Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Diplomarbeit Nr. 3425

Visuelle Filterung von Daten des Semantic Web

Steffen Bold

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. Thomas Ertl

Betreuer: Dipl.-Inf. Florian Haag
Dipl.-Inf. Steffen Lohmann

begonnen am: 10.12.2012

beendet am: 11.06.2012

CR-Klassifikation: H.2.3, H.3.3, H.3.5, H.5, I.2.4, I.3.6

Kurzfassung

Die Vision, die mit dem Semantischen Web verfolgt wird, ist, Informationen so zu strukturieren, dass diese von Computersystemen automatisiert verwertet werden können. Die Technologien hinter dem Semantischen Web, wie RDFS, OWL, RIF, SPARQL sollen diese Vision wahr werden lassen und bieten dazu starke Ausdrucksmöglichkeiten zur Speicherung und Erstellung semantischer Anfragen. Mit der Vernetzung verschiedener semantischer Datenquellen, wie beispielsweise der LOD-Cloud, die durch das Linked Open Data Projekt vorangetrieben wird, kommt man dem Ziel eines Semantischen Webs der Daten und Fakten immer näher. Dennoch gibt es außerhalb des wissenschaftlichen Bereichs nur wenige Anwendungsfälle und Zugriffsmöglichkeiten für Nutzer ohne informationstechnischem Hintergrundwissens. Visuelle Anfragesprachen können hier Abhilfe schaffen. Mithilfe von Filter-/Flow-Visualisierungskonzepten lassen sich auch komplexe Anfragen aus booleschen Ausdrücken visualisieren. Im Rahmen dieser Arbeit wurde ein Konzept für eine visuelle Anfragesprache entwickelt, das aufbauend auf Filter-/Flow-Visualisierungskonzepten, die Möglichkeiten der Visualisierung mit den Ausdrucksmöglichkeiten von SPARQL miteinander vereint. Das entwickelte Konzept wurde anhand einer Benutzerstudie evaluiert. Dabei hat sich gezeigt, dass das entwickelte Konzept sich sowohl für die Erstellung einfacher Anfragen als auch für Anfragen mit komplexen Verweisen auf Teilergebnisse einer Anfrage eignet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1. Einleitung

1.1 Motivation

Das World Wide Web ist aus unserem Alltag kaum mehr wegzudenken. Es verbindet uns, lässt uns miteinander kommunizieren und liefert uns eine nahezu unvorstellbare Menge an Informationen und Unterhaltung. Vor 20 Jahren begann die Geschichte des World Wide Web, als am 30. April 1993 die, dem Web zugrunde liegenden, Technologien für die freie und offene Nutzung am schweizer Kernforschungszentrum CERN freigegeben wurden [1]. Vor der Entwicklung des World Wide Web gab es keine Möglichkeit Informationen in digitaler Form global und miteinander vernetzt zugänglich zu machen. Ein Problem, mit dem sich auch der Erfinder des World Wide Web, Tim Berners-Lee, konfrontiert sah. Wissenschaftliche Arbeiten konnten zu dieser Zeit noch nicht einfach aus einem global und frei zugänglichen Medium, wie dem World Wide Web, heruntergeladen werden. Stattdessen war man auf eine gut ausgestattete Bibliothek angewiesen, in der die gewünschten Dokumente verfügbar waren. Wollte man auf eine referenzierte Arbeit zugreifen, so musste man hierfür ebenfalls zunächst eine Bibliothek finden, die die entsprechende Arbeit verlieh und vorrätig hatte. Um dieses Projekt zu lösen, arbeitete Tim Berners-Lee 1989 am Kernforschungszentrum CERN an einem Projekt, mit dem Ziel wissenschaftliche Arbeiten auf einfache Art und Weise in einem computergestützten System zur Verfügung zu stellen. Zur Lösung des Problems wurde auf das Hypertext-Konzept, das die Bildung von Verweisen auf andere Arbeiten ermöglichte, zurückgegriffen. Aus diesem Grundprinzip von untereinander verknüpfter Dokumente entstand das heutige World Wide Web. [2]

In den zwei Jahrzehnten seiner Existenz, hat das World Wide Web unseren Alltag, wie kaum eine andere Technologie auf dieser Welt, verändert. Das Datenvolumen, das uns dabei im World Wide Web zur Verfügung steht, wächst täglich. Man schätzt, dass sich das jährlich erzeugte Datenvolumen bis zum Jahr 2020 um den Faktor 44 des Jahres 2010 steigern wird [3]. Vergleicht man die Arbeitskosten für das Identifizieren und Löschen nicht benötigter Informationen und diejenigen des Speichervolumens, so ist das Löschen von Daten bereits heute oft teurer als der Kauf eines neuen Datenspeichers [4] [5]. Die gigantische und stetig wachsende Datenmenge im World Wide Web bringt jedoch nicht nur positive Aspekte mit sich. Durch die steigende Datenflut wird es immer schwieriger aus der gigantischen Datenmenge die gesuchten Informationen herauszufiltern. In den Anfängen des World Wide Webs konnte man dieses Problem noch mit einem Internetverzeichnis, der World Wide Web Virtual Library lösen, eine Art Telefonbuch für das Web, die von Tim Berners-Lee gestartet 1991 gestartet wurde [6]. Mit der heutigen Menge an online verfügbaren Daten und Webseiten kann es aber eine sehr zeitaufwendige Angelegenheit werden, diese in einem Art Telefonbuch nachzuschlagen. Stattdessen nutzen wir heute Suchmaschinen, rechnergestützte Sys-

teme, die das Web nach Informationen durchforschen und uns blitzschnell relevante Links zu einem Suchbegriff liefern. Dabei wird eine Eingabe aus Schlüsselwörtern mit dem Index einer Suchmaschine verglichen und die Webseiten mit übereinstimmenden Schlüsselwörtern in Form einer Liste zurückgegeben [7]. Dieses Prinzip bildet auch heute noch die Basis der meisten Suchmaschinen. Nutzer erhalten dadurch allerdings keine direkten Antworten auf Fragen, sondern nur Listen aus Webseiten, die möglicherweise eine Antwort auf die Anfrage des Nutzers enthalten.

Dieses Problem kann mit der Weiterentwicklung des World Wide Web zu einem Semantischen Web, indem die Suchanfrage eines Nutzers „verstanden“ wird, gelöst werden. Im Gegensatz zum World Wide Web, das auch als Web der Dokumente, bezeichnet wird, besteht das Semantische Web aus einer Sammlung von Fakten. Nutzer erhalten so keine Listen aus Links zu Dokumenten, sondern direkte Antworten auf ihre Anfragen. [8]

Komplexe Anfragen mit verschachtelten Bedingungen und Teilausdrücken lassen sich dennoch nur schwer in Form einer einfachen und einzeiligen Texteingabe erstellen. Textuelle Anfragen lassen sich zudem nur schwer in Einzelteile zerlegen, um beispielsweise anzeigen zu können, zu welchem Teil der Anfrage noch Ergebnisse und Antworten gefunden werden können. Ben Shneiderman erkannte dieses Problem textueller Anfragen bereits 1991 und stellte in seiner Arbeit [9] das Konzept der Filter-/Flow-Graphen vor, mit denen sich auch komplexe Anfragen anschaulich visualisieren lassen.

1.2 Zielsetzung

Im Rahmen dieser Arbeit soll ein nutzerorientierter Ansatz zur visuellen Filterung von Daten des Semantic Web entwickelt werden. Dabei wird auf Filter-/Flow-Visualisierungskonzepten aufbauend, ein Konzept zur Erstellung visueller Anfragen auf semantischen Datenquellen erarbeitet, welches auf der formalen Anfragesprache SPARQL basiert. In Rahmen einer Benutzerstudie soll das entwickelte Konzept evaluiert werden.

1.3 Kapitelüberblick

In diesem Abschnitt wird ein Überblick über die Kapitel und Struktur dieser Arbeit gegeben.

Kapitel 1, Einleitung Dieses Kapitel gibt einen Einblick in die Motivation und Zielsetzung dieser Arbeit, sowie einen Überblick über den Aufbau der weiteren Kapitel.

Kapitel 2, Grundlagen Das Grundlagenkapitel erläutert Themenfelder, die dieser Arbeit zugrunde liegen. Dabei wird insbesondere auf Formate und Sprachen zur Speicherung semantischer Informationen und der Anfrage dieser Informationen eingegangen.

Kapitel 3, Anfragesprachen und ihre Visualisierung In diesem Abschnitt der Arbeit werden verschiedene Arten von Anfragesprachen sowie deren Visualisierungsformen erläutert. Dabei wird eine Taxonomie visueller Anfragesprachen gegeben und diskutiert.

Kapitel 4, Themenverwandte Arbeiten Dieses Kapitel stellt themenverwandte Arbeiten zur visuellen Anfrageerstellung vor.

Kapitel 5, Lösungskonzept Das im Rahmen dieser Arbeit verfasste Lösungskonzept wird im 5. Kapitel vorgestellt und erläutert.

Kapitel 6, Implementierung Eine prototypische Implementierung des Lösungskonzepts wird in diesem Kapitel beschrieben.

Kapitel 7, Evaluation Das in Kapitel 5 vorgestellte Konzept zur Visualisierung semantischer Anfragen wird in diesem Teil der Arbeit anhand einer Benutzerstudie evaluiert.

Kapitel 8, Zusammenfassung und Ausblick Ein zusammenfassender Überblick über die Arbeit, sowie ein Ausblick auf weitere Fragestellungen und Ausbaumöglichkeiten, die unter anderem in fortführenden Arbeiten behandelt werden können.

2. Grundlagen

In diesem Kapitel werden die Grundlagen zum Verständnis dieser Arbeit aufgeführt und erläutert. Dabei wird auf das Semantische Web und den damit verbundenen Themengebieten eingegangen.

2.1 Das Semantische Web

Mit dem Begriff „Semantic Web“ beziehungsweise „Semantische Web“ bezeichnet man die Weiterentwicklung des World Wide Webs um eine semantische Komponente. Das Konzept hierfür geht auf Tim Berners-Lees Vorschläge zur Verbesserung des bisherigen Webs zurück, die er in seiner Arbeit [8] beschreibt.

Das semantische Web ermöglicht es, dass Computersysteme auf maschinenlesbare Informationen im semantischen Web zugreifen, sowie nicht-semantische Informationen semantisch annotieren zu können [10]. Während rein syntaktisch arbeitende Computersysteme im World Wide Web nur Anfragen mit aus Verweisen bestehenden Listen zu Dokumenten beantworten, so können semantische Systeme auf Basis des Semantischen Webs auch Anfragen mit direkten Fakten beantworten. Aus einem Web der Dokumente entsteht dadurch ein Web der Fakten und Informationen. Anfragen, wie beispielsweise „Wie hoch ist der Eiffelturm?“ lassen sich mithilfe des Semantischen Webs direkt mit der Höhenangabe des Eiffelturms (324 Meter) beantworten. Moderne Suchmaschinen, wie beispielsweise Google oder Wolfram Alpha, aber auch soziale Netzwerke, wie z. B. Facebook, setzen die Möglichkeiten des Semantischen Webs bereits heute in ihren Diensten ein. In den folgenden beiden Abbildungen 1 und 2 werden Suchanfragen, die sich bereits heute mithilfe des Semantischen Webs direkt beantworten lassen, aufgezeigt. Abbildung 1 zeigt die erwähnte Beispielanfrage „Wie hoch ist der Eiffelturm?“ und das Ergebnis dieser Anfrage mit dem Suchdienst von Google. Abbildung 2 zeigt eine Suchanfrage nach Personen des sozialen Netzwerks Facebook, die gerne Fahrrad fahren und aus der Heimatstadt des suchenden Nutzers stammen. Der semantische Suchdienst von Facebook nennt sich „Graph Search“, der von Google „Knowledge Graph“.

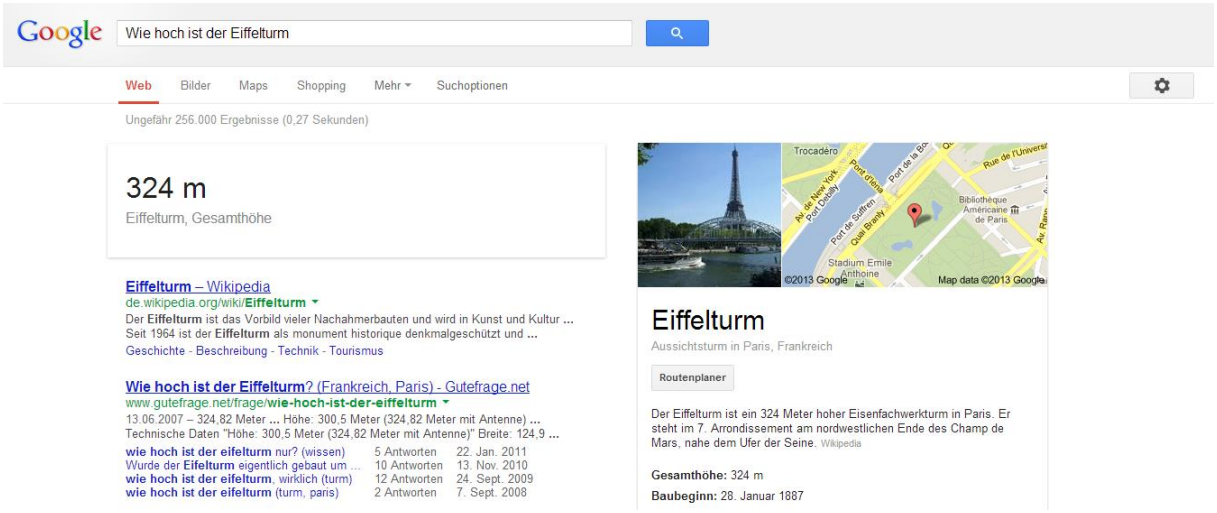


Abbildung 1: Semantische Suche am Beispiel der Suchmaschine Google. Die Semantik der Anfrage „Wie hoch ist der Eiffelturm“ wird vom Suchalgorithmus richtig interpretiert und die korrekte Antwort auf die Frage zurückgegeben.[1]

Abbildung 2: Semantische Suche am Beispiel von Facebook Graph Search. Die Semantik der Anfrage wird hier interpretiert und die Anfrage entsprechend beantwortet.[2]

Die Beispiele aus Abbildung 1 und 2 verdeutlichen, dass es sich beim Semantischen Web nicht mehr um ein reines Forschungsprojekt von Universitäten handelt, sondern um ein System, das bereits im industriellen Einsatz zu finden ist und das Potenzial besitzt unsere Suche im Web zu revolutionieren.

[1] Die abgebildete Google-Suche auf Basis des von Google genannten „Knowledge Graph“ lässt sich mit der folgenden URL aufrufen (*Stand: 09.06.2013*): https://www.google.de/search?q=Wie+hoch+ist+der+Eiffelturm

[2] Eine Einführung in Facebooks „Graph Search“ genannte Suche findet sich unter folgender URL (*Stand: 09.06.2013*): https://de-de.facebook.com/about/graphsearch

Durch die Struktur des Semantischen Web und dessen maschinenlesbare Speicherform semantischer Informationen lassen sich aber nicht nur für menschliche Nutzer intuitivere Systeme entwickeln. Auch Computersysteme können so selbst Teilnehmer des Webs werden. In diesem Zusammenhang spricht man auch vom Internet der Dinge („Internet of Things“). Ein Begriff der von Kevin Asthon entscheidend mitgeprägt wurde [9].

Für die Realisierung des Semantischen Webs müssen Computersysteme in die Lage versetzt werden semantische Informationen zu interpretieren und bearbeiten. Wie gewaltig die dabei zu lösenden Probleme sind, sieht man beispielsweise an unserer natürlichen Sprache. So einfach uns der Umgang mit unserer natürlichen Sprache fällt, so schwierig gestaltet sich das für Computersysteme. Dies liegt unter anderem daran, dass unsere natürliche Sprache eine hochgradig semantische und kontextsensitive Sprache ist. Abbildung 3 veranschaulicht die Komplexität der natürlichen Sprache an einem Beispiel.

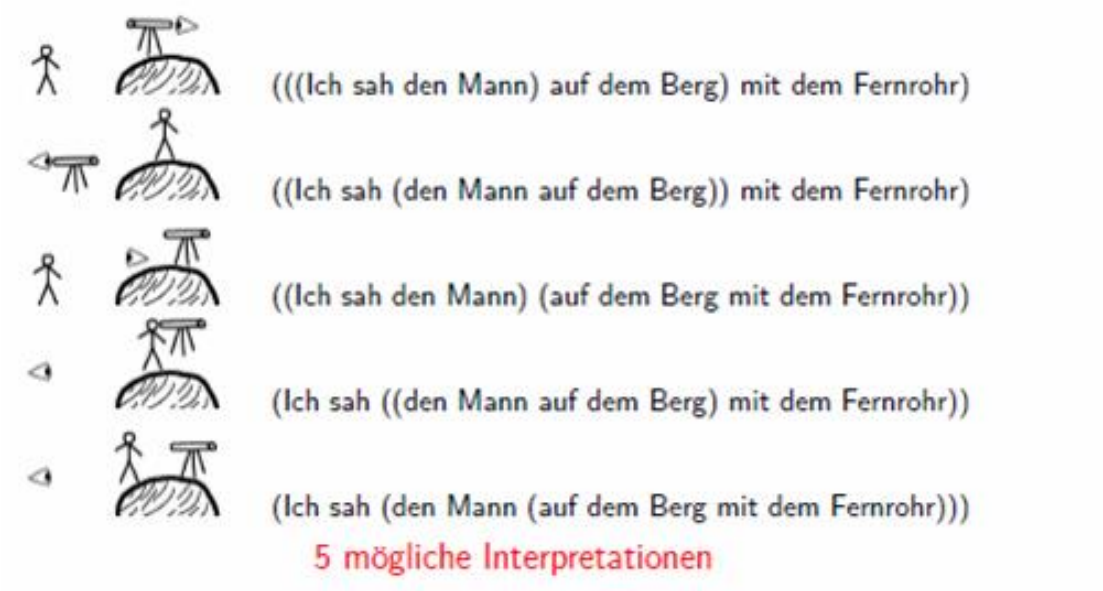


Abbildung 3: Die Komplexität unserer natürlichen Sprache wird an einem Beispiel veranschaulicht, zu dem sich fünf verschiedene Interpretationsmöglichkeiten ergeben. [12]

Ein zu Abbildung 3 weiteres Beispiel, wäre eine Person, die sagt „Ich mag Boston“. Ein einfacher Satz aus nur drei Wörtern, zudem man denken könnte, dass es keine weiteren Interpretationsmöglichkeiten, wie in Abbildung 3, gibt. Aber was hat die Person eigentlich gemeint, wenn sie sagt „Ich mag Boston“? Sie könnte meinen, dass sie die Hauptstadt des Bundesstaates Massachusetts an der Ostküste der Vereinigten Staaten mag. Sie könnte aber auch eine der anderen 26 Städte mit dem Namen Boston rund um den Globus, oder eventuell auch die Basketballmannschaft oder die Band „The Boston“, die beide ebenfalls den gleichen Namen tragen, gemeint haben.

Für einen menschlichen Gesprächspartner würde sich die Bedeutung eines Satzes, wie in Abbildung 3 oder am Beispiel „Ich mag Boston“ aus dem Kontext des Gesprächs ergeben. Unser Gehirn besitzt die Fähigkeit Zusammenhänge und andere Rahmenbedingungen, wie zum Beispiel die Tatsache, dass unser Gesprächspartner gerade aus einem Urlaubsaufenthalt in Boston zurückgekommen ist, zu erkennen. Aber auch unserem Gehirn können dabei einmal Fehler passieren, wir sprechen dann von Missverständnissen zwischen zwei Gesprächspartnern. Computersysteme dagegen stellt dies vor ein nahezu unlösbares Problem. [13]

Nicht eindeutige Bezeichner beziehungsweise Namen, wie z. B. „Boston“ stellen ein Problem für Computersysteme dar, da sich diese meist nur über den Kontext erkennen lassen. Natürliche Sprachen erlauben also Mehrdeutigkeiten. Diese Eigenschaft von natürlichen Sprachen nennt man auch Homonymie für Wörter mit unterschiedlicher Bedeutung aber mit gleicher Schreibweise beziehungsweise Aussprache und Polysemie für Wörter mit gemeinsamer Wurzel und gleicher Schreibweise aber unterschiedlicher Bedeutung. Ein Beispiel für ein Homonym ist das Wort „Tau“, das für ein Seil, morgendlicher Niederschlag oder den 19. Buchstaben des griechischen Alphabet stehen kann. Ein Beispiel für ein Polysem ist das Wort „Läufer“, das je nach Kontext einen Sportler oder eine Schachfigur bezeichnen kann. Bezeichner beziehungsweise Namen können also mehrdeutig sein. [14] [15]

Ein Objekt kann durch mehrere Bezeichner beschrieben werden, beispielsweise wenn zu einer Bezeichnung oder einem Namen mehrere Schreibweisen existieren. Das Wort „Foto“ beziehungsweise „Photo“, wäre so ein Beispiel. In der Sprachwissenschaft werden solche unterschiedlichen Schreibweisen sprachlicher Ausdrücke Allographen genannt [16]. Allein für die Universität Berkeley existieren über 50 verschiedene Schreibformen [13]. Die Schreibweise eines Wörtern kann also sehr unterschiedlich sein. Bereits beim Bilden eines Satzes wird die Schreibweise von Wörtern verändert, um Wörter in die grammatikalische Struktur eines Satzes hinzufügen zu können. Ein Wort wird in seiner Grundform, wie z. B. das Wort „gehen“, in einem Satz, wie z. B. „er geht“ grammatikalisch angepasst. Dies nennt man in der Sprachwissenschaft Morphologie [17] und stellt ein weiteres Problem für das Verständnis von Sätzen und Fragen in natürlicher Sprache da.

Im Semantischen Web geht man diese Problematik dadurch an, indem man beispielsweise Ontologien bildet, die ein Mapping zwischen einem Wort in natürlicher Sprache und einer semantischen Ressource herstellen können. Eine Ontologie bildet dabei eine Menge aus miteinander vernetzten Informationen in strukturierter und maschinenlesbaren Form. Diese Ontologien ermöglichen es so semantische Informationen zu speichern und auf diesen logische Schlussfolgerungen durchzuführen. Im folgenden Kapitel werden Ontologien, die die

Basis für semantische Informationen und deren Verarbeitung im Semantischen Web darstellen, im Detail erläutert.

2.2 Ontologien

Der Begriff „Ontologie“ kommt aus dem griechischen und bedeutet übersetzt die Lehre des Seins (on = Lehre, logos = Sein) [18]. Ontologien beschreiben dabei in der Informationswissenschaft Netzwerke aus miteinander verbundenen Informationen. Eine Ontologie beschreibt dabei einen gerichteten Graphen, der in der Informatik zur Repräsentation von Wissen verwendet werden kann. [19]

Eine einfache Ontologie, die Informationen über den Eiffelturm semantisch speichert, ist in Abbildung 4 abgebildet.

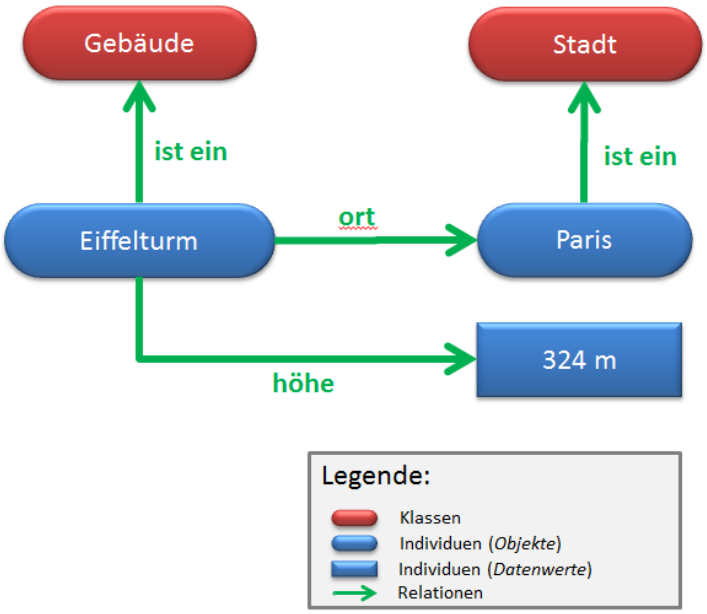


Abbildung 4: Eine einfache Ontologie mit Informationen über Höhe und Standort des Eiffelturms.

Eine Ontologie, wie in Abbildung 4 dargestellt, besteht aus über Relationen miteinander verbundenen Informationen. Die Höhe des Eiffelturms lässt sich in der abgebildeten Ontologie über die Relation „höhe“ ablesen. Relationen sind in Ontologien immer gerichtet und können entweder auf Klassen oder Individuen verweisen. Durch das Verknüpfen von Informationen über Relationen entstehen einfache Sätze aus Informationstripeln, wie z. B. „Eiffelturm höhe 324m“. Ein solches Tripel besteht dabei immer aus einem Subjekt (*hier „Eiffelturm“*), einer Relation (*hier „höhe“*) und einem Objekt (*hier „324 m“*). Eine Ontologie kann man damit auch stets als Aneinanderreihung von Tripeln betrachten.

Abbildung 5 stellt eine etwas komplexere Ontologie dar, die sich mit Informationen um den Themenbereich Kunst befasst, dar. Ein Tripel in Abbildung 5 wäre beispielsweise „Künstler erzeugt Kunstwerk“.

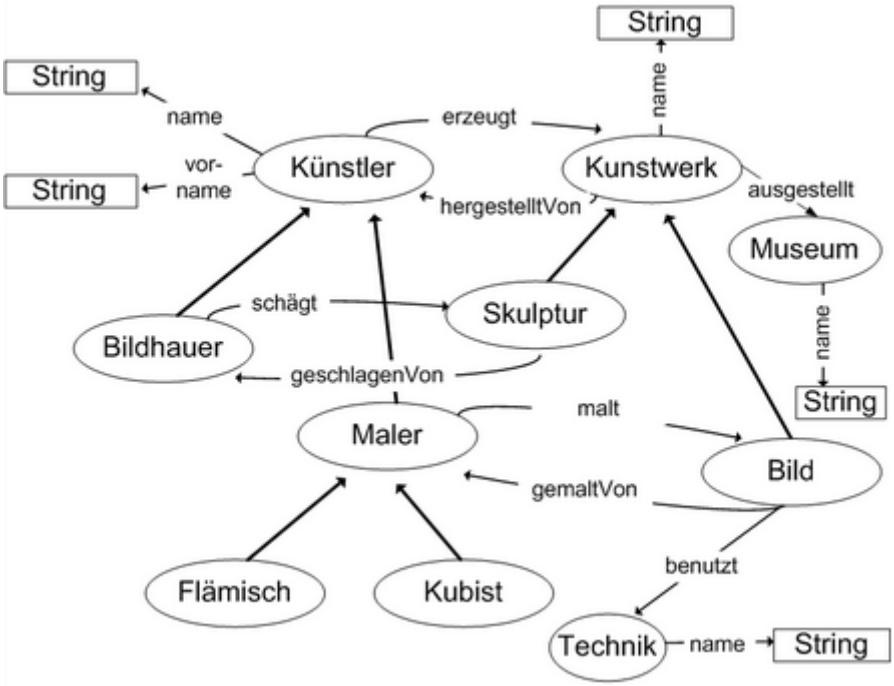


Abbildung 5: Eine Ontologie, die Informationen aus dem Themenbereich Kunst enthält. Kreise bilden hier Klassen ab, Verbindungslinien mit Beschriftung, die entsprechende Relation. Vierecke stehen für Informationen, die als einfacher Datentyp, wie z. B. einer Zeichenkette (String) vorliegt. [19]

Klassen wie z. B. „Künstler“ bilden eine Gruppe, deren Relationen auch für die Individuen, wie z. B. „Maler“ einer Klasse gelten. So lässt sich in einer Ontologie beschreiben, dass Maler Bilder malen und Bildhauer Skulpturen erschaffen, aber sowohl Maler als auch Bildhauer Künstler sind, die Kunstwerke, wie Bilder und Skulpturen erschaffen. In einigen Fällen ist es aber sinnvoller Informationen nicht in Form von Individuen oder Klassen zu speichern, sondern in Form eines einfachen Datentyps, wie z. B. einer Zeichenkette (String). Dies ist beispielsweise beim Namen eines Künstlers der Fall. Man kann daher zwischen Relationen unterscheiden, die auf Klassen und Individuen verweisen, welche als Objekt-Relationen bezeichnet werden, und denen, die auf einfache Datentypen verweisen und als Daten-Relationen bezeichnet werden.

Neben Klassen, Relationen und Individuen können Ontologien auch Axiome enthalten, die Aussagen beschreiben, die innerhalb einer Ontologie immer wahr sein müssen. Axiome erlauben es auch Vererbungsregeln zwischen Klassen oder Relationen anzuwenden, sodass ausgedrückt werden kann, dass es sich bei einem Bild um eine Unterklasse von Kunstwerken handelt. Je nachdem wie intensiv Axiome in Ontologien eingesetzt werden, spricht man von leichtgewichtigen und schwergewichtigen Ontologien. Leichtgewichtige Ontologien beschränken sich dabei auf Axiome, die nur Taxonomiebeziehungen beschreiben. Schwergewichtige Ontologien beschreiben Ontologien, die darüber hinaus weitere Axiome zur Beschreibung von Informationen und deren Beziehungen verwenden. [19]

Möchte man Ontologien, wie in Abbildung 4 und 5 dargestellt, in maschinenlesbare Form speichern, so stellt sich die Frage nach einem passenden Datenformat sowie einer dafür geeigneten Syntax. Mit dem XML-Format (eXtensible Markup Language) existiert ein bereits weitverbreitetes und standardisiertes Format zur Speicherung und zum Austausch maschinenlesbarer Informationen. In einem XML-Dokument lassen sich beliebige Informationen hierarchisch durch verschachtelte XML-Elemente strukturieren. Der Aufbau eines XML-Dokuments beginnt dabei mit einer optionalen XML-Deklaration, die die Version des verwendeten XML-Standards angibt, sowie die gewählte Zeichencodierung der XML-Datei definiert. Anschließend werden in XML-Elementen die Informationen des XML-Dokuments definiert. Im Folgenden wird der Aufbau eines XML-Dokuments an einem Beispiel veranschaulicht.

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

<data>
   <Kunstwerke>
      <Gemaelde>
         <Mona_Lisa />
         <Der_Schrei />
      </Gemaelde>
      <Filme>
         <Pulp_Fuction />
      </Filme>
      <Literatur>
         <Mobbie_Dick />
      </Literatur>
      …
   </Kunstwerke>
</data>
```

Quellcode 1: Ein XML-Dokument, das Informationen über die Taxonomie von Kunstwerken enthält.

XML-Dokumente eignen sich damit hervorragend, um Informationen in hierarchisch strukturierter und maschinenlesbarer Form zu speichern. Durch die einfache und offene Struktur eignen sich XML-Dokumente darüber hinaus als universelles plattform- und implementierungsunabhängiges Austauschformat zwischen rechnergestützten Systemen und ist entsprechend als Datenformat weit verbreitet. [20]

Zur Speicherung semantischer Informationen ist das XML-Format aber unzureichend und eignet sich daher nicht für das Semantische Web. Das liegt daran, dass XML-Elemente

außer durch ihre hierarchische Ordnung keine semantischen Informationen tragen. Die Bedeutung des Inhalts des in Quellcode 1 abgebildeten XML-Dokuments ergibt sich rein aus den in natürlicher Sprache gehaltenen Bezeichnern und der hierarchischen Struktur. Da die natürliche Sprache aber komplex und mehrdeutig sein kann, eignet sich ein solches XML-Dokument nur für menschliche Nutzer. Das folgende Beispiel macht deutlich, wie schwer sich ein XML-Dokument und dessen Werte interpretieren lassen, wenn die Semantik der XML-Elemente nicht durch die natürliche Sprache definiert wird.

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

<1232221321132>
   <2313223132121331>
      <555466645465>
         <213 />
         <214 />
      </555466645465>
      <34533534>
         <215 />
      </34533534>
      <242434323423423422442>
         <216 />
      </242434323423423422442>
      …
   </2313223132121331>
</1232221321132>
```

Quellcode 2: XML-Dokument aus Quellcode 1 ohne natürlich sprachliche Bezeichner.

Des Weiteren sind zentrale und abgeschlossene Dokumente für einen Einsatz in einem dezentral strukturierten Web ungeeignet. Ein Dateiformat im Semantischen Web muss es erlauben, abgespeicherte Ontologien dezentral miteinander zu verknüpfen und zu erweitern. Für diese Aufgaben eignet sich das XML-Format nicht. Das XML-Format kann aber aufgrund seiner Eigenschaften als Metasprache, als Ausgangsformat zur Spezifikation eines für das Semantische Web geeigneteren Formats, verwendet werden. [20]

Im folgenden Kapitel wird RDF, eine einfache Sprache zum Beschreiben und Speichern von Ontologien, vorgestellt. Spracherweiterungen wie RDFS und OWL, die die Ausdrucksmöglichkeiten erhöhen, werden in den daran anschließenden Kapiteln erläutert.

2.3 Resource Description Framework - RDF

Die Abkürzung RDF steht für Resource Description Framework und beschreibt eine formale graphenbasierte Sprache zur Beschreibung semantischer Informationen, die auf die XML-Syntax aufbaut. Der erste Entwurf für RDF wurde 1997 beim W3C eingereicht und im Februar 1999 als W3C Recommendation verabschiedet. Aktuell liegt RDF in der W3C Recommendation von 2004 vor. [21]

Durch RDF lassen sich einfache logische Aussagen in Form von Tripeln definieren, wodurch ein gerichteter Graph, wie beispielsweise in Abbildung 4 gezeigt, definiert wird. Im Folgenden wird der Aufbau eines RDF-Dokuments an einem Beispiel erläutert und dabei auf die Konzepte von RDF eingegangen. Aufbau und Konzepte lassen sich in der vom W3C verabschiedeten Spezifikation [21], auf die dieses Kapitel aufbaut, nachlesen.

2.3.1 Aufbau eines RDF-Dokuments

RDF baut auf dem XML-Format auf. Ein RDF-Dokument beginnt daher mit einer optionalen XML-Deklaration. Eine XML-Deklaration enthält Angaben zur Version des verwendeten XML-Standards und der für das Dokument verwendeten Zeichencodierung.

Anschließend kann ein optionale Dokumenttypdefinition (Document Type Decleration – kurz DTD beziehungsweise DOCTYPE) definiert werden. Wie der Name vermuten lässt, wird innerhalb einer Dokumenttypdefinition der Typ eines Dokuments spezifiziert. Für RDF-Dokumente bedeutet dies <http://www.w3.org/1999/02/22-rdf-syntax-ns#RDF> oder abgekürzt rdf:RDF. Im Fall der abgekürzten Schreibweise muss der Namespace spezifiziert werden. Namespaces können innerhalb einer Dokumenttypdefinition über eine Entity-Deklaration spezifiziert werden. Eine Entity-Deklaration besteht aus beliebig vielen Entity-Referenzen zur Spezifikation von Namespaces. Im Quellcodebeispiel 3 wird die Deklaration einer Dokumenttypdefinition veranschaulicht.

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

Quellcode 3: Ein Beispiel für eine DOCTYPE-Deklaration innerhalb eines RDF-Dokuments.

Im Anschluss an eine optionale XML-Deklaration und Dokumenttypdefinition folgt ein einleitendes RDF-Tag, das die innerhalb des Dokuments spezifizierten Informationen umschließt.

Darüber hinaus kann ein RDF-Tag die im Dokument verfügbaren XML-Namensräume (xmlns) als Attributwerte enthalten. Ein RDF-Dokument, das die Höhe und den Standort des Eiffelturms beschreibt, ist in Quellcodebeispiel 4 abgebildet.

```
<rdf:RDF
xmlns ="http://example.de#"
xml:base ="http://www.example.de"
xmlns:other-example ="http://www.ein-anderes-beispiel.de"
xmlns:owl ="http://www.w3.org/2002/07/owl#"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">

<rdf:Description rdf:about="#Eiffelturm">
   <befindetSichIn rdf:ressource="#Paris" />
   <hoehe>324m</hoehe>
</rdf:Description>

<rdf:Description rdf:about="#Paris">
   <befindetSichIn rdf:ressource="#Frankreich" />
</rdf:Description>

<rdf:Description rdf:about="#Frankreich" />

</rdf:RDF>
```

Quellcode 4: Die Spezifikation eines RDF-Dokuments an einem Beispiel. Innerhalb eines RDF-Tags werden Informationen über den Eiffelturm beschrieben. Dabei wird Standort und Höhe spezifiziert.

2.3.2 RDF Konzepte

Mit der Strukturierung von Informationen in Form eines gerichteten Graphen wurde bereits ein Konzept des Resource Description Framework im Kapitel 2.2 genannt. Weitere Konzepte werden im Folgenden aufgelistet und erläutert.

2.3.2.1 Uniform Resource Identifier (URI)

Eins der wichtigsten Konzepte im Semantischen Web ist die Verwendung von URIs (Uniform Resource Identifier) als universelle Bezeichner für Ressourcen. Die Aufgabe einer URI ist es eindeutig auf eine Ressource zu verweisen. In unserem Alltag sind wir mit URIs bereits bestens vertraut, wenn wir beispielsweise die Adresse einer Webseite (URL – „Uniform Resource Locator“) in unseren Browser eingeben oder ein Buch über dessen ISBN (International Standard Book Number) bestellen wollen. Während eine URL auf den eindeutigen Ort einer Ressource verweist, beschreibt eine URN (Uniform Resource Name), wie beispielsweise

eine ISBN, eindeutig auf den Namen einer Ressource. Bei einer URI kann es sich um eine URL oder URN oder beides zugleich handeln.

2.3.2.2 Namespace

Durch Namensräume können Namenskonflikte vermieden werden. Beispielsweise zwischen einem Eiffelturm in Ontologie A und einem Eiffelturm, der in Ontologie B deklariert wird. Da Namespaces mitunter sehr lang sein können, werden Namespaces oft mit Präfixen abgekürzt. Präfixe lassen sich dabei frei wählen und als Abkürzung für eine URI verwenden. In den vorangegangenen Beispielen wurden bereits Präfixe verwendet, so wurde z. B. der RDF-Namespace über den Präfix rdf abgekürzt. Bei der Deklaration einer Ressource kann so auf die abgekürzte Schreibweise rdf:Description zurückgegriffen werden, statt eine Ressource über die vollständige URI (http://www.w3.org/2000/01/rdf-schema#Description) deklarieren zu müssen. In Quellcodebeispiel 5 sind die wesentlichen Aspekte zum Verständnis von Namespaces abgebildet.

```
<rdf:RDF
xmlns ="http://example.de#"
xml:base ="http://www.example.de"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

…
<rdf:Description rdf:about="#Eiffelturm">
   <befindetSichIn rdf:ressource="#Paris" />
   <hoehe>324m</hoehe>
</rdf:Description>

…
</rdf:RDF>
….
```

Quellcode 5: Beispiel zur Erläuterung von Präfixen und Namespaces.

