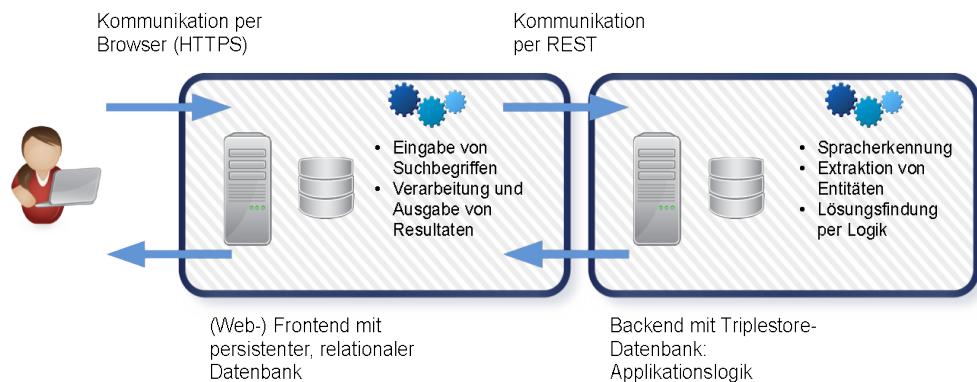


Abbildung 3.2: Kommunikation im Überblick²

²Eigene Darstellung mittels Libre Office Writer

4 Technische Umsetzung

Zur technischen Umsetzung wird, wie in der Abschlussarbeit des Moduls 7302 'Projekt 2' evaluiert, Apache Stanbol eingesetzt. Freundlicherweise steht von der BFH Infrastruktur zum Betrieb zur Verfügung. Der Server steht unter elephantsearch.bfh.ch zur Verfügung.

5 Ziel der Thesis

Als Endresultat der Thesis soll ein Dokument mit Tutorial-Charakter stehen, welches aufzeigt wie Knowledge Engineers vorgehen, um eine Problemdomäne systematisch zu modellieren und zu formalisieren.

Es soll also gezeigt werden, wie eine Speicherung von Daten, die Verknüpfung dieser sowie ihre logische Ableitungen in einer semantischen Datenbank für eine RDF/OWL-Ontologie umgesetzt wird. [2]

Dies wird mittels eines konkreten Anwendungsproblems, der Erlernung der Programmiersprache Prolog, umgesetzt.

5.1 Milestones

Die folgende Auflistung zeigt eine Übersicht, der in der Anfangsphase bereits erkennbaren Meilensteine der Arbeit:

- Anforderungsdokument
- Modellierung der Entitäten, Attribute und Regeln;
Erzeugung der Ontologie mit dem Ziel einer Domänenpezifikation
- Einbetten der Sprache der Domänenpezifikation in die technische Lösung
- Analysieren und Anwenden der Sprache SPARQL
- Installieren der nötigen Infrastruktur auf dem BFH Server
- Realisierung einer einfach handhabbaren Anwenderoberfläche
- Dokument mit Tutorial-Charakter zur Modellierung und Formalisierung einer Problemdomäne
- Abschlussdokument

Literaturverzeichnis

- [1] Sven Osterwalder, Mira G.: *Requirements of Elephant Search – A semantic search engine for children.* 2014
- [2] Eckerle, Dr. J.: *Aufgabenstellung Bachelorthesis.* 2014
- [3] *Apache Stanbol Enhancement Chain.* <http://stanbol.apache.org/docs/trunk/components/enhancer/chains/>
- [4] *Apache Stanbol Enhancement Graph.* <https://stanbol.apache.org/docs/trunk/components/enhancer/enhancementstructure.png>

Abbildungsverzeichnis

3.1 Apache Stanbol Enhancement Chain ¹	4
3.2 Kommunikation im Überblick ²	5



Tutorium

Aufbau von Wissensdomänen anhand einer semantischen Datenbank

Studiengang: Informatik

Autoren: Sven Osterwalder¹, Mira Günzburger²

Betreuer: Prof. Dr. Jürgen Eckerle³

Experte: Jean-Marie Leclerc

Datum: 14. Januar 2015

¹mira.guenzburger@students.bfh.ch

²sven.osterwalder@students.bfh.ch

³juergen.eckerle@bfh.ch

Versionen

<i>Version</i>	<i>Datum</i>	<i>Status</i>	<i>Bemerkungen</i>
0.1	03.10.2014	Entwurf	Initiale Erstellung des Dokumentes
0.2	31.10.2014	In Bearbeitung	Theoretische Grundlagen erarbeitet (OWL, RDF, Expertensysteme, semantische Netze, Graphen, Inferenz und Resolution)
0.3	01.11.2014	In Bearbeitung	Verbinden der Abschnitte (Leitfaden)
0.4	09.11.2014	In Bearbeitung	Kapitel SWRL und SPARQL erarbeitet
0.5	22.12.2014	In Bearbeitung	Beschreibung Reasoner und Tableau-Kalkül, Korrekturen
0.6	28.12.2014	In Bearbeitung	Korrekturen und Schlusswort
0.7	29.12.2014	Vorgelegt	Gegenlesen und Quellenangaben
1.0	30.12.2014	Abgestimmt	Fertigstellung des Dokumentes
1.1	11.01.2015	Abgestimmt	Anpassungen Reasoner
2.0	14.01.2015	Abgenommen	Feinschliff

Inhaltsverzeichnis

1 Einleitung	1
1.1 Wissen	2
2 Expertensysteme	3
2.1 Komponenten	3
2.2 Problemlösung	4
2.3 Wissensarten	4
2.4 Wissensrepräsentationsformalismen	6
3 Graphrepräsentationen	7
4 Wissensrepräsentationsformen	10
4.1 Semantische Netze	10
4.2 Frames	11
4.3 Wissensnetze	11
5 Wissen abbilden mittels Ontologien	14
5.1 Anwendung von Ontologien	15
6 Inferenz und Resolution	17
6.1 Inferenz	17
6.2 Resolution	17
6.3 Inferenz und Resolution zur Ziehung von Schlüssen	18
6.4 Pellet	20
7 RDF	28
7.1 RDF Data Model	28
7.2 Multiple Graphs	29
7.3 RDF Vokabular	30
7.4 RDF Formen	30
8 OWL	32
8.1 OWL Syntax	32
8.2 Wissen modellieren	33
8.3 Die wichtigsten Elemente von OWL	33
8.4 Ontologien	37
8.5 OWL Untersprachen	37
8.6 OWL-Werkzeuge	38
9 Regeln — SWRL	39
9.1 Aufbau	39
9.2 Open world assumption	42
10 SPARQL	44
10.1 Beispiel einer SPARQL Abfrage	45
10.2 Namespaces	45
10.3 Variablen	45
10.4 (Teil-) Graphen	46
10.5 Gruppierungen und Aggregationen	47
10.6 Modifikatoren	48

10.7 Abfragearten	48
10.8 Ausdrücke und Wertevergleiche	52
11 Schlusswort	55
Glossar	56
Literaturverzeichnis	56
Abbildungsverzeichnis	58
Tabellenverzeichnis	59
Auflistungsverzeichnis	61

1 Einleitung

Die relationale Datenspeicherung ist ein klassischer Ansatz zur Wissensabbildung. Häufig wird diese Art der Datenspeicherung in der objektorientierten Programmierung verwendet. Experten aus einer Fachrichtung sind dadurch fähig, die Daten zu interpretieren und Schlüsse zu ziehen. In der heutigen Zeit möchte man Zugang zum Wissen auch ohne die Hilfe von Experten haben.

Daher möchten wir eine neue, eher wenig bekannte Art der Wissensabbildung vorstellen, nämlich die Wissensmodellierung (Knowledge-Engineering) und deren Verwendung in Expertensystemen. Diese berücksichtigt das Abbilden von Objekteigenschaften und -verhalten und kann mit Hilfe von Schlussfolgerungen die Rolle des Experten einnehmen. Sie ist auch Teilgebiet der künstlichen Intelligenz.

Grundsätzlich geht es beim Knowledge-Engineering darum, nicht nur Informationen, sondern auch Wissen abzubilden. Es handelt sich um eine Wissensrepräsentation, welche zusätzliche Verarbeitungsmöglichkeiten aufweist, zum Beispiel logisches Schlussfolgern oder Beweisführungen. Ein wichtiger Aspekt ist die Darstellung in einer von Maschinen lesbaren Form (vgl. [Furrer, 2014, S. 308]).

Ein wichtiges Einsatzgebiet dieser Form der Wissensabbildung ist das semantische Web. Auch bei diesem geht es in erster Linie darum, Informationen mit ihrer Bedeutung (der sogenannten Semantik) zu verbinden.

Das nachfolgende Dokument zeigt, wie Wissen in einem wissensbasierten System abgebildet und genutzt werden kann. Anhand unseres Beispiels eines Reiseplaners wollen wir zeigen, welche Schritte eine Wissensmodellierung beinhaltet. Dafür werden wir zuerst die theoretischen Grundlagen wie Expertensysteme, Wissensrepräsentationsformen und Ontologien einführen. Zudem erklären wir im Kapitel Inferenz und Resolution wie in solchen Systemen zusätzliches Wissen gewonnen wird. Weiter zeigen wir formale Aspekte in Form formaler Sprachen auf. Es sind dies: Die Modellierungssprachen RDF und OWL, die Regelsprache SWRL und die Abfragesprache SPARQL. Aufgrund unserer Erfahrung geben wir praktische Tipps für die direkte Umsetzung.



Als Symbol führen wir die Eule ein. Im jeweils danebenstehenden Abschnitt sind praktische Hinweise zur aktuellen Thematik aufgeführt.

1



Das Symbol Elefant dient um Beispiele zu geben, wie ein Thema in der Praxis umgesetzt werden kann. Das Erscheinen dieses Symbols (am Ende eines Kapitels) zeigt das entsprechende Beispiel im nebenstehenden Text.

2

¹<https://openclipart.org/detail/17566/cartoon-owl-by-lemmling>

²<https://openclipart.org/detail/17810/-by-17810>

1.1 Wissen

Ein Knowledge-Engineer betreibt Wissensakquise mit Hilfe von Experten aus der entsprechenden Fachwelt. Dafür müssen Wissensbestände eines Sachgebietes aufgebaut werden.

Die Kommunikation von Wissensbeständen ist ebenso wichtig wie ihr Vorhandensein. In der Fachwelt hängt die Wissenskommunikation sehr von der Anerkennung der Wissensmodellierung ab.

Je nach Situation ist eine andere Form der Wissensabbildung geeignet. Wenn man sich für eine bestimmte Wissensabbildung entschieden hat, muss diese eine spezielle Aufgabe erfüllen. Hierbei muss der Nutzen der Abbildung im Mittelpunkt stehen.

Eine Information kann für eine bestimmte Problemstellung unumgänglich, für eine andere aber nutzlos sein. Sucht jemand bei der Planung einer Reise zum Beispiel ein familienfreundliches Hotel zur Übernachtung, so ist die Historie der Besitzer dabei nicht relevant. Die Information, wann ein Hotel familienfreundlich ist, jedoch schon.

Es gibt nicht "das richtige" Vorgehen beim Sammeln von Wissen. Es ist aber sinnvoll das Wissen von groben Begriffen bis hin zu detaillierten Informationen einzuteilen.

(vgl. [Busse et al., 2014, S. 287])

Das nachfolgende Kapitel zeigt einen der vielen Formalismen zum Sammeln und Nutzen von Wissen in Form eines Expertensystems auf.

2 Expertensysteme

Bei Expertensystemen handelt es sich um Systeme zur Wissensrepräsentation, welche in einem sehr eingeschränkten (Teil-) Gebiet die Leistung eines menschlichen Experten ersetzen oder diese sogar übertreffen können. Entwicklung und Erfolg von Expertensystemen führten zu einem standardisierten Prozess der Wissensdarstellung und zu Ontologien. Dies vereinfachte die Entwicklung von Expertensystemen in neuen, unerforschten Themengebieten erheblich (vgl. [Russell and Norvig, 2009, S. 257]).

2.1 Komponenten

Um ein Expertensystem aufbauen zu können, muss zuerst das Wissen über einen Problembereich formalisiert werden.

Die dazu benötigten Komponenten sind:

- Eine Wissensdatenbank, welche die Fakten des Problembereiches in formaler Sprache enthält
- Ein Verarbeitungsmechanismus zum automatischen Ziehen von Schlüssen (Inferenz-Maschine)

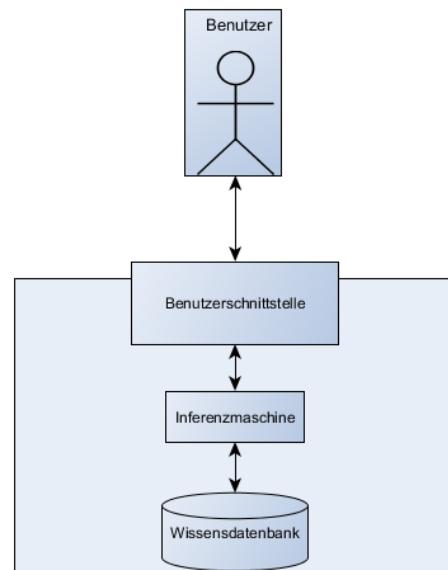


Abbildung 2.1: Aufbau eines Expertensystems.¹

(vgl. [Lämmel and Cleve, 2012, S. 23])

¹Eigene Darstellung mittels yEd, basierend auf [Lämmel and Cleve, 2012, S. 23]

2.2 Problemlösung

Bei der Problemlösung wird typischerweise in folgenden Schritten vorgegangen:

- Charakterisierung der Problemdomäne
- Symbolische Repräsentation der Objekte
- Eingabe des Wissens in den Computer
- Stellen von Fragen
- Interpretation der Antworten

Für die symbolische Repräsentation der Objekte ist es notwendig eine geeignete Sprache zu wählen. Diese Sprache kann beispielsweise aus mathematischen Relationen oder Logik bestehen, oder eine Programmiersprache sein.

In der Informatik wird das Wissen über ein Problem üblicherweise direkt mittels Lösungsalgorithmen programmiert. Bei der künstlichen Intelligenz hingegen, wird das Wissen von der Verarbeitungskomponente getrennt dargestellt. Dies hat den Vorteil, dass die Wissensbasis jederzeit ausgetauscht, die Verarbeitungskomponente jedoch bestehen bleiben kann. Ein Programm in Form eines Expertensystems kann somit also für unterschiedliche Anwendungen verwendet werden.

(vgl. [Lämmel and Cleve, 2012, S. 28 - 30])

Eine Problemlösung geht immer von dem vorhandenen, expliziten Wissen aus, z.B. in Form von Fakten. Aus dem expliziten kann implizites Wissen gewonnen werden, es können damit Aussagen gewonnen werden. Die Aufgabe der Verarbeitungskomponente besteht darin, das implizite Wissen abzuleiten (vgl. [Lämmel and Cleve, 2012, S. 30 - 31]). Die Sprache der Wissensrepräsentation und die zugehörige Verarbeitungskomponente müssen gewissen Kriterien erfüllen, Details siehe [Lämmel and Cleve, 2012, S. 31].

2.3 Wissensarten

Sofern nicht anders vermerkt, basiert der folgende Abschnitt auf [Lämmel and Cleve, 2012, S. 30 - 31].

Um Wissen abzubilden benutzt der Mensch mehrere Arten:

- Relationales Wissen
- Vererbung von Eigenschaften
- Prozedurales Wissen
- Logisches Wissen

2.3.1 Relationales Wissen

Relationales Wissen widerspiegelt einfache Beziehungen zwischen Objekten. Ein Nachteil am relationalen Wissen ist, dass nur Fakten abgebildet werden können, nicht aber logischen Abhängigkeiten.

|| Abenteuerreisen sind Reisen.
 Auflistung 2.1: Einfaches Beispiel von relationalem Wissen.

2.3.2 Vererbung von Eigenschaften

Bei der Vererbung von Eigenschaften geht es um die Weitergabe von Eigenschaften einer Oberklasse an eine Unterklasse.

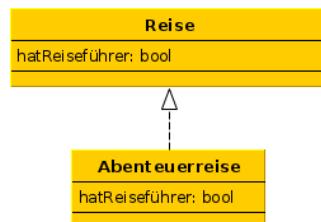


Abbildung 2.2: Einfaches Beispiel der Vererbung von Eigenschaften.²



Eine Reise hat einen Reiseführer. Definiert man, eine Abenteuer-Reise ist auch eine Reise, ist es für den Menschen klar, dass eine Abenteuer-Reise auch einen Reiseführer haben kann. Beim Computer hingegen, muss die Unterklasse "Abenteuer-Reise" zuerst die Eigenschaft der Möglichkeit eines Reiseführers von der Oberklasse erben.

2.3.3 Prozedurales Wissen

Bei prozedurelem Wissen handelt es sich um ein Wissen, welches in bestimmten Situationen bestimmte Aktionen vorschreibt. Man kann dies auch als Folge von Aktionen auffassen. So zum Beispiel das Aufschliessen einer Tür: Man steckt den (passenden) Schlüssel in das Schlüsselloch, dreht diesen, entsichert damit das Schloss, drückt die Türfalle nach unten und öffnet die Tür.

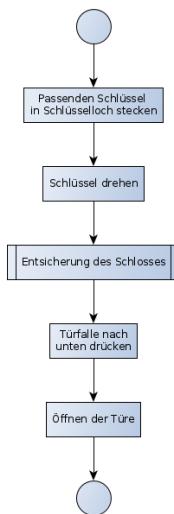


Abbildung 2.3: Einfaches Beispiel von prozedurellem Wissen.³

²Eigene Darstellung mittels yEd.

³Eigene Darstellung mittels yEd.

2.3.4 Logisches Wissen

Bei logischem Wissen geht es im Grunde genommen um eine logische Implikation. Aus A folgt B bzw. $A \rightarrow B$, was so viel heisst wie "Wenn A gilt, kann geschlossen werden, dass auch B gilt."



Möchte man darstellen, dass ein Ausflug teambildend ist, müssen folgende Voraussetzungen erfüllt sein: Für eine gewisse Anzahl Teilnehmer geeignet und gute Zusammenarbeit fördernd.

Formal ausgedrückt: $A \wedge B \rightarrow C$. Hierbei bedeutet A : geeignet für eine gewisse Anzahl Teilnehmer, B : Förderung der Zusammenarbeit und C : ist teambildend.

2.4 Wissensrepräsentationsformalismen

Sofern nicht anders vermerkt, basiert der folgende Abschnitt auf [Lämmel and Cleve, 2012, S. 32].

Formalisiert man die unter 2.3 genannten Arten des Wissens, so gelangt man zu den folgenden Wissenrepräsentationsformalismen:

- Logik
 - Aussagenlogik
 - Prädikatenlogik erster Stufe
- Semantische Netze und Schemata (Frames)
- Regelbasierte Sprachen

Wie unter 2.1 beschrieben, bildet die Basis eines Expertensystems unter anderem eine Wissensdatenbank. Zur Erleichterung des Aufbaus einer Wissensdatenbank und zur grafischen Darstellung eignen sich Graphen sehr gut. Diese werden im nachfolgenden Kapitel beschreiben.

3 Graphrepräsentationen

Sofern im Text nicht anders vermerkt, basiert das nachfolgende Kapitel auf [LinkedDataTools.com \[2010\]](#).

Graphen bilden eine der Grundlagen zur Darstellung von semantischen Daten, besonders für semantische Netze. In diesem Zusammenhang spricht man auch von Graphdatenbanken.

Der Unterschied gegenüber den gängigen Datenbanken, wie zum Beispiel den relationalen oder hierarchischen Datenbanken, liegt vor allem darin, dass Objekte beliebig verknüpft werden können.

Eine Graphdatenbank ist analog einem Graphen aufgebaut. Sie nutzt also dessen Struktur, bestehend aus Knoten, Kanten und Eigenschaften, um Daten bzw. Wissen darzustellen und abzulegen. Eine Graphdatenbank ist also ein Graph mit zyklusfreier Nachbarschaft. Dies bedeutet, dass jedes Element einen direkten Verweis auf seine benachbarten Elemente enthält und somit keine Abfragen auf dessen Relationen notwendig sind.

Die Knoten einer Graphdatenbank repräsentieren Entitäten. Eigenschaften sind auf Knoten bezogene Informationen. Kanten verbinden Knoten mit Knoten oder Knoten mit Eigenschaften und stellen die Beziehung zwischen diesen dar.

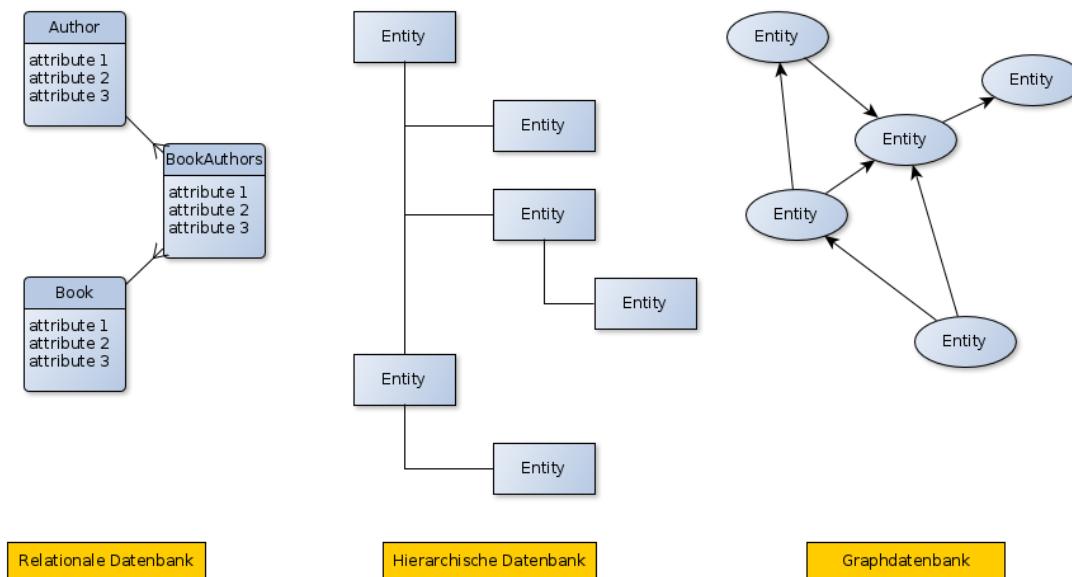


Abbildung 3.1: Darstellung der verschiedenen Datenbanktypen.¹

¹Eigene Darstellung mittels yEd, basierend auf [LinkedDataTools.com \[2010\]](#)

Gegeben seien die folgenden Aussagen über Hotels:

- ||| Ein Wellnesshotel ist ein Hotel.
Ein Familienhotel ist ein Hotel.
Wellnesshotels sind mit Familienhotels verwandt.

Auflistung 3.1: Aussagen über Hotels

Verwendet man diese Hotelangaben um daraus eine Graphdatenbank zu erstellen, so ergibt sich folgende Graphdatenbank:

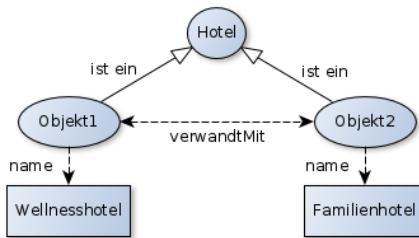


Abbildung 3.2: Aussagen über Hotels als Graphdatenbank.²

In der Graphdatenbank existieren also zwei Objekte, "Objekt1" und "Objekt2", mit den Eigenschaften "ist ein", "verwandtMit" und "name".

²Eigene Darstellung mittels yEd



Möchte man eine Graphdatenbank als Grundlage für eine semantische Datenbank aufbauen, so empfiehlt es sich, zuerst die wichtigsten Klassen und Individuen zu definieren. Es ist lohnend schon zu Beginn des Aufbaus schrittweise vorzugehen.

Nimmt man das Beispiel des Reiseplaners, ist es sinnvoll, nicht von Beginn an eine komplette Reise abzubilden, sondern zunächst mit der Abbildung lokaler Ausflüge zu beginnen. Später können diese mittels Logik zu einer Reise kombiniert werden.

Nach Überlegung ergeben sich die Klassen *Ausflug*, *Land*, *Region* und *Ort*. Doch wie gelangt man zu diesen Klassen? Nach unserer Erfahrung lohnt es sich, konkrete Fragen zu stellen, welche die semantische Datenbank schliesslich beantworten können sollte. Es ist zudem hilfreich, sich praktische Beispiele von der Anwendung solch einer Datenbank zu überlegen.

So könnte eine konkrete Anwendung eine Familie sein, welche einen eintägigen Ausflug planen möchte und deren Kinder bereits in einem Alter sind, in dem sie Beschäftigung benötigen. Dies führt schliesslich zu den Kriterien *familienfreundlich*, *regional* und *actionreich*. Dies sind ausschliesslich Überlegungen für die spätere Entwicklung des Modells. Eine Graphdatenbank kann nicht ohne Weiteres komplexe Anfragen im Sinne der genannten Kriterien oder Inhalte beantworten. Die Zusammenhänge könnte man durch Zuweisung von Kriterien an die Objekte statisch modellieren. Dies würde jedoch den Vorteil einer semantischen Datenbank zunichten machen, nämlich die Gewinnung von Schlüssen aus komplexen Zusammenhängen.

Hat man Klassen und Kriterien definiert, fällt auf, dass damit noch keine konkreten Anfragen beantwortet werden können. Es fehlen die Individuen und die Relationen.

Daher wird für einen Ausflug das Individuum „*Seilpark Balmberg*“ erstellt. Der Seilpark befindet sich in der Ortschaft *Balmberg*, welche in der Region *Solothurn* und damit in der *Schweiz* liegt.

Damit erscheint es logisch, dass wenn ein Land in Regionen unterteilt ist, diese wiederum Orte beinhalten. Daher werden für Länder, Regionen und Orte ebenso Individuen erstellt.

Dies kann über die Relationen *hatRegion* und *hatOrt* abgebildet werden.

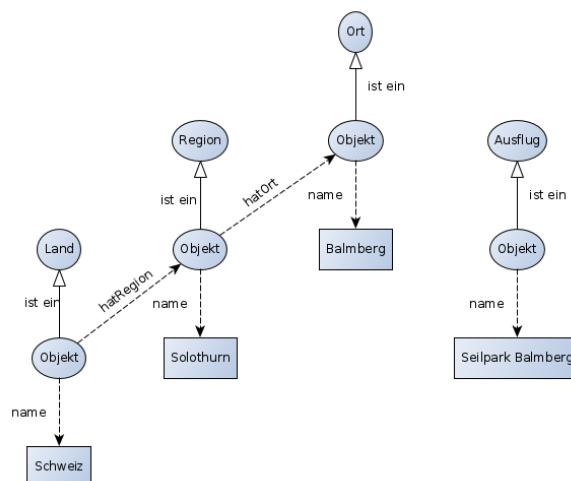


Abbildung 3.3: Beispiel einer Graphdatenbank basierend auf dem Beispiel eines Reiseplaners³

Wie aus der Grafik ersichtlich ist, hat das Individuum *Seilpark Balmberg* noch keine Relationen zu anderen Individuen. Daher ist zum jetzigen Zeitpunkt die Verknüpfung für Mehrwert bei Abfragen unklar.

Da eine Graphdatenbank eine hohe Abstraktionsebene darstellt, werden Wissensrepräsentationsformen sowie ihre Anwendung im folgenden Kapitel genauer erläutert.

³Eigene Darstellung mittels yEd⁴

4 Wissensrepräsentationsformen

Nach Darstellung der Grundlage von Graphen und Graphdatenbanken im letzten Kapitel, wollen wir jetzt Repräsentationsformen von Wissen analysieren.

In der Wissensmodellierung (Knowledge-Engineering) gibt es verschiedene Formen der Wissensrepräsentation um Wissen in wissensbasierten Systemen formal abzubilden. Auf diese Art abgelegte Informationen werden als Wissensdatenbank bzw. Wissensbasis bezeichnet (vgl. Wikipedia Foundation [2014a]).

Das folgende Kapitel basiert auf [Lämmel and Cleve, 2012, S. 85 - 90] und beschreibt einige klassische Formen der Wissensrepräsentation, nämlich semantische Netze, Wissensnetze und Frames.

Im Gegensatz zu Regeln stehen hier die Objekte und nicht Zusammenhänge und logische Abhängigkeiten im Vordergrund.

Semantische Netze und Frames versuchen das menschliche Gedächtnis abzubilden. Sie wurden hauptsächlich zur Analyse von Wörtern und Sätzen verwendet. Ein weiterer Aspekt ist die verständliche Darstellung von Klassen und ihren Beziehungen. Die Konzepte der semantischen Netze und Frames haben die Entwicklung der objektorientierten Programmierung beeinflusst.

4.1 Semantische Netze

Eine zusammengehörige Gruppe von Objekten wird als Klasse bezeichnet. Ein einzelnes Objekt heißt Individuum. Es gibt Beziehungen zwischen Objekten, zwischen Objekten und Klassen und zwischen Klassen.

Folgende Beziehungen werden unterschieden:

- "ist ein" Relation

Es handelt sich um Ober- und Unterklassen.

Beispiel: *Eine Abenteuerreise ist eine Reise.*

- "Instanz von" Relation

Die Relation sagt aus von welchem Typ ein Individuum ist.

