Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 127

Visuelle Auswahl von Ontologiebestandteilen

Sarah Oppold

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Dipl.-Inf. Florian Haag
Beginn am:	14. Mai 2014
Beendet am:	14. November 2014
CR-Nummer:	H.5.2, H.3.3

Kurzfassung

Semantische Daten gewinnen immer mehr an Bedeutung, doch es bleibt selbst für erfahrene Anwender schwer, SPARQL-Anfragen an RDF-Endpoints zu senden. Dies liegt nicht nur an der komplexen Syntax von SPARQL, sondern auch an den RDF-Daten selbst, denn die Ressourcen sind über URIs eindeutig identifizierbar. Diese sind aber für Menschen zu komplex um sie sich merken zu können und es ist auch nur selten möglich sie selbst zu erschließen. Deshalb wurde in dieser Arbeit zunächst ein Konzept für eine Visualisierung entwickelt, mit der man Ressourcen in RDF-Graphen finden kann. Dieses setzt Subgraphen ein, die der Nutzer inkrementell nach seinen Bedürfnissen erweitern kann um die Daten zu erforschen und einem Suchfeld um gezielt Ressourcen zu suchen. Das Konzept wurde in einem WPF-Prototypen umgesetzt und mit Hilfe einer kleinen Studie überprüft.
Durch die Studie stellte sich heraus, dass der im Konzept entwickelte Subgraphen zu schnell zu komplex wird um von unerfahrenen Nutzern leicht verstanden zu werden. Doch das Prinzip wurde überwiegend als sinnvoll bewertet. Darüber hinaus wurden Möglichkeiten vorgestellt, wie man das Konzept an diesen Stellen anpassen könnte um den Bedürfnissen der Nutzer gerecht zu werden. Auch die Suchleiste wurde als sinnvoll, wenn auch verbesserbar erachtet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In diesem Kapitel wird in den Hintergrund der Arbeit eingeführt und das Ziel der Arbeit erläutert. Anschließend wird ein Überblick über die Arbeit gegeben.

1.1. Hintergrund

Die Menge der Daten im Internet nimmt beständig zu. Es sind mittlerweile zu viele Informationen im Netz vorhanden und verteilt, als dass sie von Menschen sinnvoll gesichtet und zusammengefasst werden können. Dadurch können die Nutzer keinen Gewinn aus den Ergebnissen anderer Nutzer ziehen. Allerdings ist es für Maschinen auch nicht möglich, dass sie die Informationen, die im Netz stehen, lesen und sinnvoll in Beziehungen bringen können. Deshalb müssen sie in maschinenlesbare Informationen umgewandelt werden. Schon vor über 10 Jahren hat Tim Berners-Lee das Semantic Web beschrieben, das das Internet nicht ersetzen soll, sondern die bestehenden Daten um eine Beschreibung ergänzt, die Maschinen verstehen und verarbeiten können [BL98].

Dieses Konzept wird nach und nach umgesetzt und es sind schon einige große semantische Datenbanken frei verfügbar, die solche maschinenlesbaren Informationen bereitstellen. Anfragen an solche semantischen Datenbankschnittstellen können mittels der Sprache SPARQL entworfen werden. Deren Erstellung ist allerdings nicht trivial. Deswegen wurden schon einige Visualisierungen entwickelt, die den Entwurf von SPARQL-Anfragen vereinfachen sollen. Doch selbst damit ist es schwierig Anfragen zu erstellen, denn um eine bestimmte Ressource in solch einem komplexen Datensatz zu finden um sie in der Anfrage zu verwenden, muss man sich ausführlich mit dem Datensatz auseinandersetzen.

1.2. Ziel der Arbeit

Das Ziel dieser Bachelor-Arbeit ist die Entwicklung eines Visualisierungskonzepts, mit dem Ressourcen in RDF-Graphen einfach und intuitiv gefunden werden können. Dies soll sowohl für ungeübte als auch erfahrene Benutzer gelten. Außerdem soll es möglich sein, einen Überblick über den gesamten Datensatz zu erlangen, der ein besseres Verständnis der Daten ermöglichen soll. Das in der Arbeit verwendete Konzept soll in anderen Anwendungen weiterverwendet werden, in denen man die gefundenen Ressourcen weiterverwenden kann, zum Beispiel Anwendungen für SPARQL-Anfragen.