Die URI des Default-Namespace eines XML- beziehungsweise RDF-Dokuments wird durch den Wert der Deklaration xmlns = URI gebildet. Der Default-Namespace bezieht sich auf alle Deklerationen, die ohne ein Präfix beginnen. [22]

Die xml:base Deklaration bildet den Namespace für alle relativen URIs innerhalb des RDF-Dokuments, auf die durch eine rdf:about, rdf:resource, rdf:ID oder rdf:datatype Deklaration verwiesen wird. [23]

2.3.2.3 Ressourcen

Eine Ressource in RDF beschreibt ein beliebiges Objekt und wird mit der Deklaration rdf:Description eingeleitet, in der mit rdf:about auch die eindeutige URI für die Ressource spezifiziert wird. Eine Ressource lässt sich dabei wie im folgenden Quellcodebeispiel deklarieren.

```
<rdf:Description rdf:about="#Eiffelturm" />
```

Quellcode 6: Deklaration einer Ressource in RDF.

Die vollständige URI der Ressource bildet sich aus dem Base-Namespace und dem Wert des Attributs rdf:about. Ist der Base-Namespace beispielsweise http://www.example.de, dann ist die URI für die in diesem Beispiel definierte Ressource http://www.example.de#Eiffelturm.

Anstatt über das Attribut rdf:about kann die URI einer Ressource auch mit dem Attribut rdf:ID definiert werden. Beim Zusammenfügung der URI aus Base-Namespace und rdf:ID wird automatisch eine Raute vor die Zeichenkette gehängt. Mit dem rdf:ID Tag lässt sich die Ressource „Eiffelturm" äquivalent, wie folgt definieren.

```
<rdf:Description rdf:ID="Eiffelturm" />
```

Quellcode 7: Deklaration einer Ressource in RDF über rdf:ID.

2.3.2.4 Relationen

Mit Relationen lassen sich Ressourcen miteinander in Beziehung setzen. Im oberen Beispiel wurde die Ressource „Eiffelturm" über die Relation „befindetSichIn" mit der Ressource „Paris" in Verbindung gebracht. Relationen können aber nicht nur Ressourcen miteinander verbinden, sondern diese auch mit Literalen verknüpfen. Literale bezeichnen in einem RDF-Dokument einfache Datenwerte, die nicht in Form von Ressourcen vorliegen.

2.3.2.5 Literale und Datentypen

Ein Literal beschreibt innerhalb eines RDF-Dokuments einen einfachen Datenwert, wie zum Beispiel eine Zeichenkette oder eine Zahl. Im Quellcodebeispiel 5 ist die Höhenangabe des Eiffelturms beispielsweise als eine Zeichenkette angegeben. Der Standarddatentyp für Literale sind Zeichenketten. Soll ein Literal von einem bestimmten Datentyp sein, so kann man dies über das Attribut rdf:parseType deklarieren. In Quellcodebeispiel 8 sind zwei Höhenangaben über ein Literal deklariert worden. Da die beiden Werte ohne Deklaration eines Datentyps als Zeichenkette interpretiert werden, handelt es sich bei den beiden Angaben um unterschiedliche Werte („0345" != „345").

```
<hoehe>0345</hoehe>

<hoehe>345</hoehe>
```

Quellcode 8: Zwei RDF-Literale, die standardmäßig als String interpretiert werden und daher unterschiedliche Datenwerte beschreiben.

Durch die Angabe des Datentyps „Integer“ werden beide Höhenangaben als äquivalente Zahlen interpretiert (0345 = 345).

```
<hoehe rdf:parseType=“Integer“>0345</hoehe>

<hoehe rdf:parseType=“Integer“>345</hoehe>
```

Quellcode 9: Zwei als Integer interpretierte RDF-Literale, die einen äquivalenten Datenwert beschreiben.

2.4 Resource Description Framework Shema - RDFS

Das Resource Description Framework (RDF), dessen Aufbau und Konzepte in Kapitel 2.3 erläutert wurden, ermöglicht es Informationen in Form von miteinander verbundenen Ressourcen zu definieren, wodurch ein gerichteter Graph entsteht. Die Ausdrucksmöglichkeiten des Resource Description Framework (RDF) sind allerdings auf einfache Aussagen beschränkt. Schemawissen, wie beispielsweise Klassen und weitere schematische Eigenschaften von Ressourcen, lässt sich mit RDF nicht beschreiben. Aus diesem Grund führte man mit RDFS (Resource Description Framework Schema) eine Spracherweiterung zu RDF ein, mit dem sich Schemawissen innerhalb einer Ontologie beschreiben lässt. RDFS basiert dabei auf den gleichen Konzepten wie RDF. Jedes RDF konforme Dokument ist damit auch ein RDFS konformes Dokument. Informationen dieses Kapitel beruhen dabei auf der vom W3C verabschiedeten RDF-Schema Spezifikation [22] und können dort nachgelesen werden.

2.4.1 RDFS-Konzepte

Im Folgenden werden die RDFS-Konzepte vorgestellt, mit denen das Resource Description Framework (RDF) um Schemawissen erweitert wird.

2.4.1.1 Klassen und Individuen

Im folgenden Beispiel wird das bereits aus Quellcodebeispiel 4 bekannte RDF-Dokument um Klassen und Instanzen erweitert. Dabei wurden dem Beispiel die Klassen „Bauwerk“, „Stadt“, „Land“ und „Ort“ hinzugefügt. Durch rdfs:subClassOf kann für eine Klasse eine übergeordnete Klasse spezifiziert werden. Im Beispiel sind „Stadt“ und „Land“ Unterklassen der Klasse „Ort“. Jedes Individuum dieser beiden Klassen ist damit auch ein Individuum der Klasse „Ort“.

```
<rdfs:Class rdf:about="#Bauwerk" />

<rdfs:Class rdf:about="#Stadt">
   <rdfs:subClassOf rdf:resource="#Ort" />
</rdfs:Class>

<rdfs:Class rdf:about="#Land">
   <rdfs:subClassOf rdf:resource="#Ort" />
</rdfs:Class>

<rdfs:Class rdf:about="#Ort" />

<rdf:Description rdf:about="#Eiffelturm">
   <rdf:type rdf:resource="#Bauwerk" />
   <befindetSichIn rdf:ressource="#Paris" />
   <hoehe>324m</hoehe>
</rdf:Description>

<rdf:Description rdf:about="#Paris">
   <rdf:type rdf:resource="#Stadt" />
   <befindetSichIn rdf:ressource="#Frankreich" />
</rdf:Description>

<rdf:Description rdf:about="#Frankreich">
   <rdf:type rdf:resource="#Land" />
</rdf:Description>
```

Quellcode 10: Erweiterung des RDF-Dokuments aus Quellcode 4 durch das RDFS Klassen-Konzept.

2.4.1.2 Domain- und Range-Beschränkungen

Domain- und Range-Restriktionen ermöglichen es festzulegen, dass nur bestimmte Klassen und Relationen miteinander verbunden werden können. Eine Relation kann nur von den Klassen oder Individuen einer Klasse verwendet werden, die nicht im Widerspruch mit der in der Relation definierten Domain steht. Eine Relation kann nur auf eine Klasse oder ein Individuum verweisen, die nicht im Widerspruch mit dem definierten Range steht.

Im folgenden Quellcodebeispiel wird eine Relation „backen“ spezifiziert, deren Domain auf Instanzen der Klasse „Bäcker“ und deren Range auf Instanzen der Klasse „Backwaren“ beschränkt ist. Tripel der Form „Ein Bäcker backt Backwaren“ ist dadurch erlaubt. Eine Aussa-

ge mit vertauschtem Subjekt und Objekt, wie zum Beispiel „Backwaren backen den Bäcker“ ist dagegen durch die Domain- und Range-Beschränkungen nicht erlaubt.

```
<rdf:Description rdf:about=“backen“>
   <rdf:type rdf:resource=“rdf:Property“ />
   <rdfs:domain rdf:resource=“Bäcker“ />
   <rdfs:range rdf:resource=“Backwaren“ />
</rdf:Description>
```

Quellcode 11: Eine Relation mit Domain- und Range-Beschränkungen.

2.4.1.3 Klassen- und Relations-Hierarchie

RDFS erweitert die Ausdrucksmöglichkeiten von RDF unter anderem um die Möglichkeit zur Definition von Hierarchien zwischen Klassen und zwischen Relationen. Klassenhierarchien wurden bereits im Unterkapitel 2.4.1.1 vorgestellt. Das gleiche Prinzip lässt sich auch auf Relationen anwenden. Zur Deklaration von Relationshierarchien wird die Relation rdfs:subPropertyOf verwendet.

2.6 Web Ontology Language - OWL

Die Web Ontology Language (OWL) erweitert RDFS um ausdrucksstärkere Informationsbeziehungen. Gegenüber RDFS bietet OWL ein erweitertes Vokabular wie zum Beispiel symmetrische, asymmetrische und transitive Relationen. Darüber hinaus bietet OWL gegenüber RDFS Möglichkeiten zur Beschreibung von Metainformationen über Annotationen. Beispielsweise kann einer Ontologie eine Versionsnummer vergeben werden. Mit erweiterten Möglichkeiten zur Deklaration von Axiomen in OWL lassen sich auch verstärkt Schlussfolgerungen, auch Inferenzbildung genannt, aus den gespeicherten Informationen ziehen. Unter Inferenzbildung versteht man dabei die automatische Bildung neuer Relationen oder das Entfernen bestehender Relationen auf Basis der Daten und Regeln einer Ontologie. Das W3C gibt als einfaches Beispiel für eine Inferenzbildung eine Beispielontologie an, in der definiert wird, dass es sich bei der Ressource „Flipper“ um ein Individuum der Klasse „Delfin“ handelt. Wird innerhalb dieser Ontologie definiert, dass alle Individuen der Klasse „Delfin“ auch Individuen der Klasse „Säugetier“ sind, kann daraus durch Inferenzbildung geschlossen werden, dass auch das Individuum „Flipper“ zur Klasse „Säugetier“ gehört [24]. Während diese einfache Inferenz sich auch in RDFS beschreiben lässt, lassen sich mit OWL weit aus komplexere Inferenzen herstellen. [23]

Des Weiteren zeichnet sich OWL durch seinen dezentralen Aufbau aus. Damit eignet sich OWL insbesondere für den Einsatz im ebenfalls dezentral strukturierten World Wide Web. Mit OWL lassen sich Ontologien verteilt erstellen, zusammenzufügen und auch erweitern.

Die in diesem Kapitel verfassten Aussagen lassen sich in der vom W3C veröffentlichten Spezifikation zu OWL [23] nachlesen.

Die Schreibweise der Abkürzung OWL macht am Anfang wahrscheinlich etwas stutzig, müsste die abgekürzte Schreibweise doch eigentlich WOL lauten. Die Idee für die Schreibweise OWL stammte von Tim Finin, Professor an der University of Maryland (Baltimore Country), der diesen Vorschlag über die E-Mail Liste des W3C einreichte. Für OWL sprachen nach Finins Vorschlag vier Punkte [25]:

„[…]I prefer the three letter WOL to the longer SWOL. How about OWL as a variation. The words would be the same (Ontology Web Language) but it has several advantages: (1) it has just one obvious pronunciation which is easy on the ear; (2) it opens up great opportunities for logos; (3) owls are associated with wisdom; (4) it has an interesting back story. […]“

[Auszug aus http://lists.w3.org/Archives/Public/www-webont-wg/2001Dec/0169.html]

1. Die Aussprache der Abkürzung OWL ist im englischen Sprachraum klar definiert, da das Wort für Eule steht.
2. Es ist ein Akronym, das sich für die Erstellung eines Logos eignet.
3. OWL ist das englische Wort für Eule. Eulen werden mit Weisheit assoziiert.
4. Es existiert eine interessante Hintergrundgeschichte zu OWL. Worauf er auf die One World Language angespielt hat, einem ersten Versuch aus dem Jahre 1970 zur Entwicklung einer universellen Sprache für Wissensrepräsentationen.

Während sich die Namensgebung aus dem Archiv der W3C Mailingliste nachvollziehen lässt, hält sich nach wie vor das Gerücht, die Abkürzung wäre an die literarische Figur der Eule aus A.A. Milne's „Winnie the Pooh“ angelehnt, die als einziges Tier im Wald ihren Namen „Eule“ schreiben kann, dabei allerdings stets einen Buchstabendreher macht [25]. Diese Legende geht wahrscheinlich auf den Eintrag einer nicht mehr länger betreuten FAQ Seite des W3Cs zurück [26].

2.6.1 Syntax

Syntaktisch erweitert OWL RDF/RDFS um weitere Ausdrucksmöglichkeiten und ist dabei vollständig kompatibel zu den bereits in RDF und RDFS bekannten Konzepten sowie deren Syntax. OWL erweitert die syntaktischen Möglichkeiten zur Beschreibung einer Ontologie um weitere Sprachdialekte. Da Ontologien im Grunde gerichtete Graphen abbilden, lässt sich im Prinzip jede Syntax verwenden, die in der Lage ist Informationstripel zu beschreiben. Eine Übersicht über die verschiedenen Syntaxen von OWL bietet Abbildung 6.

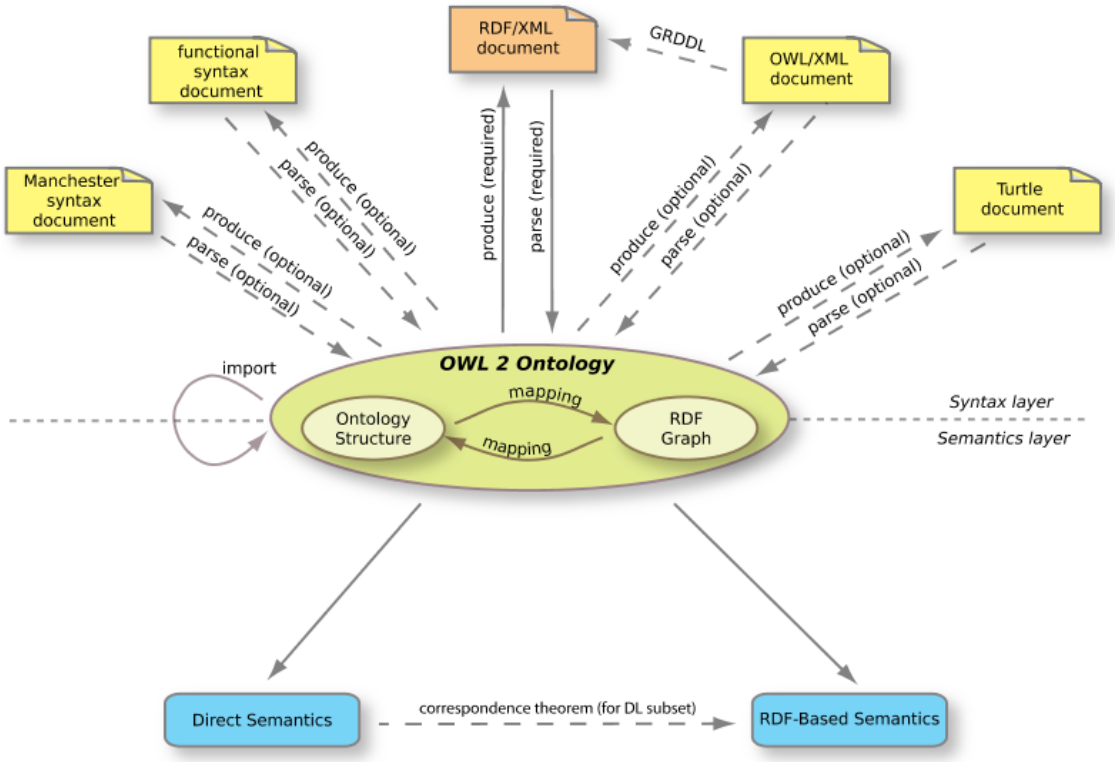


Abbildung 6: Eine Übersicht über die verschiedenen OWL-Syntaxen zur Beschreibung von Ontologien. [23]

Die am weitesten verbreitete und zu RDF und RDFS kompatible Syntax ist die RDF/XML-Syntax, die bereits zur Erläuterung von RDF und RDFS in den letzten Kapiteln vorgestellt wurde. Um keine Verwirrung zwischen unterschiedlichen Dialekten aufkommen zu lassen, wird im Rahmen dieser Arbeit die Syntax von OWL ebenfalls in der bereits vorgestellten RDF/XML-Syntax erläutert.

Eine Ontologie in OWL kann nach den W3C Recommendation, als Datei mit der Endung *.owl oder *rdf vorliegen. Der Aufbau eines OWL- und RDF- beziehungsweise RDFS-Dokuments ist bis auf die Spracherweiterungen von OWL und RDFS identisch. Dementsprechend wird in diesem Abschnitt und dessen Unterkapiteln nur auf die syntaktischen Erweiterungen von OWL eingegangen.

2.6.1.1 Ontologie-Header

Eine Ontologie in OWL enthält innerhalb des RDF-Tags einen Ontologie-Header. Dieser kann Informationen über die Ontologie selbst enthalten. Beispielsweise kann ein Ontologie-Header den Namen der Ontologie, eine Beschreibung und Informationen zur Versionierung, sowie Referenzen zu anderen Ontologien, enthalten. Das folgende Beispiel veranschaulicht die Deklaration eines Ontologie-Headers.

```
<owl:Ontology rdf:about="ontology_example">
   <rdfs:label>My Ontology</rdfs:label>
   <rdfs:comment>An example Ontology</rdfs:comment>
   <owl:versionInfo>1.1</owl:versionInfo>
   <owl:priorVersio rdf:resource="http://www.example.de/veraltete_version"/>
   <owl:imports rdf:resource="http://www.ein-anderes-beispiel.de"/>
</owl:Ontology>
```

Quellcode 12: Ein Beispiel eines Ontologie-Headers.

2.6.2 OWL-Elemente

Klassen, Relationen und Entitäten bilden die Grundbausteine von Ontologien, mit denen sich die zu speichernden Informationen modellieren lassen. Im Folgenden werden die drei Grundbausteine einer Ontologie einzeln erläutert.

2.6.2.1 Klassen

Mit Klassen lassen sich Gruppen von Individuen beschreiben, die zum Beispiel auf die gleichen Relationen zurückgreifen können.

Eine Möglichkeit eine Klasse in OWL zu erstellen ist einer RDF-Ressource über die Relation rdf:type mit der OWL-Definition für Klassen zu verbinden.

```
<rdf:Description rdf:about="#Student">
   <rdf:type rdf:resource="&owl;Class"/>
</rdf:Description>
```

Quellcode 13: Eine OWL-Klassendeklaration, die an die Syntax von RDFS Klassendeklaration angelehnt ist.

Im oberen Beispiel wird eine Klasse mit dem Namen „Student“ erzeugt. Der Ausdruck erzeugt in der Ontologie das folgende RDF-Tripel:

```
#Student        rdf:type        owl:Class.
```

Quellcode 14: Ein Informationstripel, das aus der in Quellcode 13 beschrieben Deklaration erzeugt wird.

In Form einer kürzeren Schreibweise lässt sich ein äquivalenter Ausdruck in OWL, wie im folgenden Quellcodebeispiel deklarieren.

```
<owl:Class rdf:about="#Student"/>
```

Quellcode 15: OWL-Klassendeklaration in Kurzschreibweise.

2.6.2.1.1 Äquivalente Klassen

Äquivalente Klassen lassen sich in OWL durch owl:equivalentClass deklarieren.

```
Student owl:equivalentClass Studierender
```

Quellcode 16: Eine Äquivalenzrelation zwischen zwei OWL-Klassen.

Dadurch lässt sich definieren, dass es sich bei den Klassen Student und Studierender um dieselbe Klasse handelt. Alle Individuen der Klasse Student sind damit auch Individuen der Klasse Studierender und umgekehrt.

2.6.2.1.2 Disjunkte Klassen

Eine disjunkte Beziehung zwischen zwei Klassen spezifiziert, dass es keine Individuen geben darf, die Individuen beider zueinander disjunkter Klassen sind. An einem Beispiel wird der Anwendungsfall disjunkter Klassenbeziehungen deutlich. Eine Beispielontologie soll beschreiben, dass eine Person entweder eine Frau oder ein Mann sein darf. Die Schnittmenge der Individuen beider Klassen ist damit gleich der leeren Menge. Durch die Relation owl:disjointWith lässt sich eine disjunkte Beziehung zwischen zwei Klassen, wie im folgenden Quellcodebeispiel, deklarieren.

```
<owl:class rdf:about="#Person"/>

<owl:Class rdf:about="#Mann">
   <rdfs:subclassOf rdf:resource="#Person" />
   <owl:disjointWith rdf:resource="#Frau" />
</owl:Class>

<owl:Class rdf:about="#Frau">
   <rdfs:subclassOf rdf:resource="#Person" />
   <owl:disjointWith rdf:resource="#Mann" />
</owl:Class>
```

Quellcode 17: Ein Beispiel für die Anwendung disjunkter Klassenbeziehungen in OWL.

Durch Inferenzbildung ist es ausreichend, wenn nur eine der Klassen (Frau oder Mann) über owl:disjointWith miteinander verbunden sind, um eine disjunkte Klassenbeziehung zwischen beiden Klassen herzustellen.

2.6.2.1.3 Teilmengen und Listen

Eine Klasse, die sich aus der Schnittmenge zweier anderer Klassen bildet, lässt sich über owl:intersectionOf deklarieren. Eine Vereinigungsmenge lässt sich dagegen über

owl:unionOf deklarieren. Die Klasse Person bildet sich im folgenden Beispiel aus der Vereinigungsmenge der beiden Klassen Frau und Mann.

```
<owl:Class rdf:about="#Person">
   <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Mann" />
      <owl:Class rdf:about="#Frau" />
   </owl:unionOf>
</owl:Class>
```

Quellcode 18: Beispiel einer Klasse, deren Individuen sich aus der Vereinigungsmenge von Individuen anderer Klassen ergeben.

Mit owl:oneOf lassen sich Aufzählungen, wie beispielsweise die Folgende, erstellen.

```
<owl:Class rdf:about="#GriffinFamily">
   <owl:equivalentClass>
      <owl:oneOf>
         <Person rdf:about="#Peter" />
         <Person rdf:about="#Lois" />
         <Person rdf:about="#Stewie" />
         <Person rdf:about="#Meg" />
         <Person rdf:about="#Chris" />
         <Person rdf:about="#Brian" />
      </owl:oneOf>
   </owl:equivalentClass>
</owl:Class>
```

Quellcode 19: Beispiel einer Klasse mit Angabe ihrer Instanzen in Form einer Liste.

So lässt sich beschreiben, dass die Klasse GriffinFamily aus den Individuen Peter, Lois, Stewie, Meg, Chris und Brian zusammengesetzt ist.

2.6.2.2 Thing und Nothing

Eine Ontologie verfügt nach dem OWL Standard immer über zwei vordefinierte Klassen: owl:Thing und owl:Nothing. Die Klasse owl:Thing umfasst alle Definitionen einer Ontologie. Alle Klassen, Individuen und Relationen sind immer vom Typ owl:Thing. Die Klasse owl:Nothing dagegen enthält nichts.

2.6.2.3 Individuen

Bei Individuen handelt es sich in einer Ontologie um Objekte, die einer Klasse angehören können. Beispielsweise lässt sich durch das Tripel Max_Musterstudent rdf:type Student ein Individuum Max_Musterstudent erstellen und dieses der Klasse Student zuweisen. In OWL lässt sich dieses Tripel über den folgenden Ausdruck definieren:

```
<Student rdf:about="Max_Mustermann" />
```

Quellcode 20: Kurzschreibweise einer Deklaration eines Individuums der Klasse Student.

2.6.2.4 Relationen

Relationen bilden die Verbindungen zwischen Objekten, wie zum Beispiel Klassen oder Individuen, in einer Ontologie. Relationen werden in OWL in zwei Kategorien unterteilt.

- ObjectProperties, die Relationen beschreiben, die auf Klassen oder Individuen verweisen.
- DataProperties, die Relationen beschreiben, die auf simple Datentypen, wie beispielsweise eine Zeichenkette, verweisen.

Relationen lassen sich über owl:ObjectProperty und owl:DatatypeProperty erstellen. Im folgenden Beispiel werden zwei Relationen, eine ObjectProperty und eine DataProperty, definiert.

```
<owl:ObjectProperty rdf:about="studiert">
   <rdfs:domain rdf:resource="Person"/>
   <rdfs:range rdf:resource="Studienfach"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="name">
   <rdfs:domain rdf:resource="Person" />
   <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Quellcode 21: Anwendung einer ObjectProperty und DataTypeProperty in einem Beispiel.

Der Domain und Range beider Relationen ist im Beispiel durch die entsprechenden RDFS-Tags innerhalb der Definition der Relationen eingeschränkt. Die Relation „studiert“ kann dadurch nur von der Klasse „Person“ beziehungsweise einem Individuum der Klasse auf die Klasse „Studienfach“ beziehungsweise einem Individuum der Klasse verweisen. Die Relation „name“ kann durch die Domain und Range Einschränkung nur von der Klasse „Person“ beziehungsweise einem Individuum der Klasse auf einen Datenwert vom Typ „String“ zeigen.

2.6.2.4.1 Symmetrische Relationen

Eine symmetrische Relation beschreibt eine Relation, die eine symmetrische Inferenz herstellt. Wird eine Ressource R1 mit einer anderen Ressource R2 über eine symmetrische Relation rel_sr verbunden, folgt daraus das auch R2 über rel_sr mit R1 verbunden ist. In Abbildung 7 wird die Ressource Person #1 mit einer zweiten Ressource Person #2 über eine symmetrischen Relation befreundet_mit verbunden. Aus der symmetrischen Relation folgt, dass dadurch auch Person #2 mit Person #1 über die Relation befreundet_mit verbunden sein muss.

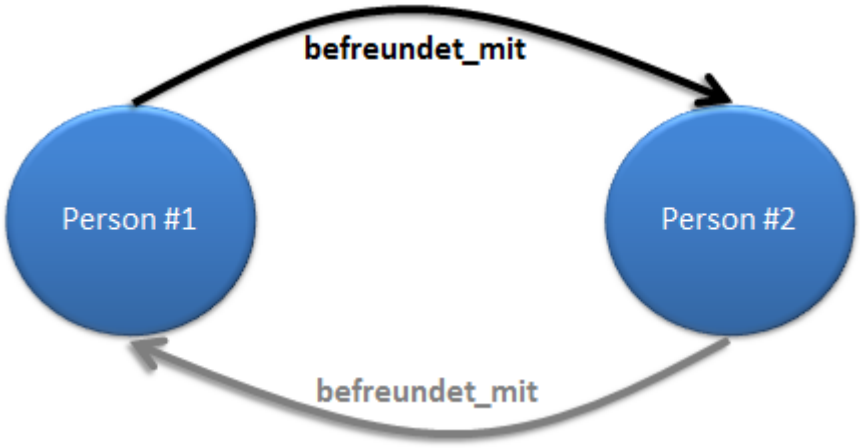


Abbildung 7: Eine symmetrische Relation, wie zum Beispiel „befreundet_mit“ (hier schwarz dargestellt) inferenziert eine gleichnamige Relation zurück (hier grau dargestellt).

```
<owl:Class rdf:about="#Person"/>

<owl:ObjectProperty rdf:about="befreundet">
   <rdf:type rdf:resource=“owl:SymmetricProperty“ />
   <rdfs:domain rdf:resource="#Person"/>
   <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<Person rdf:about=“#Person #1“>
   <befreundet rdf:resource=“#Person #2“ />
</Person>
<Person rdf:about=“#Person #2“ />
```

Quellcode 22: Deklaration einer symmetrischen Relation anhand eines Beispiels.

2.6.2.4.2 Asymmetrische Relationen

Eine asymmetrische Relation bezeichnet das Gegenteil einer symmetrischen Relation. Wird eine Ressource R1 mit einer anderen Ressource R2 über eine asymmetrischen Relation rel_asr verbunden, folgt daraus das R2 nicht über rel_asr mit R1 verbunden werden kann.

Abbildung 8 veranschaulicht ein Beispiel einer asymmetrischen Relation zwischen zwei Personen.

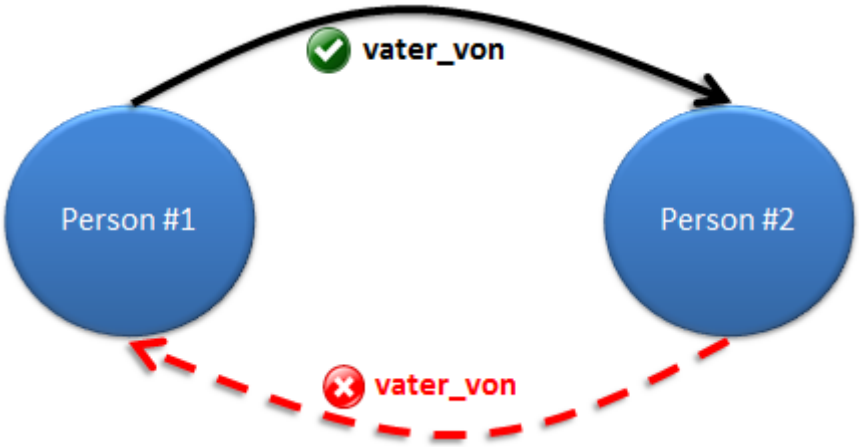


Abbildung 8: Eine asymmetrische Relation, wie zum Beispiel: „vater_von“, verbietet durch Inferenzbildung die Erstellung einer symmetrischen Beziehung zwischen zwei Ressourcen.

```
<owl:Class rdf:about="#Person"/>

<owl:ObjectProperty rdf:about="vater_von">
    <rdf:type rdf:resource=“owl:AsymmetricProperty“ />
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<Person rdf:about=“#Person #1“>
    <vater_von rdf:resource=“#Person #2“ />
</Person>
<Person rdf:about=“#Person #2“ />
```

Quellcode 23: Deklaration einer asymmetrischen Relation anhand eines Beispiels.

2.6.2.4.3 Transitive Relationen

Eine transitive Relation beschreibt eine Relation, die eine transitive Inferenz herstellt. Wird eine Ressource R1 mit einer Ressource R2 über eine transitive Relation rel_tr verbunden und R2 mit einer Ressource R3 über diese Relation, folgt daraus, dass auch R1 mit R3 über rel_tr verbunden ist. Abbildung 9 veranschaulicht dies an einem Beispiel.

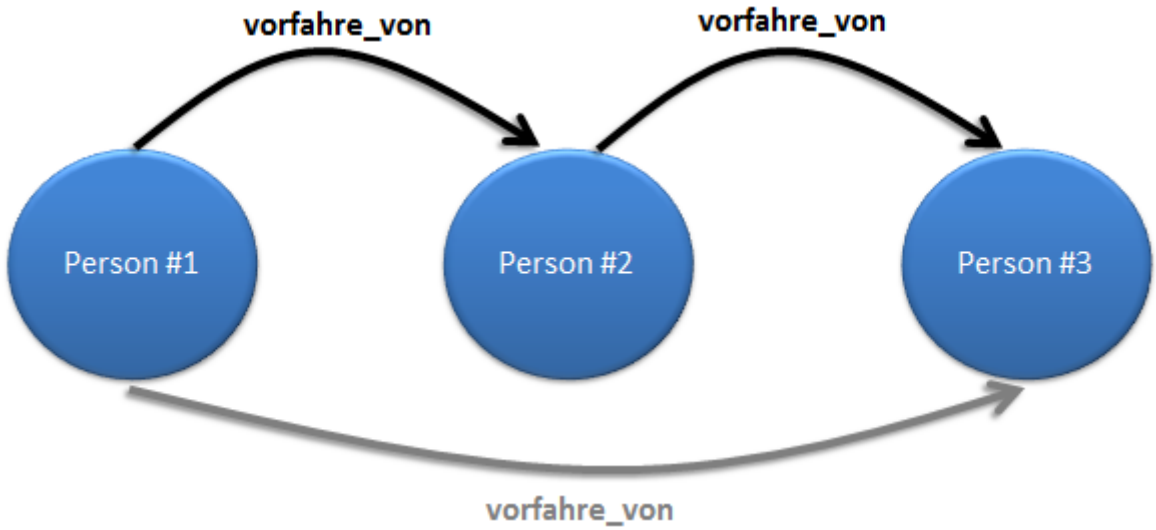


Abbildung 9: Eine transitive Relation, wie „vorfahre_von“ (hier schwarz) erzeugt durch Inferenzbildung eine entsprechende Relation „vorfahre_von“ (hier grau).

```
<owl:Class rdf:about="#Person"/>

<owl:ObjectProperty rdf:about="vorfahre_von">
   <rdf:type rdf:resource=“owl:TransitiveProperty“ />
   <rdfs:domain rdf:resource="#Person"/>
   <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<Person rdf:about=“#Person #1“>
   <vorfahre_von rdf:resource=“#Person #2“ />
</Person>
<Person rdf:about=“#Person #2“>
   <vorfahre_von rdf:resource=“#Person #3“ />
</Person>
<Person rdf:about=“#Person #3“ />
```

Quellcode 24: Deklaration einer transitiven Relation anhand eines Beispiels.

2.6.2.5 Weitere Relationen

Neben symmetrischen, asymmetrischen und transitiven Relationen sieht OWL noch equivalente, disjunkte, inverse, funktionale, invers funktionale, reflexive und irreflexive Relationen vor. Diese weiteren Relationen werden in diesem Abschnitt in einer kurzen Übersicht erläutert. Für eine detailliertere Beschreibung wird auf die OWL Spezifikation [23] verwiesen.

- **Äquivalente Relationen**
  Wird durch eine Anfrage nach Übereinstimmungen mit dem Tripel x rel1 y gesucht und die Relationen rel1 und rel2 als äquivalente Relationen deklariert, so werden auch Übereinstimmungen mit dem Tripel x rel2 y durch die Anfrage abgefragt.

- **Disjunkte Relationen**
  Sind zwei Relationen disjunkt zueinander, können sie nicht den gleichen Wert annehmen. Beispielsweise kann eine Relation name nicht den gleichen Wert wie die Relation alter annehmen.

- **Inverse Relationen**
  Sind zwei Relationen rel1, rel2 zueinander invers, bedeutet dies, dass wenn Ressource R1 mit Ressource R2 über rel1 verbunden ist, R2 auch mit R1 über rel2 verbunden sein muss.

- **Funktionale Relationen**
  Funktionale Relationen bezeichnen Relationen, die für eine Ressource nur einmal deklariert werden dürfen. Beispielsweise wäre die Relation geburtsort eine Relation, die für eine Ressource nur einmal deklariert werden sollte.

- **Invers funktionale Relationen**
  Invers funktionale Relationen beschreiben eine Kombination aus funktionalen und inversen Relationen.

- **Reflexive Relationen**
  Wird eine Relation rel1 zu einer reflexiven Relation deklariert, bedeutet dies, dass alle Individuen i mit sich selbst über rel1 verbunden sind. Ein Beispiel wäre die Relation kennt. Jedes Individuum kennt sich selbst, um dies auszudrücken kann die Relation kennt als reflexive Relation deklariert werden,

- **Irreflexive Relationen**
  Das Gegenteil einer reflexiven Relation ist eine irreflexive Relation, die aussagt, dass kein Individuum über eine irreflexive Relation mit sich selbst verbunden sein kann. Beispielsweise kann keine Person mit sich selbst verheiratet sein, die Relation verheiratet kann also als irreflexive Relation deklariert werden.

2.6.2.6 Restriktionen

Restriktionen erlauben es die Beziehung zwischen Klassen und Relationen detaillierter zu beschreiben. So lassen sich über Restriktionen, die Kardinalität oder Quantoren spezifizieren. Die Deklaration einer Restriktion wird über owl:Restriction vorgenommen, die entweder innerhalb einer Klasse definiert oder über die Relation rdf:type mit einer Klasse verbunden wird. Um zu definieren, auf welche Relation sich eine Restriktion bezieht, verwendet man die Spezifikation der Relation über owl:onProperty.

2.6.2.6.1 Kardinalitäten

OWL ermöglicht die Kardinalität einer Relation zu einer Klasse über die Deklaration von minimalen Kardinalitäten (owl:minCardinality), maximalen Kardinalitäten (owl:maxCardinality) und exakten Kardinalitäten (owl:qualifiedCardinality) einzuschränken.

Im folgenden Beispiel wird für die Klasse Person die Kardinalität der Relation name auf 1 festgelegt. Das bedeutet, dass eine Person genau einen Wert für die Relation name besitzen muss. Eine Person ohne einen Namen ist demnach keine Person.

```
<owl:Class rdf:about="#Person">
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource=“#name“ />
         <owl:qualifiedCardinality rdf:datatype=“&xsd:nonNegativeInteger“>
            1
         </owl:qualifiedCardinality>
      </owl:Restriction>
   </owl:equivalentClass>
</owl:Class>
```

Quellcode 25: Deklaration von Kardinalitätsbeschränkungen anhand eines Beispiels.

2.6.2.6.2 Quantoren

Über Quantoren kann spezifiziert werden, ob Klassen mit bestimmten Werten über Relationen verbunden sind. OWL bietet die Deklarationen owl:allValuesFrom, owl:someValuesFrom, owl:hasValue.