Beispiel: *Seilpark Balmberg ist eine Instanz der Klasse Ausflug.*

- Eigenschaft

Klassen und Objekte haben Eigenschaften.

Beispiel: *Ein Ausflug hat einen Standort.*

Eigenschaften sind transitiv: Ist also Riverrafting eine Abenteuerreise und eine Abenteuerreise eine Reise, ist auch Riverrafting eine Reise. Weiter unterliegen Eigenschaften dem Gesetz der Vererbung. Ein Reise hat einen Reiseführer, damit hat auch Riverrafting einen Reiseführer.

In semantischen Netzen werden Objekte und Klassen als Knoten abgebildet. Beziehungen und Eigenschaften werden als Kanten dargestellt. Aussagen wie Existenzaussagen und Oder-Aussagen können mit semantischen Netzen nicht abgebildet werden. Komplexe Abbildungen, wie zum Beispiel das Modellieren einer Aktion, sind trotz reinen zweistelligen Beziehungen möglich.

4.2 Frames

In Frames werden die wesentlichen Charakteristika eines Objektes als Eigenschaften abgebildet. Dabei unterstützen Frames die Konzepte der Hierarchie und der Vererbung. Frames können auch generische Informationen wie Standartwerte (Defaults) und Wertebeschränkungen (Listen) enthalten.

```
frame Abenteuerreise is a Reise :  
    default hatReiseleiter is true  
instance 'Dschungeltrip' is a kind of Abenteuerreise :  
    anbieter is dernettereiseanbieter  
    and typ is abenteuer  
    and kosten is 1000.  
constraint preise  
    when the typ of an Abenteuerreise changes to X  
    then check that X is {dschungel or abenteuer or nevernkitzel or natur}  
    otherwise write( 'Der Typ der Reise wurde nicht geändert' )  
    and nl.
```

Auflistung 4.1: Beispiel eines Frames anhand einer Reise.

4.3 Wissensnetze

Bei Wissensnetzen handelt es sich um eine bestimmte Art der Wissensrepräsentation. Dabei werden die Konzepte der semantischen Netze verwendet.

Wissen wird in Wissensnetzen objektorientiert oder mittels Frames abgebildet. Eine grafische Darstellung erfolgt zusätzlich mittels Topic Maps, auch Wissenslandkarten genannt.

"In einer Wissenslandkarte repräsentiert die Entfernung zweier Begriffe oder Wissensinhalte deren inhaltliche Nähe zueinander. Ein Wissensnetz ist somit ein Informationssystem, in dem zusätzlich die semantischen Beziehungen der Begriffe untereinander verwaltet werden. Diese Meta-Ebene ermöglicht eine **semantische Suche** – eine Suche, die über das Auffinden reiner Zeichenketten weit hinausgeht." [Lämmel and Cleve, 2012, S. 89].

Die Knoten erhalten die Bedeutung von Instanzen (Individuen) oder Klassen. Das Problem der Mehrfachvererbung wird durch Einführung des Rollenkonzeptes umgangen. Durch Erweiterung der Klassen kann eine Instanz dann eine bestimmte Rolle einnehmen.

Wissensnetze haben alle Voraussetzungen, um effektives Wissensmanagement zu bieten. Dafür muss das Wissen aber immer aktuell und umfassend sein.



Ein sehr wichtiger und zeitintensiver Teil des Knowledge-Engineerings ist die tatsächliche Modellierung, das heisst die Überlegung, ob eine abzubildende Information ein Objekt, eine Instanz oder eine Eigenschaft ist. Eventuell kann sie auch als Regel abgebildet werden. Regeln werden im Kapitel 9 Regeln — SWRL genauer erläutert.

An dieser Stelle scheint es uns wichtig zu erwähnen, dass das semantische Netz ein sehr gutes Hilfsmittel ist, um einen Überblick über die Informationen und ihre Anwendbarkeit zu erhalten. Würde man das Wissen direkt in die semantische Datenbank übertragen, wäre diese nicht viel mächtiger als eine traditionelle Wissensspeicherung. Zu einem späteren Punkt erklären wir dies genauer.



Möchte man ein semantisches Netz als Hilfsmittel zum Aufbau der sprachlich formalen Darstellung (Ontologie) eines Reiseplaners nützen, ist das Vorgehen des Aufbaues demjenigen der Graphdatenbank sehr ähnlich.

Für den Aufbau wichtig sind weiterhin die zuvor gemachten Überlegungen, das heisst die Klassen, Individuen und deren Relationen. Aktuell sind dies die *Klassen* Ausflug, Land, Region und Ort, die *Individuen* Schweiz, Solothurn, Bern und Seilpark Balmberg sowie die *Relationen* hatRegion und hatOrt.

Wie ist das weitere Vorgehen? Was unterscheidet ein semantisches Netz von einer Graphdatenbank? Im Kapitel Graphdatenbanken wurde bereits eine wichtige Entität vorweggenommen, die den Hauptunterschied zwischen einem semantischen Netz und einer Graphdatenbank darstellt: Individuen. Diese lassen sich in Graphdatenbanken zwar abbilden, aber weniger intuitiv und aufwändiger.

Bei Verwendung eines semantischen Netzes besteht jedoch die Gefahr, eine ähnliche Modellierung wie bei der Graphdatenbank zu erhalten. Durch Verwendung des Editors Protégé der Universität Stanford und durch OWL 2 als Ontologiesprache konnten wir für die Modellierung der Ontologie einen guten Rahmen finden.

Ziel ist, die bereits formulierten Kriterien *familienfreundlich*, *regional* und *actionreich* so abzubilden, dass entsprechende Abfragen gestellt werden können.

Unter den bisherigen Entitäten lässt sich in unserem Beispiel nur das Kriterium *regional* abbilden.

Wenn wir davon ausgehen, dass der Standort bzw. die Region der Familie bekannt sind — diese seien der Einfachheit halber Solothurn —, sollte die Folgerung möglich sein, dass die Region des Seilparks Balmberg dieselbe wie diejenige der Familie ist.

Man kann mittels der Relation *hatOrt* definieren, dass die *Region Solothurn* den *Ort Balmberg* hat. Dies lässt den Schluss nicht zu, dass sich der Seilpark Balmberg im selben *Ort* oder in der selben *Region* befindet. Dies kann durch die Einführung der neuen Relation *hatStandort* beseitigt werden. Das heisst, das Individuum *Seilpark Balmberg* muss mit dem Ort *Balmberg* verbunden werden: *hatStandort(SeilparkBalmberg, Balmberg)*.

Das oben Genannte kann abgebildet werden wie folgt:

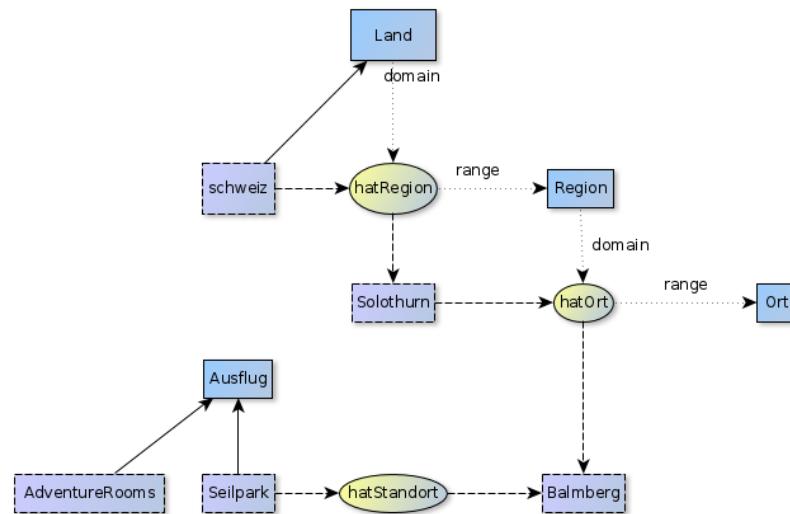


Abbildung 4.1: Abbildung von Wissen mittels eines semantischen Netzes.¹

Bei dieser Abbildung handelt es sich um eine Übersicht der *Informationen*.

Wie kann ich aber schliessen, dass das Individuum *Seilpark Balmberg* die *Region Solothurn* hat? Aus dem vorhandenen *semantischen* Netz lässt sich dies nicht folgern. Somit fehlt die Möglichkeit einer Folgerung.

¹Eigene Darstellung mittels yEd

5 Wissen abbilden mittels Ontologien

Die bisherigen Kapitel bieten einen groben Überblick über Objekte und deren Anwendung. Um bei der Wissensmodellierung von Nutzen zu sein, müssen sie aber in eine geeignete Form gebracht werden. Wissen wird im Knowledge-Engineering zum Beispiel in Form von Ontologien abgelegt.

Der Begriff Ontologie wird in verschiedenen wissenschaftlichen Bereichen verwendet, so zum Beispiel in der Philosophie, der Psychologie und der Informatik. In diesem Dokument soll nur der Aspekt der Informatik erklärt werden.

Sofern nicht anders vermerkt, basiert das folgende Kapitel auf Busse et al. [2014] sowie Furrer [2014].

In der Informatik steht Ontologie für „... eine formale Beschreibung des Wissens in einer Domäne in der Form von Konzepten der Domäne, deren Beziehung untereinander und der Eigenschaft dieser Konzepte und Beziehungen, sowie der in der Domäne gültigen Axiome und Prinzipien.“ [Furrer, 2014, S. 310].

Zum besseren Verständnis wird diese Definition im folgenden Abschnitt kurz untersucht:

- *Domäne*

Unter einer Domäne versteht man einen Ausschnitt der Welt, dessen Grenzen klar definiert sind. Die Domäne und ihre Grenzen werden durch den Anwendungsfall festgelegt.

- *Konzepte*

Konzepte werden in anderen Teilen dieses Dokumentes als Objekte oder Klassen bezeichnet. Dabei kann es sich um materielle Konzepte, wie zum Beispiel Teile eines Wagens oder um immaterielle Konzepte, wie zum Beispiel die Lösungssuche handeln.

- *Beziehungen*

Objekte stehen in Beziehungen zueinander. Es werden die Beziehungen „ist ein“ und „Instanz von“ unterschieden. In unserem Beispiel sind eine Abenteuerreise eine Reise und der Seilpark Balmberg eine Instanz eines Ausfluges (*Abenteuerreise istEin Ausflug* und *SeilparkBalmberg instanzVon Ausflug*). Die Eigenschaften eines Objektes werden auch als Beziehungen dargestellt (eine Reise hat einen Reiseführer).

- *Axiome und Prinzipien*

Damit werden die in der Domäne vorhandenen Regeln bezeichnet.

Mit diesen Elementen sind alle wichtigen Punkte einer Ontologie abgedeckt. Dabei ist es, wie in der Definition festgelegt, wichtig, dass die gesamte Abbildung in einer formalen Sprache beschrieben wird.

Ontologien werden in Basis-, Domänen- und Brücken-Ontologien unterteilt. „Basis-Ontologien (Foundational Ontologies) sind domänenübergreifende Ontologie-Schemata mit allgemeinen Begriffen wie physisches Objekt, Eigenschaft oder Bereich.“ [Busse et al., 2014, S. 291]. Domänen-Ontologien bauen auf Basis-Ontologien auf und richten sich auf ihren Verwendungszweck aus. „Zur Verbindung unterschiedlicher Basis-Ontologien werden sogenannte Brücken-Ontologien entwickelt.“ [Busse et al., 2014, S. 291-292]

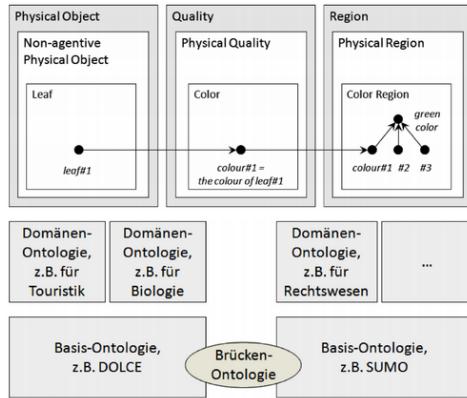


Abbildung 5.1: Ontologiestruktur¹

5.1 Anwendung von Ontologien

Ontologien werden für wissensbasierte Anwendungen verwendet. Konkret werden sie hauptsächlich dort verwendet, wo Semantik zur Formulierung von Informationen genutzt wird. Eines der bekanntesten Beispiele ist das semantische Web. Im Gegensatz zum syntaktischen Web, versucht das semantische Web Zeichen nicht nur abzulegen, sondern auch ein Verständnis und Schlussfolgerungen einzubauen.

Die Aufgabe von Ontologien ist die Ermöglichung und Verbesserung von Kommunikation zwischen Computeranwendungen untereinander und zwischen Computeranwendungen und Mensch. Einige Berufsgruppen haben bereits begonnen ihr Wissen mit Hilfe von standardisierten Sprachen abzubilden. Diese Sprachen, wie die in Kapitel 8 vorgestellte Sprache OWL, wurden vom W3C-Konsortium (World Wide Web Consortium) standardisiert.

5.1.1 Anwendung des semantischen Webs

Eine Anwendung des semantischen Webs führt zu der Gewinnung von zusätzlichem Wissen. Dies geschieht durch eine Wissensbasis in Form einer Wissensdatenbank und einer Inferenzmaschine. Ein Ontologieschema legt in der Wissensbasis fest, welche Arten von Aussagen möglich sind. Die Aussagen enthalten das konkrete Wissen. Sowohl das Schema, wie auch die Wissensabbildung werden in einer formalen Sprache, wie z.B. OWL oder RDFS, abgebildet.

|| Eine Abenteuerreise ist eine Reise.

Auflistung 5.1: Beispiel einer Aussage in einer Wissensbasis.

In der einfachsten Anwendung kann das abgelegte Wissen direkt abgefragt werden. Bei komplexeren Anfragen sind die Funktionalitäten einer *Inferenzmaschine* notwendig. Mittels Inferenzregeln ist diese fähig Schlussfolgerungen zu ziehen und dadurch neue, komplexe Aussagen zu generieren.

Die Transitivitätsregel ist eine einfache, von der Inferenzmaschine verwendete Regel: "Wird eine Relation p als transitiv deklariert und es gilt xpy und ypz , dann kann xpz gefolgt werden." [Busse et al., 2014, S. 289] Dem Entwickler ist es möglich, neben den vorhandenen Regeln, wissensdomänen spezifische Regeln zu spezifizieren. Die Inferenzmaschine berücksichtigt diese bei ihrer Auswertung.

|| Handelt es sich bei einem Objekt um eine Reise, benötigt das Objekt mindestens vier Teilnehmer/innen und ist das Objekt teambildend, so handelt es sich bei dem Objekt auch um einen Teamevent.

Auflistung 5.2: Beispiel einer Regel in einer Wissensbasis.

¹ [Busse et al., 2014, S. 292]

Damit eine Inferenzmaschine Schlüsse ziehen kann, muss die Wissensbasis in einer formalen Sprache vorliegen. Sie darf keine syntaktischen Fehler enthalten. Das Schema und die Aussagen werden von der Maschine geladen und intern als Graph gespeichert (siehe Kapitel 4).

Durch einen Algorithmus zum Abgleich von Graphen werden Abfragen implementiert. Durch wiederholtes Anwenden von Regeln können *Schlussfolgerungen* gezogen werden.

Die Anwendung dieser Regeln erfolgt wiederum mit Hilfe von Algorithmen zum Abgleich von Graphen. Die dadurch entstandenen Aussagen werden der Wissensbasis hinzugefügt und stehen für Abfragen zur Verfügung.



Wir brauchen Ontologien. Dabei spielt die Wahl der Domäne eine grosse Rolle. Bei unserer Wissensmodellierung haben wir gelernt, dass sich nicht jeder Themenbereich als Domäne eignet. Bei Umsetzung der ursprünglichen Problemdomäne, nämlich dem Erlernen des Programmierens anhand der Programmiersprache Prolog, mussten wir feststellen, dass diese nicht als Domäne geeignet ist. Zum Beispiel lässt sich das Konzept der Rekursion in einer Ontologie zwar lexikalisch abbilden, der Mehrwert (einer Ontologie gegenüber der herkömmlichen Wissensabbildung) in Form von Inferenz, ist jedoch nicht gegeben. Die Möglichkeiten einer Wissensmodellierung anhand einer Ontologie sind bei Themen, die Inferenz und damit Schlussfolgerungen zulassen, viel grösser.

Im nachfolgenden Kapitel wird eine Erklärung gegeben, wie Inferenz und Resolution zum Ziehen von Schlüssen verwendet werden.

6 Inferenz und Resolution

6.1 Inferenz

Sofern nicht anders vermerkt, basiert der folgende Abschnitt auf [Russell and Norvig, 2009, S. 163 - 165].

Grundsätzlich geht es bei Inferenz um den Prozess von Schlussfolgerungen mit Hilfe von Resolution (siehe 6.2). Die logische Inferenz ist ein Prozess der Inferenz und der Resolution, welcher die Folgebeziehungen zwischen Sätzen zum Ausdruck bringt.

6.1.1 Inferenz in Computern

Das grundsätzliche Problem bei Computern im Bezug auf Inferenz ist, dass ein Computer keine Interpretation vornehmen kann und nichts über die (Um-) Welt weiß, bzw. nur, was in seiner Wissensdatenbank gespeichert ist.

Angenommen, man möchte einen Computer fragen:

|| “Ist eine Abenteuerreise eine Reise?”

Auflistung 6.1: Beispielaufgabe an eine Wissensdatenbank eines Computers

so weiß der Computer weder was ein Abenteuerreise ist, noch kennt er das Konzept des Reisens an sich. Das Einzige, was er tun kann, ist in der Wissensdatenbank zu suchen nach:

|| “Eine Abenteuerreise ist eine Reise.”

Auflistung 6.2: Aussage in einer Wissensdatenbank eines Computers

Findet der Computer diese Aussage in der Wissensdatenbank, so spielt es keine Rolle, dass er das Konzept der Abenteuerreise oder des Reisens nicht kennt. Die Schlussfolgerung, dass eine Abenteuerreise eine Reise ist, trifft unter allen Gegebenheiten und Interpretationen zu, welche für die Wissensdatenbank zutreffen.

Zusammenfassend ist die formale Inferenz in der Lage, gültige Schlussfolgerungen zu ziehen, auch wenn der Computer die Interpretationen des Anwenders nicht kennt. Der Computer zieht immer logisch gültige Schlüsse, unabhängig von der (menschlichen) Interpretation. Da der Mensch in der Regel die Interpretation kennt, erscheinen die Schlüsse dem Menschen logisch.

6.2 Resolution

Sofern nicht anders vermerkt, basiert der folgende Abschnitt auf [Russell and Norvig, 2009, S. 277 - 279].

Resolution, aus dem Lateinischen “resolutio”, zu Deutsch “Auflösung”, ist eine Verallgemeinerung des Modus Ponens. Der Modus Ponens “... stellt eine universelle Schlussregel dar, die unabhängig vom jeweiligen Problem angewandt werden kann.” [Lämmel and Cleve, 2012, S. 41] Beispiel: *Regen = nasse Strasse. Regen gegeben, also muss die Strasse nass sein.*

Die Methode der Resolution wurde 1965 von J. A. Robinson entwickelt. Dabei handelt es sich um einen vollständigen Algorithmus des Theorembeweises für Prädikatenlogik erster Stufe. [Russell and Norvig, 2009, S. 18] In der einfachsten Form der Resolution handelt es sich um eine Inferenz-Regel der Aussagenlogik.

Eine Verallgemeinerung der einfachen Form der Inferenz-Regel zur Resolution kann als Regel zur kompletten Inferenz der Prädikatenlogik erster Stufe genutzt werden.

6.3 Inferenz und Resolution zur Ziehung von Schlüssen

Inferenz in der Semantik kann grundsätzlich als das Entdecken von neuen Beziehungen zwischen Entitäten beschrieben werden. Das bedeutet, automatische Prozeduren, in Form von so genannten *Reasonern*, leiten neue Beziehungen ab. Wie die neuen Beziehungen generiert werden, ist eine Frage der Implementation, z.B. durch Hinzufügen zu den vorhandenen Daten oder durch einfache Rückgabe derselben (vgl. [World Wide Web Consortium, 2013, Abschn. 1]).

Reasoner sind Komponenten, welche eine Folgerung von implizitem Wissen zulassen bzw. anbieten. Es handelt sich um eine Art "Verstehen" der Maschinen. Ziel ist es, aus explizitem Wissen in Form einer Ontologie implizites Wissen zu gewinnen.

Eine detaillierte Beschreibung, wie man einen Reasoner in der Praxis umsetzt, findet sich in Form des *Pellet*-Reasoners der Firma Clark & Parsia im Abschnitt 6.4. Der *Pellet*-Reasoner basiert auf Beschreibungslogik. Eine Einführung der Grundlagen der Beschreibungslogik folgt im nächsten Abschnitt.

6.3.1 Beschreibungslogik

Beschreibungslogiken sind Formalismen um Wissen darzustellen. Sie sind dabei eine Teilmenge der Prädikatenlogik und stellen den Kern von Wissensrepräsentationssystemen dar. Sie sind eine Struktur für eine Wissensbasis und den damit verbundenen Methoden zur semantischen Folgerung (vgl. Baader et al. [2003]) von neuem Wissen aus der Wissensbasis.

Die Struktur, welche Beschreibungslogiken als Wissensbasis bereitstellt, besteht aus einem Schema (Tbox, Regeln) und aus Daten (Abox, Fakten).

Details zu Beschreibungslogiken finden sich unter Baader et al. [2003].

Interpretation

"Man bezeichnet die Zuordnung von Ereignissen aus einer realen Welt zu aussagenlogischen Variablen als Interpretation." [Lämmel and Cleve, 2012, S. 36]

Sei M die Menge aller aussagenlogischen Formeln. Eine Funktion
 $I : M \rightarrow \{W, F\}$
heisst Interpretation.

Auflistung 6.3: Definition Interpretation¹

Modell

Sei I eine Interpretation und X eine aussagenlogische Formel.
Ist X unter I wahr, so bezeichnet man I als Modell von X .

Auflistung 6.4: Definition Modell²

¹[Lämmel and Cleve, 2012, S. 36]

²[Lämmel and Cleve, 2012, S. 37]

Semantische Folgerung

“Der Zusammenhang von Formeln kann durch den Begriff der semantischen Folgerung dargestellt werden.” [Lämmel and Cleve, 2012, S. 39]

Sei X eine Menge von aussagenlogischen Formeln, Y eine aussagenlogische Formel.
 Y ist eine semantische Folgerung von X falls jedes Modell von X auch Modell von Y ist.
Man schreibt dafür $X \models Y$ und sagt auch Y folgt aus X .

Auflistung 6.5: Definition semantische Folgerung³

Ableitbarkeit

Es ist erforderlich, “...dass das Folgern von Formeln auf der Ebene der Gültigkeit von der Berechnung von Formeln auf der Inferenzebene unterschieden wird. Dies wird mit dem Begriff des Ableitens getan.” [Lämmel and Cleve, 2012, S. 42]

Y ist aus X ableitbar,
 $X \vdash Y$
wenn eine endliche Folge von Inferenzschritten existiert,
so dass man von X zu Y gelangt.

Auflistung 6.6: Definition Ableitbarkeit⁴

Korrektheit und Vollständigkeit

“Der Bezug zwischen beiden Begriffen (semantische Folgerung und Ableitbarkeit, Anm. der Autoren) wird durch die Begriffe der Korrektheit und der Vollständigkeit hergestellt.” [Lämmel and Cleve, 2012, S. 43]

Ein Beweis-Verfahren heisst korrekt, wenn für beliebige Formeln X, Y gilt:
Falls $X \vdash Y$ gilt, dann gilt auch $X \models Y$.
Ein Beweis-Verfahren heißt vollständig, wenn für beliebige Formeln X, Y gilt:
Falls $X \models Y$ gilt, dann gilt auch $X \vdash Y$.

Auflistung 6.7: Definition Korrektheit und Vollständigkeit⁵

Äquivalenz von semantischer Folgerung und Widerspruchsbeweis

Um zu überprüfen, ob eine Formel wahr ist, kann getestet werden, ob Ihre Negation falsch ist. Es wird also ein Widerspruchsbeweis durchgeführt. [Lämmel and Cleve, 2012, vgl. S. 47]

Folgende Aussagen sind äquivalent:
 $X \models Y$
 $X \wedge \neg Y$ ist widersprüchlich

Auflistung 6.8: Äquivalenz von semantischer Folgerung und Widerspruchsbeweis⁶

³[Lämmel and Cleve, 2012, S. 39]

⁴[Lämmel and Cleve, 2012, S. 42]

⁵[Lämmel and Cleve, 2012, S. 43]

⁶[Lämmel and Cleve, 2012, S. 47]

6.4 Pellet

Sofern im Text nicht anders vermerkt, basiert das nachfolgende Kapitel auf Sirin et al. [2005].

Bei Pellet handelt es sich um einen Reasoner auf Basis von Beschreibungslogik. OWL ist eine syntaktische Variante der Beschreibungslogik. Bei der Entwicklung von Pellet wurde von Anfang an nur die OWL-DL⁷-Sprache berücksichtigt. Diese ist wiederum eine Teilsprache von OWL-full, siehe Kapitel 8.

Eine komplette Unterstützung des OWL-full Profils ist generell nicht möglich, da dieses nicht entscheidbar ist. Daher beschränkt sich Pellet auf die Verwendung von OWL-DL. [Sirin et al., 2005, S. 13]

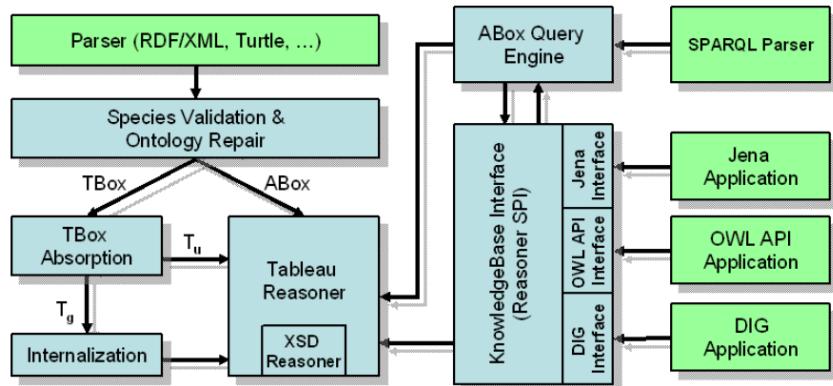


Abbildung 6.1: Hauptkomponenten des Pellet-Reasoners.⁸

Die obenstehende Abbildung zeigt die Hauptkomponenten von Pellet. Dabei ist der Tableau-Reasoner die Kernkomponente, welche die Wissensbasis auf deren Konsistenz prüft. Der Entscheid bei der Entwicklung von Pellet, ausschliesslich OWL-DL zu verwenden, führte zu einer modularen Struktur. Module ihrerseits sind z.B. ein Reasoner zur Datentypenprüfung eines XML-Schemas oder eine "Query-Engine". Genaueres dazu wird in den nachfolgenden Abschnitten beschrieben.

Hier eine Übersicht der wichtigsten Komponenten von Pellet[Sirin et al., 2005, S. 4]:

—	Tableau-Reasoner	Komponente, zum Ziehen von Schlüssen durch Konsistenzprüfung einer Ontologie.
Abox	Assertional Box	Komponente, welche Aussagen zu Individuen enthält, d.h. OWL-Fakten wie Typen, Eigenschaftswerte und logische Äquivalenz.
Tbox	Terminological Box	Komponente, welche Klassenaxiome enthält, d.h. OWL-Axiome wie z.B. Unterklassen, Gleichheit von Klassen und Klasseneinschränkungen.
KB	Knowledge Base	Eine Kombination einer Abox und Tbox, damit eine komplett OWL-Ontologie.

Tabelle 6.1: Beschreibung der wichtigsten Komponenten von Pellet⁹

6.4.1 Vorgehen des Pellet-Reasoners

Wird eine Ontologie mittels Parser geladen, wird diese auf deren Gültigkeit bezüglich OWL-DL geprüft. Während des Ladens werden Klassenaxiome in der Tbox, Aussagen über Individuen in der Abox abgelegt. Die Axiome der Tbox werden durch das Standardverfahren von OWL-DL-Reasonern vorverarbeitet. Sie werden durch verschiedene Optimierungsverfahren vereinfacht (siehe 6.4.3) und anschliessend in den Tableau-Reasoner eingespielen.

⁷<http://www.w3.org/TR/owl-ref/#OWLDL>

⁸[Sirin et al., 2005, S. 6]

⁹[Sirin et al., 2005, S. 4]

Laden und Parsen der Daten

Pellet bietet diverse Schnittstellen um Ontologien zu laden. Pellet selbst implementiert keinen RDF/OWL-Parser. Er ist aber in verschiedenen RDF/OWL-Werkzeugen, welche solch einen Parser anbieten, integriert. Dabei unterstützt Pellet die unterschiedlichen Datenstrukturen der Werkzeuge. Weiter stellt der Reasoner Schnittstellen zur Beantwortung von Anfragen der Werkzeuge zur Verfügung.

Tableau-Reasoner

Der Tableau-Reasoner hat nur eine Funktion: Eine Ontologie auf ihre Konsistenz zu prüfen. Eine Ontologie ist dann konsistent, wenn eine Interpretation der Ontologie existiert, welche alle Fakten und Axiome der Ontologie erfüllt. Patel-Schneider et al. [2004] Solch eine Interpretation wird als Modell der Ontologie bezeichnet.

Der Tableau-Reasoner sucht nach solch einem Modell durch Vervollständigung. Dieses wird inkrementell als eine Art Tafel (eben, Tableau) aufgebaut. Dieses Vorgehen wird Tableau-Algorithmus genannt. Die Vervollständigung beginnt mit einem initialen Graphen der Abox. Dabei repräsentieren die Knoten Individuen und Literale (z.B. Zeichenketten, Datumswerte oder auch Nummern). Jедem Knoten wird der entsprechende (Daten-) Typ zugewiesen. Die gerichteten Kanten zwischen den Knoten stellen die Eigenschaften dar.

Durch wiederholtes Anwenden der Transformationsregeln (siehe Tabelle 6.2) zur Erweiterung des Graphen, versucht der Reasoner einen widerspruchsfreien Graphen zu bilden. Dies tut er solange bis entweder ein Widerspruch (Kontradiktion) auftritt oder keine Regeln mehr anwendbar sind.

Der Tableau-Algorithmus basiert auf dem Tableau-Kalkül. Dabei wird mittels Widerspruchsbeweis aufgezeigt, dass eine Interpretation existiert, in der sowohl sämtliche Anforderungen (Prämissen) als auch die Negation der Folgerung (Konklusion) wahr sind. Kann ein solches Modell gefunden werden, ist damit bewiesen, dass die ursprüngliche "Formel" falsch ist. Die in der Abox definierten Daten sind dann konsistent, wenn beim Aufbau des Baumes inklusive der Negation der Folgerung ein Widerspruch auftritt. Wikipedia Foundation [2014b]

Für das Tableau-Kalkül werden Transformationsregeln benötigt. Mit diesen Regeln wird ein Argument unter einer Interpretation wahr gemacht. Es wird also ein Modell erzeugt. Wikipedia Foundation [2014b].

Die Beschreibung des Tableau-Kalküles basiert, sofern nicht anders vermerkt, auf Wikipedia Foundation [2014b].

Nachfolgend sind die Transformationsregeln zum Erzeugen von Modellen aufgezeigt.

Argument	Modell
P	$I(P) = W$
$\neg Q$	$I(Q) = F$
$P \wedge Q$	$I(P) = I(Q) = W$
$P \vee Q$	$I(Q) = W$ oder $I(P) = W$ oder beides
$P \rightarrow Q$	$I(P) = F$ oder $I(Q) = W$ oder beides

Tabelle 6.2: Transformationsregeln des Tableau-Kalküls¹⁰

Mittels nachfolgendem Beispiel soll die Anwendung des Tableau-Kalküles verdeutlicht werden.
Gegeben sind folgende Fakten:

- $S : \text{hatStandort}(\text{SeilparkBalmberg}, \text{Balmberg})$
- $O : \text{hatOrt}(\text{Solothurn}, \text{Balmberg})$

Weiter ist die folgende Regel gegeben:

- $\text{hatStandort}(\text{ausflug}, \text{standort}), \text{hatOrt}(\text{region}, \text{standort}) \rightarrow \text{hatRegion}(\text{ausflug}, \text{region})$

Die genannten Fakten bilden die *Prämissen*. Durch Anwendung des Kalküls soll der Schluss (*Konklusion*) bewiesen werden, dass sich der Seilpark in der Region Solothurn befindet. Die Regel wird also für die Fakten angewendet:

- $R : \text{hatRegion}(\text{SeilparkBalmberg}, \text{Solothurn})$

Um diese Konklusion zu belegen soll der Widerspruchsbeweis angewendet werden. Dazu wird die Negation der Folgerung (also $\neg \text{hatRegion}(\text{SeilparkBalmberg}, \text{Solothurn})$) angenommen.
Konkret wird versucht anhand den vorhandenen Fakten

- $\text{hatStandort}(\text{SeilparkBalmberg}, \text{Balmberg})$

und

- $\text{hatOrt}(\text{Solothurn}, \text{Balmberg})$
(wobei hier Solothurn eine Region ist)

zu beweisen, dass der Seilpark **nicht** in der Region Solothurn liegt. Bei diesem simplen Beispiel ist für den Leser unter Anwendung der oben genannten Regel sofort ersichtlich, dass dies zu einem Widerspruch führen wird: Wenn der Seilpark Balmberg den Standort Balmberg hat und dieser in der Region Solothurn liegt, *muss* der Seilpark Balmberg auch in der Region Solothurn liegen. Damit wird die ursprüngliche Annahme bestätigt. Durch Anwendung des Tableau-Kalküles kann dies auch mathematisch bewiesen werden.

Es soll $\Gamma \models \varphi$ gelten, wobei Γ die Menge der Prämisse und φ die Konklusion ist:

- $\Gamma : \{S \wedge O \rightarrow R, S, O\}$
- $\varphi : \{R\}$

Weil das Tableau-Kalkül darauf basiert einen Widerspruch zu erreichen, wird die Negation der Konklusion ($\neg \varphi$) mit der Menge der Prämisse (Γ) vereinigt:

- $\Gamma \cup \{\neg \varphi\}$
bzw.
- $\{S \wedge O \rightarrow R, S, O\} \cup \{\neg R\}$

ergibt: $\{S \wedge O \rightarrow R, S, O, \neg R\}$

Zur Anwendung des Kalküls werden alle Sätze der gesamten Menge untereinander aufgeschrieben:

$$(a) \quad S \wedge O \rightarrow R$$

$$(b) \quad S$$

$$(c) \quad O$$

$$(d) \quad \neg R$$

(a), (b), (c) und (d) bilden zusammen den *Wurzelknoten* des Graphen.

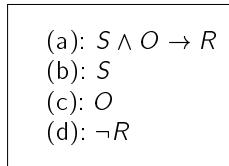


Abbildung 6.2: Wurzelknoten des Graphen

Durch Anwendung der Transformationsregeln wird der Knoten (a) zu zwei Unterknoten bzw. zu zwei Zweigen expandiert:

$$(e) \neg(S \wedge O)$$

$$(f) R$$

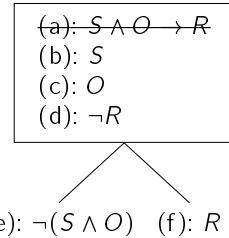


Abbildung 6.3: Graph nach Expansion des Wurzelknotens

Nach Bildung des Wurzelknotens wird der Baum anhand der Tiefensuche-Traversierung expandiert. Zuerst wird der Wurzelknoten {(a), (b), (c), (d)}, danach der linke (e) und schliesslich der rechte Teilbaum (f) durchlaufen.

Die Transformationsregeln werden auf den Wurzelknoten angewendet. Dabei wird der linke Teilbaum (e) durch einen neu gebildeten Teilbaum (bestehend aus (g) und (h)) ersetzt, der rechte Teilbaum (f) bleibt unverändert:

$$(g) \neg S$$

$$(h) \neg O$$

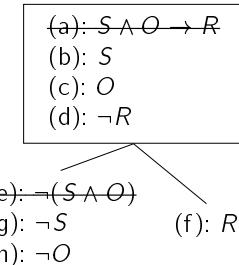


Abbildung 6.4: Graph nach Ersetzen der Teilbäume

Der linke Teilbaum ist somit vollständig expandiert und wird ausgewertet. Dabei werden die Knoten (b), (c), (d), (g) und (h) (logisch) vereint ⊕. Dabei ergibt sich, dass die Knoten (g) und (h) einen Widerspruch (F) mit den Knoten (b) und (c) bilden:

- (b) ⊕ (g) also $S \wedge \neg S = F$

und

- (c) ⊕ (h) also $O \wedge \neg O = F$

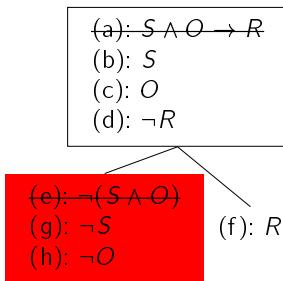


Abbildung 6.5: Graph mit markiertem Teilbaum nach Widerspruch

An dieser Stelle wird die Tiefensuche im linken Unterknoten abgebrochen, dafür im rechten Unterknoten fortgesetzt. Dieser ist bereits expandiert und wird ausgewertet. Dazu muss Knoten (f) mit den Knoten (b), (c) und (d) (logisch) vereinbar sein. Auch dort herrscht ein Widerspruch:

- (d) ⊥ (f) also $\neg R \wedge R = F$

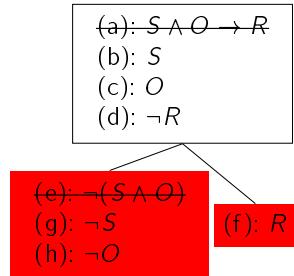


Abbildung 6.6: Graph mit markierten Teilbäumen nach Widersprüchen

Obwohl alle Knoten expandiert und ausgewertet wurden, konnte kein Modell für $\{S \wedge O \rightarrow R, S, O, \neg R\}$ gefunden werden. Dies wiederum bedeutet, dass die **Negation** $\Gamma \models \neg\varphi$ **nicht zutrifft**. Somit ist $\Gamma \models \varphi$ **bewiesen**.

Datentypenprüfung

In OWL werden Datentypen in einem XML-Schema beschrieben. Dadurch sind viele einfache Datentypen wie numerische Datentypen (Ganz- und Fließkommazahlen) und Zeichenketten gegeben. Zudem kann OWL eigene Datentypen definieren.

Das Modul zur Datentypenprüfung zeigt auf, ob die Schnittmenge von Datentypen konsistent ist. Eine Schnittmenge von Datentypen ist dann inkonsistent, wenn diese keine Elemente gemeinsam haben.

Der Tableau-Reasoner nutzt dieses Modul um zu prüfen, ob die Schnittmenge aller Datentypen für jeden Literal-Knoten des Graphens erfüllbar ist.

Schnittstelle zur Wissensdatenbank (KB)

Die Schnittstelle zur Wissensdatenbank entscheidet, wann die Konsistenz der Abox geprüft werden muss, wann alle Konzepte neu klassifiziert werden müssen und wann alle Individuen umgesetzt werden. Alle Aufgaben zum Schlussfolgern können unter geeigneter Umwandlung auf eine Prüfung der Konsistenz der Wissensdatenbank (KB) reduziert werden.

Die Schnittstelle bietet die Möglichkeit beliebige atomare Anfragen zu beantworten. Diese können Klassen, Eigenschaften oder Individuen betreffen. Wahrheitsabfragen werden in Erfüllbarkeitsprobleme umgewandelt.

Für alle Anfragen, welche mehrere Ergebnisse liefern, sind theoretisch mehrere Konsistenzprüfungen notwendig. Da dies aber sehr aufwändig ist, werden zur Optimierung im Unterabschnitt 6.4.3 beschriebene Verfahren eingesetzt.

Die Schnittstelle zur Wissensdatenbank, wie auch der Rest der Komponenten von Pellet, bauen auf der ATerm-Bibliothek¹¹ auf. ATerm (Annotated Term) ist ein abstrakter Datentyp, welcher für den Austausch von baumartigen Datenstrukturen zwischen verteilten Applikationen entwickelt wurde.

¹¹<https://strategoxt.org/Tools/ATermLibrary>

Abox Query-Engine

Die Schnittstelle zur Wissensdatenbank wird mit der Abox-Query-Engine verbunden. Diese beantwortet konjunktive (verknüpfte) Anfragen. Das Modul unterstützt in SPARQL oder RDQL¹² geschriebene Anfragen. Dabei bestehen jedoch gewisse Einschränkungen, Details siehe [Sirin et al., 2005, S. 10 - 11].

Die Abox-Query-Engine besteht im Grunde aus mehreren Query-Engines, welche Anfragen beantworten. Dabei gibt es eine zentrale Query-Engine, welche Anfragen vorverarbeitet und für die Beantwortung die entsprechende Query-Engine auswählt. Details zum Ablauf siehe [Sirin et al., 2005, S. 11].

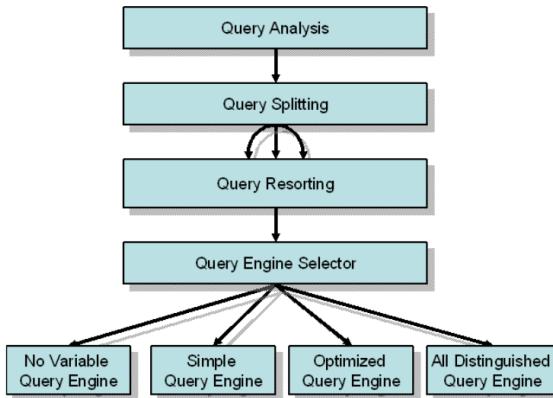


Abbildung 6.7: Ablauf der Beantwortung einer Anfrage in Abox-Query-Engine.¹³

Gültigkeitsprüfung bezüglich OWL-DL

Werkzeuge zur Modellierung und zum Export von Ontologien im OWL-Format bieten häufig Charakteristika von OWL-full zur Modellierung an. Für die Gültigkeitsprüfung einer Ontologie wird diese analysiert und gegebenenfalls mittels Heuristiken von OWL-full in OWL-DL umgewandelt.

Diese Umwandlung ist jedoch nur in gewissen Fällen möglich, so z.B. bei gleicher Bezeichnung von Klassen, Eigenschaften und Individuen, andernfalls werden die OWL-full Charakteristika ignoriert oder der Prozess wird abgebrochen.

6.4.2 Behandlung von Regeln in Pellet

Pellet ermöglicht die Verwendung von Regeln zur Schlussfolgerung unter Verwendung der SWRL-Regelsprache.

Damit in der Wissensdatenbank gespeicherte Konzepte in Regeln verfügbar sind, wird das \mathcal{AL} -Log-Framework eingesetzt. Dieses verbindet Beschreibungslogik mit Regeln. [Sirin et al., 2007, S. 4.5] \mathcal{AL} -Log ist ein integriertes System zur Repräsentation von Wissen, welches auf der Beschreibungslogik \mathcal{AL} und der deduktiven Datenbanksprache Datalog basiert. Donini et al. [1998]

Bei \mathcal{AL} handelt es sich um eine minimale Sprache der Beschreibungslogik, Details siehe [Baader et al., 2003, S. 51]. Als Implementation des \mathcal{AL} -Log-Frameworks nutzt Pellet einen Datalog-Reasoner. [Sirin et al., 2007, S. 4 - 5]

Bei Datalog handelt es sich um eine deklarative Logiksprache, in welcher jede Formel eine funktionsfreie Hornklausel ist. Im Unterschied zu Prolog muss jede Variable, die im Kopf einer Klausel vorkommt, auch im Körper der Klausel vorkommen. Die Reihenfolge der Klauseln spielt keine Rolle. Alle Anfragen terminieren und jede mögliche Antwort wird ausgegeben. Ramsdell [2004]

¹²<http://www.w3.org/Submission/RDQL/>

¹³[Sirin et al., 2005, S. 11]

In der Pellet-Implementation wird das \mathcal{AL} -Log-Framework dahingehend erweitert, dass es die $\mathcal{SHOIQ}(\mathcal{D})$ Variante der \mathcal{AL} -Beschreibungslogik nutzen kann. Zusätzlich erlaubt die Erweiterung die Verwendung von OWL-Datentypen und SWRL-Funktionen im Körper von Datalog-Regeln. [Sirin et al., 2007, S. 5]

Bei $\mathcal{SHOIQ}(\mathcal{D})$ handelt es sich um eine Erweiterung der Beschreibungslogik \mathcal{AL} . $\mathcal{SHOIQ}(\mathcal{D})$ unterstützt unter anderem zusätzliche Konzepte wie Rollenhierarchien, inverse Eigenschaften und qualifizierte Kardinalitätseinschränkungen. Diese Erweiterung erlaubt zusätzlich die Verwendung von Datentypen, Datenwerten und Datentypeneigenschaften. Wikipedia Foundation [2014c]

6.4.3 Optimierungen

Dieser Abschnitt basiert, sofern nicht anders im Text vermerkt, auf [Sirin et al., 2005, S. 16 - 19]

Beschreibungslogiken, wie $\mathcal{SHOIQ}(\mathcal{D})$ haben im ungünstigsten Fall eine sehr hohe Komplexität. Daher besteht ein grosser Unterschied zwischen Design und praktischer Umsetzung einer Entscheidungsprozedur.

Um dennoch akzeptable Leistungen bei Schlussfolgerungen mittels Beschreibungslogik zu erhalten, nutzen moderne Reasoner verschiedene Arten von Optimierungen, so z.B.:

- **Normalisierung und Vereinfachung**

Alle Konzepte der Wissensdatenbank werden in eine Form gebracht, die hilft, Widersprüche bei der Tableau-Erweiterung möglichst früh zu erkennen. Die Konzepte werden in die Normalform gebracht.

Diese Vereinfachung erkennt offensichtliche Fehler während der Normalisierung. Mehrfaches Vorkommen eines gleichen Konzeptes wird eliminiert.

- **Tbox-Absorption**

Bei dem Prozess der Tbox-Absorption wird ebenfalls versucht, gleichartige Konzepte (GCI — General Concept Inclusion) zu eliminieren, dies mittels Ersatz durch atomare Konzepte.

- **Dependency-directed Backjumping**

Der Prozess des dependency-directed Backjumpings eliminiert unproduktive Backtracking-Suchen, indem er Verzweigungspunkte identifiziert, welche Konflikte verursachen. Er springt zurück und überspringt dabei die Konfliktpunkte ohne nach Alternativen zu suchen.

Weitere Verfahren zur Optimierung sowie mehr Details finden sich unter [Sirin et al., 2005, S. 17 - 19].

6.4.4 Unterschiede zwischen Pellet und Prolog

Wie unter 6.4.2 erwähnt, nutzt Pellet einen Datalog-Reasoner als Implementation des \mathcal{AL} -Log-Frameworks. Der wohl wichtigste Unterschied zu Prolog ist daher, dass in Datalog alle Anfragen terminieren und jede mögliche Antwort ausgegeben wird. Eine Reihenfolge der Klauseln spielt daher keine Rolle. Im Gegensatz dazu ist eine Termination von Anfragen in Prolog nicht immer gegeben. So sind Endlosschleifen je nach Reihenfolge der Regeln durchaus möglich. [Lämmel and Cleve, 2012, S. 175]

Ein weiterer Unterschied ist die Art, wie Schlüsse gezogen werden. Prolog basiert auf dem SLD-Resolutionsverfahren [Details siehe Lämmel and Cleve, 2012, S. 68]. Im Gegensatz dazu kommt bei Pellet der unter 6.4.1 erklärte Tableau-Algorithmus zum Einsatz.

Durch den Aufbau von Prolog ist es möglich so genannte Constraint-Satisfaction-Probleme (Bedingungserfüllungsprobleme) zu lösen [Details siehe Lämmel and Cleve, 2012, S. 148]. Mit Pellet bzw. Ontologien und Regeln ist dies nicht der Fall. Versuche dazu finden sich den Publikationen von Xiong and Jiang [2008], Sleeman and Chalmers [2006] und Croitoru and Compatangelo [2007].



Das unter 6.4.1 genannte Beispiel beantwortet exakt die unter 4.3 gestellte Frage: Wie gelangt man zum Schluss, dass das Individuum *Seilpark Balmberg* in der *Region Solothurn* liegt.

Wie gelangt man effektiv zu dieser Information? Kann diese durch reine Folgerung erreicht werden?

Die Antwort hierzu lautet ja und nein. Modelliert man die Situation beispielsweise in Protégé und nutzt den Pellet-Reasoner für Schlussfolgerungen, so nimmt man an, dass er dies finden kann. Aufgrund seiner Möglichkeiten sollte der Reasoner dies beantworten können. Die nachfolgende Grafik zeigt jedoch, dass dies nicht immer der Fall ist.

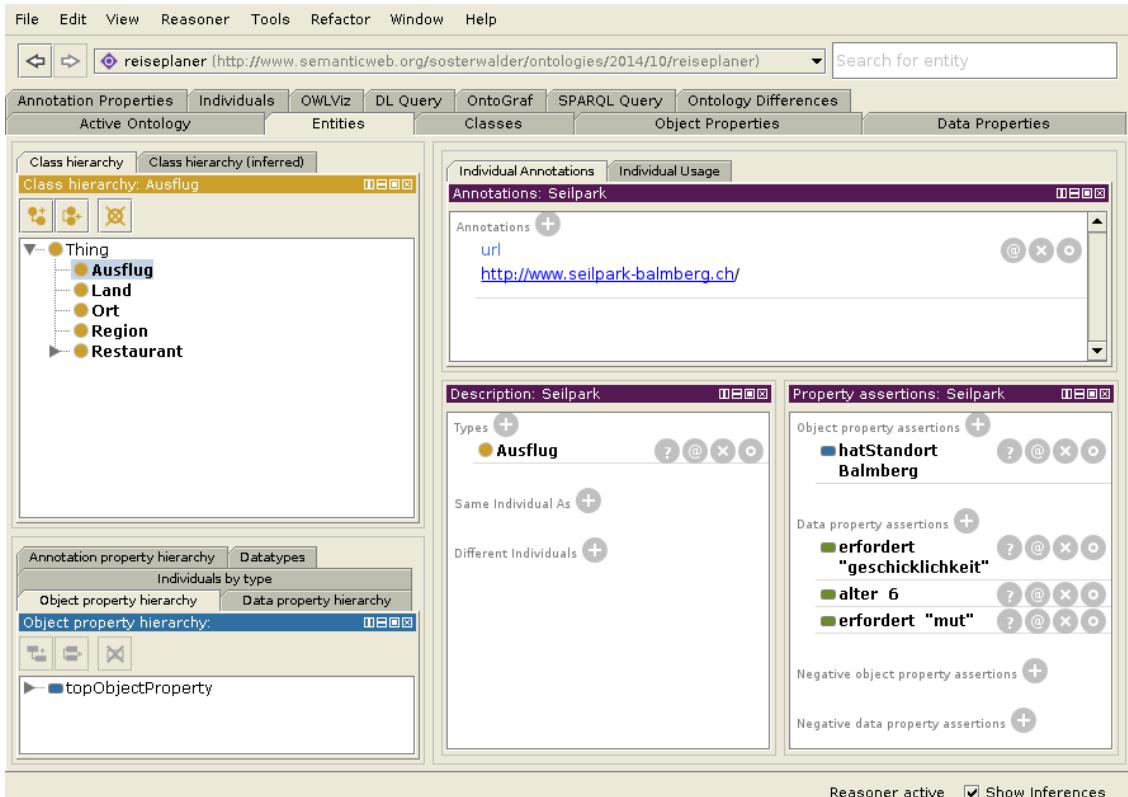


Abbildung 6.8: Darstellung des Individuums *Seilpark Balmberg* in Protégé.¹⁴

Wenn man den Relationen die entsprechenden Eigenschaften wie Symmetrie oder Transitivität gibt, ist diese Folgerung durchaus möglich. Bei unserem Beispiel haben wir die Eigenschaften bewusst weggelassen, da andernfalls irreführende Folgerungen auftreten. So wäre dann z.B. ein Ort auch ein Land und umgekehrt.

Die eigentliche Folgerung haben wir mittels einer Regel vorgenommen. Zu einem späteren Zeitpunkt erklären wir dies genauer, siehe Kapitel 9.

¹⁴Eigene Darstellung mittels Stanford Protégé Version 5.0.0 beta 15

7 RDF

Das folgende Kapitel basiert, sofern nicht anders vermerkt, auf Schreiber and Raimond [2014].

Die bisherigen Kapitel bieten einen Überblick über die verschiedenen Methoden und Angehensweisen des Knowledge-Engineerings. Die wichtigsten theoretischen Grundlagen wurden in den vorigen Kapiteln erklärt. Im Kapitel 5, "Wissen abbilden mittels Ontologien", wurde gezeigt, dass für die Darstellung von Ontologien verschiedene Sprachen entwickelt wurden.

Wir haben uns für OWL (in der Version 2) entschieden, da es sich um die am meisten benutzte Ontologiesprache handelt. Da OWL in der RDF-Syntax verfasst wird, wollen wir letztere erklären.

Das "Resource Description Framework" (RDF) ist ein Framework um Informationen aus Ressourcen zu formulieren. Als Ressourcen können Dokumente, Leute, Objekte aber auch abstrakte Inhalte dienen. Im Web können Informationen mit RDF verarbeitet werden, anstatt diese nur anzusehen. RDF bietet ein gemeinsames Framework um die Informationen zwischen Anwendungen auszutauschen ohne dabei die Bedeutung der Informationen zu verändern.

RDF ist die Grundlage des semantischen Webs, welches die Flexibilität von RDF vollumfänglich ausnutzt. Alle Daten im semantischen Web werden in RDF abgebildet. RDF ermöglicht die Verknüpfung von Daten. Dadurch werden für eine Ressource mehr Informationen zusammen gefügt. Cambridge Semantics INC (The Smart Data Company) [2014a]

7.1 RDF Data Model

In RDF werden Informationen als Aussagen abgebildet. Der Aufbau dieser ist immer gleich und weist die folgende Struktur eines Tripels auf:

|| <Subjekt> <Prädikat> <Objekt>
Auflistung 7.1: Tripel-Struktur einer RDF-Aussage

Eine RDF Aussage bildet eine Beziehung zwischen zwei Ressourcen (Entitäten) nämlich Subjekt und Objekt ab. Das Prädikat repräsentiert die Beziehung zwischen den zwei Ressourcen. Die Beziehung wird in RDF als Eigenschaft (Property) bezeichnet.

|| <Seilpark> <hatStandort> <Balmberg>
Auflistung 7.2: Beispiel einer RDF-Aussage

Eine Entität kann in mehreren Tripeln referenziert werden. Zudem ist es möglich die gleiche Ressource in einer Aussage als Objekt, in einer anderen Aussage als Subjekt zu verwenden. Damit werden Verbindungen zwischen mehreren Tripeln möglich. Dies ist ein wichtiger Aspekt von RDF.

Tripel werden in sogenannten RDF-Graphen abgebildet. Diese bestehen aus Knoten und Pfeilen. Subjekte und Objekte werden als Knoten, Prädikate als Pfeile dargestellt. Genaueres dazu findet sich im Kapitel 3, "Graphrepräsentationen".

Man unterscheidet drei Typen von RDF-Daten, welche in Tripeln auftreten: Ressourcen-Knoten (IRIs), leere Knoten (Blank) und Literale.

7.1.1 IRIs (International Resource Identifier)

Wie der Name besagt, stellt ein IRI eine Ressource dar. Dabei handelt es sich um einen globalen Identifier, IRIs können also von verschiedenen Nutzern wiederverwendet werden. Es gibt verschiedene Formen von IRIs. Zum Beispiel werden URLs als Web-Adresse verwendet. Eine andere Form der IRIs bietet eine Kennung einer Ressource ohne deren Standort oder Zugriff preiszugeben. IRIs können in allen drei Positionen eines Tripels auftreten.

Die genaue Spezifikation von IRIs findet sich unter RFC 3987.

7.1.2 Literale Knoten

Beim Begriff Literal handelt es sich um ein Synonym für Werte. Literale sind Basiswerte, die nicht IRIs entsprechen. Für die richtige Werteinterpretation wird den Literalen ein Datentyp zugeordnet. Dies können Strings, Datumswerte oder auch Nummern sein. Einem String kann zusätzlich eine Sprache zugewiesen werden.

Literale können in einem Tripel nur als Objekt verwendet werden.

7.1.3 Leere Knoten (blank nodes)

Ein leerer Knoten stellt eine Ressource ohne URI dar. Als Vorteil benötigen diese Knoten keinen globalen Identifier. Leere Knoten können mit einer einfachen Variablen in der Algebra verglichen werden. Sie bilden ein Objekt ab, wobei der Wert irrelevant ist.

Leere Knoten können in einem Tripel Subjekt oder Objekt darstellen.

7.2 Multiple Graphs

Eine der neuesten Erweiterungen von RDF sind multiple Graphen. Diese wurden eingeführt um Teilmengen einer Tripelsammlung zu definieren. Ursprünglich stammt dieser Mechanismus von der Abfragesprache SPARQL (siehe 10 SPARQL). Man unterscheidet zwischen benannten (named) und unbenannten (unnamed) Graphen. Bei unbenannten Graphen enthalten die Tripel jeweils die gesamte URI.

```
||      <http://example.org/bob> <is published by> <http://example.org>.  
Auflistung 7.3: Beispiel eines unbenannten (unnamed) Graphen
```

Bei benannten Graphen wird ein Identifikator (identifier) hinzugefügt, auf welchen referenziert wird.

```
Identifier:  
          http://example.org/bob  
Graph:  
          <Bob> <is a> <person>.  
          <Bob> <is a friend of> <Alice>.  
Auflistung 7.4: Beispiel eines benannten (named) Graphen
```

Multiple Graphen eines RDF-Dokumentes stellen eine Datenmenge dar. Sie bestehen standardmäßig aus einem unbenannten (unnamed) Graphen und mehreren benannten (named) Graphen. Dabei ist der unbenannte Graph der sogenannte Basisgraph (default).

7.3 RDF Vokabular

RDF wird typischerweise in Kombination mit einem Vokabular oder anderen Konventionen verwendet, welche semantische Informationen über verwendete Ressourcen bieten.