```
<owl:Class rdf:about="#Person">
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#name" />
         <owl:hasValue rdf:datatype="&xsd:integer">Max</owl:hasValue>
      </owl:Restriction>
   </owl:equivalentClass>
</owl:Class>
```

Quellcode 26: Klasse mit Restriktion

```
<owl:Class rdf:about="#Person">
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#vorfahre_von" />
         <owl:hasValue rdf:resource="#Clinton"
      </owl:Restriction>
   </owl:equivalentClass>
</owl:Class>
```

Quellcode 27:Klasse mit Restriktion.

2.6.2.7 Zusammenfassung

Die Web Ontology Language (OWL) ist eine vom W3C empfohlene Sprache, die ausdrucksstarke Möglichkeiten zur Modellierung und Speicherung von Informationen in strukturierter Form bietet. Durch die Beschreibung von Informationsbeziehungen in Form von Klassen- und Relationseigenschaften lassen sich durch Inferenzbildungen komplexe Ontologien erstellen und prüfen. Eine ausführliche Übersicht über alle Ausdrucksmittel der Web Ontology Language und ihre Dialekte bietet die W3C Spezifikation [23].

2.7 Rule Interchange Format - RIF

Das Rule Interchange Format (RIF) ist eine Empfehlung des W3C, an der seit 2005 gearbeitet wird. Es dient zur Formulierung und zum Austausch von Regeln innerhalb regelbasierter Systeme. [27] [28]

Ein regelbasiertes System besteht laut [28] im Wesentlichen aus drei Komponenten:

- eine Komponente zur Datenspeicherung, beispielsweise einer Datenbank,
- ein Satz an anwendbaren Regeln,
- eine Komponente zur Interpretation und Anwendung der definierten Regeln

Im Grunde lassen sich Regeln laut [28] in Form von „Wenn X dann Y“ ausdrücken, dabei gibt es innerhalb regelbasierter Systeme im Wesentlichen drei Kategorien zu Deklaration von Regeln:

- Prädikatenlogik der ersten Stufe
- Logische Programmierung
- Produktionsregeln

Durch die vielfältigen Möglichkeiten Regeln zu definieren, ist es offensichtlich, dass eine einzige Sprache kaum alle Anforderungen bieten kann, um alle Möglichkeiten zur Definition von Regeln abzudecken. Der Schwerpunkt von RIF liegt daher im Austausch von Regeln zwischen unterschiedlichen Systemen. Die W3C Arbeitsgruppe schlug daher die Entwicklung einer Sprachfamilie aus einheitlichen und erweiterbaren Dialekten vor. Zur Deklaration von Regeln innerhalb des Rule Interchange Formats bietet das W3C zwei unterschiedliche Dialekte an. Der Basic Locig Dialect dient zur Deklaration von Regeln in Form von prädikatenlogischen Horn-Formeln und der Production Rule Dialect (RIF-PRD) zur Deklaration von Regeln in Form von Produktionsregeln. Des Weiteren wird vom W3C ein Basisdialekt (RIF Core Dialect) angeboten, der eine Untermenge aus den beiden anderen Dialekten darstellt. [28]

2.8 SPARQL

SPARQL ist ein rekursives Akronym und steht für SPARQL Protocol And RDF Query Language [29]. Rekursive Akronyme werden in der Informatik häufig für Abkürzungen verwendet, die in ihrer ausgeschriebenen Form ihre eigne Abkürzung erneut enthalten [30]. SPARQL wurde vom W3C (World Wide Web Consortium) im Jahre 2008 als W3C Recommendation verabschiedet und beschreibt eine formale, graphbasierten Anfragesprache für semantische Datenquellen auf Basis von RDF/RDFS beziehungsweise OWL. Anfragen in SPARQL werden dabei an einen SPARQL-Endpoint gerichtet, der eine Anfrage interpretieren und auf Basis eines semantischen Datenbestands eine Antwort zurückgeben kann. Die W3C Spezifikation von SPARQL findet sich unter [29]. Am 21. März 2013 wurde SPARQL 1.1 als Spracherweiterung zu SPARQL in Version 1.0 vom W3C als Recommendation verabschiedet. In Version 1.1 enthält SPARQL viele neue Funktionen, wie beispielsweise Aggregationsfunktionen, Unteranfragen, Negationen, Relationspfade, erweiterte Variablendeklarationen, verteilte Anfragen, Updatefunktionen, sowie erweiterte Filterfunktionen. [31]

Der Grundaufbau einer SPARQL-Anfrage besteht aus frei definierbaren Variablen, die aus einer Zeichenkette mit voranstehendem „?“ oder „$“ gebildet werden, sowie aus den folgenden Deklarationsteilen:

- PREFIX
  Deklaration von Präfixen für eine Anfrage, die dazu dienen URIs durch Abkürzungen leichter les- und schreibbar zu gestalten.

- SELECT/ASK/DESCRIBE/CONSTRUCT
  Deklaration der Anfrageart und gegebenenfalls der zurückzugebenden Variablen beziehungsweise des zurückzugebenden Graphenschema.

- FROM
  Optionale Deklaration der Datenquelle beziehungsweise eines Graph einer Datenquelle an die sich eine Anfrage richtet. Erhält ein SPARQL-Endpoint eine Anfrage ohne Angaben zur Datenquelle beziehungsweise eines Graphen der Datenquelle, so nimmt der SPARQL-Endpoint den bei ihm hinterlegten Standardgraphen.

- WHERE
  Deklaration der Bedingungen und Filter einer Anfrage, die über UND-/ODER-Operatoren miteinander verknüpft werden können.

- ORDER BY /GROUP/LIMIT/OFFSET
  Im Anschluss an eine Deklaration von Anfragebedingungen innerhalb von WHERE, kann durch die Anfrage beispielsweise die Sortierung, Gruppierung, Begrenzung, Offset, sowie weitere Modifikationen der Ergebnismenge deklariert werden.

2.8.1 Einfache SPARQL-Anfragen

Eine SPARQL-Anfrage setzt sich aus den vorgestellten Grundbausteinen zusammen. Im folgenden Beispiel findet sich eine einfache SPARQL-Anfrage, in der nach Buchtiteln gesucht wird.

```
PREFIX example: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?title
WHERE
{
  ?books rdf:type example:book .
  ?books dc:title ?title
}
ORDER BY ?title
```

Quellcode 28: Eine einfache SPARQL-Anfrage in der Titel eines Buchs in alphabetischer Reihenfolge gesucht werden.

Im Präfixbereich der Anfrage werden Abkürzungen für URIs definiert. Diese Abkürzungen lassen sich im weiteren Verlauf der Anfrageerstellung verwenden. Durch das Schlüsselwort SELECT und der Angabe einer Variable wird spezifiziert, dass die Rückgabe der Anfrage aus einer Tabelle mit den Werten der Variable ?titel bestehen soll. Anschließend werden zwei Bedingungen gestellt, zum einen muss die Variable ?books mit der Relation rdf:type mit der Ressource example:book verbunden sein und zum anderen soll der ?titel eines Buches über die Relation dc:title abgefragt werden. Beide Bedingungen sind über eine UND-Verknüpfung miteinander verbunden. UND-Verknüpfungen zwischen zwei Bedingungen werden durch einen Punkt („.“) angegeben. ODER-Verknüpfungen werden durch folgendes Schema deklariert.

```
{ …} UNION {…}
```

Quellcode 29: Schema einer ODER-Verknüpfung von innerhalb {…} deklarierter Bedingung beziehungsweise verknüpfter Bedingungen.

Wobei {…} jeweils für die Deklaration einer oder mehrerer verknüpfter Bedingungen steht. Durch ORDER BY wird angegeben, dass das Ergebnis bezüglich der Werte der Variable ?title alphabetisch sortiert zurückgegeben werden soll.

Wird eine Anfrage an einen SPARQL-Endpoint geschickt, so erstellt dieser ein entsprechendes Graphenmuster, das er mit dem bei sich hinterlegten Datensatz vergleicht. Die Übereinstimmung zwischen Graphenmuster und Datensatz bildet die Lösung und damit den Rückgabewert der gestellten Anfrage. Die in Abbildung 10 dargestellte Grafik verdeutlicht die Suche nach Mustern in einem Graphen.

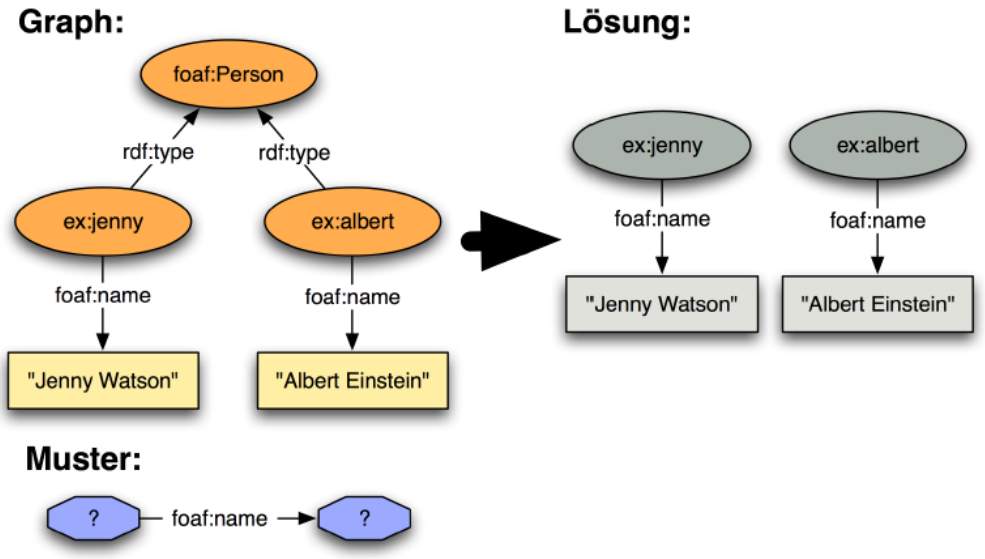


Abbildung 10: Der orange/gelbe Graph bildet die Datenquelle ab. Mit einer graphbasierten Anfragesprache, wie SPARQL, kann ein Graphmuster (blau eingefärbt) erstellt werden. In diesem Beispiel wird ein Subjekt und das zugehörige Objekt in einem Tripel gesucht, die über die Relation foaf:name miteinander verbunden sind. Die Lösung ist in der Abbildung grau unterlegt. [32]

2.8.2 Anfragearten

SPARQL bietet insgesamt vier verschiedene Arten von Anfragen, von denen wir mit SELECT bereits eine kennengelernt haben. Im Folgenden werden alle Anfragetypen von SPARQL aufgelistet und ihre Funktion beschrieben. [33]

2.8.2.1 SELECT

Über SELECT-Anfragen werden übereinstimmende Graphenmuster in einem Graphen gesucht und in relationale Form von einem SPARQL-Endpoint zurückgegeben. Werte übereinstimmender Graphenmuster werden dabei auf im WHERE-Bereich gebundenen Variablen projiziert. Über eine Auflistung von Variablen, die dem Schlüsselwort SELECT folgt, kann ausgewählt werden, welche der gebundenen Variablen im Rückgabewert des SPARQL-Endpoints vorkommen sollen. Folgt auf das Schlüsselwort SELECT ein *, so sind alle gebunden Variablen im Rückgabewert des Endpoints enthalten.

2.8.2.2 ASK

Mit ASK kann angefragt werden, ob ein übergebenes Graphenmuster im Datensatz des SPARQL-Endpoints vorhanden ist. Der Rückgabewert ist also immer entweder wahr oder falsch. So lassen sich zum Beispiel Anfragen der Form „Ist der Eiffelturm 324m hoch?“ definieren. Aufgrund des sehr einfachen Rückgabewerts (wahr/falsch) besitzt eine ASK-Anfrage keine Modifikationen für den Rückgabewert, wie zum Beispiel LIMIT oder ORDER BY.

Das folgende Beispiel zeigt eine simple ASK-Anfrage, in der abgefragt wird, ob die Höhe des Eiffelturms einem Zahlenwert von 324 beträgt.

```
ASK {
<http://dbpedia.org/resource/Eiffelturm> <http://dbpedia.org/property/gesamthoehe> ?x
FILTER(?x = 324))
}
```

Quellcode 30: Eine ASK-Anfrage, in der angefragt wird, ob die Gesamthöhe des Eiffelturms 324 beträgt.

2.8.2.3 DESCRIBE

Mit dem Schlüsselwort DESCRIBE kann von einem SPARQL-Endpoint ein vollständiger RDF-Graph angefordert werden, der die in der Anfrage spezifizierten Datensätze beschreibt. So lässt sich beispielsweise mit folgender Anfrage ein vollständiger RDF-Graph vom Endpoint der DBPedia zurückgeben, der die Programmiersprache JavaScript beschreibt.

```
DESCRIBE <http://dbpedia.org/resource/JavaScript>
```

Quellcode 31: Eine SPARQL-Anfrage über das Schlüsselwort "DESCRIBE". Angefragt wird ein RDF-Graph, der die Ressource mit der URI <http://dbpedia.org/resource/JavaScript> beschreibt.

Der durch diese Anfrage zurückgegebene RDF-Graph enthält alle Tripel in denen <http://dbpedia.org/resource/JavaScript> vorkommt.

Eine Anfrage mit dem Schlüsselwort kann natürlich auch durch einen WHERE-Ausdruck spezifiziert werden. Mit folgender Anfrage lässt sich beispielsweise ein RDF-Graph zurückgeben, der alle Programmiersprachen beschreibt, durch die JavaScript beeinflusst wurde.

```
DESCRIBE * WHERE {
<http://dbpedia.org/resource/JavaScript> <http://dbpedia.org/ontology/influencedBy> ?x
}
```

Quellcode 32: Eine SPARQL-Anfrage über das Schlüsselwort "DESCRIBE" mit einer WHERE-Bedingung. Angefragt wird ein RDF-Graph, der alle Ressourcen beschreibt, die die Programmiersprache JavaScript beeinflusst haben.

2.8.2.4 CONSTRUCT

Mit dem Schlüsselwort CONSTRUCT wird wie durch DESCRIBE ebenfalls ein RDF-Graph von einem SPARQL-Endpoint zurückgegeben. Für den RDF-Graph lässt sich in der CONSTRUCT-Anfrage ein Template definieren.

Trifft an einem SPARQL-Endpoint eine Anfrage mit dem Schlüsselwort CONSTRUCT ein, werden die übereinstimmenden Tripel des Datensatzes auf die entsprechenden Variablen des Templates gemapped und in den RDF-Graphen eingefügt, der wenn alle übereinstimmenden Tripel gefunden wurden, zurückgegeben wird.

In der folgenden Anfrage, werden die Programmiersprachen gesucht, die die Programmiersprache JavaScript beeinflusst haben und in einen RDF-Graphen eingefügt der genau diese Informationen enthalten soll.