Um Vokabulare zu definieren, bietet RDF die RDF-Schema-Sprache. Erst diese ermöglicht es semantische Eigenschaften von RDF-Daten zu definieren. Damit kann beispielsweise festgelegt werden, welche Ressourcen an welcher Position eines Tripels verwendet werden sollen.

RDF verwendet den Klassen-Bezeichner (class) um Kategorien zu definieren, welche die Klassifizierung von Ressourcen erlauben. Die Beziehung zwischen einer Instanz und ihrer Klasse wird durch den Typen-Bezeichner (type) ausgedrückt.

Mit der RDF-Schema-Sprache können Hierarchien im Bereich der Klassen und Sub-Klassen sowie der Eigenschaften und Untereigenschaften gebildet werden. Auf Subjekten bzw. Objekten können Typeneinschränkungen mittels dem Domänen- (domain) bzw. dem Bereichs-Bezeichner (range) vorgenommen werden.¹

7.4 RDF Formen

RDF kann in verschiedenen Formen dargestellt werden. Sie alle führen zu den exakt selben Tripeln, sind also logisch äquivalent. Die Einsatzgebiete sind jedoch unterschiedlich. Formen von RDF sind: Turtle-Schreibweise (N-Triples, Turtle, TriG und N-Quads), JSON-LD, RDFa und RDF/XML.

Da wir letztere Schreibweise in unserem Beispiel verwendet haben, werden wir sie genauer vorstellen.

7.4.1 RDF/XML

Bei RDF/XML handelt es sich um eine Schreibweise von RDF, welche die XML-Syntax verwendet. Bei der Entwicklung von RDF in den 1990er Jahren war RDF/XML die einzige existierende Schreibweise.

In RDF/XML werden Tripel in dem XML-Element `rdf:RDF` spezifiziert. Das XML-Element `rdf:description` definiert eine Menge von Tripeln, welche als Subjekt den IRI ihres `about`-Attributes haben.

Ein Description-Element kann Unterelemente beinhalten. Der Name des Unterelementes ist ein IRI, welcher in der RDF-Eigenschaft `rdf:type` abgebildet ist. Dabei repräsentiert jedes Unterelement ein Tripel.

Handelt es sich bei dem Objekt eines Tripels auch um einen IRI, hat das Unterelement keinen Inhalt und der Objekt-IRI-Knoten wird durch das `rdf:resource`-Attribut beschrieben.

Ist das Objekt eines Tripels ein Literal, wird der Inhalt der RDF-Eigenschaft zum Literalwert.

Der Datentyp der RDF-Eigenschaft wird als Attribut angegeben. Bei Fehlen von Datentyp und Sprache wird angenommen, dass der Literal vom Datentyp "String" ist.

```
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
  <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.semanticweb.org/mira/ontologies/2014/9/FamilyOnto#"
           xml:base="http://www.semanticweb.org/mira/ontologies/2014/9/FamilyOnto"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:swrl="http://www.w3.org/2003/11/swrl#"
           xmlns:owl="http://www.w3.org/2002/07/owl#">
```

¹<http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/#section-rdfa> Schreiber and Raimond [2014]

```

    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
<owl:Ontology rdf:about="http://www.semanticweb.org/mira/ontologies
    /2014/9/FamilyOnto"/>
<rdf:Description>
    <rdf:type rdf:resource="#IndividualPropertyAtom"/>
    <swrl:propertyPredicate rdf:resource="http://www.semanticweb.org/mira/
        ontologies/2014/9/FamilyOnto#isAncestor"/>
    <swrl:argument1 rdf:resource="urn:swrl#a"/>
    <swrl:argument2 rdf:resource="urn:swrl#x"/>
</rdf:Description>
</rdf:RD>

```

Auflistung 7.5: Beispiel RDF-Elemente²

²Eigenes Beispiel

8 OWL

Im vorherigen Kapitel wurden Voraussetzungen geschaffen um die eigentliche Ontologiesprache, OWL zu analysieren. In diesem Kapitel, auf Hitzler et al. [2012] basierend, wird OWL näher betrachtet.

Bei OWL (Web Ontology Language) handelt es sich um eine Ontologiesprache für das semantische Web. Mit dieser Sprache können Ontologien beschrieben werden. Cambridge Semantics INC (The Smart Data Company) [2014b] Wie RDF wurde auch OWL nicht nur für die Anzeige sondern auch für die Verarbeitung von Informationen entwickelt.

Durch zusätzliches Vokabular, wie Beziehungen zwischen Klassen, und erweiterter formaler Semantik hat der Benutzer mehr Möglichkeiten als bei RDF/XML oder auch XML.

8.1 OWL Syntax

Für OWL stehen verschiedene Schreibweisen zur Verfügung, welche für verschiedene Anwendungen definiert wurden:

- *RDF/XML Syntax für OWL*

Entspricht der RDF /XML Syntax mit einer spezifischen Übersetzung für OWL-Konstrukte. Es ist die einzige Syntax, welche von allen OWL2-Werkzeugen unterstützt wird (OWL2 ist die aktuellste Version von OWL).

- *OWL/XML Syntax*

XML-Syntax für OWL.

- *Functional-Style Syntax*

Sie dient der Vereinfachung der ursprünglichen Spezifikation. Sie unterstützt Grundlagen zur Implementation von OWL2 durch APIs und Reasoners.

- *Turtle Syntax*

Bei der Turtle Syntax handelt es sich um eine textuelle Syntax, welche die Beschreibung eines RDF-Graphen in kompakter und natürlicher Form erlaubt.

- *Manchester Syntax*

Vereinfacht das Lesen und Verstehen für Leser ohne Kenntnisse der Prädikatenlogik.

Anmerkung: Es existieren Werkzeuge, welche eine Übersetzung zwischen den verschiedenen Schreibweisen zulassen.

Bei OWL handelt es sich nicht um eine Schemasprache. Im Gegensatz zu XML beschreibt OWL nicht, wie ein Dokument aufgebaut sein muss. So kann das Vorhandensein eines bestimmten Elementes nicht vorgeschrieben werden.

```
<#green-goblin>
    rel:enemyOf <#spiderman> ;
    a foaf:Person ;      # in the context of the Marvel universe
    foaf:name "Green Goblin".
```

Auflistung 8.1: Beispiel der Turtle Syntax¹

¹Beispiel von <http://www.w3.org/TR/turtle>

8.2 Wissen modellieren

OWL ist eine wissensbasierte Repräsentationssprache. Sie widerspiegelt das Wissen einer spezifischen Domäne. Dabei wird versucht eine Domäne mittels OWL so abzubilden, dass sie Teile des menschlichen Wissens entspricht. Eine Möglichkeit alle Aspekte des menschlichen Wissens abzubilden besteht aber nicht. Trotzdem kann OWL als potente Modellierungssprache bezeichnet werden. Das Ergebnis einer solchen Modellierung ist eine Ontologie, wie bereits ausgeführt.

Für die Wissensabbildung über OWL werden grundlegende Konzepte wie Axiome, Entitäten und Ausdrücke (Expressions) benutzt.

- *Axiom*

Grundlegende Aussage.

Grundaussagen oder Basispropositionen, wie "Abenteuerreisen sind Reisen" werden in OWL Axiome genannt. Es handelt sich um "Teile des Wissens", welche je nach Sachlage wahr oder falsch sein können. Dies ist ein essentieller Unterschied zu Entitäten und Ausdrücken.

- *Entitäten*

Elemente, welche konkrete Objekte aus der realen Welt abbilden.

Grundbestandteile einer Aussage werden Entitäten genannt, zum Beispiel Objekte wie "Abenteuerreise" und "Reise" oder Beziehungen wie "sind".

Konkrete Objekte werden dabei als Individuen (Individuals), Kategorien (für Individuen) als Klassen und Beziehungen (zwischen Individuen) als Eigenschaften (Properties) bezeichnet.

Beziehungen werden unterteilt in Relationen zwischen zwei Individuen (ObjectProperties), spezifischen Datenwerten von Individuen (DataProperties) und zusätzliche Informationen zu einem Objekt (AnnotationProperties).

- *Expression (Ausdruck)*

Ausdrücke sind Kombinationen aus Entitäten. Sie ermöglichen den Aufbau von komplexen Beschreibungen.

Entitäten, wie z.B. Klassen, können mit Hilfe eines Konstruktors kombiniert werden. Beispiel: "Abenteuerreise" und "Reise". Diese Kombination wird als ClassExpression abgebildet und ist eine neue Entität.

Ausdrücke (Ausdruckssprache) sind für Klassen sehr vielfältig, hingegen nicht für Eigenschaften.

```
||| EquivalentClasses(  
|||   :Restaurant  
|||   :Gaststätte  
||| )
```

Auflistung 8.2: Beispiel eines Ausdrucks: Zwei Klassen bedeuten das Gleiche

8.3 Die wichtigsten Elemente von OWL

8.3.1 Klassen, Subklassen und Individuen

Klassen werden dazu verwendet, um Individuen mit Gemeinsamkeiten zu gruppieren. Klassen stellen daher eine Menge von Individuen dar. Für die Abbildung der menschlichen Denkweise werden in der Modellierung Klassen verwendet, beispielsweise Reisen oder Abenteuerreisen.

Beispiel: *Abenteuerreise* ist (nach unserer Definition) eine Klasse.

```
||| <!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/  
|||   reiseplaner#Ausflug -->  
||| <owl:Class rdf:about="#reiseplaner:Ausflug"/>
```

Auflistung 8.3: Beispiel einer Klasse: Ausflüge

Subklassen Im obigen Abschnitt wurden die Klassen Reise sowie Abenteuerreise eingeführt. Der menschliche Leser weiss intuitiv, dass eine Abenteuerreise auch eine Reiseform ist.

In OWL stehen diese beiden Klassen jedoch nicht in Beziehung. Es handelt sich um zwei unterschiedliche Klassen mit unterschiedlichen Bezeichnungen. Um die Schlussfolgerung, eine Abenteuerreise ist auch eine Reise, zu erreichen, muss dies definiert werden. In OWL geschieht dies mit dem Subklassen-Axiom *subClassOf*.

Beispiel: *Landgasthaus* ist (nach unserer Definition) Subklasse von *Restaurant*.

```
<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#Landgasthaus -->
<owl:Class rdf:about="#reiseplaner;Landgasthaus">
    <rdfs:subClassOf rdf:resource="#reiseplaner;Restaurant"/>
</owl:Class>
```

Auflistung 8.4: Beispiel einer Subklasse

Nicht nur zur Darstellung von Abhängigkeiten werden Subklassen verwendet, sondern auch zur Modellierung von Klassenhierarchien. Mittels Subklassen werden allgemeine Beziehungen der Klassen abgebildet, so zum Beispiel die Relation "ein Landgasthaus ist ein Restaurant."

Bei **Individuen** handelt es sich um *Instanzen von Klassen*. Ein Individuum ist Mitglied der Menge einer Klasse und bildet daher deren Konzept ab. Individuen können gleichzeitig Mitglied von mehreren Klassen sein.

Beispiel: *Seilpark Balmberg* ist (nach unserer Definition) ein Individuum der Klasse *Ausflug*.

```
<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#Seilpark -->
<owl:NamedIndividual rdf:about="#reiseplaner;Seilpark">
    <rdf:type rdf:resource="#reiseplaner;Ausflug"/>
    <reiseplaner:alter rdf:datatype="xsd:integer">6</reiseplaner:alter>
    <reiseplaner:erfordert>mut</reiseplaner:erfordert>
    <reiseplaner:erfordert>geschicklichkeit</reiseplaner:erfordert>
    <reiseplaner:url>http://www.seilpark-balmberg.ch/</reiseplaner:url>
    <reiseplaner:hatStandort rdf:resource="#reiseplaner;Balmberg"/>
</owl:NamedIndividual>
```

Auflistung 8.5: Beispiel eines Individuums

Mittels der *owl:sameAs-Relation* kann ausgedrückt werden, dass es sich bei zwei oder mehreren Individuen um die gleichen Individuen handelt.

```
<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#Seilpark -->
<owl:NamedIndividual rdf:about="&reiseplaner;Seilpark">
  <rdf:type rdf:resource="&reiseplaner;Ausflug"/>
  ...
  <owl:sameAs rdf:resource="&reiseplaner;Seilpark_Balmberg"/>
</owl:NamedIndividual>
```

Auflistung 8.6: Beispiel einer Gleichstellung von Individuen: Seilpark und Seilpark-Balmberg sind das gleiche Individuum

Durch die *owl:AllDifferent-Relation* kann das Gegenteil ausgesagt werden: Zwei oder mehrere Individuen sind nicht die gleichen Individuen.

```
<rdf:Description>
  <rdf:type rdf:resource="&owl; AllDifferent"/>
  <owl:distinctMembers rdf:parseType="Collection">
    <rdf:Description rdf:about="&reiseplaner;Seilpark"/>
    <rdf:Description rdf:about="&reiseplaner;Seilpark_Pilatus"/>
  </owl:distinctMembers>
</rdf:Description>
```

Auflistung 8.7: Beispiel einer Differenzierung von Individuen: Seilpark ist nicht das gleiche Individuum wie Seilpark-Pilatus

Eigenschaften

Objekt-Eigenschaften (ObjectProperties)

Objekt-Eigenschaften beschreibt die Beziehung zwischen zwei Individuen.

```
<owl:ObjectProperty rdf:about="&reiseplaner;hatOrt">
  <rdfs:range rdf:resource="&reiseplaner;Ort"/>
  <rdfs:domain rdf:resource="&reiseplaner;Region"/>
</owl:ObjectProperty>
...
<owl:Thing rdf:about="&reiseplaner;Bern">
  <rdf:type rdf:resource="&owl; NamedIndividual"/>
  <reiseplaner:distanzZuAusgangpunkt rdf:datatype="&xsd; integer">0</
    <reiseplaner:distanzZuAusgangpunkt>
  <reiseplaner:hatOrt rdf:resource="&reiseplaner;Bern"/>
  <reiseplaner:hatOrt rdf:resource="&reiseplaner;Ersigen"/>
</owl:Thing>
```

Auflistung 8.8: Beispiel einer Objekteigenschaft *hatOrt* und deren Anwendung

Durch Auswahl von Klassen kann eingeschränkt werden, auf welche Individuen eine Objekt-Eigenschaft angewendet werden darf. Dies gilt sowohl für die Quelle einer Beziehung (Domäne, Domains) als auch für das Ziel einer Beziehung (Wertebereich, Ranges).

```
<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#hatRegion -->
<owl:ObjectProperty rdf:about="&reiseplaner;hatRegion">
  <rdfs:domain rdf:resource="&reiseplaner;Land"/>
  <rdfs:range rdf:resource="&reiseplaner;Region"/>
</owl:ObjectProperty>
```

Auflistung 8.9: Beispiel von Einschränkungen der Objekteigenschaft *hatRegion*

Subeigenschaften

Analog zu Subklassen lassen sich Untereigenschaften (Subproperties) definieren.

```

<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#hatGemeinde -->
<owl: ObjectProperty rdf:about="&reiseplaner; hatGemeinde">
    <rdfs:subPropertyOf rdf:resource="&reiseplaner; hatRegion"/>
</owl: ObjectProperty >
```

Auflistung 8.10: Beispiel der Objekteigenschaft *hatGemeinde* als Subeigenschaft von *hatRegion*

Datentypen-Eigenschaften (DataProperties)

Datentypen-Eigenschaften definieren und beschreiben einen spezifischen Datenwert eines Individuums. Dies können beispielsweise das Alter oder die Größe einer Person sein. Datentypen-Eigenschaft können frei definiert werden und, bei Bedarf an vordefinierte Wertetypen gebunden werden. Eine Auflistung vordefinierter Wertetypen findet sich unter der Webseite w3.org².

```

<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#anzahlTeilnehmer -->
<owl: DatatypeProperty rdf:about="&reiseplaner; anzahlTeilnehmer">
    <rdfs:range rdf:resource="&xsd; integer"/>
    <rdfs:subPropertyOf rdf:resource="&owl; topDataProperty"/>
</owl: DatatypeProperty >
...
<!-- http://www.semanticweb.org/sosterwalder/ontologies/2014/10/
     reiseplaner#AdventureRooms -->
<owl: NamedIndividual rdf:about="&reiseplaner; AdventureRooms">
    ...
        <reiseplaner:anzahlTeilnehmer rdf:datatype="&xsd; integer">12</
        reiseplaner:anzahlTeilnehmer >
    ...
</owl: NamedIndividual >
```

Auflistung 8.11: Beispiel der Datentypen-Eigenschaft *anzahlTeilnehmer* und deren Anwendung bei einem Individuum

²http://www.w3.org/TR/owl2-syntax/#Datatype_Maps

8.4 Ontologien

Wissensdomänen zu einem Thema werden in Ontologien abgebildet. Diese können für verschiedenen Anwendungen genutzt werden.

```
<owl:Ontology rdf:about="http://www.semanticweb.org/sosterwalder/ontologies/2014/10/reiseplaner"/>
```

Auflistung 8.12: Beispiel einer Definition einer Ontologie

Ontologien werden als OWL-Dokumente abgespeichert. Dabei steht es dem Autor frei, sie im Internet zur Verfügung zu stellen. Namespaces können zusätzlich beim Schreiben einer Ontologie verwendet werden. Sie dienen der Unterscheidbarkeit von Ontologien. Beispiel: "*meineOntologie#*". Um eine Ontologie anhand deren Namespace benutzt zu können, wird ein Präfix definiert und dieser einer Ontologie zugewiesen. Beispiel: "PREFIX hallo: <*meineOntologie#*>". Somit können die Elemente der Ontologie mit dem Präfix angesprochen werden. Beispiel: "hallo:objekt1DerOntologie".

Informationen aus einer beliebigen Ontologie können in einer anderen verwendet werden. Dafür ist ein Import der Ontologie nötig.

```
<owl:Ontology rdf:about="http://example.com/owl/families">
  <owl:imports rdf:resource="http://example.org/otherOntologies/families.owl" />
</owl:Ontology>
```

Auflistung 8.13: Beispiel eines Importes einer Ontologie

Um bei der Verwendung von Objekten aus anderen Ontologien eine Umbenennung zu umgehen, kann direkt auf das entsprechende Objekt referenziert werden.

```
<owl:DatatypeProperty rdf:about="hasAge">
  <owl:equivalentProperty rdf:resource="\&otherOnt;age"/>
</owl:DatatypeProperty>
```

Auflistung 8.14: Beispiel einer Referenzierung auf ein Objekt einer externen Ontologie

8.5 OWL Untersprachen

OWL ist in drei Untersprachen aufgeteilt: OWL Lite, OWL DL und OWL full. Jede dieser Sprachen wurde für verschiedene Anwendergruppen entwickelt.

Die Version 2 von OWL unterscheidet zusätzlich zwischen OWL2 QL, OWL2 EL und OWL2 RL. Dies sind weitere Unterklassen von OWL2 DL, welche ihrerseits wiederum eine Untersprache von OWL2 full ist.

Genauere Informationen über die einzelnen Untersprachen finden sich unter der Webseite w3.org/TR/2004/REC-owl-features³, über OWL2-Profiles unter w3.org/TR/owl2-profiles⁴

Die drei wichtigsten Unterklassen werden nachfolgend kurz vorgestellt.

- *OWL Lite*

Bietet primär eine Hierarchie zur Klassifikation sowie einfache Bedingungen, so wird zwar z.B. Kardinalität geboten, jedoch nur mit den Werten 0 und 1. Hitzler et al. [2012]

- *OWL DL*

Bietet das Maximum an Ausdrucksmöglichkeiten unter Einhaltung der Vollständigkeit (alle Folgebeziehungen werden berücksichtigt und miteinbezogen) und der Entscheidbarkeit (alle Berechnungen enden nach endlicher Zeit). Hitzler et al. [2012]

³<http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3>

⁴<http://www.w3.org/TR/owl2-profiles/>

- *OWL full*

“Bietet das Maximum an Ausdrucksmöglichkeiten und die syntaktische Freiheit von RDF aber ohne Garantien betreffend Vollständigkeit und Entscheidbarkeit.” Hitzler et al. [2012]

8.6 OWL-Werkzeuge

Typischerweise unterscheidet man zwischen zwei Arten von Ontologiewerkzeugen, einem Editor und einem Reasoner. Der Editor ermöglicht das Erstellen und Ändern von Ontologien. Der Reasoner leitet logische Folgerungen aus dem bestehenden Wissen ab.



Die Ontologiesprachen erlauben Ontologien zu modellieren. Für die Formulierung und das Lesen aber sind sie umständlich. Mit Hilfe von Ontologiewerkzeugen, zum Beispiel einem Editor, lassen sich Ontologien intuitiv und übersichtlich modellieren.

In unserem Beispiel haben wir den in der Fachwelt häufig verwendeten Editor Protégé der Universität Stanford benutzt.

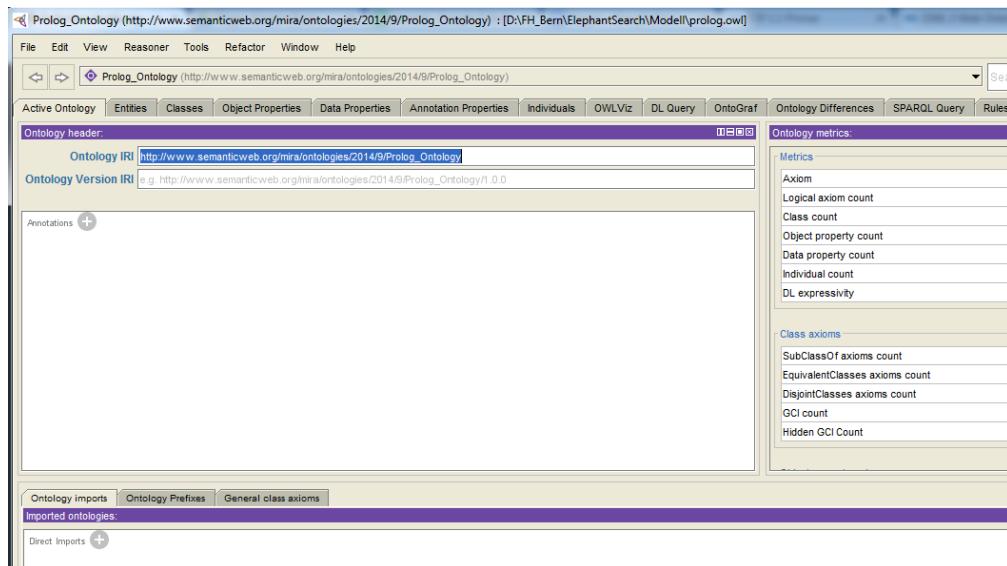


Abbildung 8.1: Übersichtsfenster des Protégé-Editors⁵

⁵Eigens erstellter Screenshot von Protégé

9 Regeln — SWRL

Das vorangehende Kapitel 8, "OWL" hat gezeigt, wie mittels der Ontologiesprache OWL in der Version 2 eine Ontologie erstellt werden kann. Weiter zeigte das Kapitel wie innerhalb einer Ontologie Klassen, Individuen und Beziehungen aufgebaut werden können. Beim Erstellen solch einer Ontologie fällt auf, dass der Nutzen der Inferenz mit den beschriebenen Entitäten nicht voll ausgeschöpft werden kann. Damit ein Reasoner die Inferenz vollenfänglich nutzen kann, mussten wir Regeln in Form der Regelsprache *SWRL* einführen.

Das folgende Kapitel basiert, sofern nicht anders vermerkt, auf Horrocks et al. [2004].

Bei SWRL handelt es sich um eine auf OWL und RuleML¹ basierende Regelsprache. Sie erlaubt es Regeln in Form von OWL-Konzepten auszudrücken und bietet dadurch vielfältige Möglichkeiten der Inferenz.

9.1 Aufbau

Eine SWRL-Regel besteht aus einem Kopf und einem Körper. Hierbei folgt der Kopf immer auf den Körper. Beide bestehen aus positiven Konjunktionen von Atomen:

|| $atom \wedge atom \dots \rightarrow atom \wedge atom$

Auflistung 9.1: Beispiel von positiven Konjunktionen von Atomen

Informal ausgedrückt heisst das: "Wenn alle Teile des Körpers wahr sind, dann ist auch der Kopf wahr". SWRL unterstützt keine negierten Atome oder Disjunktionen.

Ein Atom ist Ausdruck der Form:

|| $p(arg1, arg2, \dots argn)$

Auflistung 9.2: Beispiel eines Atoms

p ist hierbei ein Prädikat, $arg1$, $arg2$ und $argn$ sind die Argumente bzw. Terme des Ausdrucks.

Prädikate können OWL-Klassen, -Eigenschaften oder -Datentypen sein. Argumente können OWL-Individuen, -Werte oder darauf verweisende Variablen sein.



In SRWL können Atome ausschliesslich mit UND verknüpft werden. Eine ODER-Verknüpfung gibt es nicht. Die UND-Verknüpfung wird mittels eines Kommas , vorgenommen. Eine ODER-Verknüpfung wird durch mehrere Regeln mit unterschiedlichen Regelkörpern, aber gleichen Regelköpfen (Schlussfolgerungen) abgebildet.

9.1.1 Atomare Typen

SWRL unterscheidet zwischen den folgenden atomaren Typen:

- Klassen
- Eigenschaften von Individuen
- Wertespezifischen Eigenschaften
- Unterscheidung von Individuen

¹www.ruleml.org

- Gleichsetzungen von Individuen
- Wertebereichen
- Vordefinierten Atomen

Klassen

Ein Klassenatom besteht aus einem Prädikat, sowie nur einem Argument. Das Prädikat ist eine OWL-Klasse, das Argument ein OWL-Individuum oder eine Variable.

```
||    Abenteuerreise(?x)
      Ausflug(Seilpark_Balmberg)
```

Auflistung 9.3: Beispiele von Klassen-Atomen

Möchte man nun mittels SWRL z.B. aussagen, dass alle Abenteuerreisen auch Reisen sind, so geschieht dies mittels:

```
||    Abenteuerreise(?x) → Reise(?x)
```

Auflistung 9.4: Beispiel der SWRL-Regel, dass alle Abenteuerreisen auch Reisen sind

Eigenschaften von Individuen

Ein Atom, welches Eigenschaften von Individuen darstellt, besteht aus einem Prädikat und zwei Argumenten. Das Prädikat ist eine OWL-Eigenschaft bzw. eine Relation zwischen zwei Individuen. Die Argumente sind somit ein OWL-Individuum oder eine Variable.

```
||    anzahlTeilnehmer(?a, ?anzahl)
      erfordert(?a, "mut")
```

Auflistung 9.5: Beispiele von Atomen zur Darstellung von Eigenschaften von Individuen

Wir definieren als praktisches Beispiel ein Individuum als *Teamevent*, wenn es mindestens 4 maximal aber 20 Teilnehmer zulässt. Weiter muss es über die Eigenschaft der Teambildung verfügen. Drückt man dies per SWRL-Regel aus, ergibt sich folgender Ausdruck:

```
||    anzahlTeilnehmer(?a, ?anzahl), greaterThanOrEqual(?anzahl, 4), lessThanOrEqual(?anzahl, 20),
      bietet(?a, "teambildung") → teamevent(?a, true)
```

Auflistung 9.6: Beispiel der SWRL-Regel um ein Objekt *?a* als Teamevent zu definieren

Wertespezifische Eigenschaften

Ein Atom, welches wertespezifische Eigenschaften darstellt, besteht aus einem Prädikat sowie zwei Argumenten. Das Prädikat ist eine wertespezifische Eigenschaft. Das erste Argument ein OWL-Individuum, das zweite Argument ist ein OWL-Datenwert.

```
||    erfordert(?a, "mut")
      durchschnittspreis(?r, ?preis)
```

Auflistung 9.7: Beispiele von Atomen zur Darstellung von wertespezifischen Eigenschaften von Individuen

Dass ein Objekt als aufregend (actionreich) gilt, wenn es Mut erfordert, kann mit folgender Regel beschrieben werden:

```
||    erfordert(?a, "mut") → action(?a, true)
```

Auflistung 9.8: Beispiel einer Regel, welche ausdrückt, dass ein Objekt actionreich ist, wenn es Mut erfordert

Unterscheidung von Individuen

Ein Atom, welches die Unterscheidung von Individuen bezeichnet, besteht aus einem Prädikat sowie zwei Argumenten. Das Prädikat ist das *differentFrom*-Prädikat, die Argumente sind OWL-Individuen.

```
|| differentFrom(?x,?y)  
|| differentFrom(Seilpark_Balmberg,Seilpark_Pilatus)
```

Auflistung 9.9: Beispiele von Atomen zur Unterscheidung von zwei Individuen

Eine genauere Beschreibung der Verwendung bzw. des Sinnes dieses Atoms findet sich unter 9.2.

Gleichsetzung von Individuen

Das Atom, welches die Gleichheit von zwei Individuen kennzeichnet, besteht aus einem Prädikat sowie zwei Argumenten. Das Prädikat ist das *sameAs*-Prädikat, die Argumente sind OWL-Individuen.

```
|| sameAs(?x,?y)  
|| sameAs(Seilpark, Seilpark_Balmberg)
```

Auflistung 9.10: Beispiele von Atomen zur Gleichsetzung von zwei Individuen

Wertebereiche

Das Atom, welches einen Wertebereich kennzeichnet, besteht aus einem Datentyp oder einer Menge von Literalen sowie einem Argument.

```
|| xsd:int(?x)  
|| [3, 4, 5](?x)
```

Auflistung 9.11: Beispiele von Atomen zur Kennzeichnung eines Wertes

Im ersten Beispiel ist das Objekt, gekennzeichnet durch die Variable ?x, eine ganzzahlige Zahl.