```
CONSTRUCT {
<http://dbpedia.org/resource/JavaScript> <http://dbpedia.org/ontology/influencedBy> ?x
} WHERE {
<http://dbpedia.org/resource/JavaScript> <http://dbpedia.org/ontology/influencedBy> ?x
}
```

Quellcode 33: Eine SPARQL-Anfrage über das Schlüsselwort "CONSTRUCT". Angefragt wird ein RDF-Graph, der dem durch CONSTRUCT beschriebenen RDF-Graph für alle Ressourcen, die die Programmiersprache JavaScript beeinflusst haben, entspricht.

Dadurch lassen sich die Informationen spezifizieren, die im angeforderten RDF-Graphen enthalten sein sollen. Unerwünschte Informationen, wie zum Beispiel in diesem Fall der Entwickler der Programmiersprache gelangen so nicht in den angeforderten RDF-Graphen.

2.9 Linked Open Data Cloud

Die Grundidee des Semantischen Webs ist es Informationen miteinander zu verbinden. Dies beschränkt sich nicht auf die Erstellung von Ontologien, sondern betrifft auch die Datenbanken, die die Ontologien enthalten und miteinander verbunden werden können. Mit Hilfe von OWL können so Ontologien dezentral entwickelt und erweitert werden. Eine Ontologie kann so auf einem Server A gespeichert sein und durch eine weitere Ontologie auf Server B mit weiteren Informationen erweitert werden. Die Linked Open Data Cloud (LOD-Cloud) bildet so ein Netzwerk aus miteinander verbunden SPARQL-Endpoints mit einem freien und offen Zugang zu Informationen. Die Designprinzipien der LOD-Cloud beschreibt Tim Berners-Lee in [34] mit folgenden vier Regeln:

- eindeutige URIs zur Beschreibung von Objekten verwenden.
- über das http-Protokoll nachschlagbare URIs.
- Informationen über Standards bereitstellen
- Informationen mit anderen Diensten vernetzen

Die folgende Abbildung veranschaulicht die Struktur der LOD-Cloud bestehend aus miteinander verbundenen SPARQL-Endpoints.

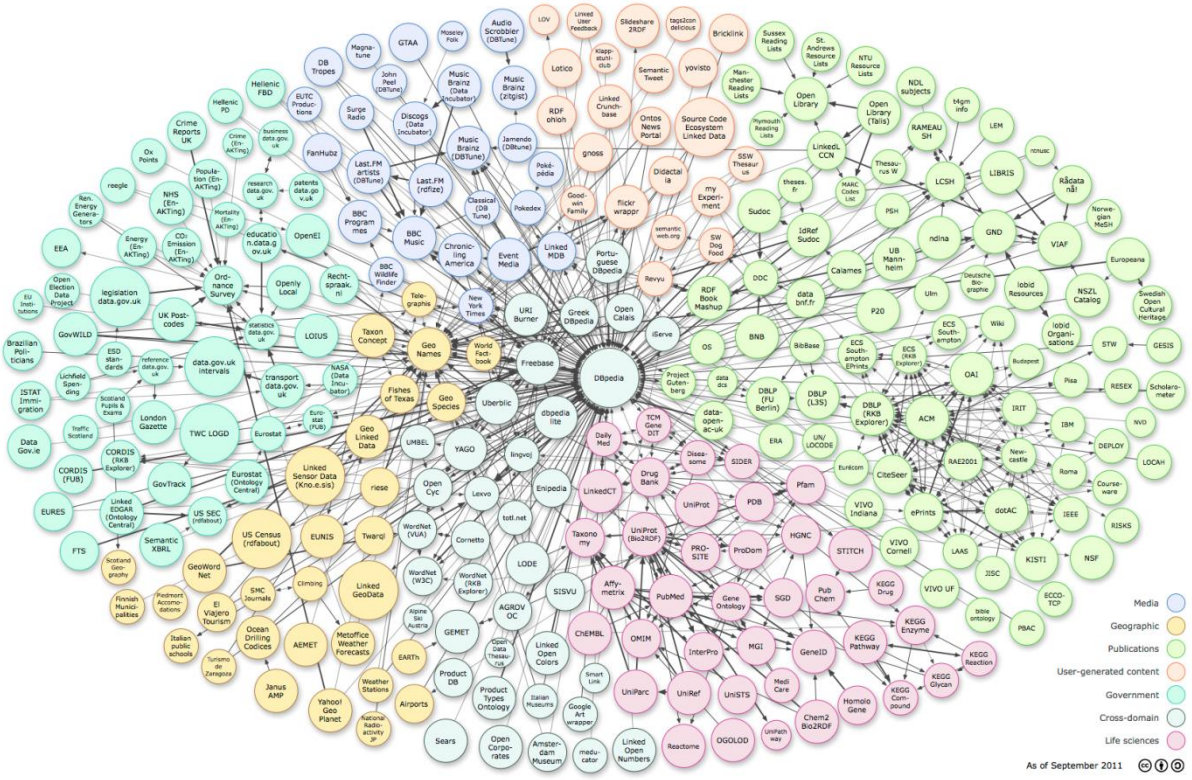


Abbildung 11: Visualisierung der LOD-Cloud (*Stand: September 2011*) als gerichteter Graph. Kreise stehen für die jeweiligen SPARQL-Endpoints von Diensten, wie beispielsweise der DBPedia oder GeoNames. Die Verbindungspfeile visualisieren die Verweise von Diensten untereinander. [35]

Der Datensatz der DBPedia, der durch die semantische Annotation des Datensatz des Wikipedia-Projekts gewonnen wird [36], gehört, wie in Abbildung 11 zu sehen ist, mitunter zu einem der wichtigsten Datensätze innerhalb der LOD-Cloud. Das im Rahmen dieser Arbeit entwickelte Lösungskonzept und dessen Implementierung wird daher insbesondere auf den Daten der DBPedia evaluiert werden.

3. Anfragesprachen und ihre Visualisierung

Mit Anfragesprache bezeichnet man in der Informatik eine Sprache zur Suche nach Informationen innerhalb einer oder mehrerer Informationsquellen. Je nach Art der Anfrage und der anzufragenden Informationsquelle können Anfragesprachen unterschiedlich strukturiert und entworfen worden sein. Anfragesprachen bestehen dabei aber im Wesentlichen immer aus Bedingungen, die mit Operatoren zu einer Anfrage verknüpft werden. [37]

In diesem Kapitel werden verschiedene Arten von Anfragesprachen behandelt, die im Kontext des Semantischen Webs angewendet werden, ohne dabei Anspruch auf Bildung einer vollständigen Taxonomie zu erheben.

3.1 Textuelle Anfragesprachen

Zu textbasierten Anfragsprachen gehören die formalen Anfragesprachen, wie beispielsweise SQL oder SPARQL sowie die natürlichen Sprachen.

3.1.1 Formale Anfragesprachen

Eine formale Anfragesprache ist eine formale Sprache, die zur Anforderung von Daten einer Informationsquelle entworfen wurde. Unter dem Begriff formale Sprache versteht man dabei eine abstrakte Sprache, die sich logischer und mathematischer Beschreibungen bedient [38]. Formale Abfragesprachen können dabei, je nach Form der anzufordernden Datensätze, unterschiedliche Gestalt annehmen. So ist beispielsweise SQL auf relationale Datenbanken spezialisiert und besitzt Eigenschaften, um diese effizient abzufragen. Beispielsweise bietet SQL Funktionen an, um mehrere relationale Tabellen miteinander zu verbinden, um so die gewünschten Informationen in einer gebündelten Anfrage anfordern zu können. Mit SPARQL existiert eine vom W3C empfohlene formale Anfragesprache, die für semantische Anfragen entworfen wurde [31].

Zu den Nachteilen formaler Anfragesprachen zählen aus Sicht von Nutzern:

- Ein hoher Lernaufwand, zum Erlernen der Sprache sowie ihrer Syntax. [39]
- Schwierigkeiten beim Umgang mit einer formalen Sprache [39], beispielsweise beim Erstellen von Anfragen mit booleschen Ausdrücken [9]
- Wenig bis gar kein Feedback beim Erstellen einer Anfrage. [39]

- Beschränkte Interaktionsmöglichkeiten, da sich die Interaktion zwischen Mensch und Maschine meist nur auf die Eingabe von Kommandos und der Ausführung dieser beschränk. [39]
- Kenntnisse über das Datenbankschema sind notewendig, um eine Anfrage zu erstellen [40]

Daher wurde im Laufe der Zeit weitere Möglichkeiten zur Erstellung von Anfragen erarbeitet. Formale Anfragesprachen, wie SPARQL oder SQL, bilden dabei meist die Basis für darauf aufbauende Anfragesprachen. Beispiele für visuelle Anfragesprachen, die auf formale Anfragesprachen aufbauen, finden sich in Kapitel 4. [40] [41]

3.1.2 Natürliche Sprache

Mit der natürlichen Sprache besitzen wir ein sehr mächtiges Ausdruckmittel, das wir schon fast uns gesamtes Leben lang zur Kommunikation nutzen. Anfragen können wir daher am leichtesten in unserer natürlichen Sprache formulieren. Charakteristisch für unsere natürliche Sprache ist ihr hoher Grad an Kontextsensitivität. Aus diesem Grund ist sie für heutige Rechnersysteme nur sehr schwer zu interpretieren. Das Semantische Web bietet Techniken, die es auch ohne Zuhilfenahme komplexer Algorithmen künstlicher Intelligenz erlaubt einfache Anfragen in natürlicher Sprache, wie zum Beispiel „Wie hoch ist der Eiffelturm?“, zu interpretieren. Bei diesem Vorgang wird die natürliche Sprache für die Verarbeitung mit rechnergestützten Systemen auf eine formale Anfragesprache, wie beispielsweise SPARQL übersetzt.

3.2 Visuelle Anfragesprachen

Visuelle Anfragesprachen nutzen Visualisierungstechniken zum Beschreiben und zur Darstellung von Anfragen. Das Ziel, das mit visuellen Anfragesprachen verfolgt wird, ist es die Erstellung und Bearbeitung von Anfragen für Nutzer zu erleichtern und Probleme gegenüber textuellen Anfragesprachen zu vermeiden. [39]

Durch visuelle Anfragesprachen kann der nötige Lernaufwand für das Erstellen von Anfragen gegenüber textuellen Anfragesprachen reduziert werden [40]. Zudem können visuelle Anfragesprachen Nutzern beim Erstellen einen Überblick über das Datenbankschema einer Informationsquelle vermitteln [40] [42].

Mithilfe visueller Anfragesprachen kann zudem sichergestellt werden, dass eine erstellte Anfrage stets valide ist und Nutzern ein ausreichendes Feedback beim Erstellen von Anfragen gegeben werden. [43]

Visuelle Anfragesprachen, wie beispielsweise [44] und [41] übersetzen visuell erstellte Anfragen zur weiteren Verarbeitung meist in formale Anfragesprachen.

Eine Gliederung unterschiedlicher Arten visueller Anfragesprachen ist aufgrund der Vielfalt visueller Anfragesprachen nicht einfach. In der Literatur [45] [46] findet sich eine Unterteilung visueller Anfragesprachen in vier Gruppen:

- Formularbasiert
- Diagrammbasiert
- Iconbasiert
- Hybride

In den folgenden Unterkapiteln werden formularbasierte, diagrammbasierte, iconbasierte und hybride Anfragesprachen und ihre Besonderheiten vorgestellt. Im Anschluss wird eine mögliche Erweiterung dieser Taxonomie vorgestellt.

3.2.1 Formularbasierte Anfragesprachen

Charakteristisch für formularbasierte Anfragesprachen sind vorgefertigte Eingabefelder zum Erstellen von Anfragen, die vom Nutzer ausgefüllt werden.

Query By Example [42], eine bei IBM von Moshé M. Zloof entwickelte Anfragesprache für relationale Datenbanken, ist ein bekanntes Beispiel für eine formularbasierte Anfragesprache. In Query By Example füllt der Nutzer Werte in einem Tabellengerüst aus, das anschließend in eine SQL-Anfrage umgewandelt wird. Eine durch Query By Example formulierte Beispielanfrage ist in Abbildung 12 dargestellt.

Alle Kunden aus Bremen

KUNDE	kdnr	kname	adresse	ort
P.				='Bremen'

Abbildung 12: Eine in Query By Example erstellte Beispielanfrage. Durch die in der Tabellenstruktur gesetzten Werte werden alle Kunden aus dem Ort Bremen angefragt. Die Tabelle stammt aus [47].

Anfragen können sich in Query By Example auch über mehrere Tabellen erstrecken. Ein Beispiel dazu veranschaulicht Abbildung 13.

Sailors	*sid*	*sname*	*rating*	*age*	*Reserves*	*sid*	*bid*	*day*
	_Id	P._S		> 25		_Id		'8/24/96'

Abbildung 13: Eine in Query By Example erstellte Anfrage, durch die Segler gesucht werden, die über 25 Jahre alt sind und am 24.08.1996 ein Boot reserviert haben. [48]

Über eine „Conditions Box“ lassen sich in Query By Example komplexere Anfragen, wie beispielsweise Anfragen mit UND-/ODER-Verknüpfungen, erstellen.

Sailors	*sid*	*sname*	*rating*	*age*	*Conditions*
		P.		_A	_A < 20 OR 30 < _A

Abbildung 14: Eine Anfrage mit einer ODER-Verknüpfung in Query By Example. Gesucht wird in der abgebildeten Anfrage der Nachname eines Seglers, der zwischen 20 und 30 Jahren alt ist. [48]

Das Erstellen von UND-/ODER-Verknüpfungen lässt sich Query By Example aber auch ohne eine „Conditions Box“ bewältigen. In Abbildung 15 sind zwei Anfragen in Query By Example dargestellt. In Anfrage (a) werden die beiden Bedingungen über ODER miteinander verknüpft. In Anfrage (b) beziehen sich beide Bedingungen auf die gleiche Variable („_Id“) und werden dadurch mit einem UND miteinander verknüpft. Eine Darstellung über eine „Conditions Box“ erscheint dabei aber übersichtlicher und leichter zu verstehen.

Sailors	*sid*	*sname*	*rating*	*age*
		P.		< 30
		P.		> 20

(a)

Sailors	*sid*	*sname*	*rating*	*age*
	_Id	P.		< 30
	_Id			> 20

(b)

Abbildung 15: UND-/ODER-Verknüpfungen lassen sich in Query By Example auch ohne eine „Conditions Box“ erzeugen. In Anfrage (a) werden beide Bedingungen mit einem ODER verknüpft (Nachname eines Seglers, der entweder über 20 oder unter 30 Jahre alt ist). In Anfrage (b) beziehen sich beide Bedingungen auf die gleiche Variable („Id“) und werden daher mit einem UND verknüpft (Nachname eines Seglers, der über 20 und unter 30 Jahre alt ist). [48]

Durch die Felder, die man beim Erstellen einer Anfrage mit Werten füllt, geben formularbasierte Anfragesprachen auch eine Übersicht über das Schema einer Datenbank. Einfache Anfragen lassen sich so schnell und unkompliziert erstellen [48]. Für komplexere Anfragen muss unter Umständen auf Hilfsmittel, wie eine „Conditions Box“ in Query By Example, zurückgegriffen werden.

3.2.2 Diagrammbasierte Anfragesprachen

Diagrammbasierte Anfragesprachen visualisieren Anfragen durch simple, geometrische Figuren, zum Beispiel durch Rechtecke, Kreise und Linien. Beispiele diagrammbasierte Anfragesprache finden sich in Abbildung 16 und 17.

Abbildung 16 stellt eine Anfrage in der diagrammbasierten Anfragesprache NITELIGHT dar [44]. Mit NITELIGHT lassen sich SPARQL-Anfragen visuell erstellen. Die Sprache wird in Kapitel 4.3 im Detail erläutert.

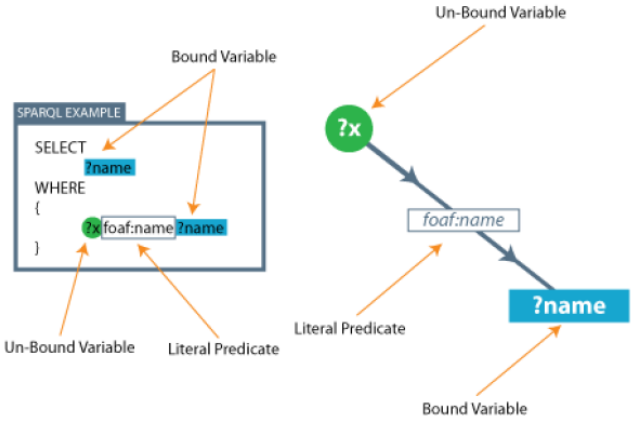


Abbildung 16: Eine in NITELIGHT erzeugte diagrammbasierte Anfragevisualisierung einer SPARQL-Anfrage. [44]

Abbildung 17 stellt eine Anfrage durch ein aus der Mengenlehre bekanntes Konzept zur Bildung von Teilmengen dar. Hier werden zwei Mengen von Studenten gebildet, deren Schnittmenge aus dem gleichen Studienfach bestehen soll, zusätzlich soll der Name der Studenten mitabgefragt werden.

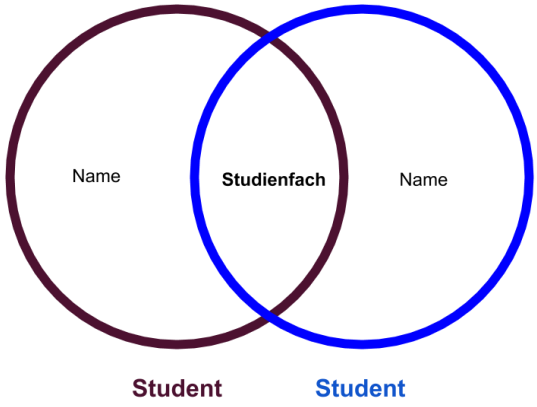


Abbildung 17: Eine mögliche Darstellungsform einer Anfrage auf Basis von Konzepten aus der Mengenlehre.

Die diagrammbasierte Visualisierung von Anfragen bildet eine sehr häufig verwendete Visualisierungsform von Anfragen und ermöglichen die anschauliche Darstellung von Relationen. [44]

3.2.3 Iconbasierte Anfragesprachen

Icons sind in der Mensch-Computer Interaktion ein beliebtes Mittel zur Informationsvisualisierung und werden beispielsweise zur Repräsentation von Objekten, Prozessen oder eines Status eingesetzt. Ein Beispiel für eine iconbasierte Anfragesprache zur Anfrage objektorientierter Datenbanken ist in Abbildung 18 dargestellt. [46]

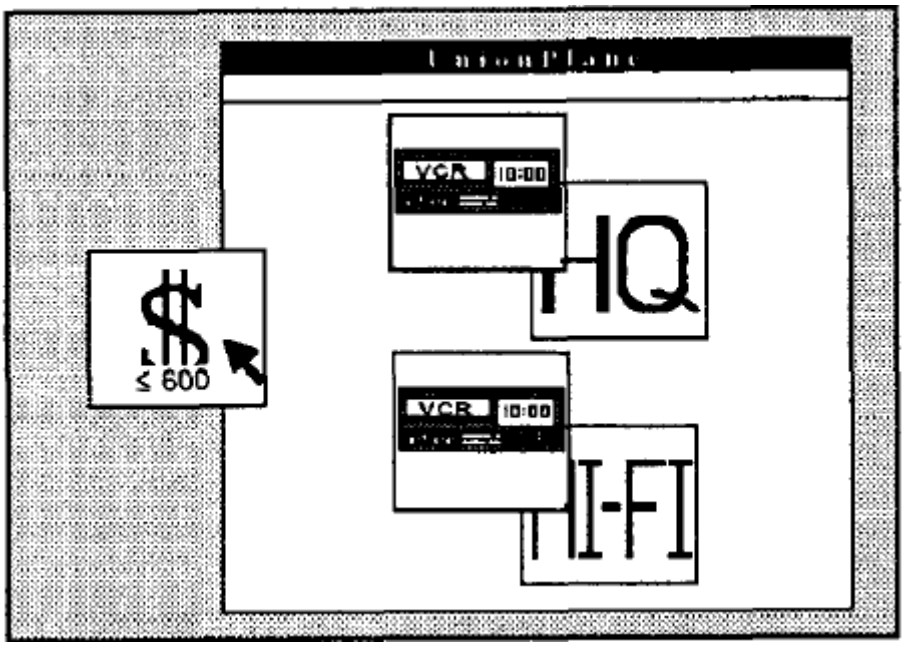


Abbildung 18: Eine beispielhafte Visualisierung einer iconbasierten Anfragesprache. Im gezeigten Beispiel werden werden Videorekorder unter 600 Dollar gesucht, die HQ oder HIFI unterstützen. [49]

Sind die verwendeten Icons einer iconbasierten Anfragesprache so gewählt, dass diese durch Nutzer leicht und schnell Objekten zugeordnet werden können, so können iconbasierte Anfragesprachen ein geeignetes Mittel zur Erstellung einfacher Anfragen darstellen. Problematisch ist in rein iconbasierten Anfragesprachen allerdings die Darstellung von Relationen zwischen verschiedenen abzufragenden Klassen oder Objekten. Das einer der beiden Videorekorder teurer sein soll als der andere, lässt sich in dem in Abbildung 18 veranschaulichten Konzept nur schwer umsetzen. [49]

3.2.4 Hybride Anfragesprachen

In hybriden Anfragesprachen werden Konzepte visueller Anfragesprachen miteinander kombiniert. Die Anfragesprache Kaleidoquery, die in Kapitel 4.3 im Detail vorgestellt wird, ist ein Beispiel für eine hybride Anfragesprache, die das Konzept der Icon- und Diagrammvisualisierung miteinander kombiniert. Dadurch lässt sich das Verständnis einer Anfrage erhöhen und dem Nutzer wird der Einstieg in den Umgang mit der visuellen Anfragesprache erleichtert [40]. Eine Beispielanfrage ist in Abbildung 19 dargestellt.

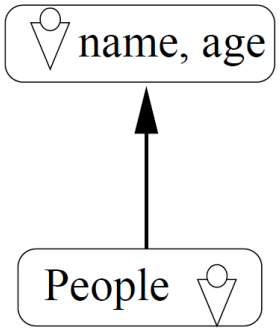


Abbildung 19: Eine durch Kaleidoquery visualisierte Anfrage, in der Name und Alter von Personen in einer Datenbank abgefragt werden sollen. [40]

3.2.4 Diskussion

Die Unterteilung in formularbasierter, diagrammbasierter, iconbasierter und hybriden Anfragesprachen erscheint in Anbetracht der Tatsache, dass die meisten Sprachen in diagrammbasierten oder hybriden Anfragesprachen einzuordnen sind [44], zu grobkörnig. Im Rahmen dieser Arbeit wird vorgeschlagen die diagrammbasierten Anfragesprachen in graphenbasierte, mengenbasierte und flussbasierte Anfragesprachen weiter zu unterteilen.

In der Regel visualisiert eine visuelle Anfragesprache eine formale Sprache. Im einfachsten Fall werden die Elemente einer formalen Sprache dabei durch visuelle Repräsentationen ersetzt. In diesem Fall sind sich visuelle und formale Anfragesprache sehr ähnlich. Soll eine visuelle Anfragesprache aber so entworfen werden, dass sie sowohl Anfragen an eine relationale, als auch an eine objektorientierte, sowie an eine semantische Datenbank stellen kann, so muss sie so generisch sein, dass sie in verschiedene formale Anfragesprachen übersetzt werden kann. Sie orientiert sich in diesem Fall durch nicht mehr so stark an den einzelnen Konzepten einer bestimmten formalen Anfragesprache und verfolgt generischere Ansätze. Die Ähnlichkeit zu einer formalen Sprache kann daher ein weiteres Unterscheidungsmerkmal visueller Anfragesprachen darstellen.

3.2.5 Filter-/Flow-Graphen

Filter-/Flow-Graphen stellen ein Visualisierungskonzept zur Darstellung von Datenflüssen dar. Filter-/Flow-Graphen bedienen sich dabei der Metapher eines Flusses, dessen Wasser auf dem Weg von der Quelle zum Ziel gefiltert wird. In einer Anfrage fließen so die Daten der Datenquelle in einem Datenfluss zur Ergebnismenge und werden unterwegs durch Knoten gefiltert. Wie ein Fluss kann sich auch ein Datenfluss verzweigen und wieder zusammenfließen. Dies lässt sich auch mit der Parallel- und Reihenschaltung von Strom vergleichen. In einer Verzweigung lassen sich die Daten im Fluss getrennt vom restlichen Fluss filtern, fließt der Fluss wieder zusammen wird der gesamte Datenfluss gefiltert. Passieren Daten im Datenfluss einen Filter oder werden diese durch ihn herausgefiltert, lässt sich dies direkt durch die Visualisierung der Flussbreite ablesen. Die Flussbreite verringert sich, wenn Daten durch einen Filter entfernt werden. Werden alle Daten herausgefiltert, so versiegt der Fluss. Bleibt die Flussbreite gleich, wurden keine Daten herausgefiltert. Filter-/Flow-Graphen bildeten damit azyklische Graphen und sind besonders zur Darstellung boolescher Ausdrücke sowie von UND-/ODER-Verknüpfungen geeignet. [50]

Das Konzept der Filter/Flow-Visualisierung geht auf Ben Shneiderman zurück, der das Konzept 1991 in seiner Arbeit [9] zum ersten Mal vorstellte und zusammen mit Degi Young in ihrer gemeinsamen Arbeit [50] 1993 durch eine Benutzerstudie evaluierte. Dabei ergab sich, dass es Nutzern deutlich leichter fiel, eine Anfrage mithilfe einer Filter-/Flow-Visualisierung zu erstellen als mit einer formalen Anfragesprache. Eine Filter-/Flow-Visualisierung einer Anfrage nach dem von Ben Shneiderman entworfenen Konzept ist in Abbildung 20 abgebildet. In der abgebildeten Anfrage wird eine Datenbank mit Informationen über Angestellte angefragt. Gesucht werden dabei Angestellte, die in Kalifornien arbeiten und 50 000 $ verdienen. Der Datenfluss verläuft in der abgebildeten Darstellung von der Datenquelle links zur Ergebnismenge rechts. Die Orientierung des Datenflusses ist dabei variabel, in [9] stellt Ben Shneiderman beispielsweise den Datenfluss in einer weiteren Beispielanfrage von oben nach unten dar.

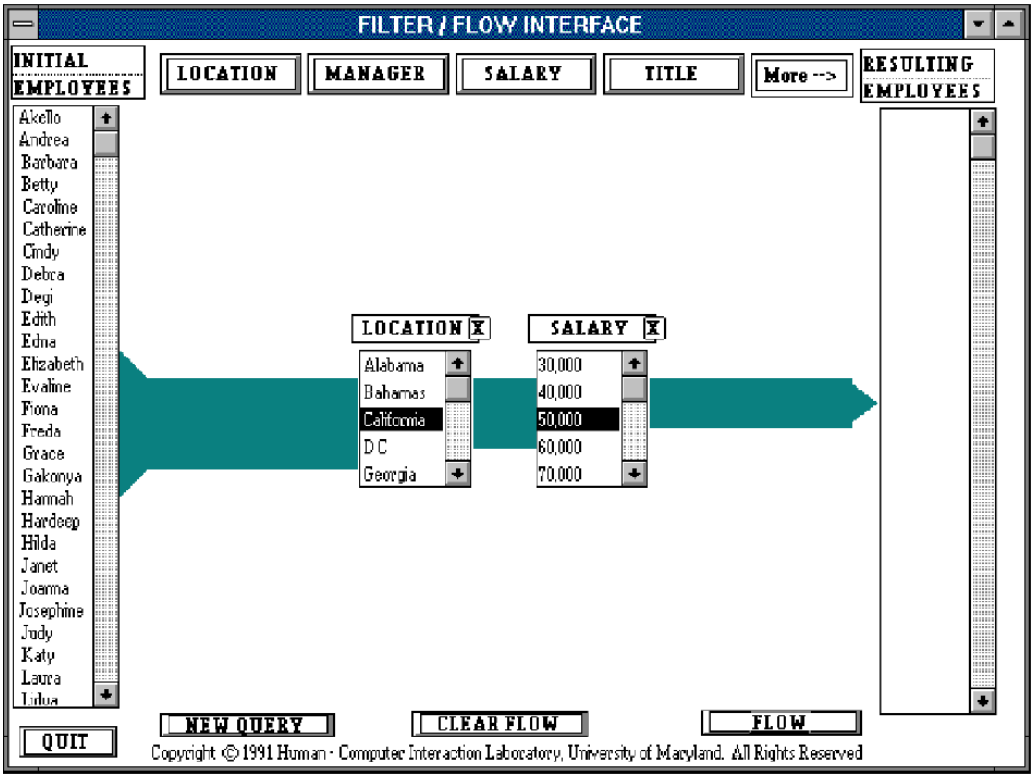


Abbildung 20: Eine Filter-/Flow-Visualisierung einer Anfrage, in der eine Datenbank aus Angestellten bezüglich Beschäftigungsort und ihrer Bezahlung gefiltert werden soll. [50]

3.2.6 Vereinfachte Filter-/Flow-Graphen

Obwohl sich Filter-/Flow-Graphen zur Visualisierung von Datenbankanfragen eignen, resultieren komplexe Anfragen mit vielen Bedingungen in ebenso komplexe Graphen, deren Übersichtlichkeit verloren geht. In der Vergangenheit wurden zahlreiche Erweiterungen für Verbesserungen am Filter-/Flow-Konzept vorgeschlagen, wie zum Beispiel die farbliche Hervorhebung von Knoten, nutzerdefiniertes Graphenlayout oder multiple Ergebnismengen. Diese Vorschläge beschränken sich auf die allgemeine Verbesserung der visuellen Darstellung des Graphen, statt auf konzeptionelle Verbesserungen. Zur Reduktion der Komplexität von Filter-/Flow-Graphen für komplexe Anfrage wurde das ursprüngliche Konzept von Shneiderman am Institut für Visualisierung und Interaktive Systeme der Universität Stuttgart konzeptionell weiterentwickelt. In der Arbeit [51] von Florian Haag, Steffen Lohmann und Thomas Ertl wird das Ergebnis dieser Weiterentwicklung in Form eines vereinfachten Filter-/Flow-Visualisierungskonzepts vorgestellt. Während das ursprüngliche Konzept von Shneiderman [9] als gerichteter Graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ betrachtet werden kann, wobei V die Filterknotenmenge und $E \subseteq V \; x \; V$ die Kantenmenge im Graph beschreiben. In dem Filterknoten einen eingehenden und einen ausgehenden Datenfluss enthalten, erweitert die vereinfachte Filter-/Flow-Visualisierung die Knoten um multiple eingehende und ausgehende Datenflüsse. Dadurch lassen sich Filterknoten zusammenfassen und so die Menge an Filterknoten reduzieren, was zu einer übersichtlicheren Visualisierung von Anfragen führt. Abbildung 21 ver-

anschaulicht die Struktur der vereinfachten Filterknoten. Eingehende Datenflüsse werden dabei als Rezeptoren und ausgehende als Emitter beschrieben.

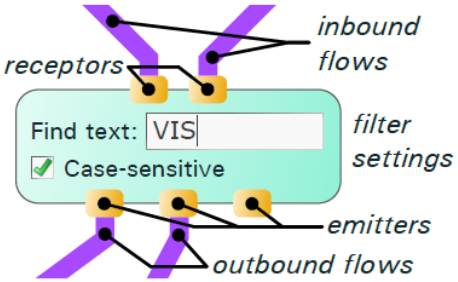


Abbildung 21: Schematische Darstellung eines Filterknotens im vereinfachten Filter-/Flow-Konzepts. [52]

Ein erweiterter Filter-/Flow-Graph lässt sich formal als ein Graph $\dot{G} = (\dot{V}, \dot{E}, I, O)$ beschreiben, wobei I für die Menge der Rezeptoren und O für die Menge der Emitter steht. Rezeptoren und Emitter sind jeweils nur einem einzigen Knoten zugewiesen. Die Filterknotenmenge wird mit $\dot{V} \subseteq P(I)x\ P(O)$ beschrieben. Jeder ursprüngliche Filter-/Flow-Graph lässt sich als vereinfachter Filter-/Flow-Graph darstellen. Wie dies formal bewerkstelligt werden kann, ist in der Arbeit [51] beschrieben. Zwei Knoten v_1 und v_2 werden als direkt miteinander verbunden bezeichnet, wenn eine Kante (o,i) mit o e O_1, $v_1 = (I_1,O_1)$ und i e I_2, $v_2 = (I_2,O_2)$ existiert, was sich durch $v_1 \rightarrow v_2$ ausdrücken lässt. Ein Graph G = (V,E) lässt sich so als ein Graph G = (V,E,I,O), unter der Annahme G = (V,E,I,O) = ($\{(\{i_n\},\{o_n\}) \mid v_n \varepsilon V\}$, $\{(o_n,i_m) \mid (v_n,v_m) \varepsilon E\}$, $\{i_n \mid v_n \varepsilon V\}$, $\{o_n \varepsilon v_n \varepsilon V\}$ }, beschreiben.

Zur Reduktion der Komplexität und Substitution der Filterknoten sieht das vereinfachte Filter-/Flow-Konzept beispielsweise folgende Regeln, von denen es aber beliebig viele geben kann, vor:

- **Bedingte Filter (Conditional Filters)**
  Identische Filterknoten mit unterschiedlichen Werten, wie beispielsweise wahr und falsch können in einem Filterknoten zusammengefasst werden.

- **Verschachtelte Mengenoperationen (Nested Set Instances)**
  Mengenoperationen, die sich auf verschiedene Variablen beziehen, lassen sich in einem einzigen Filterknoten zusammenfassen.

- **Binäre Operatoren (Binary Operator Filters)**
  Durch diese Regel lassen sich Filter mit den selben Variablen auf der linken Seite eines Ausdrucks zu einem einzigen Filter mit verschiedenen Emitterwerten für die

rechten Variablen eines Ausdrucks ersetzen.

- **Bereichsfilter (Between Filters)**
  Wertvergleiche in einem bestimmten Bereich, wie beispielsweise alle Werte für X zwischen 10 und 15 lassen sich in einem einzigen Filterknoten darstellen.

Abbildung 22 veranschaulicht die Substitutionsverfahren in einer übersichtlichen Tabellenansicht mit visualisierten Beispielen.

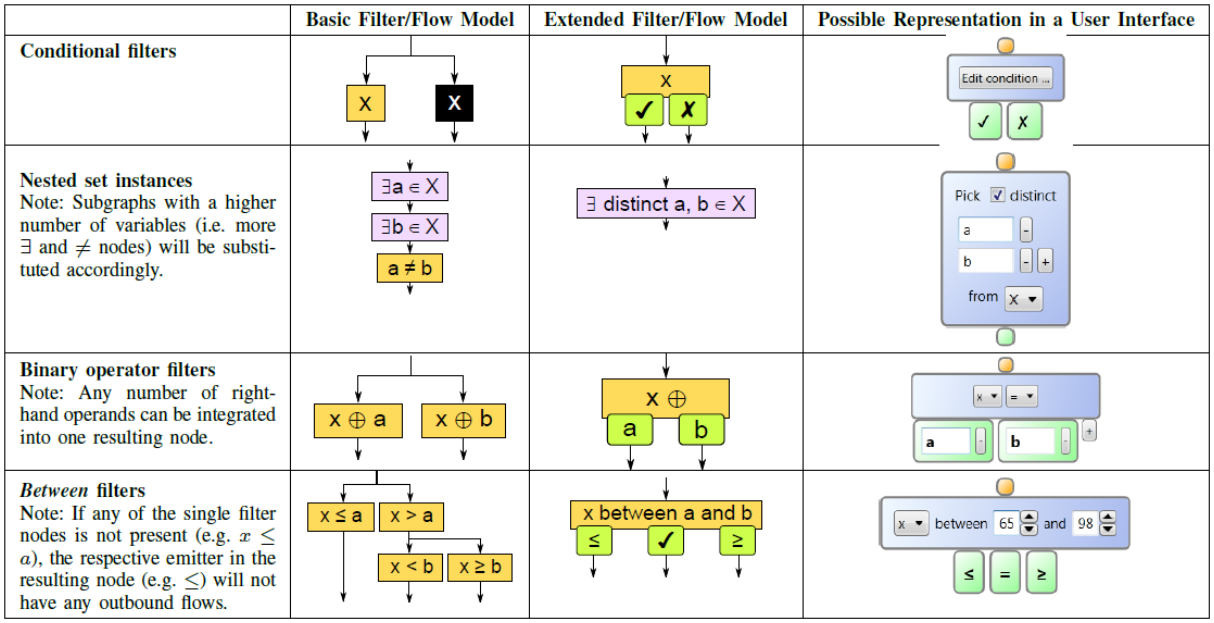


Abbildung 22: Beispiele für die Minimierung von Filter-/Flow-Graphen durch eine komprimiertere Darstellung von Knoten. [51]

Das erweiterte Konzept vereinfachter Filter-/Flow-Graph wurde im Rahmen einer Benutzerstudie [53] evaluiert. Viele der befragten Probanden empfanden die vereinfachte Filter-/Flow-Visualisierung gegenüber Anfragen, die mit dem ursprünglichen Filter-/Flow-Konzept von Shneiderman [9] visualisiert wurden, einfacher zu lesen. Aufgrund der Ergebnisse aus [53] soll das im Rahmen dieser Arbeit zu entwickelnde Visualisierungskonzept auf dem vereinfachten Filter-/Flow-Visualisierungskonzepts aufbauen.

4. Themenverwandte Arbeiten

In diesem Kapitel werden thematisch verwandte Arbeiten, die sich mit der Visualisierung von Informationsanfragen befassen, vorgestellt. Im Folgenden werden jeweils verschiedene Arbeiten und deren Visualisierungskonzepte vorgestellt und bezüglich der folgenden Fragestellungen, die auf der Arbeit [45] basieren, analysiert:

- Wie werden Anfragen erstellt und visualisiert?
- Ausdrucksmöglichkeiten einer visuellen Anfragesprache.
- Wie wird das Ergebnis einer Anfrage visualisiert?
- Für welche Nutzer ist eine visuelle Anfragesprache gedacht oder geeignet?

4.1 Yahoo! Pipes

Yahoo! Pipes wurde von den Yahoo! Mitarbeitern Pasha Sadri, Ed Ho, Jonathan Trevor, Kevin Cheng und Daniel Raffel entwickelt und erlaubt es Nutzern unterschiedliche Datenquellen des Web2.0, wie zum Beispiel RSS-Feeds, miteinander zu kombinieren. Beispielsweise können so die Daten der News-Feeds von Google und Yahoo! nach Schlüsselwörtern durchsucht werden. Eine dazu entsprechende Yahoo! Pipe findet sich in Abbildung 23.

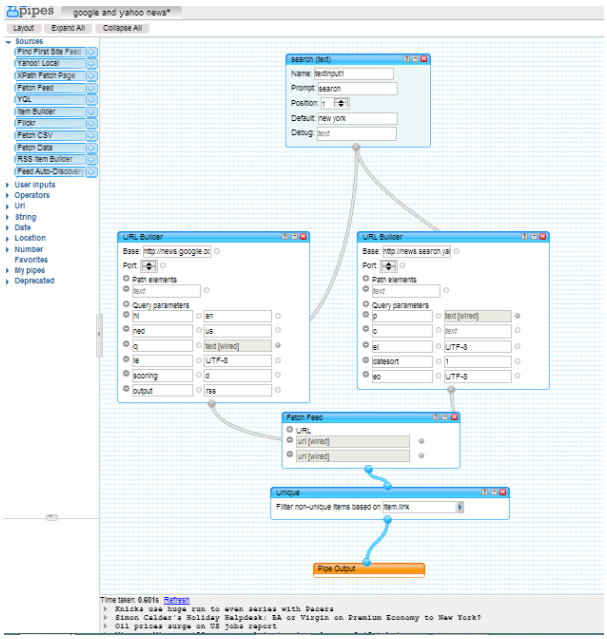


Abbildung 23: Eine Yahoo! Pipe, die die beiden News-Feeds von Google und Yahoo! miteinander kombiniert und nach dem Schlüsselwort „New York“ durchsucht.[3]

Eine Yahoo! Pipe besteht dabei aus verschiedenen miteinander kombinierbaren Modulen. Jedes Modul kann über einen oder mehrere Dateneingänge, sowie über einen oder mehrere Datenausgängen verfügen. Die Datenausgänge eines Moduls können die Datenquellen für

[3] Abgebildete Yahoo! Pipes online aufrufbar unter (*Stand: 09.06.2013*): http://pipes.yahoo.com/pipes/pipe.edit?_id=1nWYbWm82xGjQylL00qv4w

ein anderes Module darstellen, das die Daten weiterverarbeitet. Yahoo! Pipes umfasst derzeit 48 verschiedener Module und kann aufgrund eines modularen Aufbaus durch weitere Module erweitert werden.

Eine Übersicht verschiedenster Yahoo! Pipes findet sich auf der Projektseite von Yahoo![4].

Mit Yahoo! Pipes lassen sich zwar keine semantischen Datenquellen anfragen, dennoch bietet Yahoo! Pipes ein interessantes und vielseitiges Konzept zur Visualisierung von Anfragen. MashQL bietet eine Umsetzung, der in Yahoo! Pipes angewendeten Konzepte, zur Visualisierung und Erstellung semantischer Anfragen. Eine Analyse bezüglich der in Yahoo! Pipes und MashQL umgesetzten Visualisierungskonzepten bezüglich der vorgestellten vier Kriterien wird in Abschnitt 4.2 erläutert.

[4] Eine Liste verschiedener Yahoo! Pipes findet sich unter (*Stand 09.06.2013*): http://pipes.yahoo.com/pipes/pipes.popular

4.2 MashQL