Im zweiten Beispiel hat das Objekt einen der Werte 3, 4 oder 5.

Vordefinierte Atome

Bei den vordefinierten Atomen handelt es sich um Prädikate, welche ein oder mehrere Argumente entgegennehmen. Sofern alle Argumente dem Prädikat genügen, geben sie den Wahrheitswert "richtig" zurück. Andernfalls "falsch".

Die vordefinierten Atome haben das Präfix *swrl*db. Eine detaillierte Beschreibung aller vordefinierter Atome findet sich unter der Webseite [daml.org](http://www.daml.org)².

Möchte man ausdrücken, dass zum Beispiel Objekte, welche einen Durchschnittspreis zwischen 20 und 30 haben, *preiswert* (wertespezifische Eigenschaft des Attributes *preissegment*) sind, so geschieht dies mittels der SWRL-Regel:

```
|| durchschnittspreis(?r, ?preis), lessThanOrEqual(?preis, 30), greaterThan(?preis, 20) →  
|| preissegment(?r, "preiswert")
```

Auflistung 9.12: Beispiel einer SWRL-Regel, welche besagt, dass ein Objekt die wertespezifische Eingenschaft *preiswert* hat

²<http://www.daml.org/2004/04/swrl/builtins>

9.2 Open world assumption

Wie auch OWL geht SWRL von der so genannten “open world assumption” aus. Diese besagt, der Wahrheitswert einer Aussage kann unabhängig des effektiven Wahrheitwertes wahr sein.

Möchte man zum Beispiel ausdrücken, zwei OWL-Individuen sind Kollaboratoren (sie arbeiten zusammen an einer Publikation) gilt folgende Regel:

|| $Publication(?p), hasAuthor(?p,?y), hasAuthor(?p,?z) \rightarrow collaboratesWith(?y,?z)$

Auflistung 9.13: Beispiel einer SWRL-Regel zum Ausdrücken von Kollaboration

Aufgrund der “open world assumption” von OWL ist es nicht möglich auszusagen, dass zwei OWL-Individuen automatisch unterschiedlich sind, wenn sie verschiedene Namen haben. Mittels den *sameAs*- und *differentFrom*-Relationen bietet SWRL die Lösung des Problemes:

|| $Publication(?p), hasAuthor(?p,?y), hasAuthor(?p,?z), differentFrom(?y,?z) \rightarrow collaboratesWith(?y,?z)$

Auflistung 9.14: Beispiel einer SWRL-Regel zum Ausdrücken von Kollaboration mit expliziter Unterscheidung zwischen den Kollaborateuren

Analog gilt dies für die Gleichsetzung von zwei Individuen. Diese müssen explizit mit der *sameAs*-Relation gleichgesetzt werden.

Aufzählungen können beispielsweise nicht ohne Umstände ausgedrückt werden: Es ist nicht möglich zu sagen, ob eine Publikation nur von einem einzigen Autor stammt. Es kann durchaus sein, dass nur ein Autor der Publikation bekannt ist, diese aber noch über weitere Autoren verfügt.

Dies bezeichnet den Kern der “open world assumption”. Der effektive Wahrheitswert ist beispielsweise, die Publikation $?p$ hat mehrere Autoren ($?x, ?y$ und $?z$). Die Aussage, dass die Publikation $?p$ einen bestimmten Autor $?z$ hat, ist dennoch wahr, auch wenn die übrigen Autoren noch nicht bekannt sind.



In diesem Kapitel wurden *Regeln* eingeführt, die es ermöglichen die im Kapitel 3 (Graphrepräsentationen) gefundenen Kriterien *familienfreundlich*, *regional* und *actionreich* als Schlussfolgerungen zu definieren. Vorgängig müssen diese jedoch als Attribute bzw. DataProperties (spezifischer Datenwert eines Objektes) definiert werden.

Wir haben die Definition wie folgt:

Ein Objekt muss über das Attribut *alter* verfügen und der Wert dieses Attributes muss grösser oder gleich der Zahl zwei sein. Ist dies gegeben, erhält ein Objekt automatisch das Attribut *familienfreundlich* mit dem Wahrheitswert (Boolean) *wahr* bzw. *true*.

|| $alter(?a, ?alter), swrlDb:greaterThanOrEqual(?alter, 2) \Rightarrow familienfreundlich(?a, true)$

Auflistung 9.15: Beispiel der SWRL-Regel für die Eigenschaft *familienfreundlich*

Ein Objekt gilt dann als *regional*, wenn es über einen Standort verfügt und die Region dieses Standortes 50 oder weniger Kilometer vom Ausgangspunkt (welcher dynamisch bestimmt sein kann, also abhängig vom Abfragesteller) entfernt ist.

|| $hatStandort(?ausflug, ?ort), hatRegion(?land, ?region), hatOrt(?region, ?ort), distanzZuAusgangspunkt(?region, ?distanz), swrlDb:lessThanOrEqual(?distanz, 50) \Rightarrow regional(?ausflug, true)$

Auflistung 9.16: Beispiel der SWRL-Regel für die Eigenschaft *regional*

Eine Regel kann in SWRL keine ODER-Verknüpfung enthalten, wie in Abschnitt 9.1 (Aufbau) beschrieben. Die ODER-Verknüpfungen bildet man ab, durch mehrfache Definition von Regeln für die selbe Schlussfolgerung. Der Regelkopf (das heisst die Schlussfolgerung) muss immer gleich sein, die Bedingungen (das heisst die Regelkörper) dürfen verschieden sein.

Wir wenden an: Ein Objekt erhält das Attribut *actionreich*, wenn es Mut oder Geschicklichkeit erfordert oder wenn es Nervenkitzel verursacht. Die Eigenschaft *nervenkitzel* ist ihrerseits eine Folgerung aus der Erfordernis von Mut.

Es findet also eine transitive Ableitung statt: $A \sim B \wedge B \sim C \Rightarrow A \sim C$ bzw. $\text{erfordertMut} \Rightarrow \text{birgtNervenkitzel} \wedge \text{birgtNervenkitzel} \Rightarrow \text{istActionreich}$ somit gilt $\text{erfordertMut} \Rightarrow \text{istActionreich}$.

```
||  erfordert(?a, "mut") => actionreich(?a, true)
    nervenkitzel(?a, true) => actionreich(?a, true)
    erfordert(?a, "geschicklichkeit") => actionreich(?a, true)
```

Auflistung 9.17: Beispiel der SWRL-Regel für die Eigenschaft *actionreich*

Es sind jetzt alle Komponenten vorhanden, um Ausflüge abilden zu können, die den in Kapitel 3 (Graphrepräsentationen) genannten Kriterien entsprechen.

Um nach Ausflügen mit den gewählten Kriterien suchen zu können, benötigt man jedoch eine bestimmte Abfragegespräche. Diese wird im kommenden Kapitel aufgezeigt.

10 SPARQL

Im letzten Kapitel wurde aufgezeigt, wie Inferenz in einer Ontologie mittels der Regelsprache SWRL vollumfänglich genutzt werden kann.

Wie können Informationen aus der Ontologie und den darauf basierenden Inferenzen abgefragt werden? Mit SPARQL — einer Abfragesprache — ist es möglich.

Im folgenden Kapitel, welches, sofern nicht anders vermerkt, auf W3C SPARQL Working group and Harris and Seaborne [2013] basiert, stellen wir diese Abfragesprache vor.

Bei SPARQL handelt es sich um eine Abfragesprache für RDF. Sie erlaubt es, Abfragen in mehreren Datenquellen vorzunehmen. Dabei werden Anfragen über Graphen vollzogen, auch entlang derer Konjunktionen und Disjunktionen. SPARQL unterstützt weiter Aggregation, Unterabfragen, Negation sowie die Nutzung von Ausdrücken als Werte.

Resultate sind entweder eine Menge von Ergebnissen oder RDF-Graphen.



Kennt man die Abfragesprache SQL, so erscheint SPARQL auf den ersten Blick recht ähnlich — der Name lässt dies bereits vermuten. Dies ist nur teilweise der Fall. Klauseln wie *SELECT* und *WHERE* sind ähnlich. In SPARQL gibt es jedoch keine *FROM*-Klausel.

Der Hauptunterschied zwischen SPARQL und SQL ist jedoch die Verwendung von Variablen. Diese werden in SPARQL in der Regel bei jeder Abfrage verwendet. Abfragen ohne Variablen kommen in der Praxis (gemäß unserer Erfahrung) eher selten vor.

SQL enthält in der Regel sehr wenig Variablen.

Eine SPARQL-Abfrage besteht aus folgenden Komponenten:

- Einem oder mehreren Namespaces
- Variablen
- Einem (Teil-) Graphen
- Einer Menge von Gruppierungen und Aggregationen
- Einer Menge von Modifikatoren
- Abfragearten
- Ausdrücken und Wertevergleichen



Für die Praxis besteht eine SPARQL-Abfrage in der Regel aus *Namespaces*, einer *SELECT*-Klausel, einer *WHERE*-Klausel sowie *Modifikatoren*.

10.1 Beispiel einer SPARQL Abfrage

Gegeben sei die folgende Datenbasis:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL  
Tutorial".
```

Auflistung 10.1: Einfache Datenbasis direkt in SPARQL¹

Die Datenbasis beinhaltet das Objekt *book1* mit dem Attribut *title*, welches den Wert "SPARQL Tutorial" enthält.

Möchte ich nun den Titel des Buches abfragen, so kann ich folgende Abfrage verwenden:

```
SELECT  
    ?title  
WHERE {  
    <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
```

Auflistung 10.2: Beispiel einer einfachen SPARQL-Abfrage²

Sie ergibt folgendes Resultat:

title
"SPARQL Tutorial"

Tabelle 10.1: Resultat einer einfachen SPARQL-Abfrage³

10.2 Namespaces

SPARQL ist eine Abfragesprach für RDF, das seinerseits auf XML basiert. Dadurch sind XML-Namespace auch in SPARQL verfügbar. Bei Namespaces handelt es sich um Referenzen auf andere XML-basierte Dokumente, welche modular genutzt werden können.

Ein Namespace besteht aus einem Kürzel (welches auch leer sein kann), sowie aus der kompletten Adresse der Ressource (vgl. [Beckett, 2013, 2.1 Introduction]).

```
< rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns:dc="http://purl.org/dc/elements/1.1/"  
    xmlns:ex="http://example.org/stuff/1.0/">
```

Auflistung 10.3: Beispiel von Namespaces in RDF⁴

10.3 Variablen

Bei Abfragen unterstützt SPARQL die Verwendung von Variablen. Diese können dabei an einer beliebigen Stelle, wie bei (Teil-) Graphen, Gruppierungen, Aggregationen und Modifikatoren eingesetzt werden.

Die Variablen dienen als Platzhalter für abzufragende Objekte, Subjekte und Prädikate. Variablen beginnen entweder mit einem Dollarzeichen \$ oder einem Fragezeichen ?, welche selbst nicht Teil der Variable sind.

¹[W3C SPARQL Working group and Harris and Seaborne, 2013, 2.1 Writing a Simple Query]

²[W3C SPARQL Working group and Harris and Seaborne, 2013, 2.1 Writing a Simple Query]

³[W3C SPARQL Working group and Harris and Seaborne, 2013, 2.1 Writing a Simple Query]

⁴[Beckett, 2013, 2.6 Completing the Document: Document Element and XML Declaration]

```

||| SELECT DISTINCT
|||   *
||| WHERE {
|||   ?object ?predicate ?subject
||| }
||| LIMIT
|||   10

```

Auflistung 10.4: Beispiel einer SPARQL Abfrage mit den Variablen `?object`, `?predicate` und `?subject`.

Wird obige Anfrage auf einen SPARQL-Endpunkt, wie z.B. <http://dbpedia.org/>, angewendet, so ergibt dies folgendes Resultat:

object	predicate	subject
http://dbpedia.org/ontology/acceleration	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/torqueOutput	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/birthDate	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/birthYear	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/deathDate	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/deathYear	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/diameter	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/displacement	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/height	... rdf-syntax-ns#type	... owl#FunctionalProperty
http://dbpedia.org/ontology/latestReleaseDate	... rdf-syntax-ns#type	... owl#FunctionalProperty

Tabelle 10.2: Resultat einer Abfrage des DBpedia-Endpunktes limitiert auf 10 Ergebnisse.

10.4 (Teil-) Graphen

SPARQL basiert auf dem Matching eines (Teil-) Graphen in einem gegebenen Graphen. Dabei ist ein (Teil-) Graph eine Menge von Tripeln.

Man unterscheidet zwischen den folgenden Arten von Teilgraphen:

- *Grundlegende Teilgraphen*

Hierbei muss eine Menge von Tripeln derjenigen im gegebenen Graphen entsprechen

- *Gruppierende Teilgraphen*

Hierbei müssen *alle* Elemente einer Menge von Tripeln von grundlegenden Teilgraphen wiederum derjenigen im gegebenen Graphen entsprechen

- *Optionale Teilgraphen*

Zusätzliche Teilgraphen können die Lösungsmenge erweitern

- *Alternative Teilgraphen*

Versuch, ob zwei oder mehrere mögliche Teilgraphen denjenigen im gegebenen Graphen entsprechen

- *Teilgraphen in namensbasierten Graphen*

Versuch, ob Teilgraphen denjenigen im gegebenen, namensbasierten Graphen entsprechen

Ein grundlegender Teilgraph ist somit eine Menge von Tripeln, wobei jedes Tripel gemäss RDF-Spezifikation immer aus Subjekt, Prädikat und Objekt bestehen muss (vgl. [Schreiber and Raimond, 2014, 3.1 Triples]).

```
|| ?subject ?predicate ?object.
```

Auflistung 10.5: Beispiel eines grundlegenden Teilgraphen in SPARQL



Ein Teilgraph wird in SPARQL immer zur Gruppierung einer Abfrage verwendet. Dies entspricht in der SQL-Sprache der *WHERE*-Klausel.

Ein Teilgraph wird immer mit dem Punkt-Operator beendet.

Ein gruppierender Teilgraph wird in der SPARQL-Abfrage immer mit geschweiften Klammern `{}` umschlossen. Es ist möglich die Resultate einer Gruppierung zu filtern. Dazu wird *FILTER* als Schlüsselwort verwendet. Dieses bezieht sich immer auf die Gruppierung, in welcher es steht.

```
|| {
    ?x foaf:name ?name.
    ?x foaf:mbox ?mbox.
    FILTER regex(?name, "Smith")
}
```

Auflistung 10.6: Beispiel eines gruppierenden Teilgraphen mit dem *FILTER*-Schlüsselwort⁵

Im obigem Beispiel werden alle Objekte und die Werte ihrer Attribute (*name* und *mbox*, welche im Namespace *foaf* bezeichnet werden) abgefragt.

Dabei kann eine komplette leere Menge — wenn keine Objekte vorhanden sind — oder aber eine leere Teilmenge — wenn ein Objekt z.B. nicht über die beiden Attribute *name* und *mbox* verfügt — zurückgegeben werden.

Es bleiben nur Objekte zurück, welche über das Attribut *foaf:name* verfügen und dessen Inhalt “*Smith*” ist.

10.5 Gruppierungen und Aggregationen

Aggregationen bezeichnen eine bestimmte Art der Verbindung zwischen Objekten. Wikipedia Foundation [2014d] Aggregationen wenden Ausdrücke bzw. Funktionen auf eine Lösungsmenge an. Dabei enthält eine Lösung normalerweise eine einzelne Gruppe mit allen Lösungen.

Gruppierungen werden mittels *GROUP BY* angegeben.

SPARQL unterstützt aktuell die Aggregationen *COUNT*, *SUM*, *MIN*, *MAX*, *AVG*, *GROUP_CONCAT* und *SAMPLE*.

Die Aggregation wird dann verwendet, wenn ein Resultat über eine Gruppe von Lösungen berechnet werden soll (Beispiel: Maximaler Wert einer bestimmten Variablen).

```
|| PREFIX books: < http://books.example/>
|| SELECT (
||   SUM(?lprice) AS ?totalPrice
|| )
|| WHERE {
||   ?org books:affiliates ?auth .
||   ?auth books:writesBook ?book .
```

⁵[W3C SPARQL Working group and Harris and Seaborne, 2013, 5.2.2 Scope of Filters]

```

    ?book books:price ?lprice .
}
GROUP BY ?org
HAVING (SUM(?lprice) > 10)

```

Auflistung 10.7: Beispiel einer Abfrage mit Gruppierung und Aggregation⁶

10.6 Modifikatoren

Abfragen mittels SPARQL generieren eine ungeordnete Menge von Lösungen. Jede dieser Lösungen stellt eine Funktion von Variablen zu RDF-Termen dar. So gewonnene Lösungen werden als Sequenz von Lösungen behandelt, zunächst ohne spezifische Ordnung.

Um eine Sequenz von Lösungen zu ordnen, werden Modifikatoren verwendet.
Dabei unterscheidet man zwischen folgenden Modifikatoren:

- *Order*
Sortiert die Lösungen auf- oder absteigend nach einer bestimmten Variablen
- *Projection*
Erlaubt die Auswahl von spezifischen Variablen mittels der *Select*-Klausel
- *Distinct*
Stellt sicher, dass die Lösungen in der Lösungsmenge einmalig sind
- *Reduced*
Verhindert die Elimination von Duplikaten in der Lösungsmenge
- *Offset*
Definiert, ab welchem Element der Lösungsmenge die gefundenen Lösungen ausgegeben werden
- *Limit*
Definiert die maximale Anzahl an gefundenen Lösungen, die ausgegeben werden soll

Die *Modifikatoren* werden in der Reihenfolge der obigen Liste angewendet.

10.7 Abfragearten

SPARQL unterscheidet zwischen vier Abfragearten:

- *SELECT*
Liefert alle Variablen oder eine Teilmenge derselben eines Graphen
- *CONSTRUCT*
Liefert einen RDF-Graphen durch Substitution der Variablen in einer Menge von gegebenen Tripeln
- *ASK*
Liefert einen Wahrheitswert (Boolean), welcher angibt, ob die Menge von angefragten Tripeln derjenigen in dem gegebenen Graphen entspricht

⁶[W3C SPARQL Working group and Harris and Seaborne, 2013, 11.1 Aggregate Example]

- *DESCRIBE*

Liefert einen RDF-Graphen, welcher die gefundenen Ressourcen beschreibt

10.7.1 SELECT

Die *SELECT*-Abfrageart liefert direkt Variablen und deren Belegung, in Form von Projektionen. Dabei entstehen neue Variablen-Belegungen. Die spezifischen, gewünschten Variablen werden in Form einer Liste, durch Leerzeichen getrennt, angegeben.

```
||| SELECT ?a ?b ?c
  WHERE {
```

Auflistung 10.8: Beispiel der *SELECT*-Abfrageart in SPARQL

Die *SELECT**-Syntax ist eine Kurzschreibweise und bedeutet die Verwendung bzw. die Auflistung aller in der Abfrage verwendeten Variablen. Davon ausgenommen sind Variablen innerhalb von *FILTER*-Anweisungen sowie Variablen, welche rechts des Minus-Operators stehen. Die Syntax darf nur verwendet werden, wenn die Abfrage keine *GROUP BY*-Aggregation verwendet.

```
||| SELECT *
  WHERE {
```

Auflistung 10.9: Beispiel der *SELECT ** Kurzschreibweise in SPARQL

Durch die *SELECT*-Abfrageart lassen sich neuen Variablen durch Ausdrücke einführen. Diese werden mittels (*SELECT expr AS var*) eingeführt.

```
||| SELECT ?title (?p * (1 - ?discount) AS ?price)
  WHERE {
```

Auflistung 10.10: Beispiel eines Ausdrucks der *SELECT*-Abfrageart

10.7.2 CONSTRUCT

Die *CONSTRUCT*-Abfrageart gibt als Antwort einen einzigen, durch eine Vorlage definierten RDF-Graphen zurück: Jede Variable der Abfrage wird durch die passende Lösung aus der Lösungssequenz ersetzt. Die Lösungsmenge wird schliesslich durch Vereinigung der Tripel auf einen einzigen RDF-Graphen reduziert.

```
||| PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
  CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }
```

```
|| WHERE { ?x foaf:name ?name }
```

Auflistung 10.11: Beispiel einer SPARQL-Abfrage unter Nutzung der *CONSTRUCT*-Abfrageart⁷

Die obige Abfrage erzeugt folgende Ausgabe:

```
@prefix vcard: < http://www.w3.org/2001/vcard-rdf/3.0#> .
< http://example.org/person#Alice> vcard:FN "Alice" .
```

Auflistung 10.12: Beispiel der Ausgabe einer SPARQL-Abfrage unter Nutzung der *CONSTRUCT*-Abfrageart⁸

Entsteht bei diesem Vorgang ein ungültiges Tripel, wird dieses nicht im Ausgabegraphen angezeigt.
Enthält die Vorlage eines Graphen Tripel ohne Variablen, erscheinen diese dennoch im Ausgabegraphen.

```
PREFIX foaf: < http://xmlns.com/foaf/0.1/>
PREFIX vcard: < http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x vcard:N _:v .
            _:v vcard:givenName ?gname .
            _:v vcard:familyName ?fname }
WHERE
{
  { ?x foaf:firstname ?gname } UNION { ?x foaf:givenname ?gname } .
  { ?x foaf:surname ?fname } UNION { ?x foaf:family_name ?fname } .
}
```

Auflistung 10.13: Beispiel einer SPARQL Abfrage mit Nutzung der *CONSTRUCT*-Abfrageart unter Verwendung leerer Knoten⁹

Die obige Abfrage erzeugt folgende Ausgabe:

```
@prefix vcard: < http://www.w3.org/2001/vcard-rdf/3.0#> .

_:v1 vcard:N _:x .
_:x vcard:givenName "Alice" .
_:x vcard:familyName "Hacker" .

_:v2 vcard:N _:z .
_:z vcard:givenName "Bob" .
_:z vcard:familyName "Hacker" .
```

Auflistung 10.14: Beispiel der Ausgabe einer SPARQL Abfrage mit Nutzung der *CONSTRUCT*-Abfrageart unter Verwendung leerer Knoten¹⁰

⁷[W3C SPARQL Working group and Harris and Seaborne, 2013, 16.2 CONSTRUCT]

⁸[W3C SPARQL Working group and Harris and Seaborne, 2013, 16.2 CONSTRUCT]

⁹[W3C SPARQL Working group and Harris and Seaborne, 2013, 16.2.1 Templates with Blank Nodes]

¹⁰[W3C SPARQL Working group and Harris and Seaborne, 2013, 16.2.1 Templates with Blank Nodes]

10.7.3 ASK

Die ASK-Abfrageart beantwortet in der Ausgabe, ob eine Lösung für die gegebene Abfrage existiert.

Gegeben sei die folgende Datenbasis:

```
|| @prefix foaf: < http://xmlns.com/foaf/0.1/> .  
|| _:a foaf:name "Alice".  
|| _:a foaf:homepage < http://work.example.org/alice/> .  
|| _:b foaf:name "Bob".  
|| _:b foaf:mbox < mailto:bob@work.example> .
```

Auflistung 10.15: Einfache Datenbasis direkt in SPARQL¹¹

Wird nun folgende Abfrage getätigt:

```
|| PREFIX foaf: < http://xmlns.com/foaf/0.1/>  
|| ASK {  
||   ?x foaf:name "Alice"  
|| }
```

Auflistung 10.16: Beispiel einer SPARQL-Abfrage mit Nutzung der ASK-Abfrageart¹²

So antwort das System: *true*, da die Datenbasis ein Individuum mit dem Attribut *foaf : name* enthält, dessen Wert "Alice" ist.

10.7.4 DESCRIBE

Bei der DESCRIBE-Abfrageart antwortet das System mit einem RDF-Graphen, der Informationen in Form von RDF-Daten über Ressourcen enthält. Als Ressourcen kommen entweder *IRIs* oder Variablen in Betracht.

```
|| DESCRIBE  
||   ?a ?b ?c  
|| WHERE {  
||   ...
```

Auflistung 10.17: Beispiel der DESCRIBE-Abfrageart in SPARQL

Die DESCRIBE*-Syntax ist die Kurzschreibweise und bedeutet die Verwendung bzw. die Auflistung aller in der Abfrage verwendeten Variablen.

```
|| DESCRIBE *  
|| WHERE {  
||   ...
```

Auflistung 10.18: Beispiel der DESCRIBE * Kurzschreibweise in SPARQL

¹¹[W3C SPARQL Working group and Harris and Seaborne, 2013, 16.3 ASK]

¹²[W3C SPARQL Working group and Harris and Seaborne, 2013, 16.3 ASK]

10.8 Ausdrücke und Wertevergleiche

In SPARQL kann jede Abfrage mit einem Filter versehen werden. Dabei wird versucht ein bestimmtes Muster (durch Beschränkungen) auf Graphen anzuwenden um so die Graphen einzuschränken (filtern).

Jedes RDF-Literal kann durch einen bestimmten Datentyp definiert sein, wie zum Beispiel Wahrheitswerte (Boolean) oder Datum und Uhrzeit (DateTime). Der Filter in SPARQL erlaubt dabei den Wertevergleich auf typisierte RDF-Literale. Die dabei verwendeten Operanden und Datentypen finden sich unter den Webseiten [w3.org/TR/xpath-functions¹³](http://www.w3.org/TR/xpath-functions) und [w3.org/TR/sparql11-query¹⁴](http://www.w3.org/TR/sparql11-query).

Eine SPARQL-Abfrage unter Verwendung eines Filters kann wie folgt aussehen:

```
...  
SELECT  
    ?book , ?amount  
WHERE {  
    ?book :isInLibrary :BerneUniversityLibrary  
    ?book :hasAmount ?amount  
    FILTER (?amount > xsd:integer(10))  
}
```

Auflistung 10.19: Beispiel einer einfachen SPARQL-Abfrage unter Verwendung eines Filters

In diesem Beispiel wird eine Datenbank nach mehrfachen Vorkommen (mindestens 10 Mal) eines Buchexemplares in der Bibliothek der Universität Bern abgefragt. Die Anzahl der Exemplare eines Buches wird durch die Variable *?amount* zurückgegeben.

Ist man in obigem Beispiel nicht sicher, welchem Datentyp die Variable *?amount* entspricht, kann man die Funktionalität zur Umwandlung (Casting, Typenumwandlung) von Variablen nützen:

```
...  
SELECT  
    ?book , ?amount  
WHERE {  
    ?book :isInLibrary :BerneUniversityLibrary  
    ?book :hasAmount ?amount  
    FILTER (xsd:integer(?amount) > xsd:integer(10))  
}
```

Auflistung 10.20: Beispiel einer einfachen SPARQL-Abfrage unter Verwendung eines Filters mit Typenumwandlung

Die genaue Auswertung eines Filters lassen sich unter der Webseite [w3.org/TR/sparql11-query/#evaluation¹⁵](http://www.w3.org/TR/sparql11-query/#evaluation), die interne Verwendung und Auswertung von Operatoren unter der Webseite [w3.org/TR/sparql11-query/#OperatorMapping¹⁶](http://www.w3.org/TR/sparql11-query/#OperatorMapping) nachlesen.

10.8.1 Funktionen

SPARQL bietet eine Vielzahl an Operatoren und Funktionen. Diese einzeln aufführen und beschreiben zu wollen, würde den Rahmen dieser Arbeit sprengen. Daher werden nur die Operatoren und Funktionen aufgeführt, welche in der vorliegenden Arbeit hauptsächlich angewendet wurden. Eine genauere Beschreibung findet sich unter der Webseite [w3.org¹⁷](http://www.w3.org).

¹³<http://www.w3.org/TR/xpath-functions/>

¹⁴<http://www.w3.org/TR/sparql11-query/#operandDataTypes>

¹⁵<http://www.w3.org/TR/sparql11-query/#evaluation>

¹⁶<http://www.w3.org/TR/sparql11-query/#OperatorMapping>

¹⁷<http://www.w3.org/TR/sparql11-query/#Sparql10ps>

BOUND

Mittels *BOUND* lässt sich prüfen, ob ein Objekt eine Wert-Bindung hat. Man kann es als eine Art der Überprüfung auf gewisse Eigenschaften bezeichnen.

Möchte man zum Beispiel wissen, welche Personen über eine Relation zu einem Datum (z.B. ihr Erfassungsdatum) haben, so bestimmt man dies mittels:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:a foaf:givenName "Alice".
_:b foaf:givenName "Bob".
_:b dc:date "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

Auflistung 10.21: Beispiel einer simplen Ontologie zur Nutzung der *BOUND*-Funktion in einer Abfrage¹⁸

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT
    ?givenName
WHERE {
    ?x foaf:givenName ?givenName .
    OPTIONAL { ?x dc:date ?date } .
    FILTER ( bound(?date) )
}
```

Auflistung 10.22: Beispiel einer SPARQL-Abfrage zur Nutzung der *BOUND*-Funktion¹⁹

Die obige Abfrage ergibt folgendes Resultat:

givenName
"Bob"

Tabelle 10.3: Resultat einer SPARQL-Abfrage mittels *BOUND*-Filter²⁰

Es wird nur die Person "Bob" gefunden, da nur bei dieser Person eine Datum-Relation definiert ist.