Der Frage, wie sich die in Yahoo! Pipes umgesetzten Visualisierungskonzepte auf semantische Anfragen anwenden lassen, sind Mustafa Jarrar und Marios D. Dikaiakos in ihrer Arbeit [41] nachgegangen. Das Ergebnis dieser Arbeit ist MashQL und bezeichnet ein auf Yahoo! Pipes aufbauendes Programm zur visuellen Erstellung semantischer Anfragen, das als webbasierte Anwendung sowie als Plugin für den Browser Mozilla Firfox in Version 3.5 umgesetzt wurde. Die semantischen Datenquellen beschränken sich bei MashQL allerdings auf RDF/OWL-Dateien. SPARQL-Endpoints können mit der in der Arbeit [41] vorgestellten Umsetzung nicht angefragt werden. Aufbau und die Struktur von Pipes sind dabei zu Yahoo! Pipes identisch. Der schemenhafte Aufbau einer durch MashQL visualisierten Anfrage ist in Abbildung 24 dargestellt.

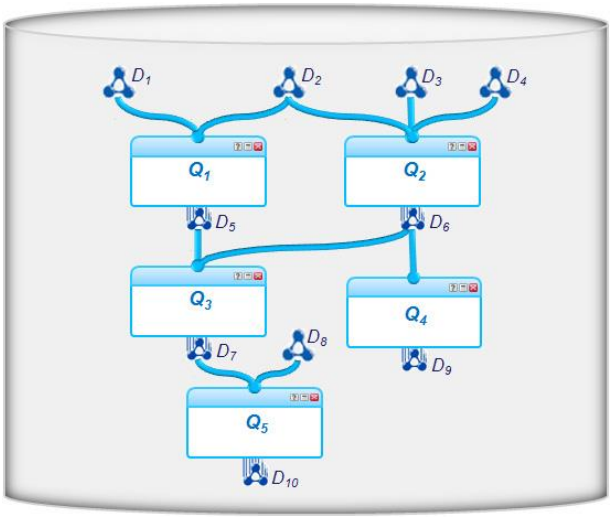


Abbildung 24: Aufbau und Struktur einer Anfrage in MashQL. Datenquellen werden in dieser Abbildung durch drei in Form eines Dreiecks verbundener Punkte, sowie einem D mit tiefgestellter Nummerierung dargestellt. Module werden in einem an ein Windows-Fenster erinnerndes Rechteck mit einem Q und einer tiefgestellten Nummerierung dargestellt. [41]

4.2.1 Wie werden Anfragen erstellt und visualisiert?

Anfragen werden in MashQL und Yahoo! Pipes auf Basis von Filter-/Flow-Visualisierungskonzepten erstellt. Dabei werden Module, die im linken Bereich der Oberfläche mittels Drag&Drop in den Graphen eingefügt werden, miteinander verbunden. Ein Modul kann von der Filterung bis zur Erzeugung von Daten unterschiedlichste Aufgaben erfüllen. Die Datenausgänge eines Moduls können dabei als Dateneingänge anderer Module dienen, wenn diese miteinander verbunden werden. Abbildung 25 veranschaulicht die Weboberfläche von MashQL anhand einer Beispielanfrage.

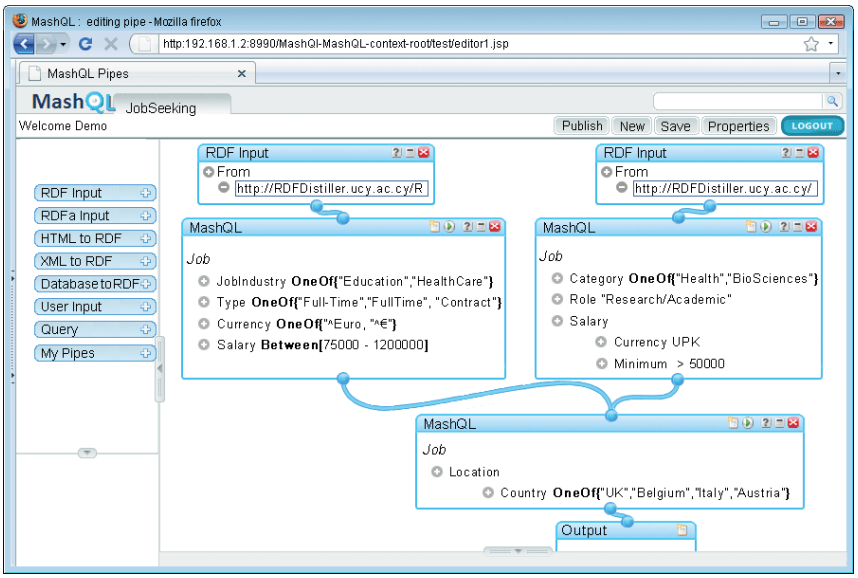


Abbildung 25: Oberfläche von MashQL zur Visualisierung semantischer Anfragen. [41]

Durch die Anwendung von Filter-/Flow-Visualisierungstechniken lassen sich UND-/ODER-Verknüpfungen sehr anschaulich darstellen. Die Deklaration einer UND-/ODER-Verknüpfung ist in MashQL aber auch innerhalb eines Moduls, wie in Abbildung 26 dargestellt, möglich.

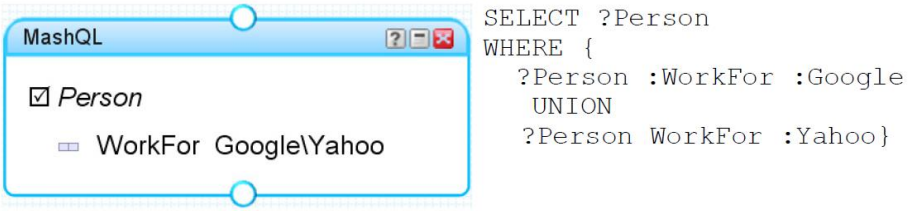


Abbildung 26: Eine ODER-Verknüpfung innerhalb eines Moduls. [41]

Durch diese Art der Darstellung lässt sich die Anzahl der benötigten Knoten innerhalb des Anfragegraphen reduzieren. Bei der Visualisierung von Datenflüssen in MashQL haben Filter keinen Einfluss auf die breite des Flusses. Die Flussbreite ist damit unabhängig von der darin enthaltenen Datenmenge.

4.2.2 Ausdrucksmöglichkeiten von MashQL

Laut [41] kann MashQL alle Anfragen in SPARQL 1.0 visuell abbilden. Dabei können aber keine SPARQL-Endpoints direkt abgefragt werden. Als Datenquellen kommen in MashQL ausschließlich RDF-Dateien zum Einsatz [41]. MashQL ermöglicht es dadurch auch Daten mehrerer RDF-Dateien und damit mehrerer Datenquellen gleichzeitig abzufragen.

Wie sich aber Anfragen mit MashQL erstellen lassen, die Ressourcen der Teilflüsse nach mehrstufigen Verweisen filtern können, wird in [41] nicht erläutert. Allerdings bietet MashQL, wie auch Yahoo! Pipes, die Möglichkeit das Konzept durch selbst erstellbare Module zu ergänzen und ist daher prinzipiell erweiterbar. Ein Beispiel soll diese Art komplexerer Anfragen und deren Anwendungsfall an einem ähnlichen Beispiel veranschaulichen. Angenommen

man bildet zwei Teilflüsse. Im ersten Teilfluss wurden Jobangebote der Firma Yahoo! herausgefiltert, im zweiten Teilfluss Jobangebote der Firma Google. Es sollen jetzt die Jobangebote der Firma Yahoo! ermittelt werden, die ein höheres Gehalt anbieten, als das am besten bezahlte Jobangebot von Google. Das Ergebnis dieser Anfrage lässt sich mit MashQL zumindest über mehrere Anfragen ermitteln. Dazu ermittelt man zunächst aus der Menge der Jobangebote von Google das Jobangebot heraus, das am besten bezahlt wird und vergleicht diesen Zahlenwert mit einer Abfrage, in der ermittelt wird, ob ein Jobangebot von Yahoo! über diesem Zahlenwert bezüglich der Vergütung liegt.

4.2.3 Für welche Nutzer ist MashQL gedacht oder geeignet?

MashQL beschreibt wie Yahoo! Pipes ein Visualisierungskonzept für Datenbankanfragen, dass sich vor allem an Nutzer ohne informationstechnisches Hintergrundwissen richtet. [41]

4.3 NITELIGHT

In ihrer Arbeit [44] präsentieren Alistair Russell, Paul R. Smart, Dave Braines und Nigel R. Shadbolteine visuelle Anfragesprache für semantische Anfragen auf Basis von SPARQL.

4.3.1 Wie werden Anfragen erstellt und visualisiert?

Eine Anfrage in NITELIGHT bildet dabei einen gerichteten Graphen, der aus vier verschiedenen Knotentypen gebildet werden kann. Zur Verfügung stehen Knotentypen für Variablen, Relationen und Werte. In Abbildung 27 werden die verschiedenen Knotentypen in NITELIGHT aufgelistet.

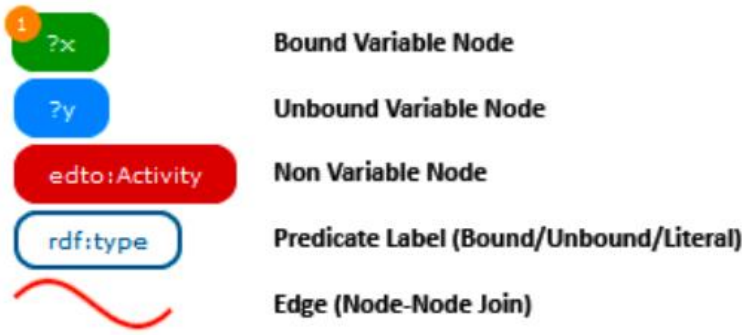


Abbildung 27: Knotentypen eines in NITELIGHT visualisierten Anfragegraphen. [44]

Ein SPARQL-Anfrage besteht im Wesentlichen aus miteinander verknüpften Tripeln aus Subjekten, Prädikaten und Objekten beziehungsweise Datenwerten. Das Grundprinzip der Visualisierung von SPARQL-Anfragen wird in Abbildung 28 dargestellt.

Abbildung 28:Visualisierungsprinzip einer Bedingung innerhalb der WHERE-Deklaration einer SPARQL-Anfrage in NITELIGHT. [44]

Alle Relationen, die sich auf die gleiche SPARQL-Variable beziehen, werden mit dieser verbunden. Daraus entsteht ein gerichteter Graph, wie in Abbildung 29 dargestellt.

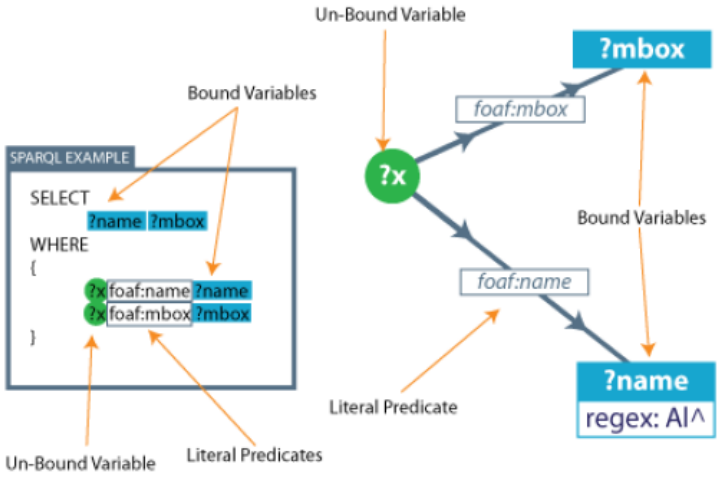


Abbildung 29: Visualisierung einer einfachen SPARQL-Anfrage mit zwei verknüpften Bedingungen, die über einen UND-Operator miteinander verbunden sind. [44]

Enthält eine Anfrage mehrere Variablen, die in keiner Relation zueinander stehen, entstehen in NITELIGHT so mehrere, nicht miteinander verbundene, gerichtete Graphen. Variablen, deren Werte im Ergebnis aufgelistet werden sollen, werden blau markiert. Filter, die sich auf eine Variable beziehen werden unterhalb der entsprechenden Variable dargestellt.

Die Reihenfolge, in der die Variablen in der Ergebnismenge aufgelistet werden, lässt sich durch eine kleine Nummer im linken oberen Bereich eines Knotens ablesen. Abbildung 30 veranschaulicht dies.

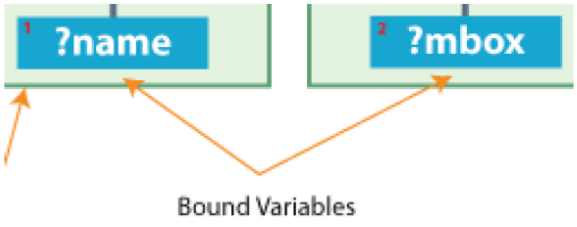


Abbildung 30: Die Reihenfolge in der Variablen in der Ergebnismenge aufgelistet werden sollen, lässt sich durch eine kleine rote Nummer ablesen. [44]

Durch die gewählte Darstellung lassen sich ODER-Verknüpfungen nicht so anschaulich darstellen, wie in einem Filter-/Flow-Graphen. Zur Visualisierung von ODER-Verknüpfungen wird eine spezielle Verbindung eingeführt. Nach dem gleichen Prinzip wird auch mit Anfragebedingungen verfahren, die über den OPTIONAL-Ausdruck in SPARQL miteinander verbunden sind. Abbildung 31 stellt eine Anfrage, die eine ODER-Verknüpfung enthält, im Visualisierungskonzept von NITELIGHT dar, Abbildung 32 eine OPTIONAL-Verknüpfung.

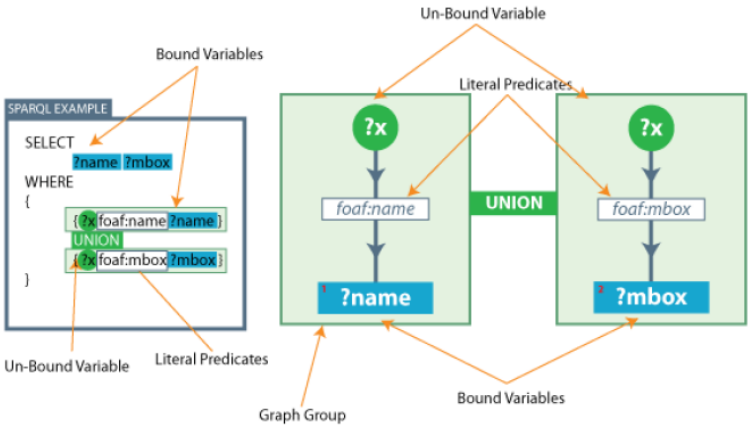


Abbildung 31: Visualisierung einer ODER-Verknüpfung in NITELIGHT. [44]

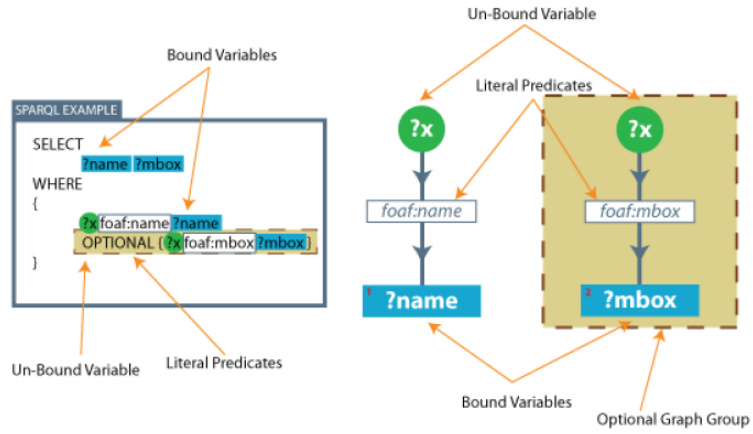


Abbildung 32: Visualisierung einer SPARQL-Anfrage, die eine OPTIONAL-Verknüpfung enthält. [44]

Das Visualisierungskonzept von NITELIGHT wurde durch einen javabasierten Prototyp realisiert. Zur Navigation innerhalb einer Ontologie wurde in der Realisierung von NITELIGHT eine Facettennavigation implementiert. Abbildung 33 veranschaulicht den Aufbau des Prototypen. Ein Knoten kann über ein Kontextmenu, wie in Abbildung 33 zu sehen, bearbeitet werden. Weitere Knoten lassen sich unter anderem auch durch Drag&Drop von Klassen, Individuen und Relationen aus der Ontologienavigation in den Anfraggraphen hinzufügen. [44].

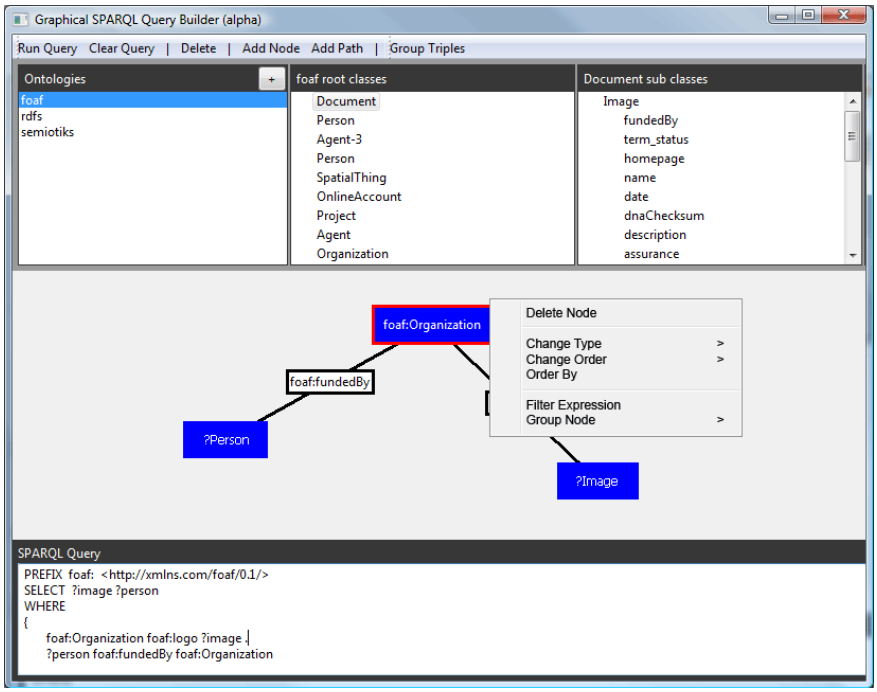


Abbildung 33: Prototypische Realisierung des NITELIGHT Konzepts, mit einer Facettennavigation zur Navigation innerhalb einer Ontologie. [44]

4.3.2 Ausdrucksmöglichkeiten von NITELIGHT

Das durch NITELIGHT umgesetzte Visualisierungskonzept ermöglicht durch seine starke Anlehnung an SPARQL, prinzipiell alle Anfragen zu visualisieren, die sich mit SPARQL ausdrücken lassen. In ihrer Arbeit [44] schreiben die Autoren, dass sie sich bisher auf SELECT-Anfragen konzentriert haben, andere Anfragearten aber eine denkbare Erweiterung von NITELIGHT darstellen könnten.

4.3.3 Wie wird das Ergebnis einer Anfrage visualisiert?

Zum Aufruf des Ergebnisses muss kein spezieller Ergebnisknoten, wie in Yahoo! Pipes oder MashQL verwendet werden. Das Ergebnis lässt sich jederzeit über einen sogenannten „ResultViewer“ in Form einer Tabellendarstellung abrufen.

4.3.4 Für welche Nutzer ist NITELIGHT gedacht oder geeignet?

NITELIGHT wurde für Nutzer konzipiert, die bereits Erfahrung im Umgang mit SPARQL haben oder SPARQL mit Unterstützung von NITELIGHT erlernen möchten. [44]

4.4 GQL – Graphical Query Language

GQL wurde von Guntis Barzdins, Sergejs Rikacovs, Martins Zviedris in ihrer Arbeit [GQL] vorgestellt und beschreibt eine visuelle Anfragesprache für semantische Datenquellen, die zur Anfragevisualisierung auf Konzepte aus UML aufbaut. Dabei hat man sich zur Visualisierung vor allem an UML-Klassendiagrammen orientiert. Abbildung 34 veranschaulicht eine Beispielanfrage in GQL.

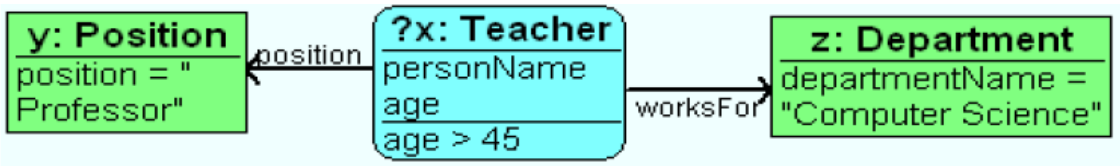


Abbildung 34: Eine Anfrage in GQL. Gesucht werden Lehrer (Teacher) über 45 Jahre, die für ein Abteilung (Department) mit der Bezeichnung "Computer Science" arbeiten und die Position "Professor" inne haben. [GQL]

4.4.1 Wie werden Anfragen erstellt und visualisiert?

Anfragen werden in GQL durch einen gerichteten Graphen nach dem Vorbild von UML-Klassendiagrammen visualisiert. Knoten im Graphen visualisieren dabei für semantische Klassen, die Klassenbezeichnung steht dabei im oberen Bereich des Knotens in hervorgehobener Schrift. Dem UML-Konzept entsprechend werden DataProperties im Knoten selbst durch Attribute abgebildet, ObjectProperties durch Verbindungslinien zwischen Knoten. Die Ergebnismenge lässt sich durch die Wahl von Knoten, die blau hervorgehoben werden, spezifizieren. Blau hervorgehobene Knoten repräsentieren dabei in der Ergebnismenge Ressourcen einer Klasse für die in der Anfrage spezifizierbare Bedingungen gelten. Datenwerte, beispielsweise Name und Alter eines Lehrers (Teachers) können durch Hinzufügen der entsprechenden DataProperties innerhalb eines blau markierten Knotens abgefragt werden. Die abzufragenden DataProperties werden, wie in Abbildung 34 dargestellt, von denjenigen DataProperties getrennt, die nur entsprechenden Anfragebedingungen einer Klassen deklarieren. Beispielsweise werden in der Anfrage in Abbildung 34 nach Lehrern (Teachers) gesucht, die über 45 Jahre alt sind. Es wird angenommen, dass sich die DataProperty „age“ dabei auf Jahre bezieht.

Über sogenannte „Context Frames“ lassen sich Anfragebedingungen über OPTIONAL und anderen Verknüpfungsarten miteinander verbinden. Abbildung 35 veranschaulicht eine Beispielanfrage mit einem „Context Frame“, in dem die Anfragebedingung FILTER (!bound(?a)) visualisiert wird.

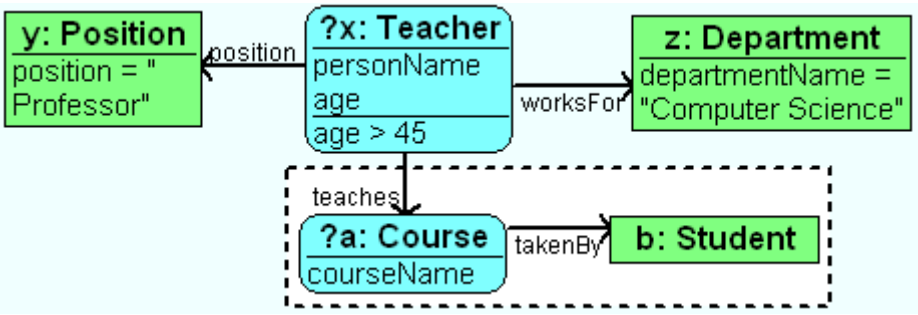


Abbildung 35: Eine in GQL visualisierte Anfrage mit einem negierten Filter (bound), der über ein „Context Frame“ aus einem gestrichelten Rechteck visualisiert wird. Angefragt wird hier die gleiche Anfrage wie aus Abbildung 34, mit der Erweiterung das für die gesuchten Lehrer (Teacher) zusätzlich gelten muss, dass sie keine Kurse unterrichten. [GQL]

4.4.2 Ausdrucksmöglichkeiten von GQL

Durch das auf UML basierte Visualisierungskonzept kann GQL nicht nur Anfragen sondern auch die Datenbasis einer semantischen Datenquelle visualisieren. Die Ausdrucksmöglichkeiten von Anfragen mit GQL sind allerdings stark begrenzt. In der Arbeit [GQL] wird keine Möglichkeit zur Erstellung von Anfragebedingungen, die über eine ODER-Verknüpfung miteinander verbunden sind, beschrieben. Die Verwendung von „Context Frames“ reduziert die intuitive Lesbarkeit von in GQL visualisierten Anfragen. Die Suche nach Individuen, sowie mehrstufigen Verweisen sind mit GQL nicht vorgesehen. So kann man beispielsweise nicht nach Lehrern (Teacher) suchen, die dasselbe Fach, wie ein bestimmter Lehrer (Teacher) in einer anderen Abteilung (Department), unterrichten.

4.4.3 Wie wird das Ergebnis einer Anfrage visualisiert?

Die Ergebnisse einer Anfrage werden in Form einer Tabelle dargestellt. Zum Erstellen eines Anfrageergebnisses wird kein spezieller Ergebnisknoten, wie in MashQL benötigt. Die Reihenfolge kann nicht innerhalb der Anfrage, wie beispielsweise in NITELIGHT manipuliert werden. Die prototypische Umsetzung in GQL bietet an Ergebnisse über eine Exportfunktion zu speichern. Abbildung 36 veranschaulicht die Ergebnisanzeige anhand einer Beispielanfrage. [GQL]

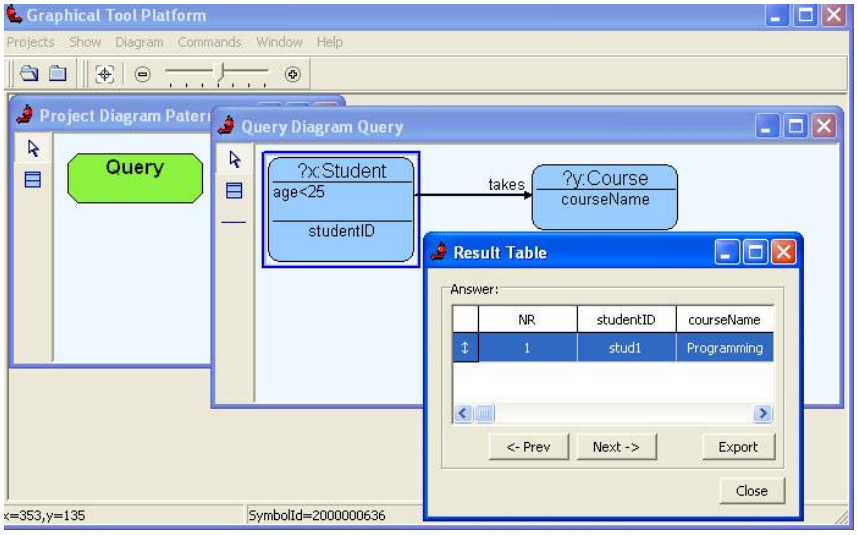


Abbildung 36: Prototypische Realisierung von GQL mit einer erstellten Anfrage und dessen Ergebnis in Form einer Tabellendarstellung. [GQL]

4.4.4 Für welche Nutzer ist GQL gedacht oder geeignet?

GQL wurde für Nutzer aus dem medizinischen Arbeitsbereich für den Umgang mit Statistiken entworfen und getestet [GQL]. Generell ist GQL aber sicherlich für alle Nutzer geeignet, die sich bereits mit UML-Klassendiagrammen auskennen.

4.5 Kaleidoquery

Kaleidoquery ist eine visuelle Anfragesprache für objektorientierte Datenbanksysteme, die auf Filter-/Flow-Graphen basierte Visualisierungsansatze aufbaut. Das Visualisierungskonzept von Kaleidoquery wird in der Arbeit [40] von Norman Murray, Norman Paton und Carole Goble vorgestellt.

4.5.1 Wie werden Anfragen erstellt und visualisiert?

Die Orientierung eines Datenflusses ist in Kaleidoquery stets von unten (Datenquelle) nach oben (Ergebnismenge) dargestellt. Zur Visualisierung setzt Kaleidoquery auf eine Kombination aus Icons und Text. Studien haben gezeigt, dass dies für Nutzer leichter verständlich ist, als ausschließlich auf Text oder Icons basierende Visualisierungsformen [40].

In Abbildung 37 ist eine Beispielanfrage in Kaleidoquery dargestellt. Gesucht werden Personen, die unter 20 Jahre und über 16 Jahre alt sind oder „Smith“ heißen. Die Ergebnismenge soll die Personen umfassen, auf die diese Eigenschaften zutreffen. Dabei wird durch die Filter-/Flow-Visualisierungstechnik auf sehr anschauliche Art und Weise ein über UND-/ODER-Verknüpfungen verbundener Datenfluss dargestellt.

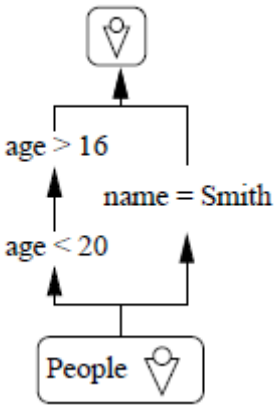


Abbildung 37: Eine Beispielanfrage in Kaleidoquery. Gesucht werden Personen in der Datenquelle, die zwischen 17 und 19 Jahre alt sind oder „Smith“ heißen. [40]

Negierte Filter werden mit schwarzer Hintergrundfarbe unterlegt. Besteht ein Teilfluss ausschließlich aus negierten Filtern, so kann dieser zur weiteren visuellen Unterstützung als gestrichelte Verbindungslinie, wie in Abbildung 38, dargestellt werden.

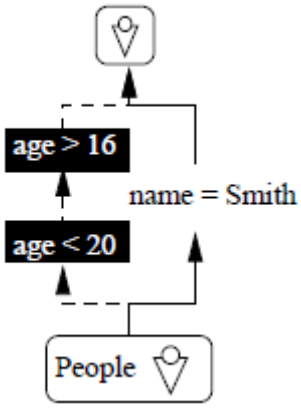


Abbildung 38: Darstellung eines Teildatenflusses, der ausschließlich aus negierten Filter besteht. [Kaledoquery]

Die im Ergebnis zurückgegebenen Attribute eines Objekts lassen sich im obersten Knoten spezifizieren. So lassen sich beispielsweise deklarieren welche Objekte und welche Attribute in der Ergebnismenge enthalten sein sollen. Werden mehr Attribute ausgewählt, als im Knoten angezeigt werden können, so wird ein Haken neben dem Symbol für den Objekttyp angezeigt. Sollen alle Attribute in der Ergebnismenge enthalten sein, so wird nur das Symbol für den Objekttyp im Knoten dargestellt. Abbildung 39 veranschaulicht dies an einem Beispiel.

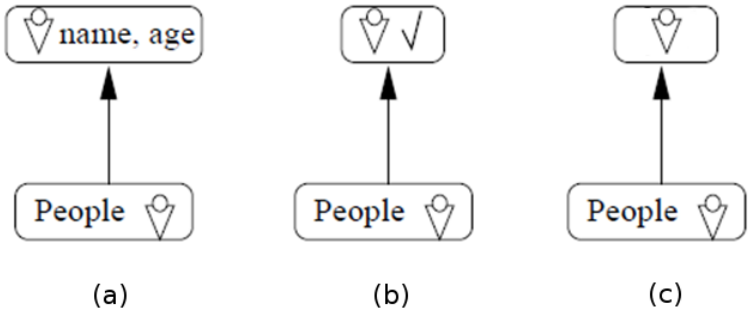


Abbildung 39: Verschiedene Formen zur Spezifikation der Ergebnismenge. [40]

In den bisherigen Beispielen wurden nur Anfragen mit Filtern erstellt, die sich auf einfache Datentypen, wie beispielsweise einer Zeichenkette oder eines Integerwerts bezogen haben. Im folgenden Beispiel wird die Visualisierung von Anfragen mit Filtern, die sich auf Objekte und deren Attribute beziehen, vorgestellt. Abbildung 40 zeigt eine Anfrage, in der der Name einer Firma gesucht wird, die Angestellte beschäftigt, die über 60 Jahre alt sind oder deren Gehalt größer oder gleich 25000 beträgt. Erfüllt mindestens ein Angestellter einer Firma diese Bedingungen, so ist der Name der Firma in der Ergebnismenge enthalten.

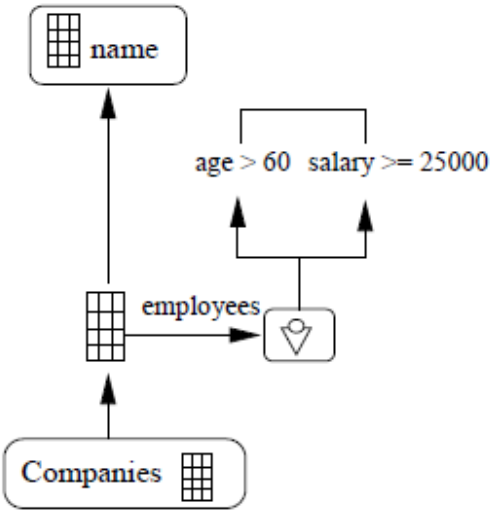


Abbildung 40: Visualisierung einer Anfrage mit einem Objektvergleich. Gesucht wird der Name aller Firmen, die Mitarbeiter beschäftigen, die über 60 (Annahme: Jahre) alt sind oder mehr als 25000 (Annahme: Dollar) verdienen. [40]

Mit Kaleidoquery lassen sich Bedingungen und Teilflüsse auch Quantifizieren. So ist es möglich die Anfrage aus Abbildung 40 umzuformulieren, so dass beispielsweise nur Namen von Firmen gesucht werden, die ausschließlich Mitarbeiter beschäftigen, die über 60 Jahre alt sind oder deren Gehalt größer oder gleich 25000 beträgt. In Abbildung 41 werden zwei Anfragen mit einer „all“ und einer „exists“ Quantifizierung eines Teilflusses einer Anfrage dargestellt. In Anfrage (a) aus Abbildung 34 muss die Bedingung für alle Angestellte einer Firma erfüllt sein, Anfrage (b) aus Abbildung 41 ist dagegen äquivalent mit der Anfrage aus Abbildung 40.

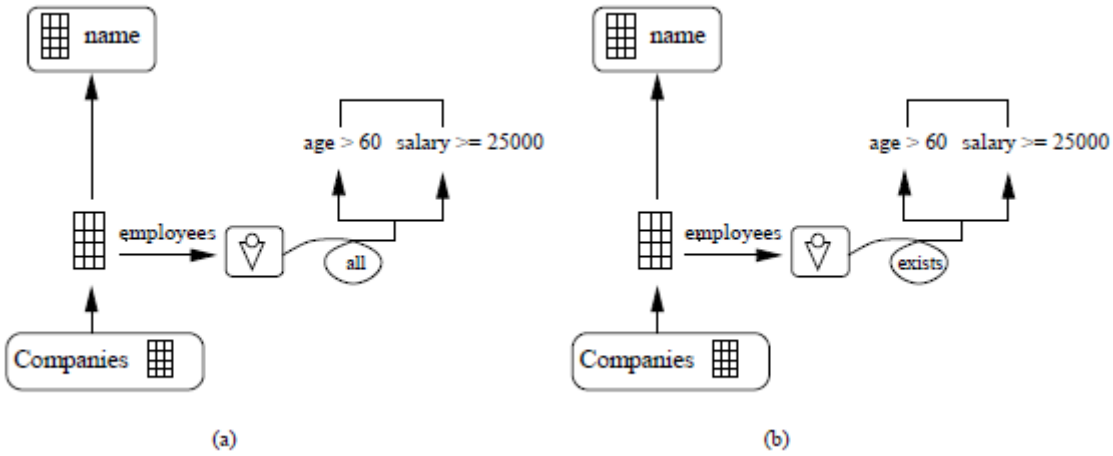


Abbildung 41: In Anfrage (a) wird der Name von Firmen gesucht, deren Mitarbeiter alle entweder über 60 (Annahme: Jahre) alt sind oder über 25000 (Annahme: Dollar) verdienen.
In Anfrage (b) wird der Name von Firmen gesucht, die mindestens einen Mitarbeiter beschäftigen, der über 60 (Annahme: Jahre) alt ist oder über 25000 (Annahme: Dollar) verdient. [40]

Darüber hinaus bietet Kaleidoquery auch Möglichkeiten zur Deklaration von Anfragen, die Aggregatsfunktionen enthalten können, sowie zur Strukturierung und Sortierung der Ergebnismenge. Abbildung 42 veranschaulicht eine Anfrage, in der die Anzahl der Personen innerhalb einer Datenbank über die Funktion „count“ ermittelt werden sollen.

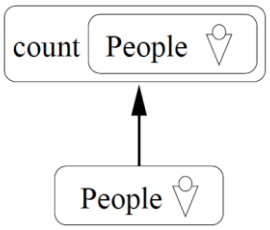


Abbildung 42: In dieser Anfrage soll die Anzahl der Personen in einer Datenbank ermittelt werden. [40]

Bei der Visualisierung von Datenflüssen in Kaleidoquery haben Filter keinen Einfluss auf die breite des Flusses. Die Flussbreite ist damit unabhängig von der darin enthaltenen Datenmenge. Auch haben Flüsse, die sich verzweigen Einfluss aufeinander, auch wenn sie in der Visualisierung nicht mehr zusammengeführt werden. In Abbildung 40 verzweigt sich beispielsweise der Datenfluss und die Bedingungen für die Mitarbeiter einer Firma werden deklariert. Beide Datenflüsse werden zur Erzeugung des Ergebnisses nicht wieder vereint, obwohl beide Flüsse Einfluss auf die Ergebnismenge nehmen. In diesen Punkten weicht das Visualisierungskonzept von Kaleidoquery von dem Filter-/Flow-Konzept von Shneiderman [9] ab.

4.5.2 Ausdrucksmöglichkeiten von Kaleidoquery

Da Kaleidoquery für objektorientierte Datenbanksysteme und nicht für Anfragen an semantische Datenquellen konzipiert wurde, wird an dieser Stelle nicht weiter auf die Ausdrucksmöglichkeiten von Kaleidoquery eingegangen. Grundsätzlich lassen sich die meisten der vorgestellten Visualisierungskonzepte von Kaleidoquery auch im Rahmen einer visuellen Anfragesprache für semantische Datenquellen übertragen.

4.5.3 Wie wird das Ergebnis einer Anfrage visualisiert?

Über die Visualisierung von Ergebnissen werden in der Arbeit [40] keine Aussagen getroffen.

4.5.4 Für welche Nutzer ist Kaleidoquery gedacht oder geeignet?

Das Ziel von Kaleidoquery ist es auch Nutzer anzusprechen, die bisher keine Erfahrung im Bereich diagrammbasierter Darstellungen von Relatioonen, wie beispielsweise in Form von Entity-Relationship-Diagrammen, haben. Damit eignet sich Kaleidoquery auch für Nutzer mit nur geringem informationstechnischen Hintergrundwissens.

4.6 Zusammenfassung

In diesem Kapitel wurden einige themenverwandte Arbeiten vorgestellt, die sich mit der Umsetzung visueller Anfragesprachen beschäftigen. Es gibt trotz einiger Vorteile von Filter-/Flow-Visualisierungstechniken gegenüber rein graphbasierter Visualisierungsformen, wie beispielsweise eine anschaulichere Darstellung von UND-/ODER-Verknüpfungen, nur wenige praktische Umsetzungen dieser Visualisierungstechnik. Zur Navigation innerhalb von Ontologien und zur Auswahl von Klassen und Relationen werden, wie in [44] und [41] textbasierte beziehungsweise listenbasierte Darstellungsformen gewählt. Ergebnisse einer Studie [54], in der verschiedene Visualisierungstechniken mit dem Ontologie-Editor Protégé verglichen wurden, legen den Schluss nahe, dass sich listenbasierte Darstellungsformen zur Navigation in Ontologien eignen und das Probanden auf Basis dieser Darstellungsform einfache Fragen, wie beispielsweise “What is the year of birth of the Professor named Constantin Halatsis?” im Vergleich zu anderen Visualisierungstechniken schneller beantworten können [54].

5. Lösungskonzept

In diesem Kapitel wird das im Rahmen dieser Arbeit entwickelte Lösungskonzept beschrieben. Problemstellung, Zielsetzung und thematisch verwandte Arbeiten wurden in den vorangegangen Kapiteln bereits vorgestellt.

5.1 Konzept

Aufbauend auf die in den vorangegangen Kapiteln bereits vorgestellten Konzepte der Filter-/Flow-Visualisierung von Shneiderman, sowie auf die an der Universität Stuttgart entwickelte vereinfachte Darstellung von Filter-/Flow-Graphen, soll ein nutzerorientierter Ansatz zu Erstellung semantischer Anfragen entwickelt werden.

Das in diesem Kapitel vorgestellte Konzept visualisiert einen Datenfluss im Filter-/Flow-Graphen von der Datenquelle, die sich im oberen Bereich einer Anfrage befindet, zur Ergebnismenge, die sich im unteren Bereich einer Anfrage befindet. Die Orientierung des Flusses ist also bei Anfragen stets von oben nach unten gerichtet. Dieser Ansatz entspricht am meisten der dem Filter-/Flow-Konzept zugrunde liegenden Flussmetapher. Auch das Wasser in einem Fluss fließt von einer höher gelegenen Quelle flussabwärts, dies soll auch für den Datenfluss einer Anfrage gelten. In anderen Arbeiten wurde ebenfalls bereits eine solche Orientierung vorgestellt, wie beispielsweise in [50], [41], aber auch abweichende Flussorientierungen, wie beispielsweise in [9], [40].

Ein Datenfluss einer SPARQL-Anfrage kann nach Klassen, Individuen und Relationen und deren Werte gefiltert werden. Da SPARQL vielfältige Möglichkeiten zur Filterung von Daten bietet, setzen einige Visualisierungskonzepte, wie beispielsweise [40] und [41], zur Filterung auf eine vielfältige Auswahl an unterschiedlichen, miteinander kombinierbaren Knotentypen. In dem hier vorgestellten Konzept soll die Anzahl an Knoten zur Datenfilterung auf wenige unterschiedliche Knotentypen reduziert werden. Das Beschränken der Knotenanzahl gegenüber anderen Konzepten, wie MashQL [41] und Kaleidoquery [40], soll dem Nutzer die Bedienung erleichtern. Die unterschiedlichen Knotentypen werden dabei auf zwei Knoten sowie einen Datenquellknoten und Ergebnisknoten beschränkt. Klassenknoten ermöglichen die Filterung des Flusses nach Klassen und deren Individuen, Relationsknoten ermöglichen die Filterung nach Relationen und deren Werte. Auch lassen sich Knotentypen nicht wahllos miteinander kombinieren. So folgt auf einen Datenknoten stets ein Klassenknoten, auf einen Klassenknoten stets ein Relationsknoten oder ein Ergebnisknoten und auf einen Relationsknoten stets ein weiterer Relationsknoten oder ein Ergebnisknoten. Wird eine Anfrage zur Ermittlung alle Städte mit mehr als 1 000 000 Einwohnern erstellt, so wird der Datenfluss eines Datenquellknotens zunächst nach Ressourcen der Klasse „Stadt“ und anschließend nach Ressourcen der Klasse „Stadt“, die mit der entsprechenden Relation zur Ermittlung der

Einwohnerzahl und dem Datenwert vom Betrag 1 000 000 verbunden sind, gefiltert. Das Ergebnis der Anfrage lässt sich durch einen Ergebnisknoten einsehen.

Die vier Knotentypen, aus denen eine Anfrage bestehen kann, werden im Folgenden aufgelistet:

- Datenquellknoten
- Klassenknoten
- Relationsknoten
- Ergebnisknoten

Alle Knotentypen sind nach dem gleichen Grundprinzip aufgebaut. Sie bestehen aus den drei grundlegenden Komponenten:

- **Rezeptor**
  Ein Knoten kann einen Rezeptor besitzen, in den beliebig viele Datenflüsse hineinfließen können, die durch den Knoten gefiltert werden sollen. Ein Datenquellknoten besitzt keinen Rezeptor, da dieser als Datenquelle keine Daten aufnehmen muss.

- **Filter**
  Über den Filter eines Knotens lässt sich einstellen, nach welchen Kriterien die eingehenden Daten des Datenflusses gefiltert werden sollen.

- **Emitter**
  Ein Knoten kann beliebig viele Emitter besitzen. Über ein Emitter lässt sich ein Wert für den Filter definieren. Daten, die den Filterwert erfüllen, können den Knoten am Emitter passieren, Daten, die den Filterwert nicht erfüllen, werden herausgefiltert.

Eine Beispielanfrage zur Suche nach einem Buch mit dem Titel „Moby-Dick“ lässt sich mit diesem hier vorgestellten Konzept, wie in Abbildung 43 dargestellt, visualisieren.

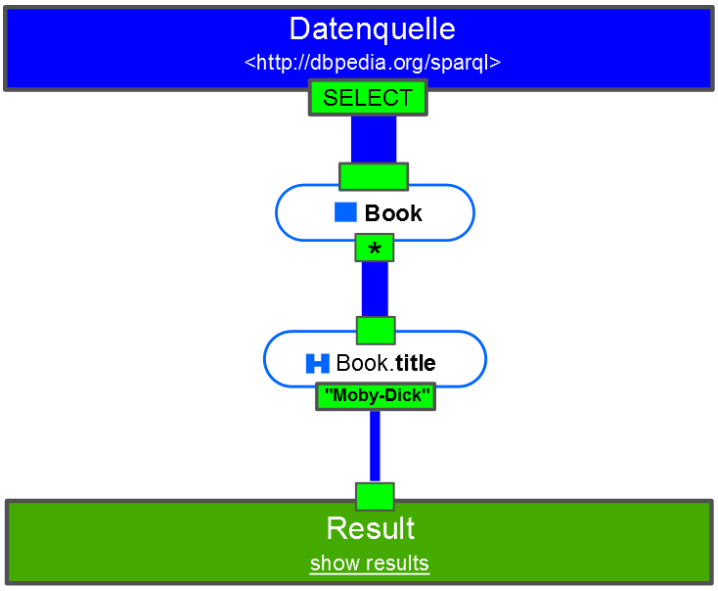


Abbildung 43: In dieser Beispielanfrage wird nach einem Buch mit dem Titel „Moby-Dick“ gesucht. Der Datenfluss fließt dabei von der Quelle durch zwei Filter, die durch die Filterung des Datenflusses dessen Datenmenge und damit dessen visuelle breite beeinflussen.

In Abbildung 43 wird der Datenfluss zunächst nach Ressourcen der Klasse „Book“ durchsucht. Durch das Ausfiltern von anderen Ressourcen, die nicht von der Klasse „Book“ sind, wird die Datenmenge im Fluss reduziert. Dies macht sich in der Breite des Datenflusses bemerkbar, der den Emitter des Klassenknotens verlässt. Im nächsten Knoten werden Ressourcen der Klasse „Book“ weitergehend nach Ressourcen, die über eine Relation „title“ mit dem Wert „Moby-Dick“ verbunden sind, gefiltert. Ressourcen, die alle Filterbedingungen erfüllen, landen in der Menge des Ergebnisknotens und können dort eingesehen werden.