IF

Die *IF*-Funktion gibt Argumente aufgrund einer Bedingung, welche selbst auch ein Argument ist, zurück. Die Funktion verwendet drei Argumente als Parameter. Die Argumente sind:

- Argument 1: Bedingung
- Argument 2: Rückgabewert wenn die Bedingung (Argument 1) zutrifft
- Argument 3: Rückgabewert wenn die Bedingung (Argument 1) nicht zutrifft

```
|| IF(?a = 2,?b,?c)
```

Auflistung 10.23: Beispiel der *IF*-Funktion in SPARQL

¹⁸[W3C SPARQL Working group and Harris and Seaborne, 2013, 17.4.1.1 BOUND]

¹⁹[W3C SPARQL Working group and Harris and Seaborne, 2013, 17.4.1.1 BOUND]

²⁰[W3C SPARQL Working group and Harris and Seaborne, 2013, 17.4.1.1 BOUND]

In prozeduraler Programmierung ausgedrückt:

```
|| if (a == 2) {  
||   return b;  
|| } else {  
||   return c;  
|| }
```

Auflistung 10.24: Beispiel der *IF*-Funktion in SPARQL anhand prozeduraler Programmierung ausgedrückt

IN

Die *IN*-Funktion prüft, ob ein Element in einer Liste vorkommt. Das zu prüfende Element wird der *IN*-Funktion vorangestellt, danach folgt die zu prüfende Liste. Kommt das Element in der Liste vor, gibt die Funktion den Wahrheitswert *wahr* bzw. *true* zurück, andernfalls *falsch* bzw. *false*.

```
|| 2 IN (1, 2, 3) % Wahr  
|| 2 IN () % Falsch
```

Auflistung 10.25: Beispiel der *IN*-Funktion in SPARQL

In den obigen Beispielen ist der erste Ausdruck wahr, da 2 in der gegebenen Liste von (1, 2, 3) vorkommt. Der zweite Ausdruck ist falsch, da 2 nicht in der leeren Liste () vorkommt.

Analog zur *IN*-Funktion existiert die *NOT IN*-Funktion. Dabei handelt es sich um die Negation der *IN*-Funktion.



Wie beschrieben sind jetzt alle Komponenten vorhanden, um Ausflüge abilden zu können, die den in Kapitel 3 (Graphrepräsentationen) genannten Kriterien entsprechen.

In diesem Kapitel haben wir die bisher fehlende Abfragesprache abgehandelt.

Um auf das beschriebene Beispiel der Familie, welche einen Ausflug plant, zurückzukommen: Mit der nachfolgenden Abfrage gelingt es, alle Ausflüge mit den Kriterien *familienfreundlich*, *regional* und *actionreich* zu finden.

```
||  
||   SELECT  
||   *  
||   WHERE {  
||     ?object :familiengerecht true;  
||     :regional true;  
||     :action true.  
|| }
```

Auflistung 10.26: Beispiel einer Abfrage um alle familiengerechten, actionreichen und regionalen Ausflüge der Ontologie auszugeben

²⁰[W3C SPARQL Working group and Harris and Seaborne, 2013, 17.4.1.9 IN]

11 Schlusswort

In der vorliegenden Arbeit zeigen wir auf, wie bei der Modellierung sowie Formalisierung der Wissensmodellierung mittels Ontologien vorgegangen werden kann. Dabei bieten wir verschiedene Sichten auf das Vorgehen.

Es war uns bei der Arbeit wichtig, die theoretischen Grundlagen fundiert und bis zu einem gewissen Detailgrad aufzuzeigen.

Am Beispiel Reiseplanung haben wir schrittweise eine Ontologie aufgebaut und führen den Leser so durch den gesamten Prozess der Modellierung. Aufgrund unserer Erfahrung geben wir praktische Tipps für die direkte Umsetzung. Zusätzlich zeigen wir anhand von Beispielen auf, wie die einzelnen Themen der Arbeit in der Praxis umgesetzt werden können.

Ontologien bilden die Grundlage für eine semantische Datenbank, welche wiederum die Wissensbasis für ein Expertensystem ist. Ontologien beinhalten sowohl Fakten, als auch Regeln. Die Fakten werden mittels einer Ontologiesprache, wie zum Beispiel OWL beschrieben. Regeln werden in der Regelsprache SWRL abgebildet. Die Inferenzmaschine (Reasoner) ist eine weitere Komponente des Expertensystems. Ein Reasoner ist fähig mittels Inferenz und Resolution Schlüsse zu ziehen und neues Wissen zu generieren.

Diese Eigenschaft führt zu dem Mehrwert dieser Form der Wissensmodellierung. Die Daten werden so mit einer gewissen Intelligenz mittels Schlussfolgerungen angereichert.

Während herkömmliche (relationale) Datenbanken nur Beziehungen aufzeigen, liegt der Vorteil der semantischen Datenbank in ihrer Flexibilität und Semantik (Möglichkeit den Relationen eine Bedeutung zu geben). Es werden also nicht nur Informationen, sondern auch Wissen abgebildet.

Eine Ontologie wird immer auf eine spezifische Problemdomäne angewendet. Eine wichtige Erkenntnis war für uns, dass sich nicht jedes Themengebiet als Problemdomäne eignet. So macht eine Abbildung einer solchen Domäne mittels Ontologien nur dann Sinn, wenn ein Experte benötigt wird um das vorhandene Wissen zu interpretieren. Theoretische Gebiete auf hoher Abstraktionsebene sind nicht geeignet.

Zur Modellierung bieten sich verschiedene Hilfsmittel an. Unserer Erfahrung nach sind semantische Netze eine sehr nützliche und übersichtliche Art der grafischen Darstellung.

Nutzt eine Applikation eine semantische Datenbank als Datenmodell, erfordern Anpassungen (Modellierungen) des Datenmodells keine Programmänderungen — bei geschickter Programmierung.

Modellierungen sind z.B. das Hinzufügen, Bearbeiten oder Löschen von Entitäten (Klassen, Individuen, Relationen oder Eigenschaften). Im Gegensatz hierzu benötigen Änderungen in relationalen Datenbanken meistens sehr aufwendige Programmänderungen.

Nach einer gewissen Einarbeitungsphase wird die Modellierung mittels Ontologien sehr intuitiv und dem menschlichen Denken ähnlich. Dank der vorhandenen Werkzeuge kann eine Ontologie übersichtlich und klar strukturiert abgebildet werden.

Glossar

Abox Assertional Box; Komponente eines Tableau-Reasoners, welche Aussagen zu Individuen enthält, d.h. OWL-Fakten wie Typen, Eigenschaftswerte und logische Äquivalenz. (vgl. Sirin et al. [2005]).

KB Knowledge Base; Eine Kombination einer Abox und Tbox, damit eine komplette OWL-Ontologie. (vgl. Sirin et al. [2005]).

OWL Web Ontology Language; Ontologiesprache für das semantische Web. Mit dieser Sprache können Ontologien beschrieben werden. (vgl. Cambridge Semantics INC (The Smart Data Company) [2014b]).

RDF Resource Description Framework; Framework um Informationen aus Ressourcen zu formulieren. Im Web können Informationen mit RDF verarbeitet werden, anstatt diese nur anzuzeigen. RDF bietet ein gemeinsames Framework um die Informationen zwischen Anwendungen auszutauschen ohne dabei die Bedeutung der Informationen zu verändern. (vgl. Schreiber and Raimond [2014]).

RDFS Resource Description Framework Schema; RDF Schema bietet ein Vokabular zur Datenmodellierung von RDF-Daten. Es stellt eine Erweiterung des RDF-Vokabulars dar. (vgl. Brickley and Guha [2014]).

RDQL RDF Data Query Language; Abfragesprache um Informationen von RDF-Graphen zu extrahieren. Die Syntax ist der SPARQL-Syntax sehr ähnlich. (vgl. Seaborne [2004]).

RuleML Rule Markup Language; XML-Sprache zur Beschreibung von Regeln. (vgl. Seaborne [2004]).

SPARQL SPARQL Protocol And RDF Query Language; Bei SPARQL handelt es sich um eine Abfragesprache für RDF. Sie erlaubt es, Abfragen in mehreren Datenquellen vorzunehmen. Dabei werden Anfragen über Graphen vollzogen, auch entlang derer Konjunktionen und Disjunktionen. SPARQL unterstützt weiter Aggregation, Unterabfragen, Negation sowie die Nutzung von Ausdrücken als Werte. Resultate sind entweder eine Menge von Ergebnissen oder RDF-Graphen. (vgl. W3C SPARQL Working group and Harris and Seaborne [2013]).

SQL Structured Query Language; "SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen." Wikipedia Foundation [2015].

SWRL Semantic Web Rule Language; Bei SWRL handelt es sich um eine auf OWL und RuleML basierende Regelsprache. Sie erlaubt es Regeln in Form von OWL-Konzepten auszudrücken und bietet dadurch vielfältige Möglichkeiten der Inferenz. (vgl. Horrocks et al. [2004]).

Tbox Terminological Box; Komponente eines Tableau-Reasoners, welche Klassenaxiome enthält, d.h. OWL-Axiome wie z.B. Unterklassen, Gleichheit von Klassen und Klasseneinschränkungen. (vgl. Sirin et al. [2005]).

Literaturverzeichnis

- Frank J. Furrer. Eine kurze geschichte der ontologie. *Informatik-Spektrum*, 37(4):308–317, 2014. ISSN 0170-6012. doi: 10.1007/s00287-012-0642-3. URL <http://dx.doi.org/10.1007/s00287-012-0642-3>.
- Johannes Busse, Bernhard Humm, Christoph Lübbert, Frank Moelter, Anatol Reibold, Matthias Rewald, Veronika Schlüter, Bernhard Seiler, Erwin Tegtmeier, and Thomas Zeh. Was bedeutet eigentlich ontologie? *Informatik-Spektrum*, 37(4):286–297, 2014. ISSN 0170-6012. doi: 10.1007/s00287-012-0619-2. URL <http://dx.doi.org/10.1007/s00287-012-0619-2>.
- S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009. ISBN 978-1-292-02420-2.
- U. Lämmel and J. Cleve. *Künstliche Intelligenz*. Carl Hanser Verlag München, 4rd edition, 2012. ISBN 978-3-446-42748-7.
- LinkedDataTools.com. Introducing graph data, May 2010. URL <http://www.linkeddatatools.com/introducing-rdf>. Abgerufen am 29. Oktober 2014.
- Wikipedia Foundation. Wissensrepräsentation. Website, October 2014a. URL <http://de.wikipedia.org/wiki/Wissensrepräsentation>. Abgerufen am 23. Oktober 2014.
- World Wide Web Consortium. Inference. Website, 2013. URL <http://www.w3.org/standards/semanticweb/inference>. W3C; Abgerufen am 17. Oktober 2014.
- Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0-521-78176-0.
- Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner, 2005.
- Patel-Schneider, Haes, and Horrocks. Owl web ontology language abstract syntax and semantics, w3c recommendation. Website, February 2004. URL www.w3.org/TR/owl-semantics/. W3C Specification; Abgerufen am 19. Oktober 2014.
- Wikipedia Foundation. Baumkalkül. Website, 2014b. URL <http://de.wikipedia.org/wiki/Baumkalk%C3%BC>. Abgerufen am 22. Dezember 2014.
- Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner, 2007.
- Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Al-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998. ISSN 0925-9902. doi: 10.1023/A:1008687430626. URL <http://dx.doi.org/10.1023/A%3A1008687430626>.
- John D. Ramsdell. *Datalog User Manual*. MITRE Corporation, <http://datalog.sourceforge.net/datalog.html>, 2.3 edition, 2004. Abgerufen am 04. Dezember 2014.
- Wikipedia Foundation. Description logic. Website, 2014c. URL http://en.wikipedia.org/wiki/Description_logic. Abgerufen am 22. Dezember 2014.
- Haoyi Xiong and Ying Jiang. Constraint satisfaction problem solving based on owl reasoning. Conference proceeding, 2008. Proceedings of the IADIS International Conference on WWW/IInterne;Jan2008, p. 177.
- Derek Sleeman and Stuart Chalmers. Assisting domain experts to formulate and solve constraint satisfaction problems. In Steffen Staab and Vojtěch Svátek, editors, *Managing Knowledge in a World of Networks*, volume 4248 of *Lecture Notes in Computer Science*, pages 27–34. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-46363-4. doi: 10.1007/11891451_5. URL http://dx.doi.org/10.1007/11891451_5.

- Madalina Croitoru and Ernesto Compatangelo. Ontology constraint satisfaction problem using conceptual graphs. In Max Bramer, Frans Coenen, and Andrew Tuson, editors, *Research and Development in Intelligent Systems XXIII*, pages 231–244. Springer London, 2007. ISBN 978-1-84628-662-9. doi: 10.1007/978-1-84628-663-6_17. URL http://dx.doi.org/10.1007/978-1-84628-663-6_17.
- Schreiber and Raimond. Rdf 1.1 primer. Website, June 2014. URL <http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/>. W3C Specification; Abgerufen am 10. Oktober 2014.
- Cambridge Semantics INC (The Smart Data Company). Rdf 101. Website, 2014a. URL <http://www.cambridgesemantics.com/semantic-university/rdf-101>. Abgerufen am 10. Oktober 2014.
- Hitzler, Krötzsch, Parsia, Patel-Schneider, and Rudolph. Owl 2 web ontology language. Website, December 2012. URL <http://www.w3.org/TR/2012/REC-owl2-primer-20121211/>. W3C Specification; Abgerufen am 10. Oktober 2014.
- Cambridge Semantics INC (The Smart Data Company). Owl 101. Website, 2014b. URL <http://www.cambridgesemantics.com/semantic-university/owl-101>. Abgerufen am 10. Oktober 2014.
- I. Horrocks, Peter F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. Website, May 2004. URL <http://www.w3.org/Submission/SWRL/>. W3C Specification; Abgerufen am 02. November 2014.
- W3C SPARQL Working group and Harris and Seaborne. Sparql 1.1 query language. Website, March 2013. URL <http://www.w3.org/TR/sparql11-query/>. W3C Specification; Abgerufen am 10. Oktober 2014.
- Beckett. Rdf/xml syntax specification (revised). Website, March 2013. URL <http://www.w3.org/TR/REC-rdf-syntax/>. W3C Specification; Abgerufen am 19. Oktober 2014.
- Wikipedia Foundation. Aggregation. Website, 2014d. URL [http://de.wikipedia.org/wiki/Aggregation_\(Informatik\)](http://de.wikipedia.org/wiki/Aggregation_(Informatik)). Abgerufen am 23. Dezember 2014.
- Dan Brickley and R.V. Guha. Rdf schema 1.1. Website, February 2014. URL <http://www.w3.org/TR/rdf-schema/>. W3C Specification; Abgerufen am 11. Januar 2015.
- Andy Seaborne. Rdql - a query language for rdf. Website, January 2004. URL <http://www.w3.org/Submission/RDQL/>. W3C Specification; Abgerufen am 11. Januar 2015.
- Wikipedia Foundation. Sql. Website, January 2015. URL <http://de.wikipedia.org/wiki/SQL>. Abgerufen am 11. Januar 2015.

Abbildungsverzeichnis

2.1	Aufbau eines Expertensystems. ¹	3
2.2	Einfaches Beispiel der Vererbung von Eigenschaften. ²	5
2.3	Einfaches Beispiel von prozedurellem Wissen. ³	5
3.1	Darstellung der verschiedenen Datenbanktypen. ⁴	7
3.2	Aussagen über Hotels als Graphdatenbank. ⁵	8
3.3	Beispiel einer Graphdatenbank basierend auf dem Beispiel eines Reiseplaners ⁶	9
4.1	Abbildung von Wissen mittels eines semantischen Netzes. ⁷	13
5.1	Ontologiestruktur ⁸	15
6.1	Hauptkomponenten des Pellet-Reasoners. ⁹	20
6.2	Wurzelknoten des Graphen	22
6.3	Graph nach Expansion des Wurzelknotens	23
6.4	Graph nach Ersetzen der Teilbäume	23
6.5	Graph mit markiertem Teilbaum nach Widerspruch	23
6.6	Graph mit markierten Teilbäumen nach Widersprüchen	24
6.7	Ablauf der Beantwortung einer Anfrage in Abox-Query-Engine. ¹⁰	25
6.8	Darstellung des Individuums <i>Seilpark Balmberg</i> in Protégé. ¹¹	27
8.1	Übersichtsfenster des Protégé-Editors ¹²	38

Tabellenverzeichnis

6.1	Beschreibung der wichtigsten Komponenten von Pellet ¹³	20
6.2	Transformationsregeln des Tableau-Kalküls ¹⁴	21
10.1	Resultat einer einfachen SPARQL-Abfrage ¹⁵	45
10.2	Resultat einer Abfrage des DBpedia-Endpunktes limitiert auf 10 Ergebnisse.	46
10.3	Resultat einer SPARQL-Abfrage mittels <i>BOUND</i> -Filter ¹⁶	53

Auflistungsverzeichnis

2.1	Einfaches Beispiel von relationalem Wissen	4
3.1	Aussagen über Hotels	8
4.1	Beispiel eines Frames anhand einer Reise	11
5.1	Beispiel einer Aussage in einer Wissensbasis	15
5.2	Beispiel einer Regel in einer Wissensbasis	15
6.1	Beispielanfrage an eine Wissensdatenbank eines Computers	17
6.2	Aussage in einer Wissensdatenbank eines Computers	17
6.3	Definition Interpretation ¹⁷	18
6.4	Definition Modell ¹⁸	18
6.5	Definition semantische Folgerung ¹⁹	19
6.6	Definition Ableitbarkeit ²⁰	19
6.7	Definition Korrektheit und Vollständigkeit ²¹	19
6.8	Äquivalenz von semantischer Folgerung und Widerspruchsbeweis ²²	19
7.1	Tripel-Struktur einer RDF-Aussage	28
7.2	Beispiel einer RDF-Aussage	28
7.3	Beispiel eines unbenannten (unnamed) Graphen	29
7.4	Beispiel eines benannten (named) Graphen	29
7.5	Beispiel RDF-Elemente ²³	30
8.1	Beispiel der Turtle Syntax ²⁴	32
8.2	Beispiel eines Ausdrucks: Zwei Klassen bedeuten das Gleiche	33
8.3	Beispiel einer Klasse: Ausflüge	33
8.4	Beispiel einer Subklasse	34
8.5	Beispiel eines Individuums	34
8.6	Beispiel einer Gleichstellung von Individuen: Seilpark und Seilpark-Balmberg sind das gleiche Individuum	35
8.7	Beispiel einer Differenzierung von Individuen: Seilpark ist nicht das gleiche Individuum wie Seilpark-Pilatus	35
8.8	Beispiel einer Objekteigenschaft <i>hatOrt</i> und deren Anwendung	35
8.9	Beispiel von Einschränkungen der Objekteigenschaft <i>hatRegion</i>	35
8.10	Beispiel der Objekteigenschaft <i>hatGemeinde</i> als Subeigenschaft von <i>hatRegion</i>	36
8.11	Beispiel der Datentypen-Eigenschaft <i>anzahlTeilnehmer</i> und deren Anwendung bei einem Individuum	36
8.12	Beispiel einer Definition einer Ontologie	37
8.13	Beispiel eines Importes einer Ontologie	37
8.14	Beispiel einer Referenzierung auf ein Objekt einer externen Ontologie	37
9.1	Beispiel von positiven Konjunktionen von Atomen	39
9.2	Beispiel eines Atoms	39
9.3	Beispiele von Klassen-Atomen	40
9.4	Beispiel der SWRL-Regel, dass alle Abenteuerreisen auch Reisen sind	40
9.5	Beispiele von Atomen zur Darstellung von Eigenschaften von Individuen	40
9.6	Beispiel der SWRL-Regel um ein Objekt ?a als Teamevent zu definieren	40
9.7	Beispiele von Atomen zur Darstellung von wertespezifischen Eigenschaften von Individuen	40
9.8	Beispiel einer Regel, welche ausdrückt, dass ein Objekt actionreich ist, wenn es Mut erfordert	40
9.9	Beispiele von Atomen zur Unterscheidung von zwei Individuen	41

9.10 Beispiele von Atomen zur Gleichsetzung von zwei Individuen	41
9.11 Beispiele von Atomen zur Kennzeichnung eines Wertes	41
9.12 Beispiel einer SWRL-Regel, welche besagt, dass ein Objekt die wertespezifische Eingenschaft <i>preiswert</i> hat	41
9.13 Beispiel einer SWRL-Regel zum Ausdrücken von Kollaboration	42
9.14 Beispiel einer SWRL-Regel zum Ausdrücken von Kollaboration mit expliziter Unterscheidung zwischen den Kollaborateuren	42
9.15 Beispiel der SWRL-Regel für die Eigenschaft <i>familienfreundlich</i>	42
9.16 Beispiel der SWRL-Regel für die Eigenschaft <i>regional</i>	42
9.17 Beispiel der SWRL-Regel für die Eigenschaft <i>actionreich</i>	43
 10.1 Einfache Datenbasis direkt in SPARQL ²⁵	45
10.2 Beispiel einer einfachen SPARQL-Abfrage ²⁶	45
10.3 Beispiel von Namespaces in RDF ²⁷	45
10.4 Beispiel einer SPARQL Abfrage mit den Variablen <i>?object</i> , <i>?predicate</i> und <i>?subject</i>	46
10.5 Beispiel eines grundlegenden Teilgraphen in SPARQL	47
10.6 Beispiel eines gruppierenden Teilgraphen mit dem <i>FILTER</i> -Schlüsselwort ²⁸	47
10.7 Beispiel einer Abfrage mit Gruppierung und Aggregation ²⁹	47
10.8 Beispiel der <i>SELECT</i> -Abfrageart in SPARQL	49
10.9 Beispiel der <i>SELECT *</i> Kurzschreibweise in SPARQL	49
10.10 Beispiel eines Ausdrucks der <i>SELECT</i> -Abfrageart	49
10.11 Beispiel einer SPARQL-Abfrage unter Nutzung der <i>CONSTRUCT</i> -Abfrageart ³⁰	49
10.12 Beispiel der Ausgabe einer SPARQL-Abfrage unter Nutzung der <i>CONSTRUCT</i> -Abfrageart ³¹	50
10.13 Beispiel einer SPARQL Abfrage mit Nutzung der <i>CONSTRUCT</i> -Abfrageart unter Verwendung leerer Knoten ³²	50
10.14 Beispiel der Ausgabe einer SPARQL Abfrage mit Nutzung der <i>CONSTRUCT</i> -Abfrageart unter Verwendung leerer Knoten ³³	50
10.15 Einfache Datenbasis direkt in SPARQL ³⁴	51
10.16 Beispiel einer SPARQL-Abfrage mit Nutzung der <i>ASK</i> -Abfrageart ³⁵	51
10.17 Beispiel der <i>DESCRIBE</i> -Abfrageart in SPARQL	51
10.18 Beispiel der <i>DESCRIBE *</i> Kurzschreibweise in SPARQL	51
10.19 Beispiel einer einfachen SPARQL-Abfrage unter Verwendung eines Filters	52
10.20 Beispiel einer einfachen SPARQL-Abfrage unter Verwendung eines Filters mit Typenumwandlung	52
10.21 Beispiel einer simplen Ontologie zur Nutzung der <i>BOUND</i> -Funktion in einer Abfrage ³⁶	53
10.22 Beispiel einer SPARQL-Abfrage zur Nutzung der <i>BOUND</i> -Funktion ³⁷	53
10.23 Beispiel der <i>IF</i> -Funktion in SPARQL	53
10.24 Beispiel der <i>IF</i> -Funktion in SPARQL anhand prozeduraler Programmierung ausgedrückt	54
10.25 Beispiel der <i>IN</i> -Funktion in SPARQL	54
10.26 Beispiel einer Abfrage um alle familiengerechten, actionreichen und regionalen Ausflüge der Ontologie auszugeben	54

C. Beispiele

C.1. Allgemein

C.1.1. Allgemein

```
?p a owl:Class?
    rdfs:subClassOf pg:Programm?
    rdfs:subClassOf ?restriction .
        ?restriction owl:onProperty pg:verwendetProgrammiersprache .
        ?restriction owl:hasValue pg:Deklarativ .

?p a owl:Class?
    rdfs:subClassOf pg:Programm?
    rdfs:subClassOf ?restriction .
        ?restriction owl:onProperty pg:bestehtAus .
        ?restriction owl:someValuesFrom pg:Axiom .
```

Auflistung C.1: Abfragen auf Relationswerte

C.1.2. Abfragen auf Relationswerte

```
PREFIX rdf: <http://www.w3.org/1999/02/22rdfsyntaxns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX pg:
<http://www.semanticweb.org/sosterwalder/ontologies/2014/9/programming#>
SELECT *
WHERE {
    ?ps pg:wirdVerwendetVon ?o .
    ?o pg:bestehtAus pg:axiom .
#pg:Programm ?xx ?object
}
```

Auflistung C.2: Alle Programmiersprachen, welche von einem Programm verwendet werden, welches aus Axiomen besteht

C.1.3. Alle Einsatzgebiete welche in Programmen zum Einsatz kommen, welche aus Axiome bestehen

```
PREFIX rdf: <http://www.w3.org/1999/02/22rdfsyntaxns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX pg:
<http://www.semanticweb.org/sosterwalder/ontologies/2014/9/programming#>
SELECT *
WHERE {
    ?es a owl:Class ?
        rdfs:subClassOf pg:Einsatzgebiet .
    ?r a owl:Restriction ?
```

```

        owl:onProperty pg:hatEinsatzgebiet?
        owl:allValuesFrom ?es.

?x a ?es.
?p a pg:Programm?
    a ?r?
    pg:bestehtAus pg:axiom.
}

```

Auflistung C.3: Alle Einsatzgebiete welche in Programmen zum Einsatz kommen, welche aus Axiome bestehen

C.1.4. Alle Einsatzgebiete welche in deklarativen Programmen zum Einsatz kommen

```

PREFIX rdf: <http://www.w3.org/1999/02/22 rdfsyntaxns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX pg:
<http://www.semanticweb.org/sosterwalder/ontologies/2014/9/programming#>
SELECT DISTINCT ?x
WHERE {
?es a owl:Class? rdfs:subClassOf pg:Einsatzgebiet .
?r a owl:Restriction?
    owl:onProperty pg:hatEinsatzgebiet?
    owl:allValuesFrom ?es.

?x a ?es.
?p a pg:Programm?
    a ?r.
    #pg:bestehtAus pg:axiom.
FILTER ( regex( str(?p) , "deklarativesProgramm" , "i" ) )
}

```

Auflistung C.4: Alle Einsatzgebiete welche in deklarativen Programmen zum Einsatz kommen

C.2. Wie ist Prolog aufgebaut?

C.2.1. Aus was besteht Prolog?

```

PREFIX rdf: <http://www.w3.org/1999/02/22 rdfsyntaxns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX pl:
<http://www.semanticweb.org/mira/ontologies/2014/9/untitledontology7#>
SELECT * WHERE {
?x rdfs:subClassOf pl:Programmiersprache .
?r a owl:Restriction?
    owl:onProperty pl:hatSyntaxElement?
    owl:someValuesFrom ?t .
?e rdfs:subClassOf ?t .
}

```

Auflistung C.5: Aus was besteht Prolog?

C.2.2. Aus was bestehen logische Elemente?

```
PREFIX rdf: <http://www.w3.org/1999/02/22rdfsyntaxns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX pl: <http://www.semanticweb.org/mira/ontologies/2014/9/untildeontology7#>
SELECT * WHERE {
    ?x rdfs:subClassOf pl:Programmiersprache .
    ?r a owl:Restriction?
        owl:onProperty pl:hatSyntaxElement?
        owl:someValuesFrom ?t .
    ?e rdfs:subClassOf ?t .
}
```

Auflistung C.6: *Aus was bestehen logische Elemente?*

C.2.3. Was für sprachliche Elemente verwendet Prolog?

```
PREFIX rdf: <http://www.w3.org/1999/02/22rdfsyntaxns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX pl: <http://www.semanticweb.org/mira/ontologies/2014/9/untildeontology7#>
SELECT * WHERE {
    ?x rdfs:subClassOf pl:Programmiersprache .
    ?r a owl:Restriction?
        owl:onProperty pl:hatSyntaxElement?
        owl:someValuesFrom ?t .
    ?e rdfs:subClassOf ?t .
    ?el rdfs:subClassOf ?e .
    ?xa a ?el .
}
```

Auflistung C.7: *Was für sprachliche Elemente verwendet Prolog?*

C.3. Wie funktioniert Unifikation?

Erwartete Antwort: In Prolog wird versucht mit Hilfe von Unifikation Anfragen mit Regeln oder Fakten identisch zu machen.

Da wir von der Fragestellung her wissen, dass wir die Klasse Unifikation wollen, geben wir diese bereits als Filter an.

C.3.1. Schritt 1: Beziehungen der Klasse Unifikation nach oben herausfinden

```
SELECT DISTINCT * WHERE {
    ?u a owl:Class .
    ?p ?b ?u .
    ?t ?a ?p .

    FILTER (?u=pl:Unifikation)
}
```

Auflistung C.8: *Beziehungen der Klasse Unifikation nach oben herausfinden*

Unifikation	u	p	b	Prolog	t	a
		● verwendet some Unifikation	someValuesFrom			subClassOf

Abbildung C.1.: Output: Beziehungen der Klasse Unifikation nach oben herausfinden.¹

C.3.2. Schritt 2: Beziehungen der Klasse Unifikation nach unten herausfinden

```
SELECT DISTINCT * WHERE {
    ?u a owl:Class .
    ?u rdfs:subClassOf ?y .

    FILTER (?u=pl:Unifikation)
}
```

Auflistung C.9: Beziehungen der Klasse Unifikation nach unten herausfinden

u	y
Unifikation	● nutzt some Substitution
Unifikation	● wirdAngewendetAuf value Anfrage
Unifikation	● setztGleich value Fakt
Unifikation	● setztGleich value Regel

Abbildung C.2.: Output: Beziehungen der Klasse Unifikation nach unten herausfinden.²

C.3.3. Schritt 3: Details der Beziehungen anzeigen

```
SELECT distinct * WHERE {
    ?u a owl:Class .
    ?u rdfs:subClassOf ?y .
    ?y owl:onProperty ?y3 .
    ?y owl:someValuesFrom ?v .
    FILTER (?u=pl:Unifikation)
}
```

Auflistung C.10: Details der Beziehungen anzeigen

u	y	y3	v
Unifikation	● nutzt some Substitution	nutzt	Substitution

Abbildung C.3.: Output: Details der Beziehungen anzeigen.³

Feststellung:

Die Beziehungen setztGleich und wirdAngewendetAuf scheinen keine weiteren Relationen zu haben.

Zwischenfazit

Mit den Schritten 1 bis 3 haben wir festgestellt, dass Unifikation auf Anfragen angewendet wird und diese Fakten und Regeln gleich setzt.

Als Verfahren scheint dabei Substitution zum Einsatz zu kommen. Dies muss aber in weiteren Schritten genauer analysiert werden.

¹Output Abfrage Protégé

²Output Abfrage Protégé

³Output Abfrage Protégé

C.3.4. Schritt 4: Beziehungen der Klasse Substitution herausfinden

```
SELECT distinct * WHERE {
    ?u a owl:Class .
    ?u rdfs:subClassOf ?y .
    ?y owl:onProperty ?y3 .
    ?y owl:someValuesFrom ?v .
    ?v rdfs:subClassOf ?v2 .
    FILTER ( ?u=pl:Unifikation )
}
```

Auflistung C.11: Beziehungen der Klasse Substitution herausfinden

	u	y	y3	v	v2
Unifikation		● nutzt some Substitution ● nutzt some Substitution	nutzt: nutzt	Substitution	● substituiert value Variable ● substituiert mit value Atom
Unifikation					

Abbildung C.4.: Output: Beziehungen der Klasse Substitution herausfinden.⁴

C.3.5. Schritt 5: Details der Beziehungen anzeigen

```
SELECT distinct * WHERE {
    ?u a owl:Class .
    ?u rdfs:subClassOf ?y .
    ?y owl:onProperty ?y3 .
    ?y owl:someValuesFrom ?v .
    ?v rdfs:subClassOf ?v2 .
    ?v2 owl:onProperty ?y3 .
    FILTER ( ?u=pl:Unifikation )
}
```

Auflistung C.12: Details der Beziehungen anzeigen

u	y	y3	v	v2

Abbildung C.5.: Output: Details der Beziehungen anzeigen.⁵

Feststellung:

Die Beziehungen substituiert und substituiertMit scheinen keine weiteren Relationen zu haben.

Zwischenfazit

Mit den Schritten 4 und 5 haben wir festgestellt, dass Substitution Variablen mit Atomen substituiert.

C.3.6. Schritt 6: Individuen extrahieren

```
SELECT distinct * WHERE {
    ?u a owl:Class .
    ?u rdfs:subClassOf ?y .
    ?y owl:onProperty ?y3 .
    ?y owl:someValuesFrom ?v .
    ?v rdfs:subClassOf ?v2 .
    ?v2 owl:hasValue ?v4 .
```

⁴Output Abfrage Protégé

⁵Output Abfrage Protégé

```

        FILTER  (?u=pl:Unifikation )
}

```

Auflistung C.13: Individuen extrahieren

u	y	y3	v	v2	v4
Unifikation	● nutzt some Substitution ● nutzt some Substitution	nutzt nutzt	Substitution Substitution	● substituiert value Variable ● substituiertMit value Atom	Variable Atom
Unifikation					

Abbildung C.6.: Output: Individuen extrahieren.⁶

Fazit

Dadurch, dass keine weitere Beziehungen vorhanden sind, haben wir somit alle Beziehungen/Informationen der Unifikation erhalten.

Es stellt sich nun die Frage, wie weitere Informationen zu den Individuen (Variable, Atom, Fakt, Regel und Anfrage) abgerufen werden können.

C.4. Was sind Atome?

Erwartete sparql'sche Antwort: Bei den Atomen handelt es sich um einfache Tokens, wobei diese wiederum Sprachelemente sind. Sprachelemente sind Tokens, woraus Prolog besteht.

Erwartete menschliche Antwort: Atome sind Sprachelemente von Prolog, welche mit einem Kleinbuchstaben oder einem Apostroph beginnen. Atome sind einfache Tokens.

Da wir von der Fragestellung her wissen, dass es sich bei Atom um ein sog. owl:NamedIndividual handelt, selektieren wir dieses direkt per Filter.

C.4.1. Schritt 1: Atom-Objekt selektieren

```

SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    FILTER ( regex( str(?i) , "atom" , "i" ) )
}

```

Auflistung C.14: Atom-Objekt selektieren

Atom	i
------	---

Abbildung C.7.: Output: Atom-Objekt selektieren.⁷

⁶Output Abfrage Protégé

⁷Output Abfrage Protégé

C.4.2. Schritt 2: Klasse/Typ des Individuums herausfinden

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c .
    FILTER ( regex( str(?i) , "atom" , "i" ) )
}
```

Auflistung C.15: *Klasse/Typ des Individuums herausfinden*

i	c
Atom	EinfachesToken
Atom	NamedIndividual

Abbildung C.8.: *Output: Klasse/Typ des Individuums herausfinden*.⁸

Feststellung:

Bei den Atomen handelt es sich um einfache Tokens (und um eine NamedIndividual, was aber bereits im Vorfeld klar war).

C.4.3. Schritt 3: Beziehungen des einfachen Token herausfinden

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c .
    ?c ?xx ?z .      # Gibt alle Beziehungen aus
    FILTER ( regex( str(?i) , "atom" , "i" ) && ?c=pl : EinfachesToken )
}
```

Auflistung C.16: *Beziehungen des einfachen Token herausfinden*

i	c	yyy	z
Atom	EinfachesToken		
Atom	EinfachesToken	type	Class
		subClassOf	SprachElement

Abbildung C.9.: *Output: Beziehungen des einfachen Token herausfinden*.⁹

Feststellung:

Wir sehen, dass einfache Tokens die Prädikate type und subClassOf mit den Subjekten Class bzw. SprachElement hat. Dies erlaubt nun die Einschränkung des Queries auf den Teil von Interesse, also subClassOf.

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c .
    ?c rdfs:subClassOf ?z .
    FILTER ( regex( str(?i) , "atom" , "i" ) && ?c=pl : EinfachesToken )
}
```

Auflistung C.17: *Beziehungen des einfachen Token herausfinden 2*

⁸Output Abfrage Protégé

⁹Output Abfrage Protégé

i	c	z
Atom	EinfachesToken	SprachElement

Abbildung C.10.: Output: Beziehungen des einfachen Token herausfinden 2.¹⁰

Feststellung:

Einfache Tokens scheinen also Sprachelemente zu sein.

C.4.4. Schritt 4: Beziehungen der Klasse SprachElement herausfinden

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c.

    ?c rdfs:subClassOf ?z .
    ?z rdfs:subClassOf ?e .

    FILTER ( regex( str(?i) , "atom" , "i" ) && ?c=pl:EinfachesToken )
}
```

Auflistung C.18: Beziehungen der Klasse SprachElement herausfinden

i	c	z	e
Atom	EinfachesToken	SprachElement	Token

Abbildung C.11.: Output: Beziehungen der Klasse SprachElement herausfinden.¹¹

C.4.5. Schritt 5: Beziehungen zu Token

Nachdem wir festgestellt haben, dass Token selbst keine weiteren Relationen mehr hat, wird in diesem Schritt die Anfrage umgedreht.

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c.

    ?c rdfs:subClassOf ?z .
    ?z rdfs:subClassOf ?e .
    ?o ?w ?e .

    FILTER ( regex( str(?i) , "atom" , "i" ) && ?c=pl:EinfachesToken )
}
```

Auflistung C.19: Beziehungen zu Token

i	c	z	e	o	w
Atom	EinfachesToken	SprachElement	Token	hatSyntaxElement some Token	someValueFrom
				LogischesElement	subClassOf
				SprachElement	subClassOf
				hatSyntaxElement	range

Abbildung C.12.: Output: Beziehungen zu Token.¹²

Feststellung:

Token wird von einer Klasse/einem Objekt mittels der Beziehung hatSyntaxElement als range verwendet.

¹⁰Output Abfrage Protégé

¹¹Output Abfrage Protégé

¹²Output Abfrage Protégé

C.4.6. Schritt 6: Objekt(e) mit Relation hatSyntaxElement herausfinden

Es soll herausgefunden werden, welche Objekte das Prädikat hatSyntaxElement als Eigenschaft und Token als Wert einer Einschränkung verwenden.

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c .
    ?c rdfs:subClassOf ?z .
    ?z rdfs:subClassOf ?e .
    ?bez rdfs:range ?e .
    ?r a owl:Restriction ;
        owl:onProperty ?bez ;
        owl:someValuesFrom ?e .
    ?cl rdfs:subClassOf ?r .

FILTER ( regex( str(?i) , "atom" , "i" ) && ?c=pl : EinfachesToken )
}
```

Auflistung C.20: *Objekt(e) mit Relation hatSyntaxElement herausfinden*

i	c	z	e	bez	r	cl	d
Atom	EinfachesToken	SprachElement	Token	hatSyntaxElement	hatSyntaxElement son Prolog		

Abbildung C.13.: Output: *Objekt(e) mit Relation hatSyntaxElement herausfinden*.¹³

Feststellung:

Es handelt sich bei dem gefundenen Objekt um Prolog!

C.4.7. Schritt 7: Beziehungen zu Prolog herausfinden

Es soll herausgefunden werden, welche Objekte das Prädikat hatSyntaxElement als Eigenschaft und Token als Wert einer Einschränkung verwenden.

```
SELECT distinct * WHERE {
    ?i a owl:NamedIndividual .
    ?i a ?c .
    ?c rdfs:subClassOf ?z .
    ?z rdfs:subClassOf ?e .
    ?bez rdfs:range ?e .
    ?r a owl:Restriction ;
        owl:onProperty ?bez ;
        owl:someValuesFrom ?e .
    ?cl rdfs:subClassOf ?r .
    ?cl rdfs:subClassOf ?ccl .
FILTER ( regex( str(?i) , "atom" , "i" ) && ?c=pl : EinfachesToken )
}
```

Auflistung C.21: *Beziehungen zu Prolog herausfinden*

i	c	z	e	bez	r	cl	ccl
Atom	EinfachesToken	SprachElement	Token	hatSyntaxElement	hatSyntaxElement son Prolog		Programmiersprache
Atom	EinfachesToken	SprachElement	Token	hatSyntaxElement	hatSyntaxElement son Prolog		hatSyntaxElement some Token
Atom	EinfachesToken	SprachElement	Token	hatSyntaxElement	hatSyntaxElement son Prolog		verwendet some Unifikation

Abbildung C.14.: Output: *Beziehungen zu Prolog herausfinden*.¹⁴

¹³Output Abfrage Protégé

Feststellung:

Prolog hat als übergeordnetes Objekt die Klasse Programmiersprache, ist als Unterklasse von dieser. Weiter ist ersichtlich, dass Prolog aus Token Elementen besteht und Unifikation verwendet.

Zwischenfazit

Die gewonnenen Erkenntnisse decken die Fragestellung grösstenteils ab, was aber nicht ersichtlich ist, ist, dass Atome über einen kleinen Anfangsbuchstaben verfügen.

C.4.8. Schritt 8: Kommentar(e) der Individuen ausgeben

Als Idee zur Lösung zum Zwischenfazit wird die Annotation comment von allen NamedIndividual Objekten ausgegeben.

```
SELECT DISTINCT * WHERE {
    ?i a owl:NamedIndividual .
    ?i rdfs:comment ?x .
    FILTER ( regex( str(?i) , "atom" , "i" ) )
}
```

Auflistung C.22: Kommentar(e) der Individuen ausgeben

i	x
Atom	"beginnt mit Kleinbuchstaben oder mit Apostroph" @

Abbildung C.15.: Output: Kommentar(e) der Individuen ausgeben.¹⁵

Analog kann dies natürlich in der Abfrage von Schritt 7 verwendet werden..

Fazit

Die ursprüngliche Frage konnte also erfolgreich wie folgt beantwortet werden:

- Atome sind Sprachelemente
Schritt 4
- von Prolog
Schritt 6
- welche mit einem Kleinbuchstaben oder einem Apostroph beginnen.
Schritt 8
- Atome sind einfache Tokens.
Schritt 3

¹⁴Output Abfrage Protégé

¹⁵Output Abfrage Protégé

C.5. Familienbeispiel

Damit wir eine Bestätigung bekommen, dass unser Semantisches System die gleiche Mächtigkeit wie Prolog (also alle notwendigen Funktionalitäten) besitzt, haben wir ein FamilienProlog Beispiel abgebildet.

Die Situation aus family.pl (siehe C.5.3) soll abgebildet werden.

Fakten werden mit Klassen resp. Objekt oder Data Properties abgebildet. Die Argumente in Prolog werden als Individuen abgebildet. (siehe C.5.4). Die Regel können eins zu eins übernommen werden.

Die Beispielenfragen wurden genau gleich wie in Prolog beantwortet.

```
ASK WHERE {
    :hans :isAncestor :tea .
}
```

Auflistung C.23: Familienbeispiel in Prolog

C.5.1. Alle Nachfahren von Doris

```
SELECT *
WHERE {
    ?subject :isAncestor ?object .
    #FILTER regex(?subject , "doris" , "i")
}
```

Auflistung C.24: Alle Nachfahren von Doris

C.5.2. Ist hans vorfahre von tea?

```
ASK WHERE {
    :hans :isAncestor :tea .
}
```

Auflistung C.25: Ist hans vorfahre von tea?

Zusätzlich zu der Möglichkeit anfragen zu stellen zeigt der Reasoner in Protégé seine Folgerungen direkt bei den Objekten an. Der Reasoner konnte in diesem Beispiel dank unserer Regeln direkt erkennen das doris eine Mutter ist und Verfahren von lisa, hans, luca, tea.

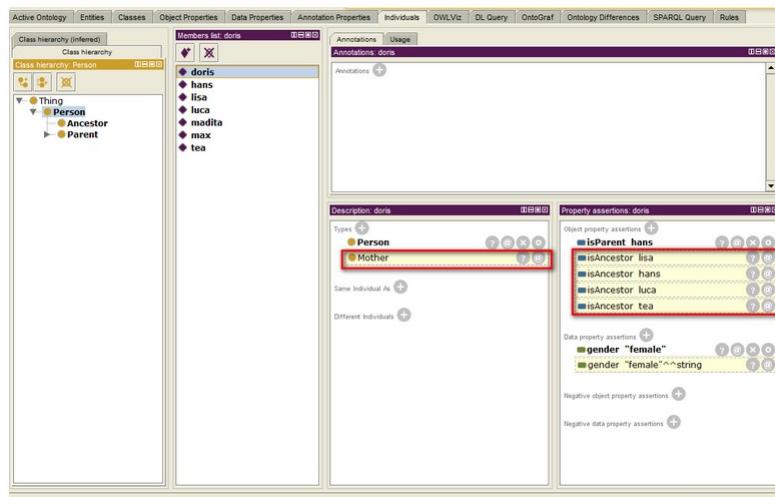


Abbildung C.16.: Output: Familienbeispiel Anzeige.¹⁶

C.5.3. Familienbeispiel in Prolog

```
:- module(family,[  
    male/1,  
    female/1,  
    parent/2,  
    mother/2,  
    father/2,  
    ancestor/2  
]).  
  
male(max).  
male(luca).  
male(hans).  
female(lisa).  
female(tea).  
female(madita).  
female(doris).  
  
parent(max,luca).  
parent(max,tea).  
parent(lisa,luca).  
parent(lisa,tea).  
parent(madita,lisa).  
parent(hans,lisa).  
parent(doris,hans).  
  
mother(P,C):-  
    female(P), parent(P,C).  
  
father(P,C):-  
    male(P), parent(P,C).  
  
child(C,P):-  
    parent(P,C).  
  
ancestor(A,C):-  
    parent(A,C).  
ancestor(A,C):-  
    parent(A,X), ancestor(X,C).  
  
% father(max,luca).  
% father(hans,luca).  
% mother(lisa,X).  
% ancestor(X,tea).  
  
% ancestor(doris,X).
```

¹⁶Reasoning in Protégé

C.5.4. Familienbeispiel in OWL

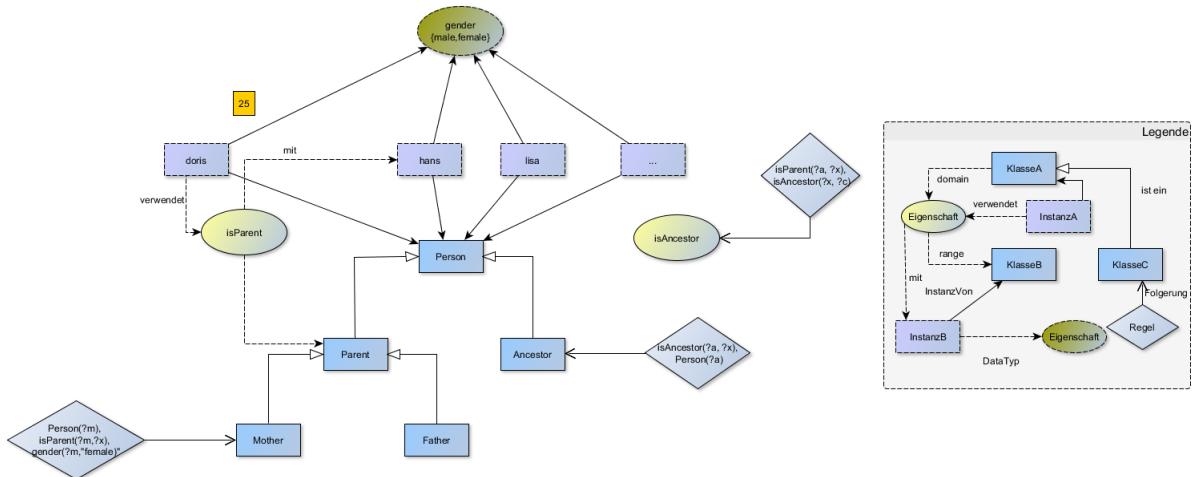
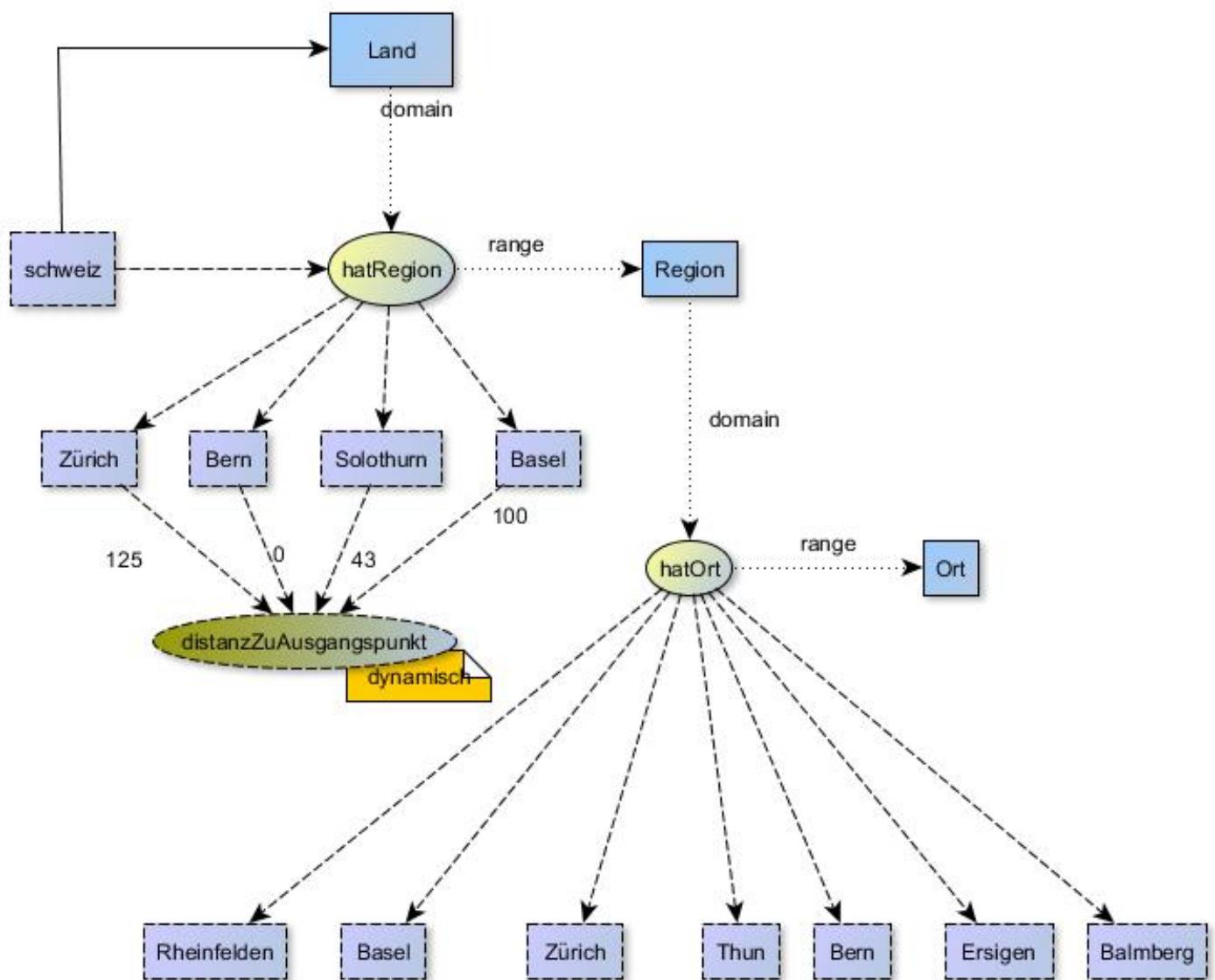


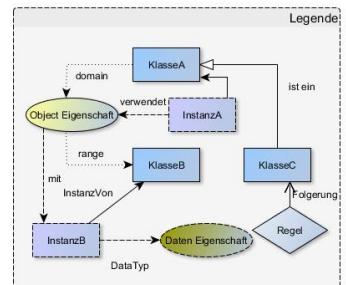
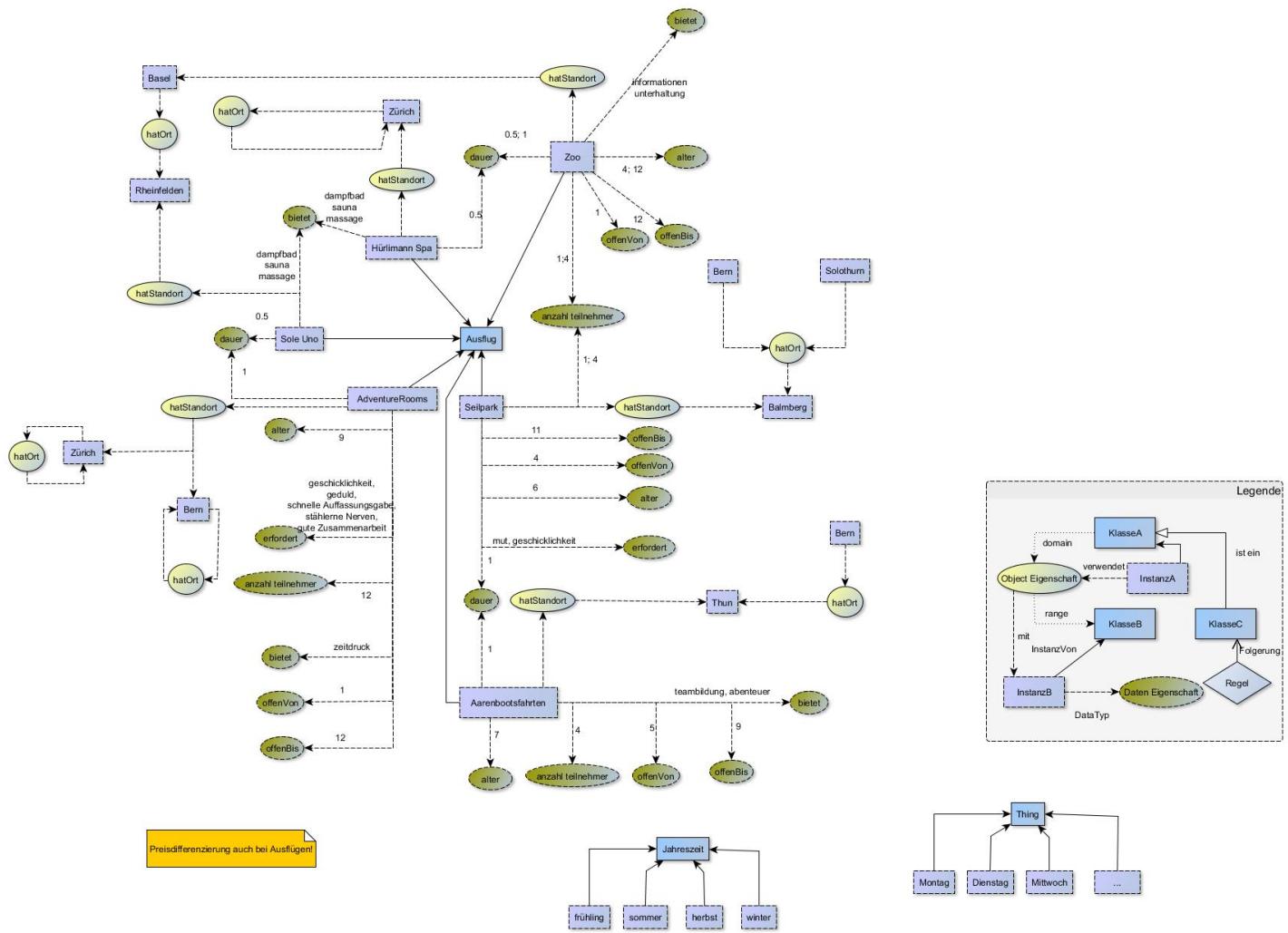
Abbildung C.17.: Semantisches Netz für das Familienbeispiel¹⁷

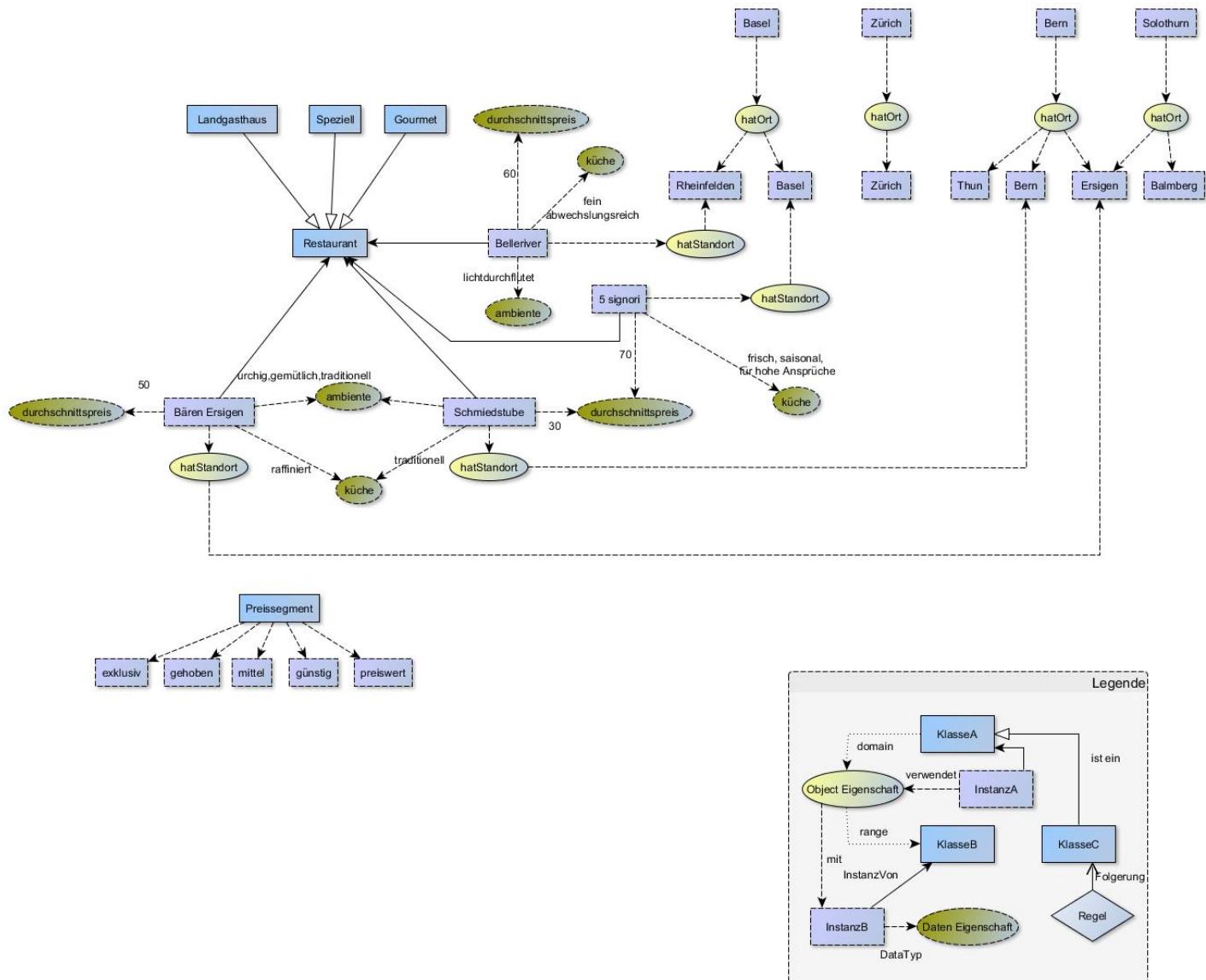
¹⁷Eigene Darstellung mittels yEd.