In Abbildung 43 wurde eine UND-Verknüpfung betrachtet. UND- sowie ODER-Verknüpfungen werden in diesem Visualisierungskonzept entsprechend dem vereinfachten Filter-/Flow-Konzepts [51] dargestellt. UND-Verknüpfungen werden als Aneinanderreihung von Filterbedingungen (siehe Abbildung 43), ODER-Verknüpfungen als Verzweigung des Datenflusses, wie in Abbildung 44 dargestellt, visualisiert.

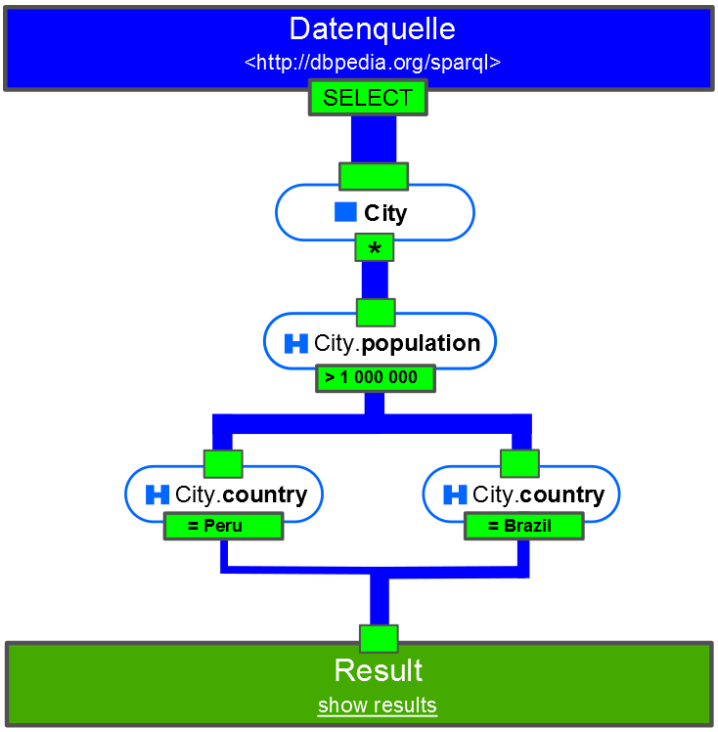


Abbildung 44: Eine Anfrage mit einer ODER-Verknüpfung. Gesucht werden Städte mit mehr als 1 000 000 Einwohnern aus Peru oder Brasilien.

In Abbildung 44 werden zwei Knoten, die sich auf die Relation „country“ einer Ressource der Klasse „City“ beziehen, erstellt. Knoten, die sich auf die gleichen Relationen einer Klasse beziehen, können zu einem Knoten, wie in Abbildung 45 dargestellt, zusammengefasst werden. Das Ergebnis der in Abbildung 45 dargestellte Anfrage ist dabei äquivalent zum Ergebnis der in Abbildung 44 dargestellten Anfrage.

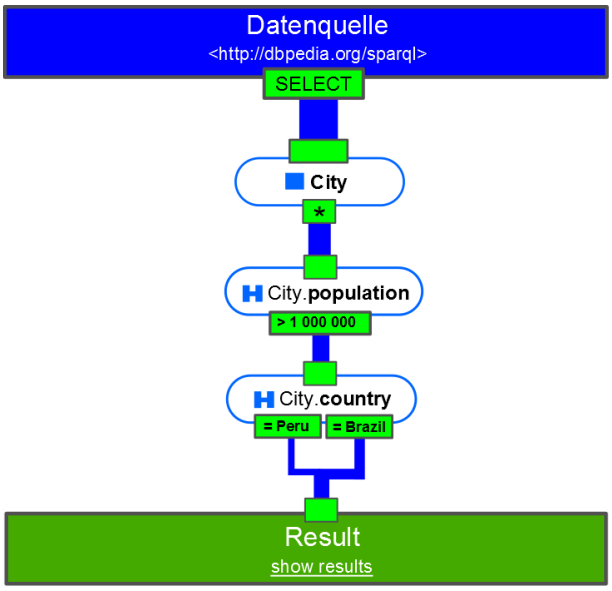


Abbildung 45: Eine zur in Abbildung 44 dargestellten äquivalente Anfrage.

Zwei Knoten, die sich auf die gleiche Relation einer Klasse beziehen und über eine UND-Verknüpfung miteinander verbunden sind, lassen sich ebenfalls zu einem Knoten zusam-

menfassen. In Abbildung 46 sind zwei Knoten, die sich auf die Relation „influenced_by“ einer Klasse „programming_language“ beziehen, abgebildet. Die Knoten sind dabei über eine UND-Verknüpfung miteinander verbunden. Abbildung 47 veranschaulicht, wie sich solche Knoten, die über eine UND-Verknüpfung miteinander verbunden sind, zusammenfassen lassen.

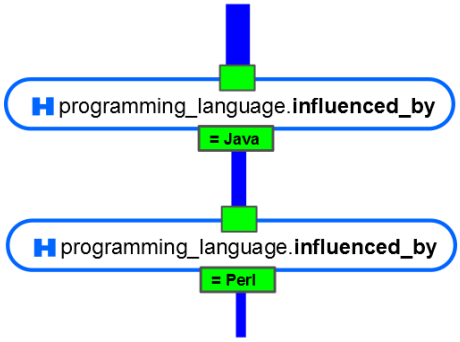


Abbildung 46: Zwei Knoten, die über eine UND-Verknüpfung miteinander verbunden sind. Gesucht wird durch die dargestellten Bedingungen eine Programmiersprache, die sowohl durch die Programmiersprache Java als auch durch die Programmiersprache Perl beeinflusst wurde.

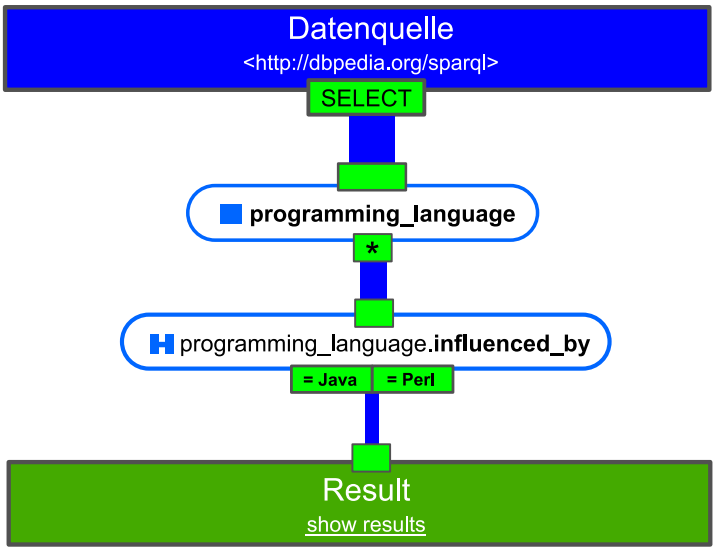


Abbildung 47: Eine Anfrage mit einem Knoten, dessen Emitter mit einer UND-Verknüpfung verbunden sind. In der abgebildeten Anfrage werden Programmiersprachen gesucht, die sowohl von der Programmiersprache Java als auch von der Programmiersprache Perl beeinflusst wurden.

Nur wenige Arbeiten, wie in Kapitel 4 beschrieben, ermöglichen in ihren Konzepten die Referenzbildung auf Objekten beziehungsweise Variablen, die sich aus Teilausdrücken der Anfrage ergeben. Verweise dieser Art ermöglichen es aber erst komplexe Anfragen zu erstellen, wie beispielsweise die Suche nach einem Buch dessen Autor für einen Verlag mit mehr als 1 000 000 € Umsatz im Jahr arbeitet. Mehrstufige Verweise, die sich auf Teilergebnisse

einer Anfrage beziehen, sollen in diesem Konzept berücksichtigt werden. Dabei soll aber auf Variablen, wie in [44] verwendet, verzichtet werden. Variablen sind ein Konzept, das vor allem im informationstechnischen Bereich Anwendung findet. Es wird im Rahmen dieser Arbeit angenommen, dass Nutzer, die nur wenig bis kein informationstechnisches Hintergrundwissen besitzen, mit der Verwendung von Variablen nicht vertraut sind. Stattdessen werden SPARQL-Variablen als unterschiedlich gefärbte Objekte behandelt, die im Fall von gleichen Labels durchnummeriert werden. Der Umgang mit unterschiedlichen Objekten, die auch den gleichen Bezeichner haben können, sollte jedem Nutzer vertraut sein. In Abbildung 48 wird eine Beispielanfrage, mit einem mehrstufigen Verweis auf Teilergebnisse der Anfrage veranschaulicht.

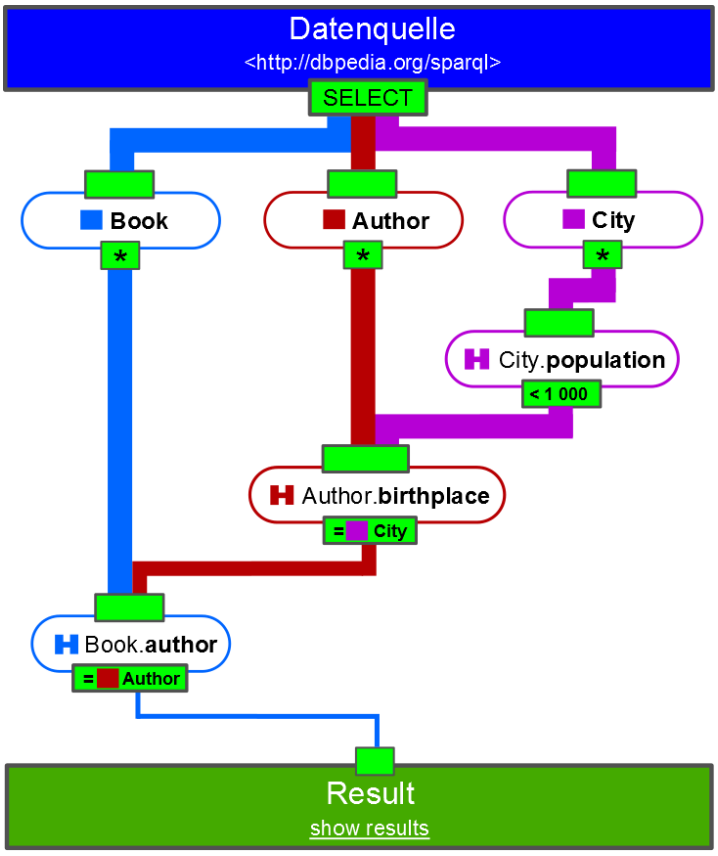


Abbildung 48: Eine Beispielanfrage mit mehrstufigen Verweisen auf Teilergebnisse einer Anfrage. Gesucht werden dabei Bücher von Autoren, die in einer Kleinstadt mit weniger als 1 000 Einwohnern stammen.

Um zu verhindern, dass Objekte mit gleichem Bezeichner verwechselt werden, besitzt jedes Objekt seine eigene Farbe, die auch die Einfärbung des Datenflusses beeinflusst. Zudem werden Objekte mit dem gleichen Bezeichner durchnummeriert. Abbildung 49 veranschaulicht eine dies.

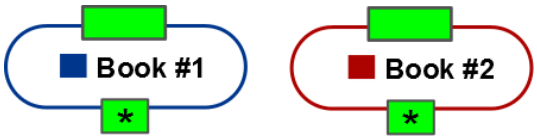


Abbildung 49: Klassenknoten mit gleichen Bezeichnern „Book“ sind nicht nur farblich unterschiedlich hervorgehoben, sondern lassen sich über ihre Nummerierung unterscheiden.

SPARQL 1.1 erweitert die Ausdrucksmöglichkeiten von SPARQL-Anfragen um viele Funktionen, wie beispielsweise arithmetische Operatoren, Unteranfragen (Subqueries), Aggregationsfunktionen oder Relationspfade. Diese erweiterten Ausdrucksmöglichkeiten sollen auch in diesem Konzept berücksichtigt werden. Die folgenden Beispiele veranschaulichen, wie diese SPARQL 1.1 Spracherweiterungen visualisiert werden können.

Abbildung 50 veranschaulicht eine Anfrage mit arithmetischem Operator. Gesucht werden Bücher, deren Endpreis unter einem Betrag von 20 $ liegen soll. Die Währung, auf die sich eine DataProperty bezieht, ergibt sich aus dem Kontext. Man verwendet daher in Ontologien häufig Relationen, die solche Informationen im Namen der Relation tragen, wie beispielsweise „priceInDollar“ oder „priceInEuro“. Der Endpreis ergibt sich in Abbildung 50 aus dem regulären Preis des Buches, sowie einem für ein Buch individueller Rabatt. Arithmetische Operatoren werden dabei im Emitter eines Knotens visualisiert.

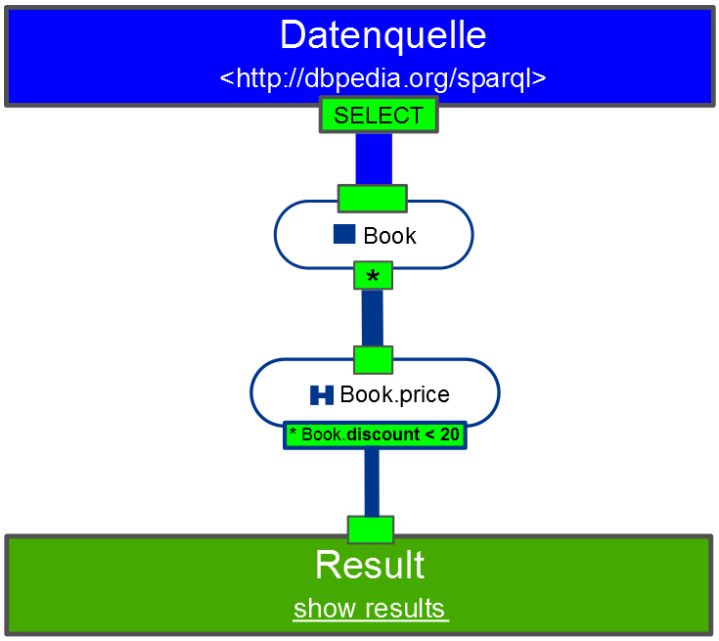


Abbildung 50: Emitter können arithmetische Operatoren enthalten, wie beispielsweise die hier dargestellte Multiplikation des Relationswerts für den Preis eines Buches mit dem Rabatt eines Buches. Das Produkt soll dabei kleiner als 20 betragen. In dieser Abbildung sind keine Währungsinformationen enthalten. Es ist unerheblich, ob es sich hierbei um eine bestimmte Währung wie zum Beispiel Dollars oder Euros handelt.

Aggregationsfunktionen werden wie arithmetische Operatoren im Emitter eines Knotens visualisiert. Eine Aggregationsfunktion, wie zum Beispiel „count“ wird dabei immer in den Emitter eines Knotens eingefügt, dessen Wert gezählt werden soll. Abbildung 51 veranschaulicht die Verwendung von Aggregationsfunktionen an einer Beispielanfrage, in der die Anzahl der Büchern in einer Datenquelle ermittelt werden soll.

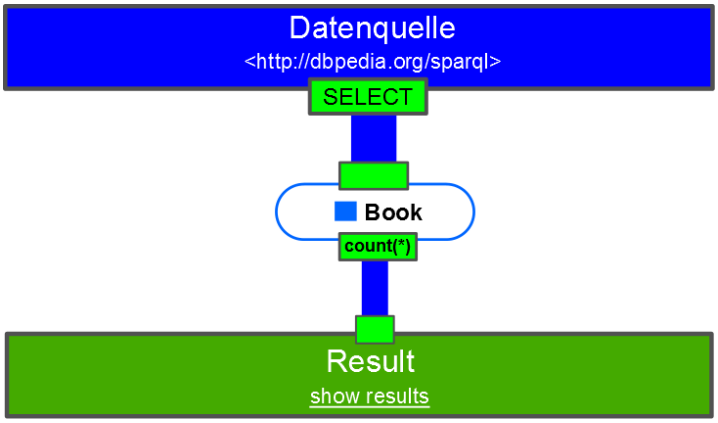


Abbildung 51: In dieser Anfrage wird die Anzahl der Ressourcen der Klasse „Book“ ermittelt. Hierfür wird die Aggregationsfunktion „count“ verwendet.

Anfragen über OPTIONAL werden in anderen Visualisierungskonzepten, wie beispielsweise in NITELIGHT als eine weitere Verbindungsart dargestellt. Statt weitere Verbindungsarten einzufügen und das Konzept dadurch nur unnötig komplizierter zu gestalten, werden Anfragen über OPTIONAL genauso wie ein Filter oder eine Aggregationsfunktion, wie in Abbildung 51 dargestellt, in Form eines Emitters mit der Beschriftung „OPTIONAL“ visualisiert.

Relationspfade [31] gehören ebenfalls zu einem sehr mächtigen Mittel in SPARQL 1.1. Die Zeichen, mit denen Relationspfade in SPARQL 1.1 angegeben werden, also beispielsweise „/“ für eine Pfadsequenz oder „^“ für einen inversen Relationspfad, werden in diesem Visualisierungskonzept übernommen. Damit wird vermieden, dass Nutzer, die bereits Erfahrung mit dem Schreiben von SPARQL-Anfragen haben, umlernen müssen. In Abbildung 52 wird eine Anfrage mit einem Relationspfad visualisiert. Hier werden alle Bücher des Autors Herman Melville, der u. a. Moby-Dick schrieb, gesucht werden.

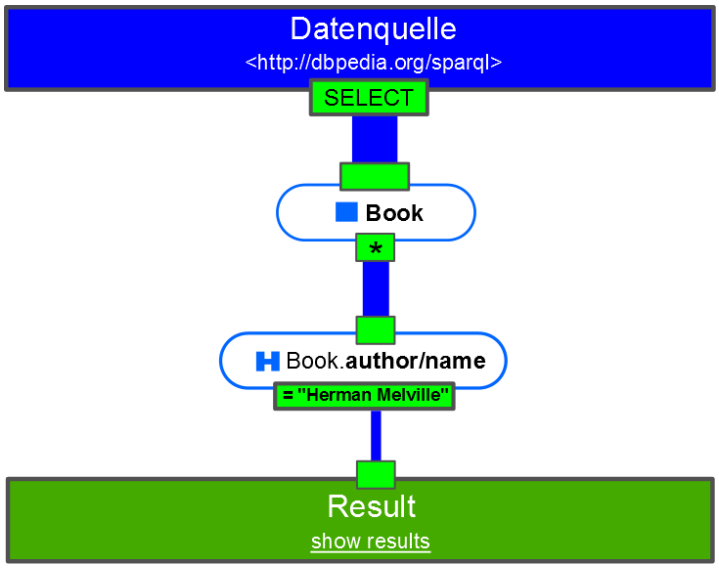


Abbildung 52: In der abgebildeten Anfrage wird ein Feature aus SPARQL 1.1 angewendet. Über einen Relationspfad kann über die Beziehung Book →author→name gleich auf den Namen des Autors eines Buches referenziert werden. Gesucht werden alle Bücher des Autors Herman Melville.

Anfragen, die auf dem Gesamtergebnis einer anderen Anfrage aufbauen und auch Unteranfragen beziehungsweise Subqueries genannt werden, werden durch SPARQL 1.1 ermöglicht. Subqueries bestehen dabei aus mindestens einer inneren und einer äußeren Anfrage. [31]

In diesem Visualisierungskonzept werden Subqueries durch das Hinzufügen weiterer Anfragen an einen Ergebnisknoten dargestellt. Die obere Anfrage stellt dabei immer die innere und die untere die äußere Anfrage da. Abbildung 53 veranschaulicht die Visualisierung von Subqueries in diesem Konzept an einem Beispiel.

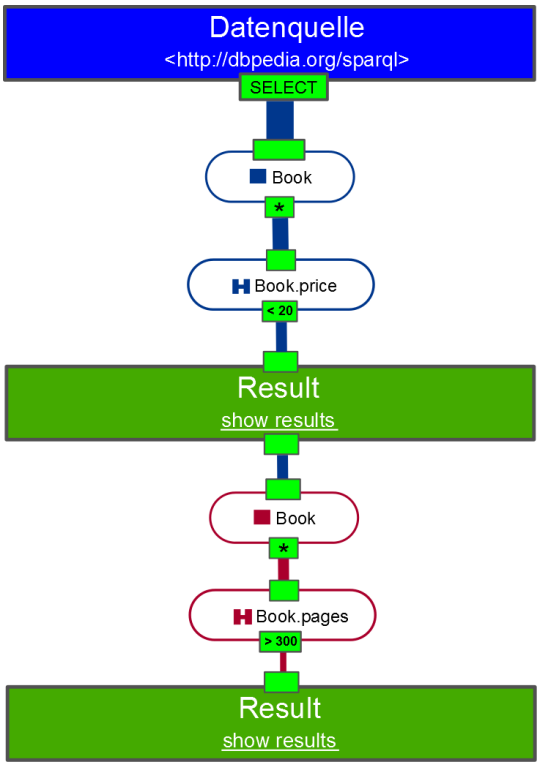


Abbildung 53: Eine visualisierte Subquery. In der inneren Anfrage wird nach Büchern gesucht, deren Preis unter einem Betrag von 20 (Dollar) liegt. Das Ergebnis der Anfrage wird in einer weiteren Anfrage nach den Büchern durchsucht, die über 300 Seiten haben.

Bisher wurden nur Anfragen betrachtet, in denen Ressourcen einer Klasse angefragt wurden, deren Relationswerte entweder aus einem Objekt oder einem einfachen Datenwert, wie beispielsweise einer Zeichenkette, bestand. Im folgenden Beispiel wird eine Anfrage betrachtet, die den Relationswert eines Individuums einer Klasse anfragt. In Abbildung 54 wird im Emitter des Klassenknotens das Individuum „Java“ ausgewählt. Der Datenfluss enthält dadurch nur noch die Ressource „Java“. Anschließend wird der Datenfluss nach Werten der Relation „influenced_by“ durchsucht, deren Wert im Ergebnis aufgelistet werden soll. Das Werte einer Relation in der Ergebnismenge aufgelistet werden soll, wird durch ein Fragezeichen im Emitter der Relation visualisiert.

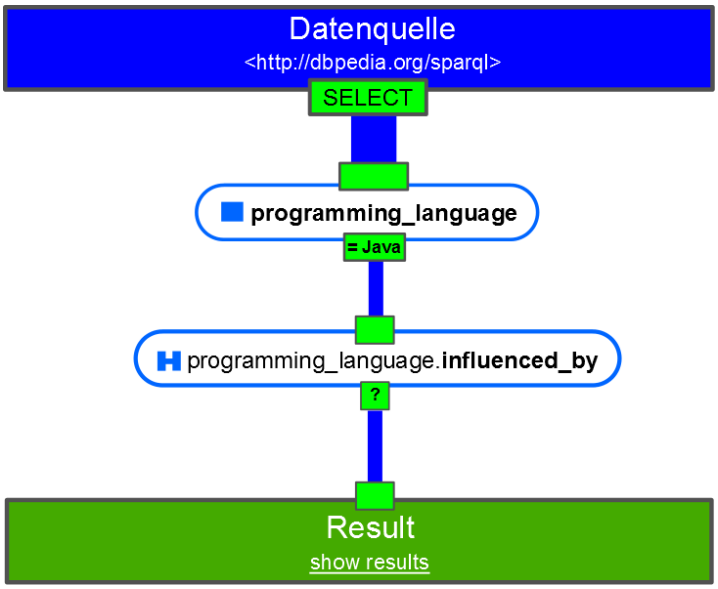


Abbildung 54: In dieser Anfrage sollen die Programmiersprachen gesucht werden, die die Programmiersprache Java beeinflusst haben.

Die SPARQL-Anfrageart lässt sich über den Emitter des Datenquellknoten auswählen. So lassen sich neben Anfragen über SELECT auch Anfragen über ASK, DESCRIBE und CONSTRUCT erstellen.

Eine SPARQL-Anfrage mit ASK ergibt ein Ergebnis, das entweder wahr oder unwahr sein kann. Entsprechend wird der Ergebnisknoten im Falle einer solchen Anfrage grün für ein wahres Ergebnis und rot für ein unwahres Ergebnis eingefärbt. Zudem werden auch die Emitter der Knoten entsprechend eingefärbt, je nachdem ob die durch den Knoten ausgedrückte Bedingung wahr oder unwahr ist. In Abbildung 55 werden zwei identische ASK-Anfragen dargestellt. Je nach Datensatz ist die Anfrage entweder wahr oder unwahr. Im ersten Fall wird angenommen, dass die Anfrage auf einem fiktiven Datensatz ein wahres Ergebnis zurückgibt und im zweiten Fall ein unwahres Ergebnis zurückgegeben wird. Die Visualisierung passt sich in diesem Konzept entsprechend dem Ergebnis an.

Abbildung 55: Dargestellt werden zwei ASK-Anfragen. In beiden Fällen wird angefragt, ob es eine Person gibt, die Alice kennt. In Anfrage (a) existiert im Datensatz eine Person, die Alice kennt. In Anfrage (b) existiert im Datensatz keine Person, die Alice kennt.

Die Visualisierung einer Anfrage über DESCRIBE ist mit einer Anfrage über SELECT nahezu identisch. Die Unterschiede beschränken sich darauf, dass im Emitter der Datenquelle das Schlüsselwort „DESCRIBE“ steht und die Ergebnismenge aus einer RDF-Datei besteht, die vom SPARQL-Endpoint zurückgegeben wird.

Anfragen über CONSTRUCT sind ähnlich wie Subqueries aufgebaut. Auch CONSTRUCT Anfragen lassen sich durch zwei hintereinander geschalteter Anfragen darstellen. Eine der beiden Anfragen definiert dabei das Schema des Rückgabegraphen, die andere den Graphen, der die Anfragebedingungen enthält.

5.2 Benutzeroberfläche

Zur Erstellung einer Anfrage kann ein Benutzer die SPARQL-Anfrageart festlegen, Knoten hinzufügen, löschen, sowie deren Filterwerte definieren und die Ergebnismenge am Ergebnisknoten einsehen. Diese Funktionen werden über Kontextmenus, Links und Auswahlfenster realisiert. Im Folgenden werden die Aspekte der Benutzeroberfläche erläutert.

5.2.1 Kontextmenu

Alle Knoten besitzen ein Kontextmenu, das Zugriff auf alle Funktionen, die ein Benutzer an einem Knoten ausführen kann, bietet. Über das Kontextmenu kann ein Benutzer einen Knoten löschen oder dessen Filterwert spezifizieren. Darüber hinaus kann er über das Kontextmenu weitere Knoten hinzufügen. In Abbildung 56 ist der Aufbau eines Kontextmenus anhand eines Klassenknotens dargestellt.

Abbildung 56: Ein Kontextmenu am Beispiel eines Klassenknotens. Kontextmenus lassen sich über einem Element durch Klicken der rechten Maustaste aufrufen.

5.2.2 Links

Links kennen Benutzer bereits durch den alltäglichen Umgang mit Webseiten. Die Auswahl des Filters eines Knotens, sowie dessen Emitterwerte können durch Links direkt aufgerufen werden. Abbildung 57 veranschaulicht einen Link am Beispiel eines Klassenknotens.

Abbildung 57: Über Links lassen sich Aktionen direkt ohne den Aufruf eines Kontextmenus ausführen.

5.2.3 Auswahlfenster

Filterwerte oder Klassen lassen sich durch Auswahlfenster, die bei deren Aufruf den Filter-/Flow-Graph überlagern, auswählen. Diese Fenster bieten eine Suchfunktion mit den sich Listen, wie beispielsweise einer Liste der auswählbaren Relationen einer Klasse, durchsuchen lassen. Übersicht und Navigation können über eine Breadcrumb-Navigation bereitge-

stellt werden. Abbildung 58 stellt eine schemenhafte Darstellung eines solchen Auswahlfensters für die Selektion einer Relation dar.

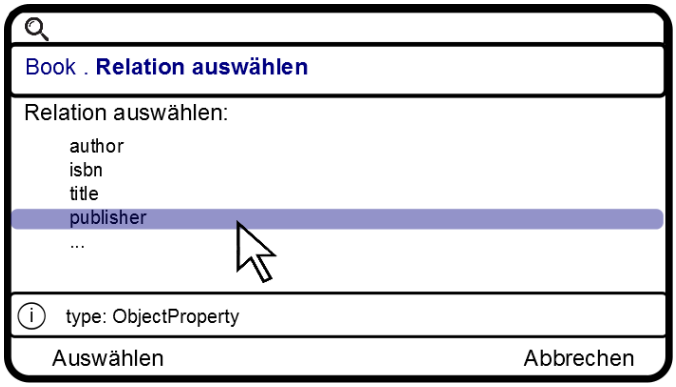


Abbildung 58: Schematische Darstellung eines Auswahlfensters für Relationen.

6. Implementierung

Die technische Realisierung des in Kapitel 5 vorgestellten Konzepts erfolgt auf Basis eines bestehenden Projekts, das in C# mit dem WPF-Framework umgesetzt wurde. Mit dem bestehenden Projekt lassen sich Filter-/Flow-Graphen erstellen. Im Rahmen dieser Arbeit wurde der Funktionsumfang des Projekts zur Anfragenerstellung auf SPARQL-Endpoints erweitert.

In diesem Kapitel wird zunächst ein grober Überblick über die Struktur des bestehenden Projekts gegeben und im Anschluss auf die Erweiterung der Implementierung eingegangen.

6.1 Überblick über die bestehende Implementierung

In diesem Abschnitt werden wichtige Komponenten des bestehenden Projekts aufgelistet und deren Funktion erläutert.

6.1.1 WPFFilterEditor

Die WPFFilterEditor-Komponente stellt das ausführbare Projekt zur Erstellung von Anfragen auf Basis der Filter-/Flow-Visualisierung da. Dabei greift es auf die Klassen der Komponenten FilterSolutions, FilterSolutions.Presentation, sowie FilterSolutions.Presentation.WPF zu.

6.1.2 FilterSolutions

Mit der FilterSolutions-Komponente werden die Datenobjekte für die Erstellung eines Filter-/Flow-Graphen bereitgestellt. Komponenten zur Realisierung einer bestimmten Aufgabe, wie zum Beispiel die Erstellung einer semantischen Anfrage, können diese Datenobjekte gemäß ihrer Aufgabe anpassen und erweitern.

Graph.cs	Implementiert das Datenmodell eines Graphen.
Node.cs	Implementiert das Datenmodell für Knoten in einem Graphen.
Connector.cs	Basisklasse für Verbindungsobjekte, mit denen Knoten miteinander verbunden werden können.
Emitter.cs	Verbindungsobjekte für ausgehende Datenflüsse.
Receptor.cs	Verbindungsobjekte für eingehende Datenflüsse.
Flow.cs	Datenmodell für Datenflüsse. Ein Datenfluss setzt sich dabei aus einem oder mehreren FlowPortions zusammen.

FlowPortion.cs	Eine FlowPortion ist ein Teilelement eines Datenflusses. FlowPortions lassen sich dazu einsetzen unterschiedliche Datenströme in einem Datenfluss zu beschreiben.
Serialization	Projektverzeichnis, das Klassen zur Speicherung von Graphen enthält.
Config	Projektverzeichnis, das Klassen zur Konfiguration des Filter-/Flow-Systems enthält.
IControlLink	Interface das Zugriff auf Zeichenfunktionen zur Erstellung, Löschung und Bearbeitung von Knoten und Flüssen ermöglicht.

6.1.3 FilterSolutions.Presentation

Diese Komponente bietet eine Abstraktionsschicht für die Darstellung von Datenobjekten der FilterSolutions-Komponente.

VisualGraph.cs	Visuelle Repräsentation einer Instanz der Klasse Graph.cs.
VisualNode.cs	Visuelle Repräsentation einer Instanz der Klasse Node.cs.
VisualNodeFactory.cs	Erstellt aus einem gegebenen Datenmodell-Knoten eine visuelle Repräsentation des Knotens.
FlowDisplayMode.cs	Eine Enumeration für die unterschiedlichen Arten der Visualisierung von Datenflüssen. • Flows: Nicht unterteilbare Datenflüsse • Portions: gleichmäßig unterteilbare Datenflüsse • WeigthedPortions: unterteilbare Datenflüsse mit unterschiedlicher Gewichtung.

6.1.4 FilterSolutions.Presentation.WPF

Die Implementierung der Abstraktionsschicht für die Darstellung von Datenobjekten der FilterSolutions-Komponente mit Hilfe des WPF-Frameworks.

FlowDisplay.cs	Klasse, die Funktionen zum Visualisieren eines Filter-/Flow-Graphen mit Hilfe von WPF bietet. Darunter fallen Funktionen wie beispielsweise das Erstellen und Laden

	eines Graphen, sowie das Hinzufügen und Entfernen von Graphelementen.
FlowElement.cs	Visualisierung eines Elements in einem Datenfluss mit Hilfe von WPF.
WPFGraph.cs	Implementierung einer Instanz der VisualGraph-Klasse mit Hilfe des WPF-Frameworks.
WPFNode.cs	Implementierung einer Instanz der VisualNode-Klasse mit Hilfe des WPF-Frameworks.
WPFConnectorElement	Implementierung eines Verbindungselement mit Hilfe des WPF-Frameworks.

6.1.5 Flow-Komponenten

Das Projektverzeichnis Flows enthält die C#-Projekte zur Realisierung von Filter-/Flow-Graphen für bestimmte Anwendungsgebiete, wie beispielsweise Filter-/Flow-Graphen für semantische Anfragen.

DBFilter	Implementiert verschiedene Knoten zur Erstellung von Datenbankanfragen mit Filter-/Flow-Graphen. Die Basisklasse für DBFilter-Knoten bildet dabei die DBFilter-Term-Klasse, die von der Node-Klasse aus dem Filter-Solutions-Projekt abgeleitet wurde.
DBFilterWPF	Projekt zur Visualisierung der DBFilter-Knoten mit Hilfe des WPF-Frameworks.

6.1.6 Backend-Komponenten

Im Projektverzeichnis Backends finden sich C#-Projekte zur Anfrage eines SPARQL-Endpoints und Hilfsfunktionen zur Erstellung von Anfragen. Zum Anfragen von SPARQL-Endpoints wird auf die „dotNetRDF“ Bibliothek zurückgegriffen.

6.2 Erweiterung der bestehenden Implementierung

Die Erweiterung der bestehenden Implementierung findet im Wesentlichen durch die SparqlDataHandler-Komponente statt. Darüber hinaus wurden auch Klassen und Funktionen bestehender Komponenten der bestehenden Implementierung erweitert, sowie neue hinzugefügt. Dies ist zum einen notwendig, damit Klassen in der modularen Architektur des Projekts miteinander kommunizieren können, sowie um neue Funktionen in bestehenden Kom-

ponenten bereitzustellen. Dieser Abschnitt gibt eine Übersicht und Erläuterung der Aufgaben, die durch die SparqlDataHandler-Komponente erfüllt werden.

6.2.1 SparqlDataHandler-Komponente

Die SparqlDataHandler-Komponente befindet sich im Projektverzeichnis Flows und realisiert in Zusammenspiel die folgenden Aufgaben:

- Konfigurationsoptionen
- Datenmodell und Knotentypen
- Erstellung asynchroner Anfragen an SPARQL-Endpoints
- Bereitstellung abgefragter Informationen
- Visualisierung und Benutzeroberfläche

Zentrales Element der Komponente bildet die SparqlDataHandler-Klasse mit ihren Funktionen, die die Konfiguration der Komponente ermöglicht und Schnittstellen zu anderen Projektkomponenten bildet. Um Aufbau und Zusammenspiel der Komponente und ihrer Klassen und Funktionen zu verstehen, wird die Implementierung anhand der aufgelisteten Aufgaben im Folgenden erläutert.

6.2.1.1 Konfigurationsoptionen

Die SparqlDataHandler-Klasse bietet Konfigurationsmöglichkeiten für die SparqlDataHandler-Komponente. Beispielsweise lässt sich die Komponente hier aktivieren und deaktivieren, sowie die Default-URL für den anzufragenden SPARQL-Endpoint festlegen. Die Konfiguration erfolgt dabei über das Festlegen von Variablenwerten im Quellcode.

6.2.1.2 Datenmodell und Knotentypen

In den Unterverzeichnissen „Data“ und „WPF“ finden sich die Klassen für die Bereitstellung des Datenmodells und der Knotentypen. Die Unterteilung ermöglicht eine schnelle Unterscheidung zwischen Daten- und Knoten-Klassen, sowie der Visualisierung dieser unter Verwendung des Windows Presentation Frameworks (WPF).

Das „Data“ Verzeichnis enthält in dessen Unterverzeichnis „Classes“ Klassen zur Implementierung des Datenmodells. Im Unterverzeichnis „Nodes“ befinden Klassen zur Implementierung der Knoten.

Das „WPF“ Verzeichnis enthält Klassen zur Visualisierung von Elementen mit Hilfe des Windows Presentation Frameworks. Klassen zur Visualisierung von Knoten befinden sich im Unterverzeichnis „Nodes“.

SparqlObject

Die SparqlObject-Klasse dient als Datenobjekt zur Speicherung semantischer Daten, wie beispielsweise einer semantischen Klasse, Relation oder eines Individuums. Eine Instanz der Klasse SparqlObject verfügt über die folgenden Eigenschaften:

Datentyp	Name der Eigenschaft	Funktion
String	URI	Speichert die URI einer semantischen Ressource.
String	Label	Speichert den Wert der Relation „rdfs:label“ der jeweiligen Ressource.
String	Prefix	Speichert den Präfix einer semantischen Ressource.

Tabelle 1: Eigenschaften der Klasse SparqlObject

SparqlObjekte werden in der SparqlDataHandler-Klasse in einer Liste gespeichert.

DataHandlerVariable

Die DataHandlerVariable-Klasse dient zur Speicherung von Klassen, die im Anfrage-Graph eingefügt wurden. Sie enthalten das SparqlObject, dass die Daten der SPARQL-Klasse enthält, als auch einen Verweis zum Knoten, sowie eine ID für die Erstellung von Sparql-Ausdrücken.

GenericSparqlObject

Das GenericSparqlObject dient dazu Listen aus Objekten der Klassen DataHandlerVariable und SparqlObject zu erstellen. Die Klasse wird dann eingesetzt, wenn eine Liste aus Objekten beider Klassen erstellt werden soll.

SparqlExpression

Die SparqlExpression-Klasse dient zur Speicherung eines semantischen Ausdrucks in Form eines Tripels aus Subjekt, Prädikat und Wert, der aus einem Objekt, Datenwert, einer verketteten Relation oder einer Verknüpfung mit einem weiteren semantischen Ausdrucks.

Datentyp	Name der Eigenschaft	Funktion
DataHandlerVariable	SparqlSubject	Zur Speicherung eines semantischen Subjekts.
DataHandlerVariable	SparqlProperty	Zur Speicherung einer semantischen Relation.
List<PropertyValue>	PropertyValueList	Zur Speicherung des Werts eines semantischen Ausdrucks.

Tabelle 2: Eigenschaften der Klasse SparqlExpression

PropertyValue

Die PropertyValue-Klasse dient zur Speicherung des Werts eines semantischen Ausdrucks. Zur Speicherung der unterschiedlichen Werte, die ein Ausdruck SPARQL 1.1 annehmen kann, bietet die Klasse verschiedene Klasseneigenschaften, die in der folgenden Tabelle 3 aufgelistet werden. Dabei kann ein PropertyValue selbst wieder auf eine SparqlExpression verweisen, um SPARQL-Ausdrücke, wie beispielsweise Relationspfade oder arithmetische Operationen, auszudrücken.

Datentyp	Name der Eigenschaft	Funktion
SparqlExpression	SparqlExpression	Ein Wert eines SPARQL-Ausdrucks kann einen weiteren SPARQL-Ausdruck enthalten. Weitere SPARQL-Ausdrücke können daher über die Eigenschaft SparqlExpression gespeichert werden.
String	Operator	Operator, wie beispielsweise „=“, mit dem der Wert eines semantischen Ausdruck mit Subjekt und Relation verbunden ist.
String	StringValueOfProperty	Der Wert eines semantischen Ausdrucks als Zeichenkette.
Int	IntValueOfProperty	Der Wert eines semantischen Ausdrucks als Zahlenwert.
DataHandlerVariable	SparqlSubjectOfProperty	Das Subjekt auf das sich der Wert eines semantischen Ausdrucks bezieht.
DataHandlerVariable	SparqlObjectOfProperty	Der Wert eines semantischen Ausdrucks als SparqlObject.
DataHandlerVariable	SparqlPropertyOfPropery	Der Wert eines semantischen Ausdrucks als SparqlProperty, beispielsweise für verkettete Relationen.

Tabelle 3: Eigenschaften der Klasse PropertyValue

QueryObject

Eine Instanz der Klasse QueryObject dient zur Speicherung einer SPARQL-Anfrage.

ClassNode

Ein Knoten zur Repräsentation eines Klassenknotens in der Datenstruktur eines Graphen. Die WPF-Visualisierung wird durch die VisualClassNode-Klasse realisiert.

DataSourceNode

Diese Knotenklasse dient zur Repräsentation einer Datenquelle innerhalb der Datenstruktur des Graphen. Die WPF-Visualisierung wird durch die VisualDataSourceNode-Klasse realisiert.

RelationNode

Diese Knotenklasse dient zur Repräsentation einer Relation innerhalb der Datenstruktur des Graphen. Die WPF-Visualisierung wird durch die VisualRelationNode-Klasse realisiert.

ResultNode

Diese Knotenklasse dient zur Repräsentation eines Ergebnisknotens innerhalb der Datenstruktur des Graphen. Die WPF-Visualisierung wird durch die VisualResultNode-Klasse realisiert.

6.2.1.3 Erstellung asynchroner Anfragen an SPARQL-Endpoints

Zum Abfragen einer semantischen Datenquelle verwendet die SparqlDataHandler-Komponente das dotNetRDF-Framework, einem OpenSource .Net-Framework unter MIT-Lizenz[5] mit dem sich SPARQL-Endpoints anfragen lassen.

Die Funktionen zur Abfrage von SPARQL-Endpoints finden sich im Projektverzeichnis „Utilities“ innerhalb der SparqlQueryBackend-Klasse. Da aufgrund der Datenmenge, die bei Anfragen an semantische Datenquellen, wie zum Beispiel der DBPedia, anfallen können, müssen SPARQL-Anfragen in mehrere Anfragen unterteilt werden, um einen Timeout am Endpoint zu vermeiden. Anfragen lassen sich in SPARQL über den Offset einer Anfrage in separate Anfragen unterteilen, wie im folgenden Beispiel.

```
SELECT * WHERE { ?s ?p ?o }

äquivalent zum Ergebnis folgender Anfragen:

SELECT * WHERE { ?s ?p ?o } LIMIT 500 OFFSET 0
SELECT * WHERE { ?s ?p ?o } LIMIT 500 OFFSET 500
SELECT * WHERE { ?s ?p ?o } LIMIT 500 OFFSET 1000
SELECT * WHERE { ?s ?p ?o } LIMIT 500 OFFSET 1500
…
```

Quellcode 34: Aufspaltung einer SPARQL-Anfrage in mehrere kleine SPARQL-Anfragen.

Zum Versenden von Unterteilten, wie nicht unterteilten Anfragen bietet die SparqlQueryBackend-Klasse, die Methoden QueryStepByStep und Query an. Bei einer Anfrage über

[5] MIT-Lizenz http://opensource.org/licenses/mit-license.php

Query wird die Anfrage nicht unterteilt, bei einer Anfrage über QueryStepByStep wird die Anfrage über das Hinzufügen eines Anfragelimits und eines Wertes für Offset in mehrere Anfragen unterteilt. Um alle Ergebnisse, die ein Endpoint auf eine Anfrage zur Verfügung stellt, über QueryStepByStep abzufragen, wird die Anfrage so oft unterteilt bis der Endpoint keine weiteren Daten mehr zurückgibt. Anfragen über QueryStepByStep können daher mitunter sehr lange andauern, bis alle gesuchten Daten vom Endpoint abgefragt wurden.

Zur Optimierung der Performance werden zudem nur einfach konstruierte SPARQL-Anfragen eingesetzt, bei denen auf SPARQL-Funktionen, wie zum Beispiel DISTINCT verzichtet wird. Damit die Ergebnismenge keine doppelten Einträge enthält, werden Dopplungen durch die Programmlogik der SparqlDataHandler-Komponente herausgefiltert. Dadurch lässt sich der angefragte Endpoint entlasten und die Performance einer Anfrage optimieren.

Die SparqlLibary-Klasse bietet eine Sammlung an verschiedenen SPARQL-Anfragen, wie beispielsweise zur Abfrage einer Klassenhierarchie. Wird die SparqlDataHandler-Klasse angefragt, so wird zunächst für die Aufgabe entsprechende SPARQL-Anfrage aus der Sammlung der SparqlLibary-Klasse geladen und anschließend ein QueryStepByStep durchgeführt.

Zur asynchronen Verarbeitung von Anfragen und deren Ergebnis werden BackgroundWorker eingesetzt. Für jede Anfrage wird ein eigener BackgroundWorker gestartet, der asynchron arbeitet.

6.2.1.4 Bereitstellung abgefragter Informationen

Die von SPARQL-Endpoints abgefragten Informationen werden durch die SparqlDataHandler-Klasse in Form von Listen aus SparqlObjekten bereitgestellt.

Über einen Cache werden einmal abgefragte Daten persistent gespeichert. Das Cache-System wird durch die CacheHandler-Klasse im „Utilitites“-Projektverzeichnis realisiert.

6.2.1.5 Visualisierung und Benutzeroberfläche

Im Projektverzeichnis „WPF“ sind alle Klassen gruppiert, die im Rahmen der SparqlDataHandler-Klasse zur Visualisierung des Anfragegraphen und Realisierung der Benutzeroberfläche dienen. Knoten werden durch die im Unterverzeichnis „Nodes“ befindlichen Klassen visualisiert. WPF-Fenster zum Hinzufügen und Bearbeiten von Knoten finden sich im Unterverzeichnis „ExpressionWindows“.

6.3 Implementierungsumfang

Der Implementierungsumfang des Prototyps umfasst die wesentlichen Aspekte des vorgestellten Visualisierungskonzept für semantische Anfragen. Mit der prototypischen Umsetzung lassen sich Wertvergleiche durchführen, die sich auf primitive Datentypen (zum Beispiel einer Zeichenkette), Individuen oder Klassen beziehen. Darüber hinaus können auch kom-

plexe Verweise durchgeführt werden, die sich auf Teilergebnisse einer Anfrage beziehen können. Der Prototyp unterstützt dabei Anfragen über SELECT und ASK. Nicht unterstützt werden derzeit Anfragen über CONSTRUCT und DESCRIBE, sowie Subqueries und weiterer SPARQL 1.1 Funktionen, wie beispielsweise arithmetischer Operatoren.

7. Evaluation

Das durch den Prototyp realisierte Visualisierungskonzept wurde im Rahmen von zwei Benutzerstudien evaluiert. In einer ersten Studie mit 6 Probanden wurde das grundlegende Konzept getestet. Daraus gewonnene Informationen wurden bei der Weiterentwicklung des Konzepts und der prototypischen Realisierung berücksichtigt. In einer zweiten Expertenstudie mit 10 Probanden wurde im Anschluss eine verbesserte prototypische Realisierung des Konzepts evaluiert. Der verbesserte Prototyp erleichterte im Wesentlichen die Erstellung von Objektreferenzen. Im Rahmen der Studien wurden sowohl die Lesbarkeit als auch die Erstellbarkeit semantischer Anfragen mit der prototypischen Realisierung evaluiert. Durchführung und Ergebnisse der Benutzerstudien werden in diesem Kapitel beschrieben.

7.1 Durchführung

In diesem Abschnitt wird die Durchführung der Studien beschrieben. Beide Studien wurden dabei auf dieselbe Art und Weise in Form einer Expertenstudie durchgeführt. Das bedeutet, dass das prototypisch umgesetzte Visualisierungskonzept immer nur von einer Person gleichzeitig unter Anleitung des Studienleiters in einem neutralen Raum durchgeführt wurde.

Zu Beginn der Studie wurden den Probanden Fragebögen, mit Fragen bezüglich Alter, Geschlecht, Ausbildungsgrad sowie Angaben zu den Englischkenntnissen und inwieweit Probanden mit der Thematik des Semantischen Webs bereits vertraut sind, ausgeteilt. Der Fragenbogen ist im Anhang dieser Arbeit angefügt. Alle Probanden haben zudem einen Test zur Erkennung einer vorhandenen Rot-Grün-Sehschwäche durchgeführt. Anschließend wurden den Probanden in einer kurzen Präsentation die Grundprinzipien des Semantischen Webs und der Filter-/Flow-Visualisierung vorgestellt, sowie auf die grundlegende Bedienung der Benutzeroberfläche eingegangen. Die Erläuterung der Benutzeroberfläche beschränkte sich dabei ausschließlich auf die Funktionen von Links und den Aufruf des Kontextmenus. Weitere Aspekte der Benutzeroberfläche, wie beispielsweise das eingefärbte Relationsknoten sich auf Klassenknoten gleicher Farbe beziehen oder Objekte mit gleichem Namen durch unterschiedlicher Einfärbung und Nummerierung zu unterscheiden sind, mussten die Probanden selbstständig erkennen. Die Präsentation ist im Anhang dieser Arbeit angefügt. Die Probanden wurden aufgefordert, während der Studiendurchführung Gedanken und Anregungen zur Benutzeroberfläche laut auszusprechen. Gedanken und Anregungen wurden mit anderen

Notizen vom Studienleiter aufgeschrieben. Während der Studiendurchführung war es Probanden erlaubt Fragen, deren Antwort keinen Rückschluss auf die Lösung einer Aufgabe ermöglichte, an den Studienleiter zu stellen. Beispielsweise konnten Probenden sich den Unterschied zwischen verschiedenen Relationstypen (ObjectProperty, DataProperty und Property) erläutern lassen. Am Ende der Studie wurden die Probanden interviewt und haben einen abschließenden Bewertungsbogen ausgefüllt, der zusammen mit dem Fragebogen im Anhang dieser Arbeit angefügt ist.

7.2 Aufgaben

Die Probanden haben im Rahmen beider Benutzerstudien vier Aufgaben, die aus jeweils einer Lese- und einer Bearbeitungsaufgabe bestanden, absolviert. Bei den Leseaufgaben wurde ermittelt, ob Nutzer Anfragen, die mit der prototypischen Umsetzung des Visualisierungskonzepts erstellt wurden, erkennen können. Bei den Bearbeitungsaufgaben wurde ermittelt, ob Nutzer Anfragen mit der prototypischen Umsetzung des Visualisierungskonzepts erstellen können. Die Aufgabenstellung wurde den Probanden ausgedruckt Form vorgelegt. Paare aus Lese- und Bearbeitungsaufgaben waren dabei ähnlich konstruiert, sodass Nutzer eine bestandene Leseaufgabe als Beispiel für das Bearbeiten einer Anfrage verwenden konnten. Der Schwierigkeitsgrad der Aufgaben war so konzipiert, dass mit jedem Aufgabenpaar die Aufgabenstellung leicht erhöht wurde. In der letzten Aufgabe mit dem höchsten Schwierigkeitsgrad wurden Anfragen mit Verweisen auf Teillösungen der Anfrage bearbeitet. Die Probanden hatten für die Bearbeitung beliebig lange Zeit. Keiner der Probanden benötigte zur Bearbeitung der Aufgabenstellungen und zur Abgabe seiner Bewertung mehr als 30 Minuten. Die Aufgabenstellungen sind im Anhang dieser Arbeit angefügt. Folgende Aufgaben wurden den Probanden gestellt:

In Aufgabe 1 a) sollte eine Anfrage erkannt werden, in der nach Filmen gesucht wurde, die ein Budget von über 15 000 000 $ hatten.
In Aufgabe 1 b) sollte eine Anfrage erstellt werden, in der nach Städten mit mehr als 1 000 Einwohnern gesucht werden soll.

In Aufgabe 2 a) sollte eine Anfrage erkannt werden, in der nach Filmen gesucht werden sollte, die von Quentin Tarantino gedreht wurden und in denen Bruce Willis als Hauptdarsteller mitspielte.
In Aufgabe 2 b) sollte eine Anfrage erstellt werden, in der nach Städten aus Brasilien mit mehr als 1 000 000 Einwohnern gesucht werden sollte.

In Aufgabe 3 a) sollte eine Anfrage erkannt werden, in der nach Städten aus Brasilien oder Peru mit mehr als 500 000 Einwohnern gesucht wurde.
In Aufgabe 3 b) sollte eine Anfrage erstellt werden, in der nach Programmiersprachen ge-

sucht wurde, die entweder durch die Programmiersprache Java oder Basic beeinflusst wurden.

In Aufgabe 4 a) sollte eine Anfrage erkannt werden, in der nach Filmen gesucht wurde, in denen der Regisseur auch im Film selbst mitgespielt hat.
In Aufgabe 4 b) sollte eine Anfrage erstellt werden, in der eine Programmiersprache gesucht werden sollte, die von einer anderen Programmiersprache beeinflusst wurde, die von Microsoft entwickelt wurde.

7.3 Studienteilnehmer

In diesem Abschnitt finden sich Informationen zu den Probanden, die an einer der beiden Studien teilgenommen haben. Zwischen den Probanden der Studien gibt es keinerlei Überschneidungen.

7.3.1 Studienteilnehmer der ersten Studie

Im Folgenden finden sich Informationen zu den Probanden der ersten Studie.

An der Studie haben insgesamt sechs Probanden im Alter zwischen 24 und 30 Jahren teilgenommen. Im Schnitt waren die Probanden 27 Jahre alt und hatten zu 83,33% bereits ein abgeschlossenes Hochschulstudium oder studierten derzeit. Fünf der sechs Probanden waren männlich und eine weiblich. Ihre Englischkenntnisse schätzten die Probanden nach eigenen Angaben in einer Skala von eins bis zehn im Schnitt auf 6,83. 33% der Probanden kannten den Begriff „Semantic Web“ bereits. Auf einer Skala von 1 bis 10 schätzten die Probanden ihre Kenntnisse in diesem Themenbereich im Schnitt auf einen Wert von 2,33 ein, wobei 1 für gar keine und 10 für hervorragende Kenntnisse steht. Bei einem Probanden wurde eine Rot-Grün-Sehschwäche festgestellt.

7.3.2 Studienteilnehmer der zweiten Studie

Im Folgenden finden sich Informationen zu den Probanden der zweiten Studie.

An der Studie haben insgesamt zehn Probanden im Alter zwischen 24 und 44 Jahren teilgenommen. Im Schnitt waren die Probanden 27,9 Jahre alt und hatten zu 80% bereits ein abgeschlossenes Hochschulstudium oder studierten derzeit. Fünf Probanden waren weiblich, fünf männlich.. Ihre Englischkenntnisse schätzten die Probanden nach eigenen Angaben in einer Skala von eins bis zehn im Schnitt auf 8,3. Den Begriff „Semantic Web“ kannten bereits 50% der Probanden. Auf einer Skala von 1 bis 10 schätzten die Probanden ihre Kenntnisse in diesem Themenbereich im Schnitt auf einem Wert von 3,4 ein. Bei keinem der Probanden wurde eine Rot-Grün-Sehschwäche festgestellt.

7.4 Resultate

In diesem Abschnitt sind die Ergebnisse der Studien sowie die abschließenden Bewertungen der Probanden aufgeschlüsselt.

7.4.1 Resultate der ersten Studie

Die Aufgabenpaare 1 bis 3 haben alle Probanden der ersten Studie korrekt gelöst. Aufgabe 4a haben 5 von 6 Personen korrekt gelöst, Aufgabe 4b haben 3 von 6 Probanden korrekt gelöst.

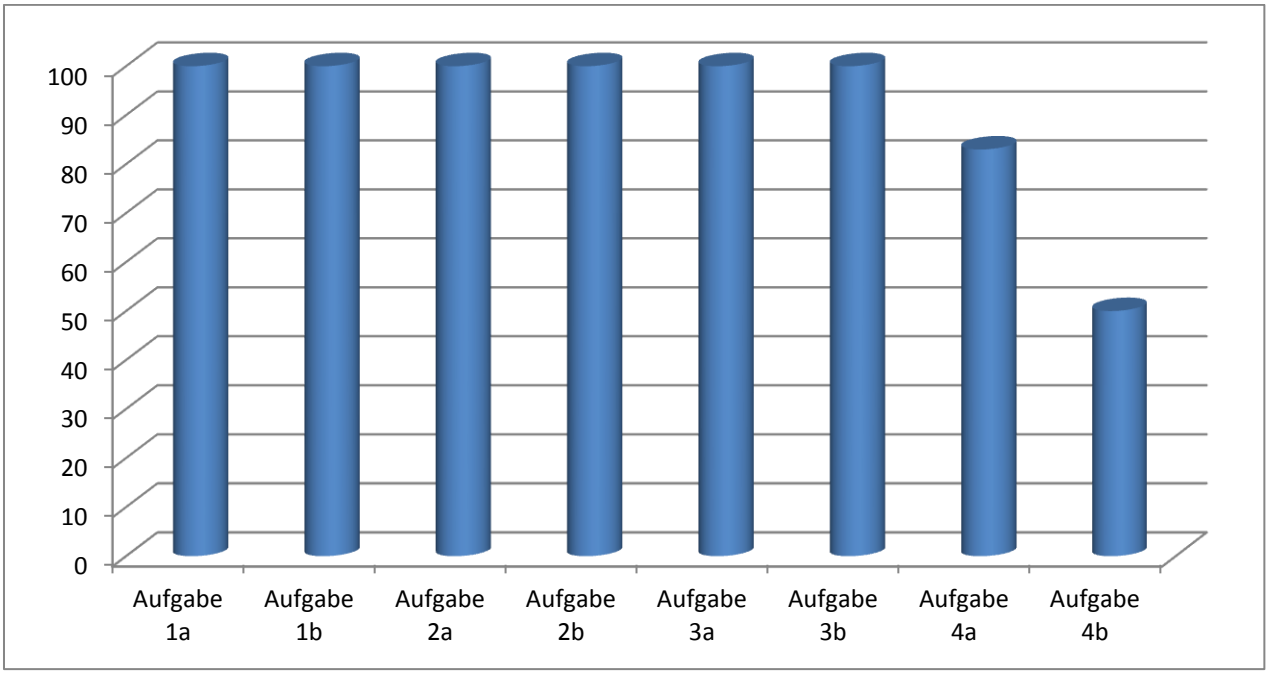


Abbildung 59: Aufschlüsselung der korrekt beantworteten Aufgaben der ersten Studie in Prozent.

Die Lesbarkeit semantischer Anfragen in Form von Filter-/Flow-Graphen wurde von den Probanden in der ersten Studie im Schnitt mit 9 von 10 Punkten bewertet.

Die Erstellbarkeit semantischer Anfragen in Form von Filter-/Flow-Graphen bewerteten die Probanden in der ersten Studie im Schnitt mit 7,5 von 10 Punkten.

Das grundlegende Konzept der Datenfilterung durch Filter-/Flow-Graphen wurde von den Probanden im Schnitt mit 8,33 von insgesamt 10 Punkten bewertet.

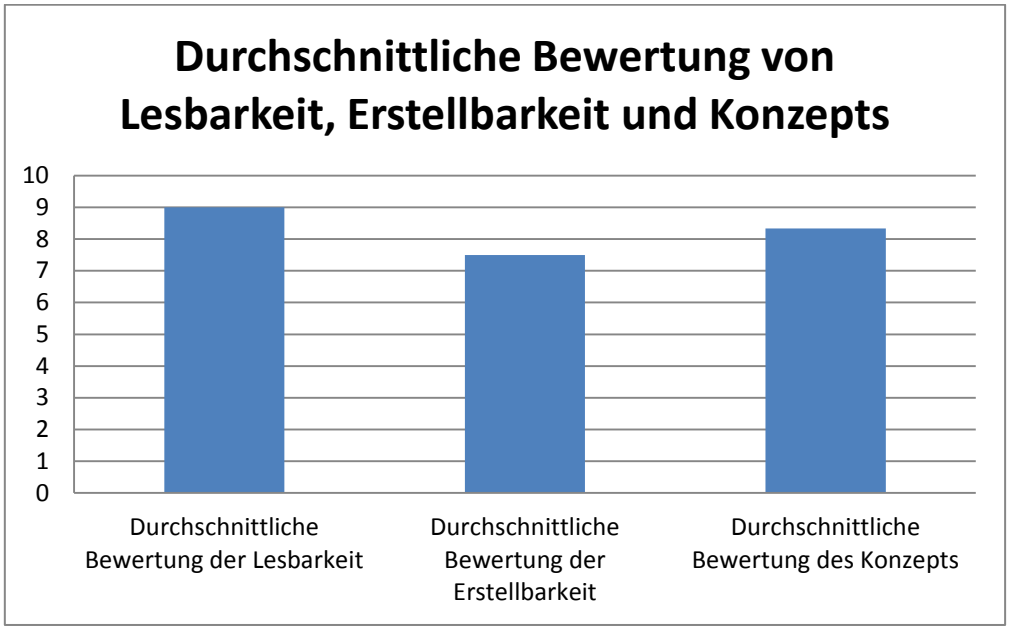


Abbildung 60: Durchschnittliche Bewertung der Lesbarkeit und Erstellbarkeit semantischer Anfragen auf Basis der prototypischen Realisierung, sowie die durchschnittliche Bewertung des grundlegenden Konzepts der Datenfilterung semantischer Anfragen auf Basis von Filter-/Flow-Graphen im Rahmen der ersten Benutzerstudie.

7.4.2 Resultate der zweiten Studie

In der zweiten Studie wurden alle Aufgaben, bis auf Aufgabe 4b, von allen Probanden korrekt beantwortet. Aufgabe 4b konnten 8 von 10 Probanden korrekt lösen. Zum Einsatz kam dabei ein gegenüber der ersten Studie verbesserter Prototyp, in dem eine verbesserte Suche implementiert wurde. Dadurch konnten Probanden unter anderem auch nach Referenzen, die sich auf Teilergebnisse einer Anfrage beziehen, suchen. Dadurch ließen sich Referenzen leichter gestalten.

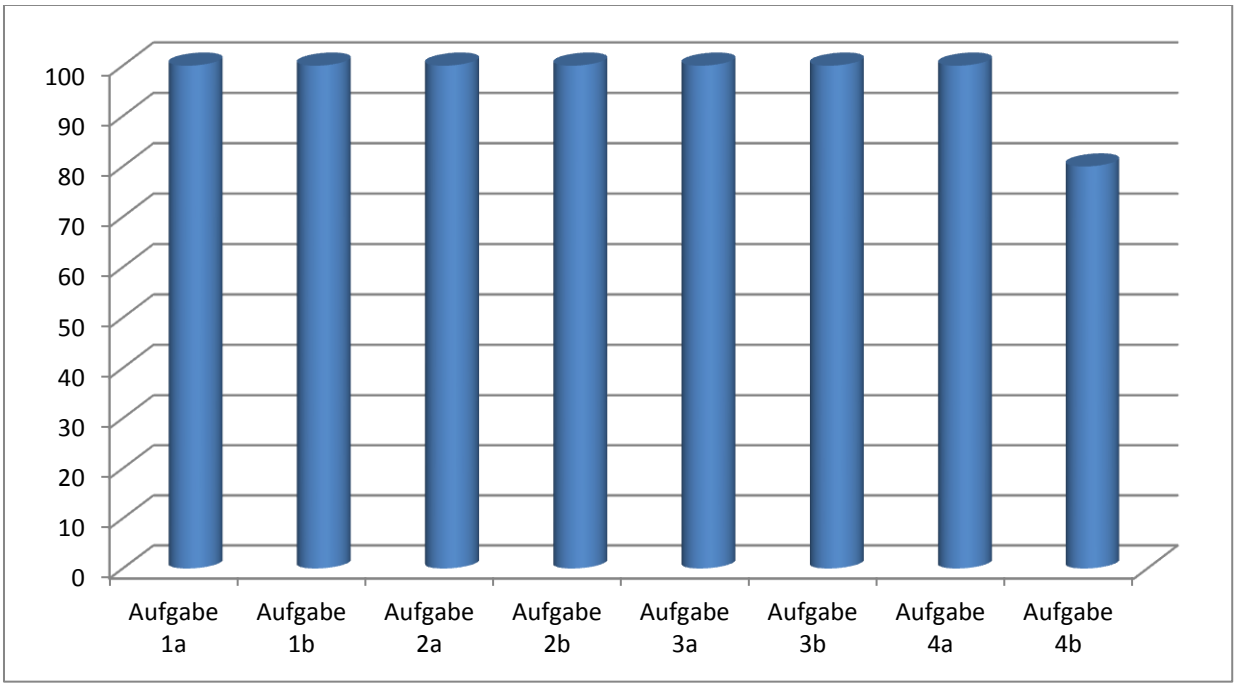


Abbildung 61: Aufschlüsselung der korrekt beantworteten Aufgaben der zweiten Studie in Prozent. Die Aufgaben von Aufgabenstellung 1a bis 4a konnten von allen Probanden korrekt gelöst werden. Aufgabe 4b konnten 8 von 10 Probanden korrekt lösen.

Die Lesbarkeit semantischer Anfragen in Form von Filter-/Flow-Graphen bewerteten die Probanden im Schnitt mit 9 von 10 Punkten.

Die Erstellbarkeit semantischer Anfragen in Form von Filter-/Flow-Graphen bewerteten die Probanden im Schnitt mit 8,4 von 10 Punkten.

Das grundlegende Konzept der Datenfilterung durch Filter-/Flow-Graphen wurde von den Probanden im Schnitt mit 9 von insgesamt 10 Punkten bewertet.

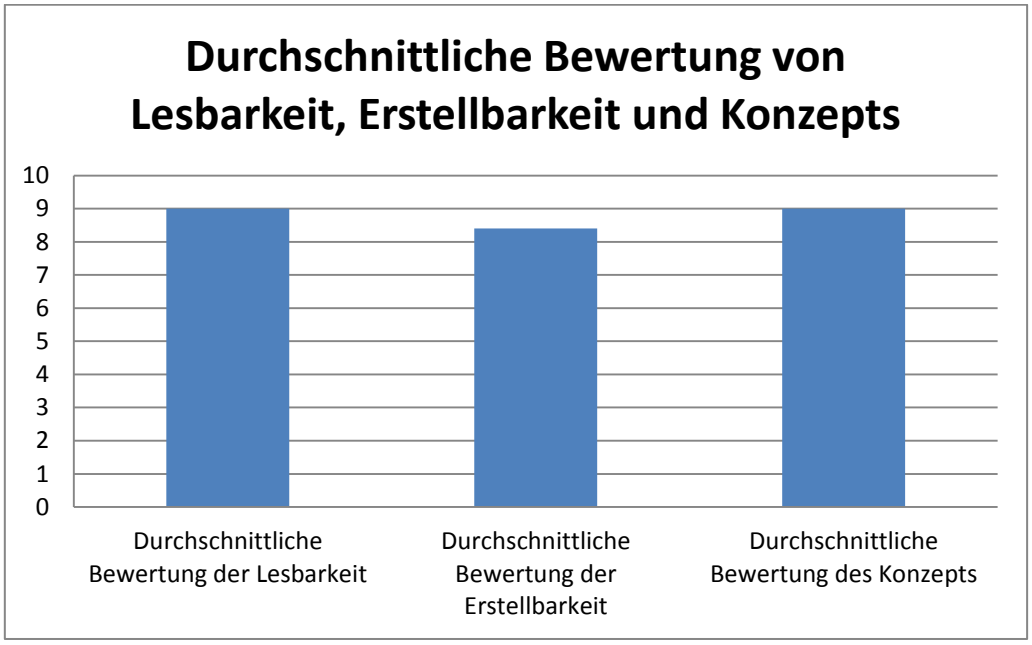


Abbildung 62: Durchschnittliche Bewertung der Lesbarkeit und Erstellbarkeit semantischer Anfragen auf Basis der prototypischen Realisierung, sowie die durchschnittliche Bewertung des grundlegenden Konzepts der Datenfilterung semantischer Anfragen auf Basis von Filter-/Flow-Graphen im Rahmen der zweiten Benutzerstudie.

7.5 Fazit & Diskussion

Die Ergebnisse der Benutzerstudie zeigen, dass sich mit der prototypischen Umsetzung des in Kapitel 5 vorgestellten Lösungskonzepts semantische Anfragen auch von Benutzern ohne informationstechnischem Hintergrundwissen lesen, sowie erstellen lassen. Insbesondere für einfache Anfragen mit direkten Wert- oder Objektvergleichen, wie in den Aufgabenpaaren 1 bis 3 gefragt, eignet sich die Umsetzung. Anfragen dieser Art konnten alle Probanden in beiden durchgeführten Studien sowohl korrekt lesen als auch erstellen. Dabei hatten die Probanden überwiegend nur wenig bis gar keine Erfahrung im Bereich des Semantischen Webs und erhielten vor der Bearbeitung der Aufgaben nur eine sehr oberflächliche Einführung in das Themengebiet und die Bedienung des Prototyps in Form einer im Anhang dieser Arbeit beigefügten Präsentation.

Auch komplexere Anfragen mit mehrstufigen Verweisen, in denen auf Teilergebnissen einer Anfrage verwiesen wurde, wie beispielsweise in Aufgabe 4b konnten von der überwiegenden Mehrheit der Nutzer gelesen und erstellt werden. Insgesamt konnten über beide Studien betrachtet Aufgabe 4a 15 von 16 Probanden und Aufgabe 4b 11 von 16 Personen korrekt beantworten. Vergleicht man beide Studien, so hat sich gezeigt, dass durch eine verbesserte Suchfunktion die Erstellbarkeit komplexer Anfragen, wie durch Aufgabe 4b gestellt, die korrekten Antworten von 50% auf 80% gesteigert werden konnte. Dabei bestand der Vorteil der verbesserten Suche, das in die Resultate der Suche sowohl Individuen, Klassen, sowie Referenzen auf bereits eingefügte Klassen beziehungsweise Teilergebnisse einer Anfrage mit eingeflossen sind. So konnten Probanden aus der Suche heraus neue Klassen anlegen und referenzieren. Dennoch zeigen die Ergebnisse, dass in der Gestaltung der Benutzeroberfläche, insbesondere zur Erstellung mehrstufiger Verweise, noch Potenzial für Verbesserungen besteht. Die Lösungen und deren Lösungswege, sowie die Probleme, die die meisten Probanden dabei hatten, werden im Folgenden genauer betrachtet. Für Aufgabe 4b gibt es dabei zwei mögliche Lösungen. Die erste der beiden möglichen Lösungen ist in Abbildung 63, die Zweite in Abbildung 64 dargestellt.

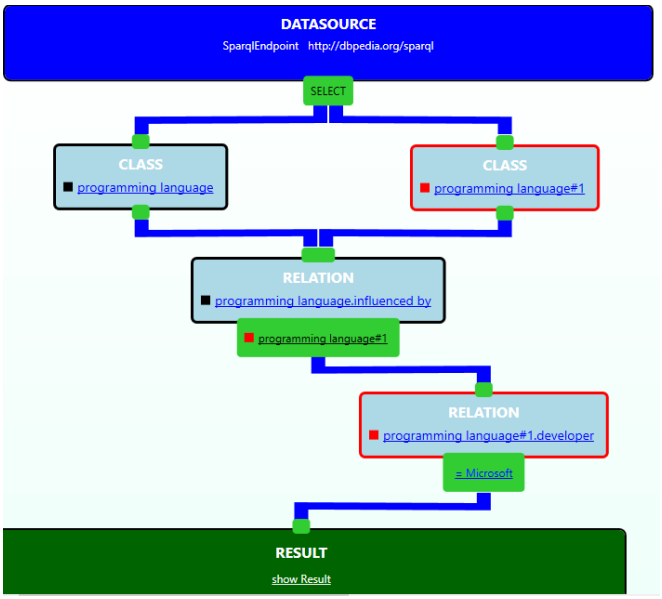


Abbildung 63: Darstellung der ersten von zwei möglichen Lösungen von Aufgabe 4b.

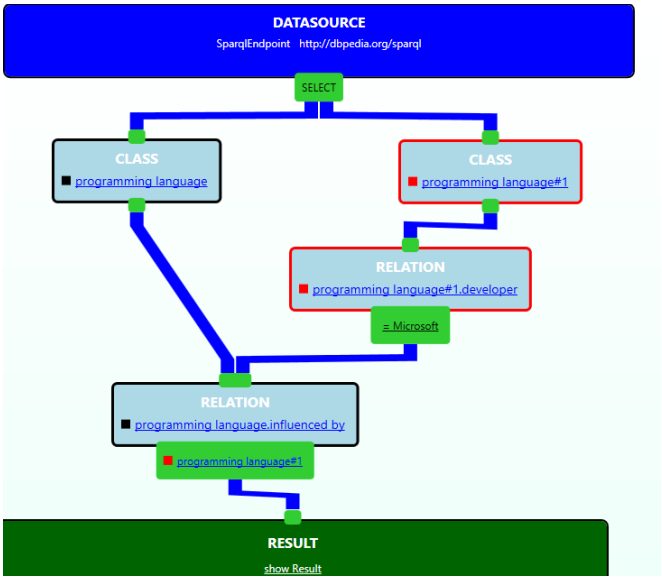


Abbildung 64: Darstellung der zweiten Lösung von Aufgabe 4b.

Beide Lösungen von Aufgabe 4b unterscheiden sich dabei nur in der Anordnung der Relation „developer“. Diese kann direkt unterhalb der Klasse, auf die sie sich bezieht, (Abbildung 63) oder als letzte Relation im Filter-/Flow-Graphen (Abbildung 64) positioniert werden. Das Ergebnis beider Anfragen ist dabei äquivalent.

Die Probanden hatten beim Lösen von Aufgabe 4b im Wesentlichen Probleme, die Referenz zwischen der „programming_language“ und der „programming_language#1“, wie in Abbildung 65 dargestellt, herzustellen.

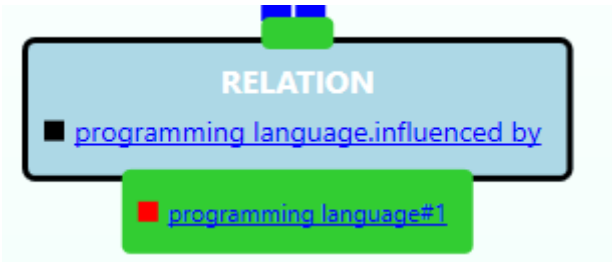


Abbildung 65: Verweis zwischen zwei Programmiersprachen über die Relation „influenced_by“.

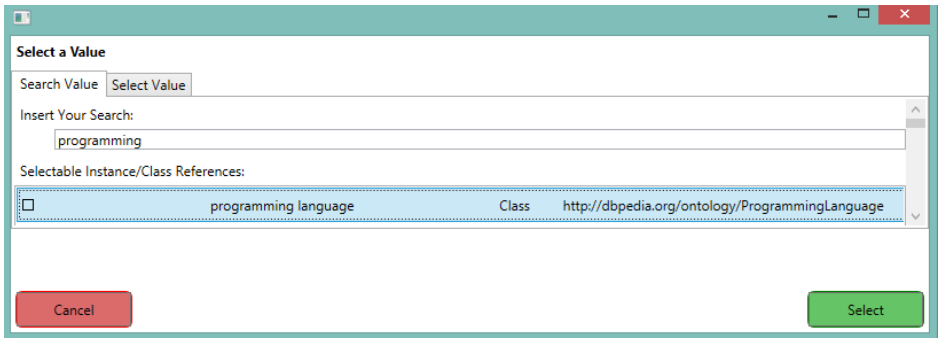


Abbildung 66: Auswahl einer neuen Klasse vom Typ „programming language“ über die Suchfunktion.

Statt über die Suchfunktion, wie in Abbildung 66 dargestellt, eine Klasse auszuwählen oder auf eine bereits hinzugefügte Klasse zu verweisen, haben einige Probanden versucht Emitter unterschiedlicher Teilflüsse miteinander zu verbinden. Um einen Verweis, wie in Abbildung 64 dargestellt, zu erstellen, haben sie dabei den Emitter der Relationsknoten „influenced_by“ mit dem Emitter des Relationsknotens „developer“, wie in Abbildung 67 dargestellt, zu verbinden versucht. Da Emitter im vorgesehenen Konzept eigentlich nur Datenausgänge visualisieren sollen, wurde dieses Vorgehen von der prototypischen Umsetzung nicht unterstützt.

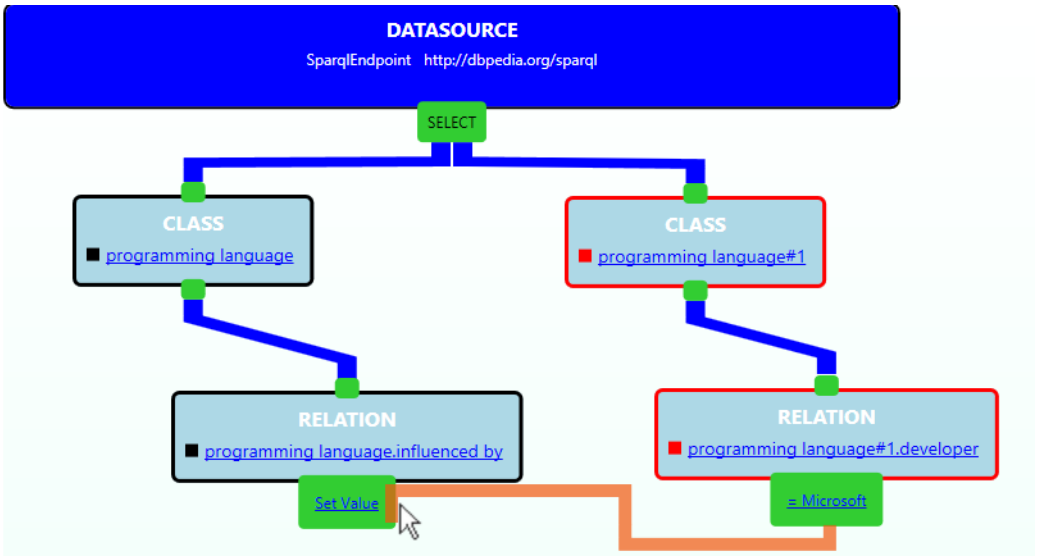


Abbildung 67: Zur Lösung von Aufgabe 4b haben Probanden oft versucht Verweise zwischen Teilergebnissen einer Anfrage durch das verbinden von Emittern durchzuführen.

Einige Probanden, die den Verweis zwischen den beiden Datenflüssen über die Suchfunktion herstellten, hatten im Anschluss Probleme die Relation „developer“ in die Anfrage einzufügen. Statt eine Lösung, in der die Relation „developer“ am Ende des Flusses eingefügt wird (Abbildung 63), bevorzugten die Probanden eine Darstellung wie in Abbildung 64. Im evaluierten Prototyp konnten Flüsse allerdings noch nicht über ein Kontextmenu gelöscht werden, was manche Probanden irritierte.

Aus diesen Beobachtungen ergeben sich mindestens zwei Verbesserungsmöglichkeiten für die Erstellung von Verweisen zwischen Teilflüssen einer Anfrage. Zum einen sollten Flüsse über ein Kontextmenu das Einfügen von Relationsknoten erleichtern. Zum anderen sollte sich durch das Verbinden von zwei Emittern einen Verweis zwischen beiden Teilflüssen herstellen lassen.

8. Ausblick und Zusammenfassung

Das vorgestellte Visualisierungskonzept bildet eine weitere visuelle Anfragesprache für semantische Datenquellen auf Basis der Anfragesprache SPARQL. Das Visualisierungskonzept basiert dabei auf den Arbeiten [50] [9] [51] zur Filter-/Flow-Visualisierung. Techniken zur Darstellung von Datenflüssen werden dabei mit der Mächtigkeit von SAPRQL verknüpft. In SPARQL 1.1 hinzugekommene Features, wie beispielsweise Relationspfade oder arithmetische Operationen werden dabei im Visualisierungskonzept berücksichtigt. Ein besonderes Augenmerk wird auf die Möglichkeit zum Ausdruck mehrstufiger Verweise auf Teilergebnisse einer Anfrage geworfen. Mit dem Fokus auf wenige Knotentypen und einer einfach gestalteten Benutzeroberfläche soll sich das Visualisierungskonzept auch für Nutzer mit geringen, bis gar keinen Kenntnissen über Themen wie dem Semantischen Web oder SPARQL eignen. Im Rahmen zweier Benutzerstudien konnte gezeigt werden, dass sich die prototypische Realisierung des Visualisierungskonzepts für einfache Anfragen eignet und auch Nutzer ohne Kenntnisse im Themenbereich des Semantischen Webs Anfragen mit mehrstufigen Verweisen erstellen können. Dennoch hat sich gezeigt, dass noch einige Verbesserungsmöglichkeiten am Konzept und der prototypischen Realisierung bestehen.

8.1 Ausblick

In diesem Abschnitt werden Vorschläge zur Verbesserung und Weiterentwicklung des vorgestellten Visualisierungskonzepts und dessen prototypischer Realisierung vorgestellt.

8.1.1 Ausbau des Prototyps