D. Modell

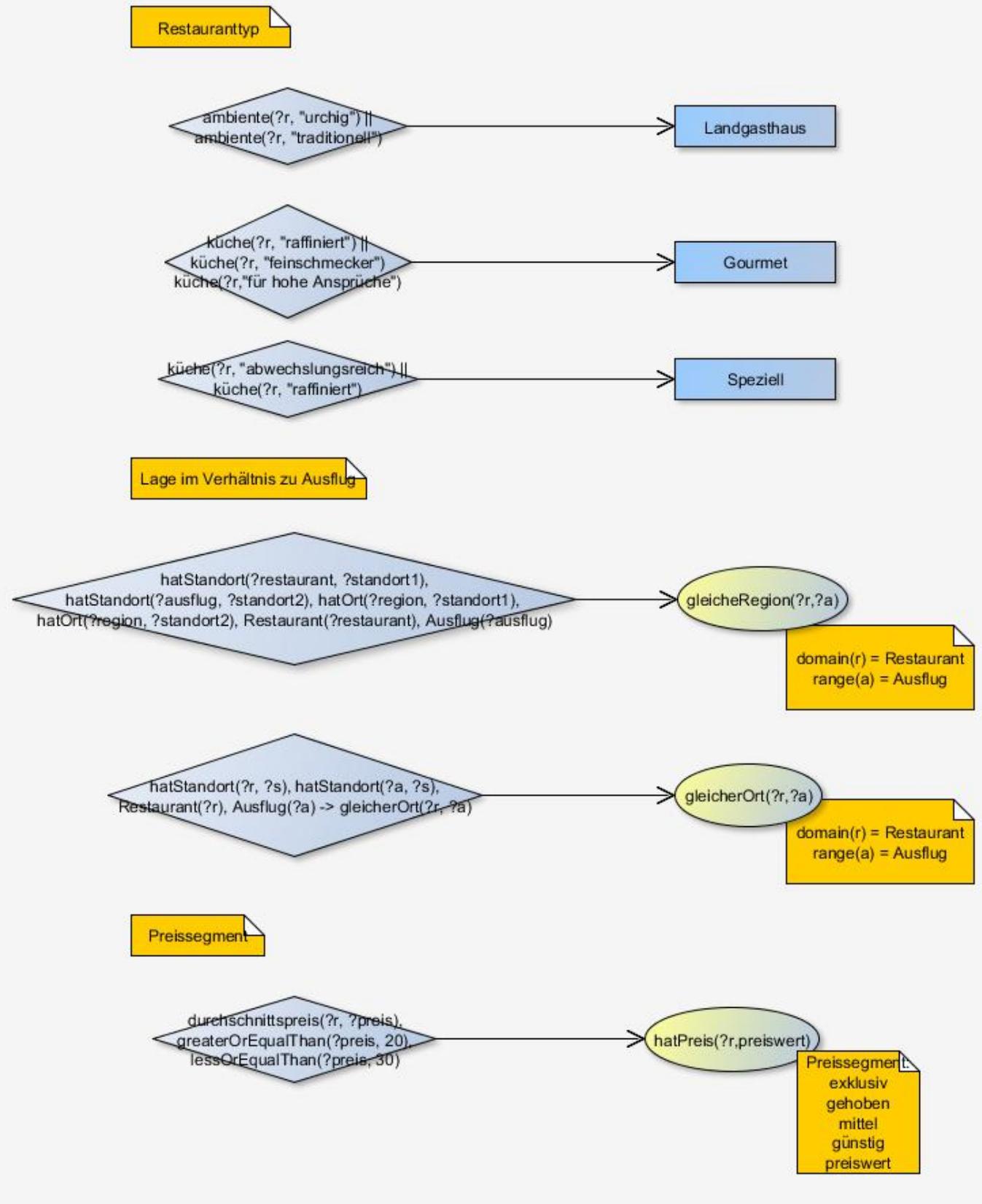
D.1. Grafiken

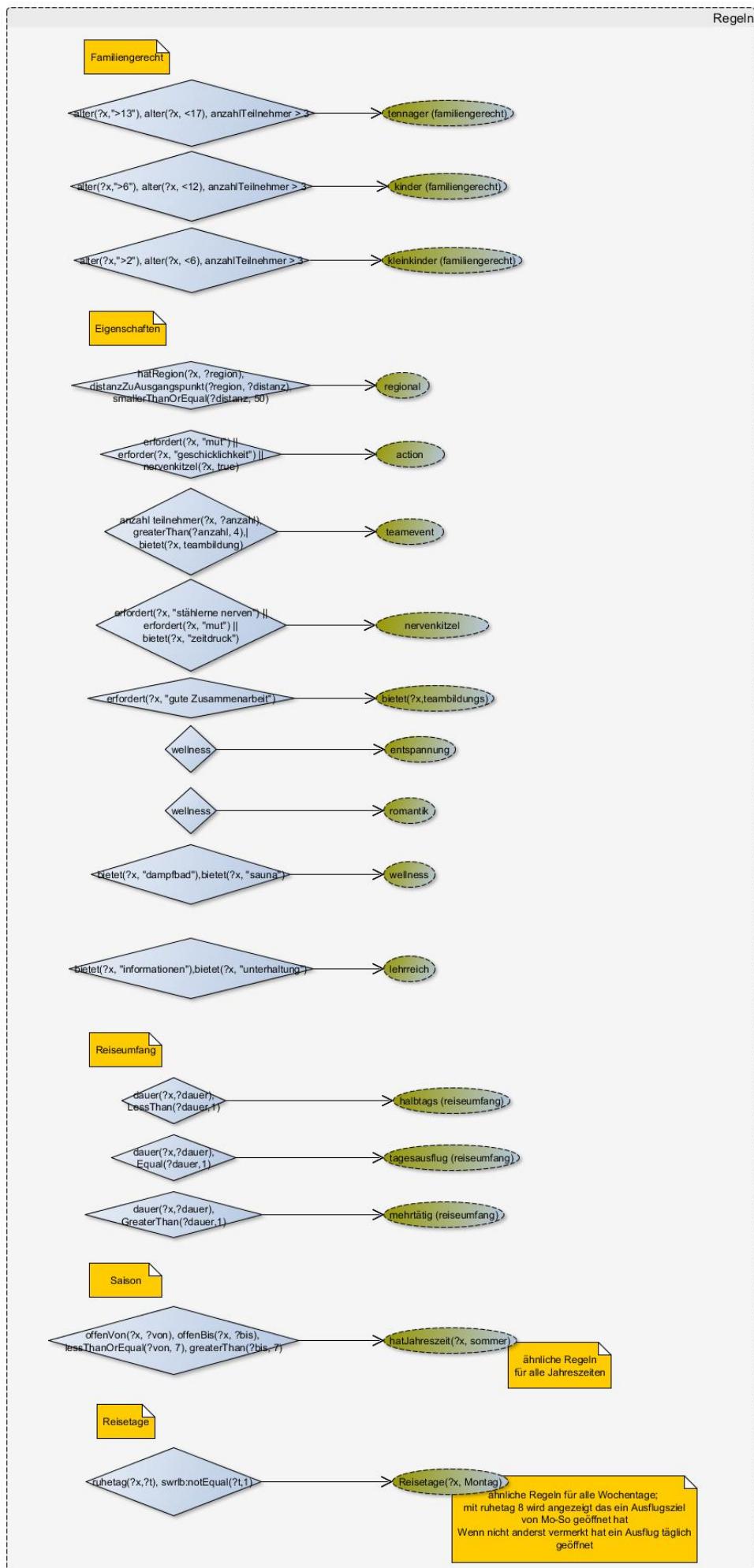






Regeln





E. Installationshandbuch

Das nachfolgende Dokument beschreibt die Installation der Lösung zur Bachelor-Thesis "Semantische Datenbanken" von Mira Günzburger und Sven Osterwalder. Bedingt durch die Projektstruktur ist das Dokument in die Kapitel *Frontend* sowie *Backend* gegliedert. Mit Backend ist die semantische Datenbank in Form des Produktes Stardog gemeint. Mit Frontend die umgesetzte Webapplikation zur Reiseplanung, bestehend aus Webserver und Applikation.

Das **Frontend** ist zum Zeitpunkt der Fertigstellung der Bachelor-Thesis bis auf Weiteres unter <http://elephantsearch.bfh.ch> verfügbar.

Das **Backend** ist zum Zeitpunkt der Fertigstellung der Bachelor-Thesis bis auf Weiteres unter <http://elephantsearch.bfh.ch:5820/> verfügbar.

Bei diesem Handbuch wird als Basis für das Backend als auch das Frontend Debian Linux in der Version 7.7 angenommen.

E.1. Backend

Beim Backend handelt es sich um die Graphdatenbank Stardog der Firma Clark & Parsia. Diese steht unter Stardog.com zur Verfügung. Es wird die Verwendung der Community-Variante empfohlen. Diese deckt alle benötigten Anforderungen und ist kostenlos verfügbar.

E.1.1. Anforderungen

Als Grundlage für das Backend kann ein beliebiger auf Unix (Linux und Apple Mac OS X) oder Microsoft Windows basierender Server zum Einsatz kommen. Die Architektur kann dabei 32- als auch 64-Bit sein.

Stardog benötigt Oracle Java in der Version 6, 7 oder 8. Die Installation und Konfiguration von Oracle Java ist nicht Teil dieses Benutzerhandbuchs.

E.1.2. Installation

Nach dem Herunterladen von Stardog in Form eines ZIP-Archivs muss in einen beliebigen Ordner entpackt werden. Als Beispiel wird `/opt/stardog` verwendet.

```
$ mkdir /opt/stardog  
$ unzip ~/downloads/stardog-2.2.6.zip /opt/stardog
```

Weiter muss die Umgebungsvariable `STARDOG_HOME` auf das zuvor erstellte Verzeichnis gesetzt werden:

```
$ export STARDOG_HOME=/opt/stardog
```

Schliesslich muss der mit der Registrierung erhaltene Lizenzschlüssel in das `STARDOG_HOME`-Verzeichnis kopiert werden:

```
$ cp ~/downloads/stardog-license-key.bin $STARDOG_HOME
```

E.1.3. Starten und Beenden des Backends

Um das Backend zu starten wird folgender Befehl verwendet:

```
$ cd $STARDOG_HOME/bin  
/opt/stardog/bin$ ./stardog-admin server start
```

Nach dem Aufruf des Befehls steht der Server via SNARL- und HTTP-Schnittstellen unter Port 5820 zur Verfügung. So kann der Server z.B. mittels *http://www.servername.tld:5820* angesprochen werden.

Um das Backend schliesslich wieder zu beenden, wird folgender Befehl verwendet:

```
$ cd $STARDOG_HOME/bin  
/opt/stardog/bin$ ./stardog-admin server stop
```

Der Status des Backends kann mittels dem folgenden Befehl abgefragt werden:

```
$ cd $STARDOG_HOME/bin  
/opt/stardog/bin$ ./stardog-admin server status
```

E.1.4. Importieren der Daten

Um die mitgelieferte Datenbank nutzen zu können, wird mit folgendem Befehl eine Datenbank erstellt und die Daten importiert:

```
$ cd $STARDOG_HOME/bin  
/opt/stardog/bin$ ./stardog-admin db create -n reiseplaner reiseplaner.owl
```

Nach dem Import der Datei steht die Datenbank unter *http://www.servername.tld:5820/reiseplaner* zur Verfügung.

E.1.5. Verwaltung

Um die Graphdatenbank zu verwalten kann entweder die Kommandozeile oder die grafische Oberfläche verwendet werden. Eine genaue Beschreibung hierzu findet sich unter docs.stardog.com¹.

E.2. Frontend

Beim Frontend handelt es sich um eine Webapplikation, welche einen lauffähigen Webserver benötigt. Um die Installation des Frontends möglichst benutzerfreundlich zu gestalten, wurde eine virtuelle Umgebung basierend auf Vagrant geschaffen. Mittels dieser kann das Frontend durch wenige Befehle initialisiert und verwendet werden. Das Frontend kann jedoch auch ohne virtuelle Umgebung genutzt werden. Nachfolgend wird jedoch nur die Nutzung mittels der virtuellen Umgebung beschrieben.

E.2.1. Anforderungen

Um das Frontend (und die virtuelle Umgebung) nutzen zu können, werden die folgenden Komponenten benötigt. Die Installation und Konfiguration dieser ist nicht Teil dieses Benutzerhandbuchs.

- *SCM Git*; eine frei verfügbare Software zur Versionskontrolle
- *Oracle VirtualBox*; eine frei verfügbare Software zur Virtualisierung
- *HashiCorp Vagrant*; eine frei verfügbare Software zur Automatisierung von Virtualisierungslösungen

¹http://docs.stardog.com/#_adminstering_stardog

E.2.2. Installation

Alle benötigten Dateien des Frontends befinden sich unter Versionskontrolle und sind unter GitHub.com verfügbar. Das Frontend kann, bedingt durch die Nutzung der virtuellen Umgebung, auf einem beliebigen Rechner installiert und genutzt werden — sofern die genannten Anforderungen erfüllt sind.

Die benötigten Dateien werden mittels den folgenden Befehlen heruntergeladen:

```
$ cd $HOME  
$ git clone https://github.com/gunzm1/ElephantSearch  
$ cd ElephantSearch
```

Nach dem Herunterladen der Dateien findet sich das Frontend im Unterordner *Solution*. Die virtuelle Umgebung wird durch die Datei *Vagrantfile* definiert und mittels folgenden Befehlen aufgesetzt:

```
$ cd $HOME/ElephantSearch/Solution  
~/ElephantSearch/Solution $ vagrant up
```

Hinweis: Je nach verwendeter Hardware und verfügbarer Internetverbindung nimmt der Prozess bis zu ca. 15 Minuten in Anspruch.

E.2.3. Starten und Beenden des Frontends

Um das Frontend zu starten werden folgende Befehle verwendet:

```
~/ElephantSearch/Solution $ vagrant ssh  
vagrant@traveling-owl:~$ sudo service nginx restart  
vagrant@traveling-owl:~$ exit
```

Nach dem Aufruf des Befehles steht das Frontend auf dem eigenen Rechner via HTTP-Schnittstelle unter <http://traveling-owl.vm> zur Verfügung. So kann das Frontend per Browser genutzt werden.

Hinweis: Ab der zweiten Verwendung des Frontends (z.B. nach Herunterfahren des Rechners) muss die virtuelle Umgebung zuerst gestartet werden:

```
~/ElephantSearch/Solution $ vagrant up  
vagrant@traveling-owl:~$ vagrant ssh  
vagrant@traveling-owl:~$ sudo service nginx restart  
vagrant@traveling-owl:~$ exit
```

Um das Frontend schliesslich zu beenden wird folgender Befehl verwendet:

```
~/ElephantSearch/Solution $ vagrant halt
```

E.2.4. Konfiguration des Frontends

Die Konfiguration des Frontends erfolgt über die Datei *Solution/frontend/js/config.js*. Es handelt sich um eine Datei im JSON-Format. Es wird empfohlen nur die Sektionen *appName* und *sparql* anzupassen. Die restlichen Sektionen betreffen rein die Applikation (in Form von Abhängigkeiten der JavaScript-Bibliotheken).

Die Sektion *appName* definiert den internen Namen der Applikation. Dies hat *nichts* mit dem Server-Namen der virtuellen Umgebung zu tun. Die Sektion *sparql* definiert die Parameter für die Verbindung zum Backend.

Arbeitsjournal

Datum	Tasks	Stunden	Bemerkungen
19.09.2014	* Anforderungsdokument überarbeitet * SSH Schlüssel vorbereitet * Erstes Einlesen in OWL/RDF und SPARQL	16	
21.09.2014	* Grundgerüst Thesis erstellt	1	
24.09.2014	* Besprechung mit Herr Eckerle * Anforderungsdokument überarbeitet	6	
26.09.2014	* Apache Stanbol auf dem BFH-Server aufgesetzt	7	
01.10.2014	* In OWL/SPARQL Grundlagen eingelesen * Erste Bausteine der Ontologie mittels Protégé entwickelt --> Als Grundlage wird die ISeminararbeit von M.Günzburger verwendet	8	
03.10.2014	* Ontologie weiterentwickelt -> Analysieren von SPARQL * Anforderungsdokument abgeschlossen * Grundgerüst des Tutorials erstellt * Fachartikel gelesen	16	
05.10.2014	* Dokumentationen gelesen * RDF * OWL * Broccoli-Artikel	7	
08.10.2014	* Tutorial begonnen * RDF * Inferenz * Div. E-Mails gesendet	6	
10.10.2014	* Tutorial weitergeschrieben * RDF * OWL * Inferenz * Resolution * SPARQL * 2. Meeting mit Herr Eckerle * Weiterarbeit an Ontologie (SPARQL)	16	
15.10.2014	* Weiterer Aufbau der Ontologie * Syntaxbaum * Klauselbaum * Unifikationsbaum * Fragen beantwortet * Wie Funktioniert Unifikation? * Was ist ein Atom? * Grafische Abbildung der Ontologie erstellt	16	
17.10.2014	* Konzept für Experten vorbereitet und Erweiterung der Ontologie * Theisdokument erweitert; Inhaltplanung * Tutorial überarbeitet, weitergeschrieben * OWL * Inferenz/Resolution * Graph-Erzeugung mit Yed Graph Editor (Exporte aus Protégé werden unübersichtlich sobald mehr Objekte drin sind) * Besprechung mit Herr Eckerle	17	
18.10.2014	* Tutorial OWL * Grafische Darstellung Syntaxbaum	4	
19.10.2014	* Tutorial SPARQL	4	
22.10.2014	* Modellierung Unifikation überarbeitet * Modellierung: Resolution/ Problemlösungsprozess/ Formelbaum * Grafische Darstellung Klauselbaum * Test Ontologie in Stanbol integriert	17	
23.10.2014	* Tutorial * Text OWL fertiggestellt * Semantische Netze fertiggestellt	4	
24.10.2014	* Tutorial: Kapitel SPARQL/Expertensysteme erweitert * Analyse und Neuorientierung nach E-Mail von Herrn Eckerle * Neubeginn Modellierung anhand des KI-Buches	12	
25.10.2014	* Tutorial: Kapitel Ontologie	4	
26.10.2014	* Tutorial * Kapitel SPARQL/Expertensysteme erweitert * Kapitel Graphen erweitert	4	
29.10.2014	* Recherchen nach besser geeigneten Tools * Stardog * Protégé mit Regeln * Technikdokumentation	16	
30.10.2014	* Modellierung Familienstammbaum (als Übung und zum Test ob unsere Tools nun wirklich alle Ihre Aufgaben erfüllen) * Grundlage Prolog * Beginn Modellierung Prolog (um die Sichtweise zu ändern) --> 1. Modellierungsschritt Prolog, dann Versuch der Umsetzung in Protégé * Präsentation Experten überarbeitet	16	
31.10.2014	* Treffen mit Experten, bisherigen Arbeit präsentiert * Weitere Modellierung mittels Prolog und Protégé * Treffen mit Herrn Eckerle --> neue Wissensdomäne * Weiterarbeit Tutorial-Dokument * Festhalten der Erkenntnisse (Fragen, Schnipsel)	16	

01.11.2014	<ul style="list-style-type: none"> * Überlegungen zu neuer Wissensdomäne * Lesen div. Artikel zu semantischen Netzen / bayesischen Netzen * Analyse div. Ontologien aus dem medizinischen Bereich * Weiterarbeit an Tutorial: Ontologie, Einführung, Wissen, Semantische Netze mit Rotem Faden versehen 	5,5
02.11.2014	<ul style="list-style-type: none"> * Dokumente nachgeführt * Weiterarbeit an Tutorial 	3,5
05.11.2014	<ul style="list-style-type: none"> * Suche nach einer neuen Wissensdomäne. Recherchen der verschiedenen Themen * Theorie Reasoner * Weiterarbeit an Tutorial 	16
07.11.2014	<ul style="list-style-type: none"> * Konzept neue Domäne: Reisen * Tutorial Dokument * Abschlussdokument Kapitel Arbeitsprozess * Besprechung mit Herrn Eckerle 	16
08.11.2014	* Tutorial: SPARQL-Kapitel erweitert	3
09.11.2014	* Tutorial: Erweiterung SWRL-Kapitel, Anpassung Formatierung	5
12.11.2014	* Weiterarbeit Modellierung des Reiseplaners: Ausflüge und Restaurants	16
14.11.2014	<ul style="list-style-type: none"> * Aufbau Vagrant * Recherchen Technologien (Javascript, JQuery, Backbones, Wizard) * Beginn Aufbau Frontend 	16
15.11.2014	<ul style="list-style-type: none"> * Thesis: Kapitel Komponenten erarbeitet * Überarbeitung Tutorial-Dokument (Korrekturen, Ergänzungen etc.) 	8
16.11.2014	<ul style="list-style-type: none"> * Backbone: Recherchen / Tests * Überarbeitung Tutorial-Dokument (Korrekturen, Ergänzungen etc.) 	4
18.11.2014	* Überarbeitung Tutorial-Dokument (Korrekturen, Ergänzungen etc.)	1
19.11.2014	<ul style="list-style-type: none"> * Analyse Technologie Benutzeroberfläche * Überarbeitung Tutorial-Dokument (Korrekturen, Ergänzungen etc.) 	16
21.11.2014	<ul style="list-style-type: none"> * Weiterentwicklung Logik Benutzeroberfläche * Design Benutzeroberfläche * Modellierung Fall 3 	8
22.11.2014	* Weiterentwicklung Logik Benutzeroberfläche	5,5
23.11.2014	<ul style="list-style-type: none"> * Fehlersuche in Stardog * Git-Repository aufgeräumt * Modellierung angepasst * Stardog Bug-Report verfasst 	7
26.11.2014	<ul style="list-style-type: none"> * Weiterarbeit an Tutorial (Einführung eines neuen Charakters: Elephant) * Starten von Stardog-Server (BFH-IT hat wohl unseren Server neu gestartet..) 	8,5
28.11.2014	<ul style="list-style-type: none"> * Weiterentwicklung Logik Benutzeroberfläche * Weiterarbeit an Thesisdokument * Besprechung mit Herrn Eckerle * Weiterentwicklung der Modellierung 	16
29.11.2014	<ul style="list-style-type: none"> * Modellierung: Attribute "familiengerecht" unterteilt, "anzahlTeilnehmer" und "saisonale" eingeführt * Allgemeine Erweiterungen der Modellierung 	5
30.11.2014	* Weiterarbeit Benutzeroberfläche: Checkboxen prüfen, Aufbau teilweise dynamischer Queries	2
02.12.2014	* Nötige Abfagen für Benutzeroberfläche ermittelt und Onto angepasst	2
03.12.2014	<ul style="list-style-type: none"> * Weiterarbeit Benutzeroberfläche * Präsentation vorbereitet * E-Mails an Herrn Anrig und Herrn Leclerc 	16
04.12.2014	<ul style="list-style-type: none"> * Weiterentwicklung GUI * Korrekturen Modell 	7
05.12.2014	<ul style="list-style-type: none"> * Präsentation Status für Herrn Leclerc * Besprechung mit Herrn Eckerle * Weiterentwicklung GUI * Planung weiteres Vorgehen * Korrekturen Modell 	16
06.12.2014	* Überarbeitung Modell und Grafik	4
07.12.2014	<ul style="list-style-type: none"> * Überarbeitung Modell und Grafik * Überarbeitung Applikation/GUI 	9,5
10.12.2014	<ul style="list-style-type: none"> * Erste Version der Book-Seite * Planung Thesisdokument * Fehlerbehebung Modell * Benutzeroberfläche * Weiterschreiben Vorgehen Thesis 	16
12.12.2014	<ul style="list-style-type: none"> * Vorgehen Thesis * Benutzeroberfläche * Besprechung mit Herrn Eckerle 	16
13.12.2014	* Administratives Thesis	1
14.12.2014	<ul style="list-style-type: none"> * Administratives Thesis überarbeiten * Lösung Thesis-Entwurf 	6
15.12.2014	<ul style="list-style-type: none"> * Vorgehen Thesis überarbeiten * Einleitung Entwurf 	6
16.12.2014	<ul style="list-style-type: none"> * Weiterarbeit Thesis * Anhang * Bibliographie * Administratives * Aufgabenstellung * Format * Korrekturen 	14,5

		<i>Soll (Ende)</i>
Total	763,25	720
<i>Total pro Person</i>	<i>381,625</i>	360
17.12.2014	* Überarbeitung Thesis (Korrekturen) * Beginn neuschreiben Kapitel Reasoner	16
18.12.2014	* Tutorial: Reasoner Kapitel weiterarbeiten	16
19.12.2014	* Grundgerüst Verteidigung erarbeitet * Entwurf Präsentation erstellt * Besprechung mit Herrn Eckerle * Tutorial: Weiterarbeit Reasoner	16
20.12.2014	* Überarbeiten Tutorial * Tableau-Kalkül analysieren	10
21.12.2014	* Überarbeiten Tutorial * Tableau-Kalkül analysieren * Tutorial: Fazit Entwurf Reasoner	5,5
22.12.2014	* Tutorial: Tableau-Kalkül eingearbeitet * Fazit Tutorial überarbeitet * TODOS über alle Dokumente abgearbeitet * Korrekturen Tutorial	18
23.12.2014	* Referenzen in Lösung Thesis korrigiert * Kapitel OWL in Tutorial fertig ergänzt * Modellierung erweitern um Ruhetage; dauer dopplet Nutzbar; Zoo; Hürlimaa * Lösung nachgeführt gemäss Model * Bilder in Lösung - Modellierung * Überarbeitung Komponenten * TODOS abgearbeitet * Weiterarbeit Korrektur Tutorial	20,5
24.12.2014	* Anpassung Benutzeroberfläche * Ausgabe von (sinnvollen) Fehlermeldungen * Implementation OR-Verknüpfungen für Relationen	2,5
26.12.2014	* Verfassen des Schlusswortes des Tutorials * Beginn Korrekturen Thesis	2,25
27.12.2014	* Fazit Thesis überarbeitet * Tutorial durchlesen (noch nicht ganz fertig)	5
28.12.2014	* Tutorial durchlesen fertig * Korrekturen Thesis	3,5
29.12.2014	* Anhänge in Thesisdokument eingearbeitet * Quellen Tutorial überarbeitet * Korrekturen Thesis	16
30.12.2014	* Quellen Thesis und Tutorial angepasst * Tutorial überarbeitet * Anhänge Thesis und Tutorial angepasst * Komponenten Thesis fertiggestellt * Schlusswort Thesis verfasst	16
03.01.2015	* Dokumente fertiggestellt * Quellen Thesis und Tutorial überarbeitet * Referenzen Thesis und Tutorial angepasst * Anhänge angepasst * Book und Poster fertiggestellt	16
04.01.2015	* Schlusskorrektur Tutorial und Thesis * Installationshandbuch * Tutorial als Projekt eingebunden	8
06.01.2015	* Ontologie Gross und Kleinschreibung überarbeitet * Bilder ersetzt in beiden Dokumenten * Schlusswort Tutorial nochmal überarbeitet	5
07.01.2015	* Schlusswort Tutorial entgültige Fassung * Glossar/ Abkürzungsverzeichnis vorbereitet * Offene Punkte Input Eltern M. Günzburger eingearbeitet	10
09.01.2015	* Abgabe vorbereitet * Schlussbesprechung mit Herrn Eckerle * Glossar/ Abkürzungsverzeichnis fertiggestellt * Installationshandbuch	16
11.01.2015	* Anpassungen nach Besprechung mit Herrn Eckerle eingearbeitet * Dokumente abschliessen	16
14.01.2015	* Arbeit gedruckt und gebunden * Präsentation für Finaltag geübt	16
16.01.2015	* Finaltag	16 geplant
23.01.2015	* Verteidigung vorbereitet und geübt	16 geplant
30.01.2015	* Verteidigung	8 geplant