Für weitere Studien zur Untersuchung und Verbesserung einzelner Aspekte des Visualisierungskonzepts bedarf es eines weiteren Ausbau der prototypischen Umsetzung, um beispielsweise zu untersuchen wie sich Anfragen, die sich an mehrere Datenquellen gleichzeitig

richten, am sinnvollsten visualisieren und implementieren lassen. Verbesserungen an der Benutzeroberfläche, wie beispielsweise die in Kapitel 7.5 beschriebenen Erleichterungen zum Erstellen von Verweisen auf Teilergebnisse einer Anfrage können Thema weiterer Arbeiten an der prototypischen Realisierung darstellen.

8.1.2 Zusammenfassen verschiedener Relationstypen

Für Nutzer, die nur wenig bis keine Erfahrung mit SPARQL oder dem Semantischen Web haben, kann es schwierig sein, zwischen den verschiedenen Relationstypen zu unterscheiden. Aber auch für erfahrene Nutzer kann die Unterscheidung zwischen Relationstypen beim Erstellen von Anfragen störend sein. Eine Relation, wie beispielsweise „gender“, kann für unterschiedliche Ressourcen einmal mit einer Zeichenkette oder mit einem Objekt verbunden sein. Bei den Charakteren der Zeichentrickserie „The Simpsons“ ist dies im Datensatz der DBPedia beispielsweise der Fall. Manche der Charakteren sind mit einer ObjectProperty „gender“ mit einem Objekt und manche über eine DataProperty „gender“ mit einer Zeichenkette verbunden. Möchte man nun eine Liste aller weiblicher beziehungsweise aller männlicher Simpsons-Charakteren erstellen, so muss man eine Anfrage über beide Relationstypen (ObjectProperty und DataProperty) erstellen. Aus Nutzersicht wäre es einfacher, wenn das Anfragesystem dies erkennt und dies entsprechend bei der Erstellung einer Anfrage berücksichtigt oder den Nutzer alternativ darauf hinweisen würde, dass es mehrere gleichnamige Relationen unterschiedlicher Relationstypen existieren.

8.1.3 Suche über inverse Relationspfade

Da eine Ontologie einen gerichteten Graphen abbildet, kann es vorkommen, dass Datensätze Relationen von einer Ressource zu einer anderen Ressource enthalten, aber keinerlei Relationen in die andere Richtung. Beispielsweise könnte eine Datenquelle Städte und Länder über die Relation „befindet sich in“ miteinander verbinden. So kann man beispielsweise das Land, in dem sich eine Stadt, wie Paris befindet, aus der Datenquelle abfragen. Existieren aber keine Relationen zwischen Ländern und Städten, kann man allerdings nicht ohne Weiteres abfragen, welche Städte in Frankreich liegen. Mit inversen Relationspfaden lässt sich dies durch die Spracherweiterung SPARQL 1.1 bewerkstelligen. Wenn in der Suche nach auswählbaren Relationen inverse Relationspfade standardmäßig berücksichtigt werden, ließen sich Probleme, die aufgrund eines unvollständigen Datensatzes entstehen, vermindern.

8.1.4 Implementierung eines Modulsystems für weitere Arbeiten

In Anbetracht der Größe des Projekts und des Zusammenspiels zahlreicher Unterprojekte würde sich die Implementierung eines Modulsystems oder Pluginsystems anbieten, um die Pflege und Erweiterbarkeit des Funktionsumfangs leichter zu gestalten.

8.1.5 Realisierung des Prototyps als webbasierter Dienst

Bisher ist der Prototyp in Form einer nativen Applikation in C# und WPF implementiert. Eine webbasierte Lösung kann insbesondere bei weiteren Evaluationen Vorteile bieten. Studien lassen sich so verteilt durchführen und Benutzer haben so einen leichteren Zugang zum Tool, da sie es sich nicht extra herunterladen müssen. Nutzer können so auch Fragestellungen am heimischen Rechner bearbeiten. Mit einer webbasierten Lösung könnte dadurch eventuell bei Evaluationen höhere Teilnehmerzahlen erreicht werden. Eine webbasierte Lösung bietet zudem die Möglichkeit, die Anwendung mit wenig Aufwand und mit der selben Codebasis auch auf mobile Endgeräte verschiedenster Plattformen zu bringen.

9. Anhang

9.1 Fragebogen

Im Folgenden ist der im Rahmen der beiden Benutzerstudien an die Probanden ausgeteilte Fragebogen angefügt.

Fragebogen zur Benutzerstudie „Visuelle Filterung von Daten des Semantic Web“

Vielen Dank, dass Sie an der Studie zur visuellen Filterung von Daten des Semantic Web teilnehmen. Der folgende Fragebogen wird nur für wissenschaftliche Zwecke im Rahmen dieser Studie verwendet. Alle Angaben, die Sie machen, werden streng vertraulich behandelt. Ihre Daten bleiben anonym, ein Rückschluss auf Ihre Person ist somit nicht möglich. Bei diesem Fragebogen gibt es keine richtigen und falschen Antworten.

Angaben zur Person

1. **Bitte geben Sie Ihr Alter an:** _____ Jahre

2. **Bitte geben Sie Ihr Geschlecht an:**

   - ☐ männlich
   - ☐ weiblich

3. **Bitte geben Sie Ihren Schulabschluss an:**

   - ☐ bisher noch kein Schulabschluss
   - ☐ Schullaufbahn ohne Abschluss beendet
   - ☐ Haupt-/Volksschulabschluss
   - ☐ Realschulabschluss / Mittlere Reife
   - ☐ Fachhochschulreife / Abitur
   - ☐ Hochschulabschluss (Bachelor, Master, Diplom,...)

4. **Falls Sie studieren oder studiert haben, geben Sie bitte Ihr Studienfach an:**

_____

_____

_____

5. **Wie hoch würden Sie Ihre Englischkenntnisse auf einer Skala von 1-10 einschätzen?**

Gering	1	2	3	4	5	6	7	8	9	10	*sehr gut*

6. **Kennen Sie den Begriff „Semantic Web“ bereits?**

- ☐ Ja
- ☐ Nein

7. **Wie vertraut sind sie mit dem Begriff „Semantic Web“?**

gar nicht vertraut	1	2	3	4	5	6	7	8	9	10	*sehr vertraut*

8. **Ist bei Ihnen eine Fehlsichtigkeit festgestellt worden (zum Beispiel Rot-Grün-Blindheit)?**

- ☐ Ja
- ☐ Nein
- ☐ Keine Angaben

Studiendurchführung:

9. Bemerkungen während der Studiendurchführung:

Abschließende Bemerkungen:

10. Hatten Sie Probleme die Aufgabenstellung zu verstehen?

☐ Ja
☐ Nein

Wenn ja, was hat Ihnen besondere Probleme bereitet?

_____

_____

_____

11. Hatten Sie Probleme beim Bearbeiten der Aufgabenstellung?

☐ Ja
☐ Nein

Wenn ja, was hat Ihnen besondere Probleme bereitet?

_____

_____

_____

12. Wie empfanden Sie die Lesbarkeit der dargestellten Filter-/Flow-Graphen?

unlesbar	1	2	3	4	5	6	7	8	9	10	*sehr gut*

13. Wie empfanden Sie die Erstellbarkeit von Filter-/Flow-Graphen?

unmöglich	1	2	3	4	5	6	7	8	9	10	*sehr gut*

14. Wie empfanden Sie das grundlegende Konzept der Filterung von semantischen Daten?

sehr schlecht	1	2	3	4	5	6	7	8	9	10	*sehr gut*

15. Haben Sie noch irgendwelche Anmerkungen zur Studie?

_____

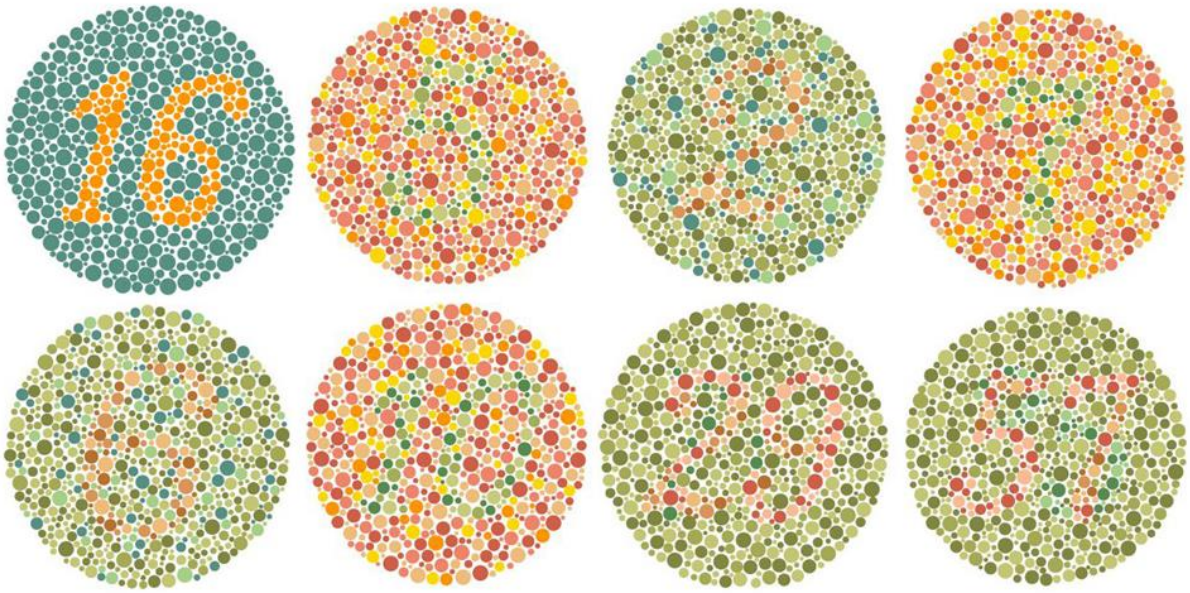
_____

_____

VIELEN DANK FÜR IHRE MITARBEIT

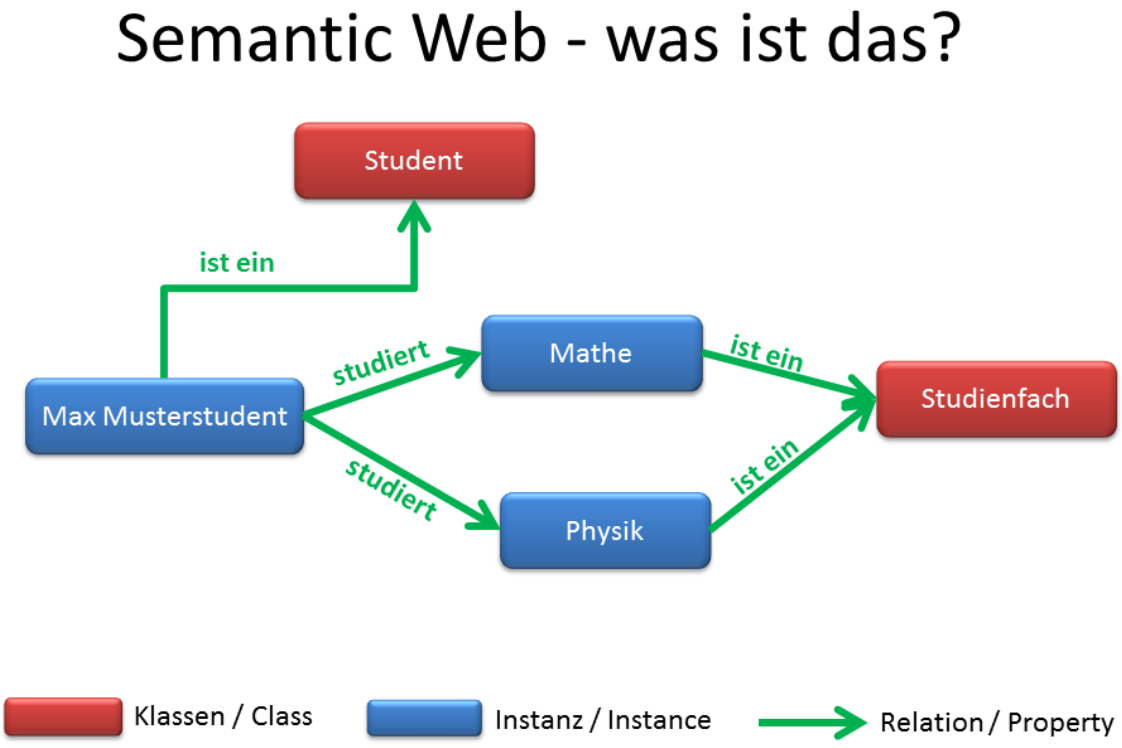
9.2 Einleitende Studienpräsentation

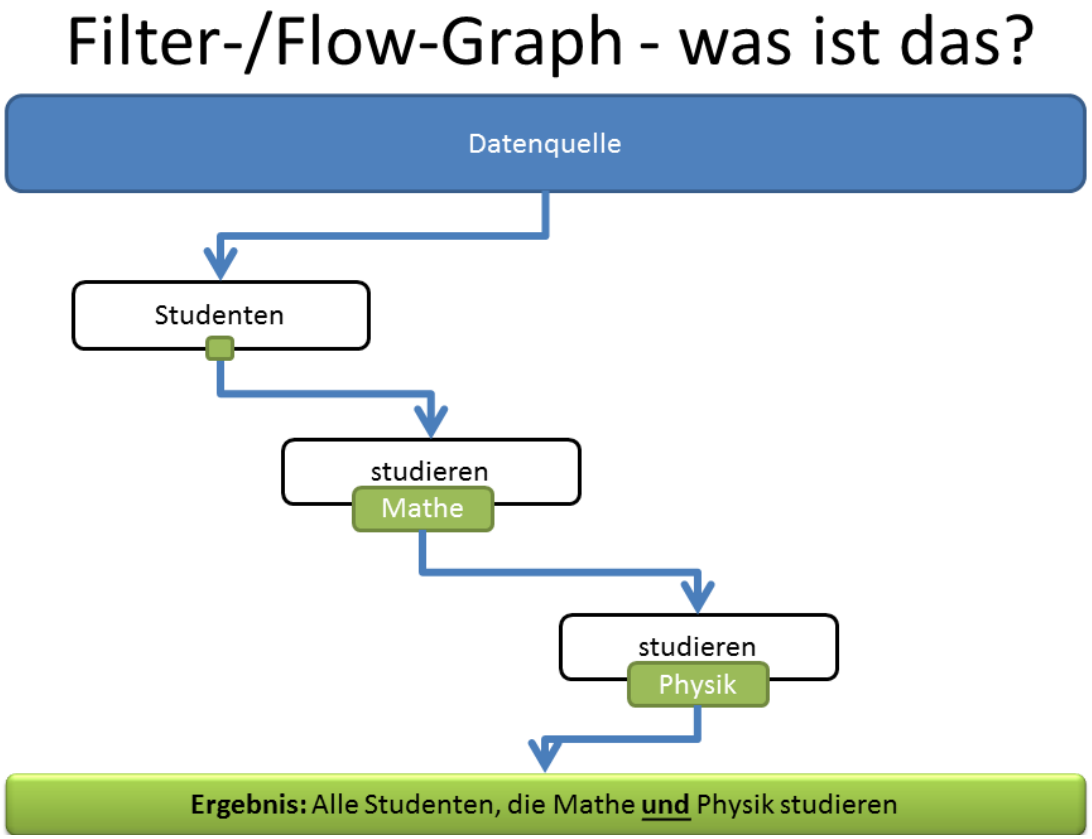
Im Folgenden ist die im Rahmen der beiden Benutzerstudien verwendete Präsentation zur Einführung der Probanden in das Thema „Semantisches Web“ angefügt.



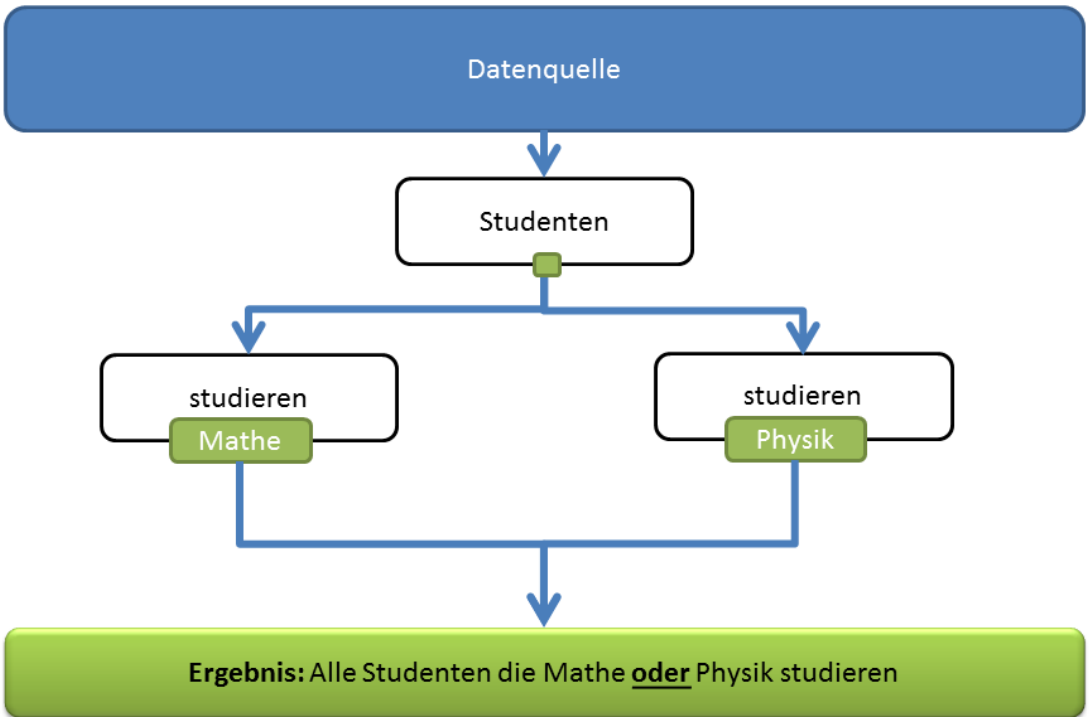
Visuelle Filterung von Daten des Semantic Web

Benutzerstudie

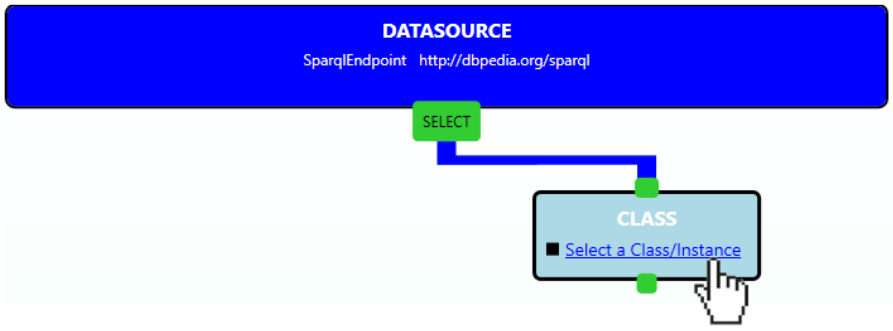

Semantic Web - was ist das?
Student
ist ein
Max Musterstudent
studiert
Mathe
ist ein
Studienfach
studiert
Physik
ist ein
Klassen / Class
Instanz / Instance
Relation / Property


Filter-/Flow-Graph - was ist das?
Datenquelle
Studenten
studieren
Mathe
studieren
Physik
Ergebnis: Alle Studenten, die Mathe und Physik studieren

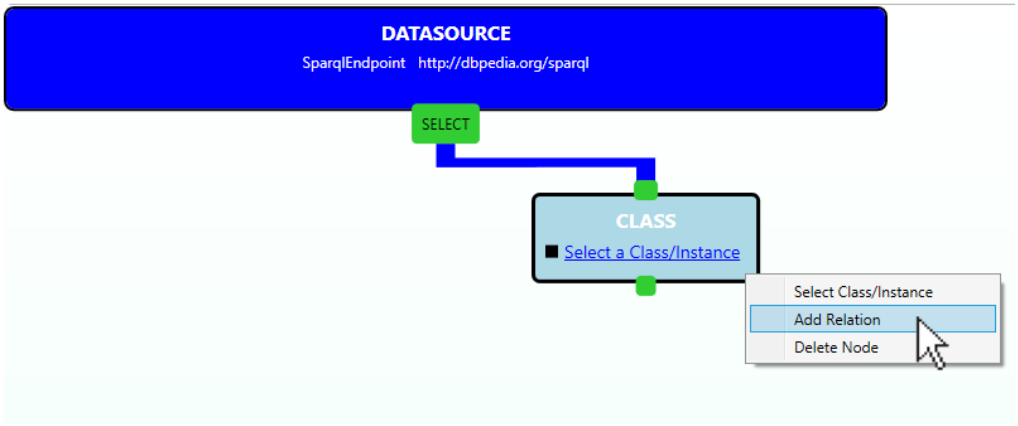
Filter-/Flow-Graph - was ist das?



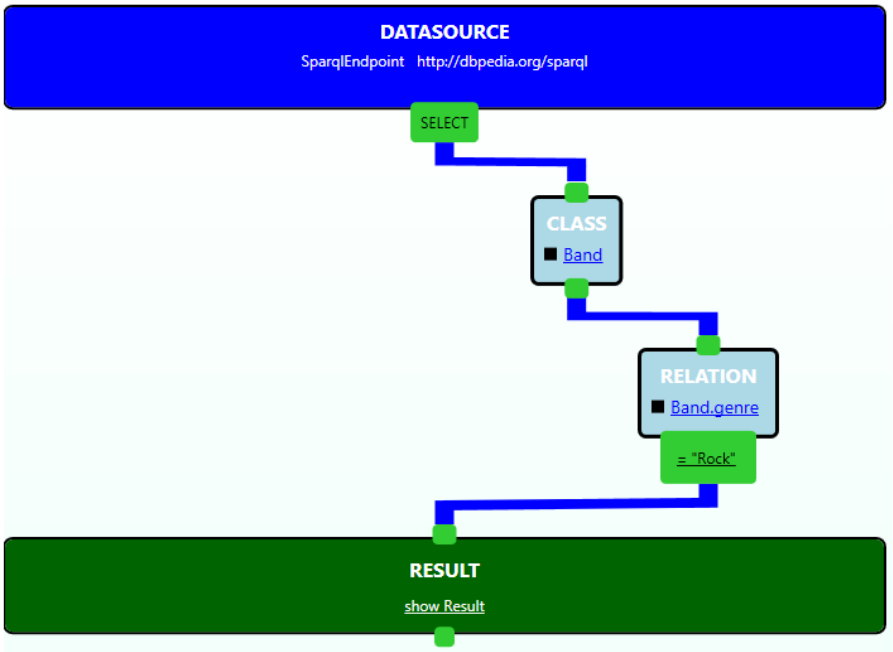
Wie bediene ich das Programm?



Wie bediene ich das Programm?



Wie bediene ich das Programm?



Schlussbemerkung

- Gedanken einfach laut aussprechen.
  Was denkst du beim Bedienen des Programms?

- Kein Prüfungsstress.
  Wir testen das Tool nicht den Probanden!

9.3 Aufgabenstellung

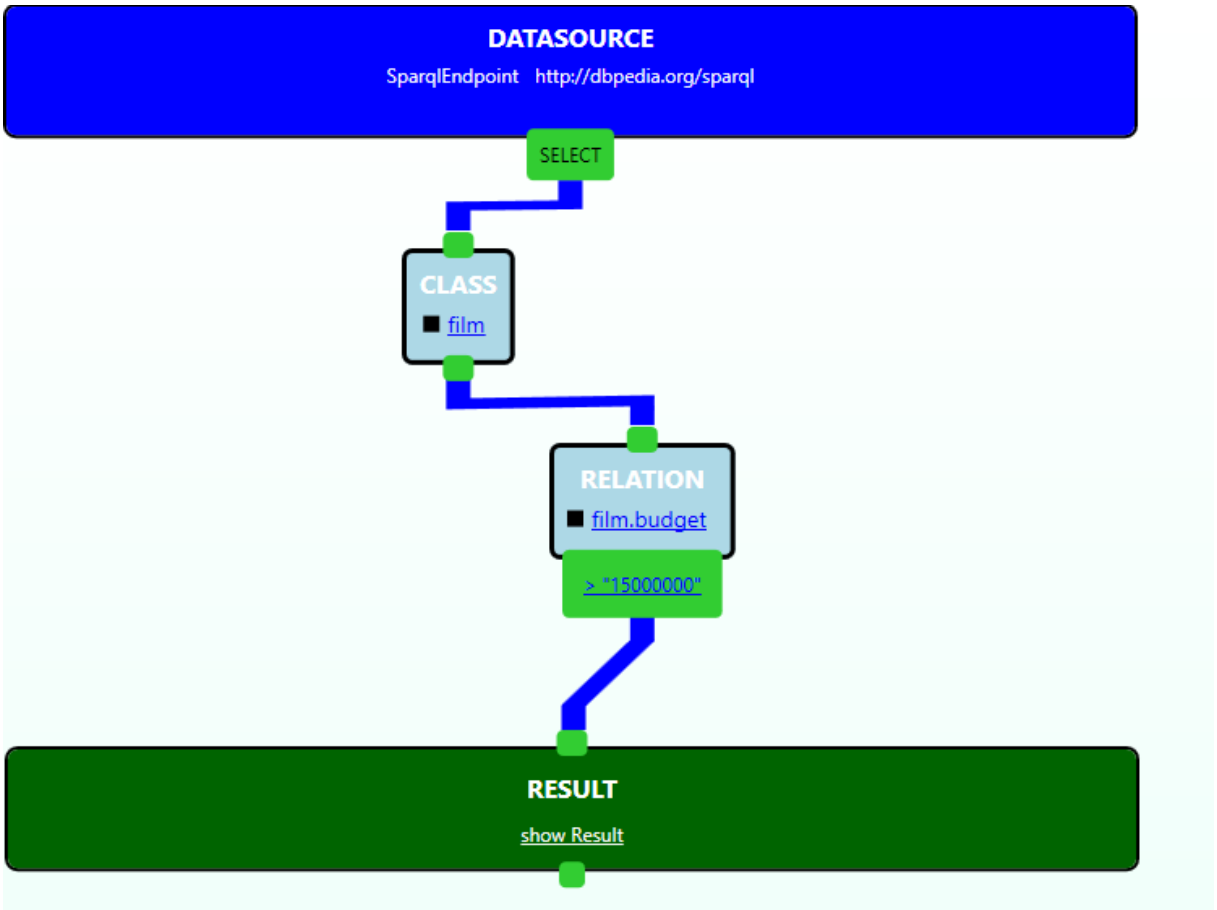
Im Folgenden sind die im Rahmen der beiden Benutzerstudien gestellten Aufgabenstellungen angefügt.

Aufgabe 1 a)

Was wird durch die folgende Anfrage gesucht?

Folgende Daten befinden sich in der Datenquelle:

Klasse	Individuum	Relation
Film	„Der Pate“	budget = 6 000 000 $
Film	„Pulp Fiction“	budget = 8 500 000 $
Film	„Up“	budget = 175 000 000 $
Film	„Toy Story“	budget = 30 000 000 $



Aufgabe 1 b)

Gesucht werden Städte (*City*) mit mehr als 100 000 Einwohnern (*population total*).

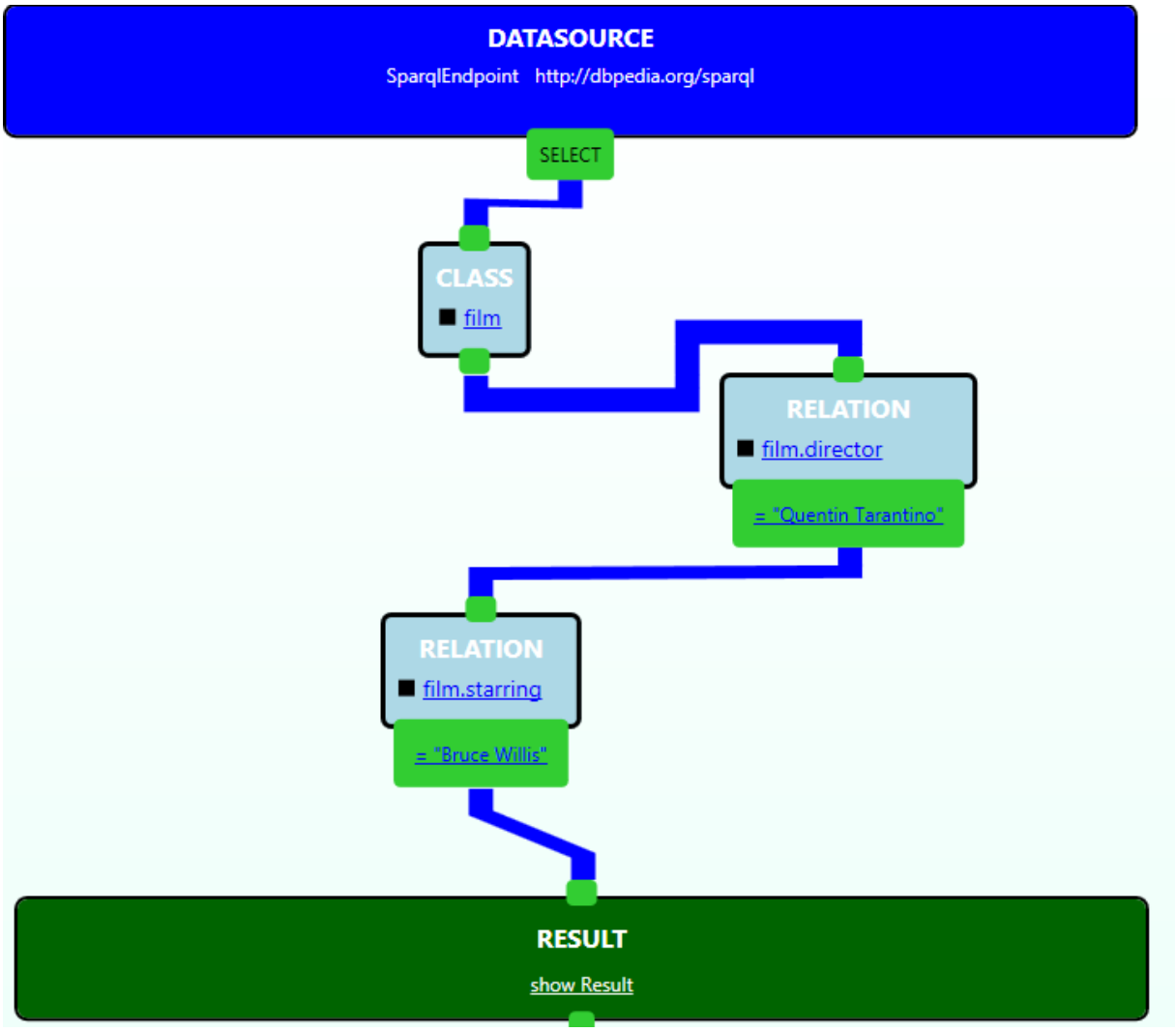
Erstellen Sie bitte hierfür eine entsprechende Anfrage.

Aufgabe 2 a)

Was wird durch die folgende Anfrage gesucht?

Folgende Daten befinden sich in der Datenquelle:

Klasse	**Individuum**	**Relation**
Film	„Der Pate“	budget = 6 000 000 $
Film	„Der Pate“	director = „Francis Ford Coppola“
Film	„Der Pate“	starring = „Al Pacino“
Film	„Pulp Fiction“	budget = 8 500 000 $
Film	„Pulp Fiction“	director = „Quentin Tarantino“
Film	„Pulp Fiction“	starring = „Bruce Willis“
Film	„Toy Story“	budget = 30 000 000 $
Film	„Toy Story“	director = „John Lasseter“
Film	„Up“	budget = 175 000 000 $
Film	„Up“	director = „Bob Peterson“
Actor	„Bruce Willis“	name = „Bruce Willis“


DATASOURCE
SparqlEndpoint http://dbpedia.org/sparql
SELECT
CLASS
film
RELATION
film.director
= "Quentin Tarantino"
RELATION
film.starring
= "Bruce Willis"
RESULT
show Result

Aufgabe 2 b)

Gesucht werden Städte (*City*) aus dem Land (*country*) Brasilien (*Brazil*) mit mehr als 1 000 000 Einwohnern (population total)

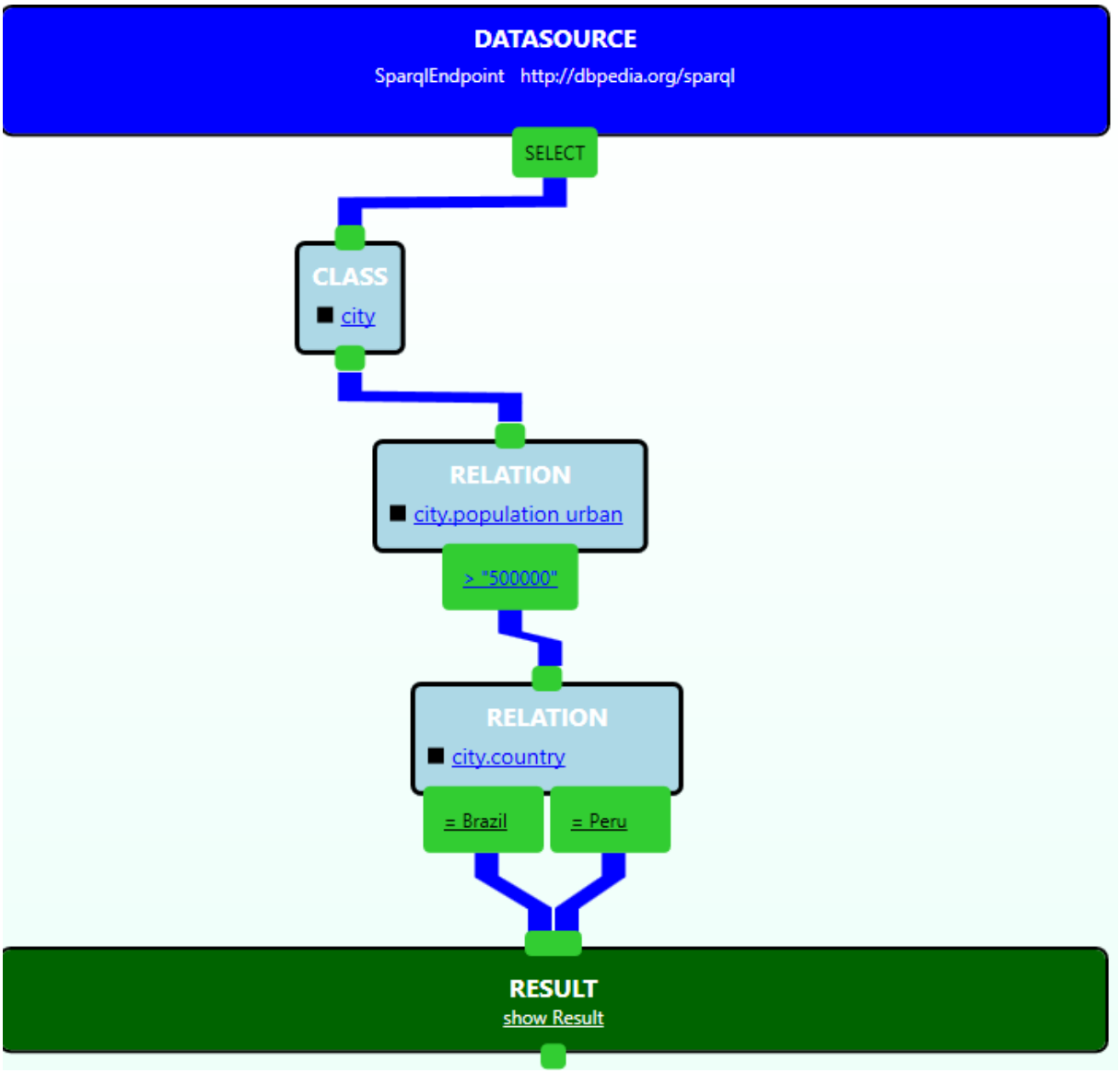
Erstellen Sie bitte hierfür eine entsprechende Anfrage.

Aufgabe 3 a)

Was wird durch die folgende Anfrage gesucht?

Folgende Daten befinden sich in der Datenquelle:

Klasse	**Individuum**	**Relation**
City	Campo Grande	population = 787,204
City	Campo Grande	country = „Brazil“
City	Lima	population = 8 486 866
City	Lima	country = „Peru“
City	Puno	population = 119 116
City	Puno	country = „Peru“
City	Santa Luzia	population = 231,607
City	Santa Luzia	country = „Brazil“
City	Sao Paulo	population = 11 244 369
City	Sao Paulo	country = „Brazil“



Aufgabe 3 b)

Gesucht wird eine Programmiersprache (*Programming Language*), die durch „Java (programming language)“ oder „BASIC“ beeinflusst (*influenced by*) wurde.

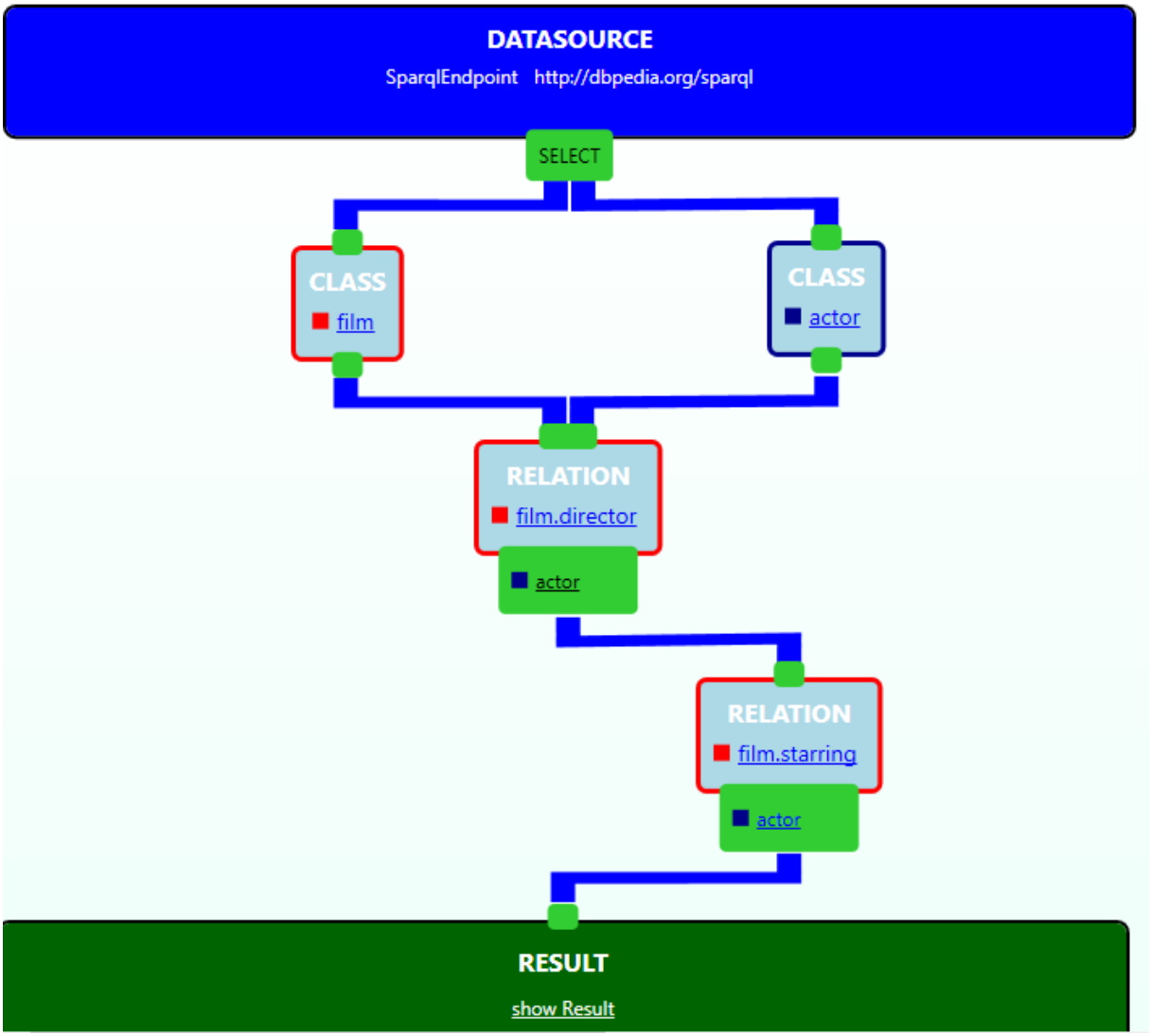
Erstellen Sie bitte hierfür eine entsprechende Anfrage.

Aufgabe 4 a)

Was wird durch die folgende Anfrage gesucht?

Folgende Daten befinden sich in der Datenquelle:

Klasse	**Individuum**	**Relation**
Film	„Der Pate“	budget = 6 000 000 $
Film	„Der Pate“	director = „Francis Ford Coppola“
Film	„Der Pate“	starring = „Al Pacino“
Film	„Pulp Fiction“	budget = 8 500 000 $
Film	„Pulp Fiction“	director = „Quentin Tarantino“
Film	„Pulp Fiction“	starring = „Quentin Tarantino“
Film	„Toy Story“	budget = 30 000 000 $
Film	„Toy Story“	director = „John Lasseter“
Film	„Up“	budget = 175 000 000 $
Film	„Up“	director = „Bob Peterson“


DATASOURCE
SparqlEndpoint http://dbpedia.org/sparql
SELECT
CLASS
film
CLASS
actor
RELATION
film.director
actor
RELATION
film.starring
actor
RESULT
show Result

Aufgabe 4 b)

Eine Programmiersprache (*Programming Language*), die durch eine andere Programmiersprache (*Programming Language*) beeinflusst (influenced_by) wurde, die von der Firma „Microsoft“ entwickelt (*developer*) wurde.

Erstellen Sie bitte hierfür eine entsprechende Anfrage.

Literaturverzeichnis

[1] Karadeniz, Besim (2013): netplanet - Geschichte des Internet - Das Phänomen World Wide Web. Online verfügbar unter http://www.netplanet.org/geschichte/worldwideweb.shtml, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[2] The original proposal of the WWW, HTMLized (2001). Online verfügbar unter http://www.w3.org/History/1989/proposal.html, zuletzt aktualisiert am 20.06.2001, zuletzt geprüft am 09.06.2013.

[3] EMC-Studie: jährlich erzeugte Datenmenge steigt bis 2020 um Faktor 44 (2013). Online verfügbar unter http://www.marketing-boerse.de/News/details/EMC-Studie-jaehrlich-erzeugte-Datenmenge-steigt-bis-2020-um-Faktor-44/22270, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[4] Esposito, Elena. Die Formen des Web-Gedächtnisses. Medien und soziales Gedächtnis. In: *Formen und Funktionen sozialen Erinnerns*. Springer Fachmedien Wiesbaden, 2013. S. 91-103.

[5] dpa; online, heise (2012): Innenminister: Es gibt keinen digitalen Radiergummi. Heise Zeitschriften Verlag. Online verfügbar unter http://www.heise.de/newsticker/meldung/Innenminister-Es-gibt-keinen-digitalen-Radiergummi-1569776.html, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[6] History of the Virtual Library (2012). Online verfügbar unter http://vlib.org/admin/history, zuletzt aktualisiert am 21.12.2012, zuletzt geprüft am 10.06.2013.

[7] Brin, Sergey; PAGE, Lawrence. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 1998, 30. Jg., Nr. 1, S. 107-117.

[8] Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web."*Scientific american* 284.5 (2001): 28-37.

[9] Shneiderman, Ben: Visual User Interfaces for Information Exploration. In: *Proceeding of the 54th Annual Meeting of The Amercian Society for Information Sciences,vol. 28 (Washington, DC, Oct. 27-31, 1991)* 28 (1991), August, S. 379–384

[10] Pasca, Marius, et al. Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge. In: *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006. S. 1400.

[11] Ashton, Kevin. That ‘Internet of Things’ Thing. *RFiD Journal*, 2009, 22. Jg., S. 97-114.

[12] Ich sah den Mann auf dem Berg mit dem Fernrohr (2013). Online verfügbar unter http://www.jbusse.de/semauth/Mann_Berg_Fernrohr.html, zuletzt aktualisiert am 12.03.2013, zuletzt geprüft am 10.06.2013.

[13] What is Metaweb? (2013). Online verfügbar unter http://www.youtube.com/watch?v=tBSdYi4EY3s, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[14] Wikipedia (Hg.) (2013): Homonym. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=118982383, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[15] Wikipedia (Hg.) (2013): Polysemie. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=116908411, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[16] Dietrich Homberger: *Sachwörterbuch zur deutschen Sprache und Grammatik.* Diesterweg, Frankfurt/Main 1989, Stichwort: *Graphem,* Seite 53.

[17] Baker, Mark & Jonathan Bobaljik (2002): Introduction to Morphology. Ms. Rutgers University and McGill University.

[18] Wikipedia (Hg.) (2013): Ontologie. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=119245040, zuletzt aktualisiert am 09.06.2013, zuletzt geprüft am 10.06.2013.

[19] Hideaki Takeda: Ontologies. Online verfügbar unter http://www-kasm.nii.ac.jp/~takeda/lecture-kss/ontologies-for-lecture07.pdf, zuletzt geprüft am 10.06.2013.

[20] Hitzler, Pascal (2008): Semantic web. Grundlagen. 1. Aufl. Berlin: Springer Berlin (EXamen.press). S.17-38

[21] RDF/XML Syntax Specification (Revised) (2004). Online verfügbar unter http://www.w3.org/TR/rdf-syntax-grammar/, zuletzt aktualisiert am 10.02.2004, zuletzt geprüft am 10.06.2013.

[22] RDF Vocabulary Description Language 1.0: RDF Schema (2004). Online verfügbar unter http://www.w3.org/TR/rdf-schema/, zuletzt aktualisiert am 10.02.2004, zuletzt geprüft am 10.06.2013.

[23] OWL 2 Web Ontology Language Document Overview (Second Edition) (2012). Online verfügbar unter http://www.w3.org/TR/owl2-overview/, zuletzt aktualisiert am 09.12.2012, zuletzt geprüft am 10.06.2013.

[24] Inference - W3C (2013). Online verfügbar unter http://www.w3.org/standards/semanticweb/inference, zuletzt aktualisiert am 02.03.2013, zuletzt geprüft am 09.06.2013.

[25] Hitzler, Pascal (2008): Semantic web. Grundlagen. 1. Aufl. Berlin: Springer Berlin (EXamen.press). S.125

[26] Frequently Asked Questions on W3C's Web Ontology Language (OWL) (2008). Online verfügbar unter http://www.w3.org/2003/08/owlfaq, zuletzt aktualisiert am 21.05.2008, zuletzt geprüft am 09.06.2013.

[27] Chris Welty; IBM Research: Rules Interchange Format Basic Logic Dialect. Online verfügbar unter http://www.w3.org/2005/rules/wiki/images/b/b0/W3C_RIF-CW-9-09.pdf, zuletzt geprüft am 09.06.2013.

[28] RIF Overview (Second Edition) (2013). Online verfügbar unter http://www.w3.org/TR/rif-overview/, zuletzt aktualisiert am 04.02.2013, zuletzt geprüft am 09.06.2013.

[29] SPARQL Query Language for RDF (2013). Online verfügbar unter http://www.w3.org/TR/rdf-sparql-query/, zuletzt aktualisiert am 26.03.2013, zuletzt geprüft am 09.06.2013.

[30] Wikipedia (Hg.) (2013): Rekursives Akronym. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=118171705, zuletzt aktualisiert am 07.05.2013, zuletzt geprüft am 09.06.2013.

[31] SPARQL 1.1 Overview (2013). Online verfügbar unter http://www.w3.org/TR/sparql11-overview/, zuletzt aktualisiert am 21.03.2013, zuletzt geprüft am 09.06.2013.

[32] Jäckle Dominik: SPARQL, Sparql Protocol And RDF Query Language (2009). Seminar im Rahmen des Studienprojekts SemSor am Institut für Visualisierung und Interaktive Systeme - Universität Stuttgart. Online verfügbar unter http://www.vis.uni-stuttgart.de/htdocs/ger/teaching/lecture/ws09/stupro/seminar/4_sparql_ausarbeitung.pdf, zuletzt geprüft am 09.06.2013.

[33] SPARQL 1.1 Query Language (2013). Online verfügbar unter http://www.w3.org/TR/2013/REC-sparql11-query-20130321/#QueryForms, zuletzt aktualisiert am 21.03.2013, zuletzt geprüft am 09.06.2013.

[34] Bizer, Christian; Heath, Tom; Berners-Lee, Tim. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009, 5. Jg., Nr. 3, S. 1-22.

[35] (2013). Online verfügbar unter http://www.lod-cloud.net/versions/2011-09-19/lod-cloud_colored.html, zuletzt aktualisiert am 15.02.2013, zuletzt geprüft am 09.06.2013.

[36] Bizer, Christian, et al. DBpedia-A crystallization point for the Web of Data.*Web Semantics: Science, Services and Agents on the World Wide Web*, 2009, 7. Jg., Nr. 3, S. 154-165.

[37] Willenborg, Josef: *Anfragesprachen für Internet-Informationssysteme.* Berlin, Humboldt-Univ., Diss., 2001 http://www.josef-willenborg.de/publications/dissertation.pdf Seite 29ff.

[38] U. Hedtstück: *Einführung in die Theoretische Informatik - Formale Sprachen und Automatentheorie*. Oldenbourg Verlag, München 2000.

[39] Benzi, Francesca, Dario Maio, and Stefano Rizzi. "VISIONARY: a viewpoint-based visual language for querying relational databases." *Journal of Visual Languages and Computing* 10.2 (1999): 117-145.

[40] Murray, Norman, Norman Paton, and Carole Goble. "Kaleidoquery: a visual query language for object databases." *Proceedings of the working conference on Advanced visual interfaces*. ACM, 1998.

[41] Jarrar, M., and Dikaiakos, M. D.: MashQL: A Query-By-Diagram Language for Data Mashups. Technical Article (No. TAR200805). University of Cyprus, 2008 http://www.jarrar.info/publications/JD08.pdf

[42] Zloof, Moshe M. "Query-by-example: A data base language." *IBM systems Journal* 16.4 (1977): 324-343.

[43] Czejdo, Bogdan, et al. "An algebraic language for graphical query formulation using an extended entity-relationship model." *Proceedings of the 15th annual conference on Computer Science*. ACM, 1987.

[44] Russell, Alistair, et al. "NITELIGHT: A Graphical Tool for Semantic Query Construction." (2008).

[45] Batini, C., et al. "Visual Query Systems: A Taxonomy." *Visual database systems, II: proceedings of the IFIP TC 2/WG2. 6 Second Working Conference on Visual Database Systems, Budapest, Hungary, 30 September-3 October, 1991*. Vol. 7. North Holland, 1992.

[46] Catarci, Tiziana, et al. "Visual query systems for databases: A survey." *Journal of visual languages and computing* 8.2 (1997): 215-260.

[47] Wikipedia (Hg.) (2013): Query by Example. Online verfügbar unter http://de.wikipedia.org/w/index.php?oldid=116844665, zuletzt aktualisiert am 02.06.2013, zuletzt geprüft am 09.06.2013.

[48] Maurer, Hermann, et al. Query-By-Example (QBE). In: *From Databases to Hypermedia*. Springer Berlin Heidelberg, 1998. S. 136-151.

[49] Tsuda, Kazuyuki, et al. "Iconicbrowser: An iconic retrieval system for object-oriented databases." *Journal of Visual Languages & Computing* 1.1 (1990): 59-76.

[50] Young, Degi ; Shneiderman, Ben: A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. In: *Technical Reports*

[51] Haag, Florian, Steffen Lohmann, and Thomas Ertl. "Simplifying filter/flow graphs by subgraph substitution." *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*. IEEE, 2012.

[52] Haag, Florian, Steffen Lohmann, and Thomas Ertl. A Flexible Architecture for Filter/Flow-Based Visual Querying. In: Graphics Interface 2013 Posters

[53] Haag, Florian, Steffen Lohmann, and Thomas Ertl: Evaluating the Readability of Extended Filter/Flow Graphs. In: Proceedings of Graphics Interface 2013. S.33 -36.

[54] Katifori, Akrivi, et al. "A comparative study of four ontology visualization techniques in protege: Experiment setup and preliminary results." *Information Visualization, 2006. IV 2006. Tenth International Conference on*. IEEE, 2006.

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

(Steffen Bold)