1.3. Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 –Grundlagen In diesem Kapitel werden Semantic Web, das daraus entstandene RDF und RDF Schema beschrieben. Außerdem wird die dafür entwickelte Anfragesprache SPARQL erklärt. Diese Themen stellen die Grundlagen für diese Arbeit dar.

Kapitel 3 – Verwandte Arbeiten Hier werden verwandte Konzepte und Visualisierungen vorgestellt und diskutiert inwiefern sie sich Teile davon für die Verwendung in diesem Konzept eignen.

Kapitel 4 – Konzept In diesem Kaptiel wird das Visualisierungskonzept, das im Rahmen dieser Arbeit aus den Recherchen entstanden ist vorgestellt und genauer definiert.

Kapitel 5 – Implementierung Um das Visualisierungskonzept in einer Studie zu testen, wurde ein Prototyp implementiert, der das Konzept umsetzt.

Kapitel 6 – Studie Um die Verwendbarkeit des Konzeptes zu untersuchen, wurde der Prototyp von Probanden getestet und evaluiert inwiefern das Konzept für Nutzer geeignet ist, um Ressourcen in RDF-Graphen zu finden.

Kapitel 7 – Zusammenfassung und Ausblick Abschließend wird diese Arbeit zusammengefasst und ein Ausblick auf die weitere Arbeit mit diesem Konzept gegeben.

2. Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte und Technologien erläutert, auf denen diese Arbeit aufbaut. Zunächst wird das Semantic Web beschrieben, danach RDF, das wegen des Semantic Webs entwickelt wurde. Anschließend wird auf das zugrundeliegende Schema von RDF eingegangen und die darauf operierende Anfragesprache SPARQL.

2.1. Das Semantic Web

Viele Menschen verwenden heutzutage das Internet. Sie kommunizieren mit anderen Menschen, präsentieren Informationen auf Webseiten und speichern Daten, um die Möglichkeit zu haben, sie von einem beliebigen Ort über das Internet wieder zu erreichen. Dabei wird nur Wert darauf gelegt, dass auch andere Menschen diese Informationen lesen und verstehen können. Die Daten, die Maschinen lesen können, betreffen meist nur das Layout.

Trotz Fortschritten in Forschungsgebieten wie künstlicher Intelligenz und maschineller Sprachverarbeitung ist es für Computer noch nicht möglich, die Semantik der komplexen menschlichen Sprachen zu verstehen und zu verarbeiten. Doch schon 1998 hat Tim Berners-Lee, der Erfinder des World Wide Web, eine Möglichkeit ausgearbeitet, inwiefern man das Internet mittels Aufbereitung der Daten ergänzen kann, um Maschinen in diesem Punkt entgegenzukommen. Dieses Konzept ist das Semantic Web.

Berners-Lees Idee war es, eine Sprache bereitzustellen, mit deren Hilfe man sowohl Daten als auch Regeln über diesen Daten ausdrücken kann und die es zusätzlich erlaubt sie mit Regeln von anderen Wissenssystemen zu kombinieren. Er schlug vor, ein einfaches, generell gehaltenes Modell zu verwenden, dessen Elementen mithilfe eines Schemas Regeln zugeordnet werden können. Zusätzlich sollte es innerhalb einer Logik-Schicht möglich sein, Regeln und Daten zu validieren und mittels Anfragesprachen auf die Daten zuzugreifen. Damit sollen die bereits bestehenden Inhalte erweitert werden, um ihnen für Maschinen lesbare Strukturen, Zusammenhänge und Bedeutungen zuzuordnen.

Mithilfe dieser Konstrukte sollen nun Maschinen die gesamten Inhalte des Internets verstehen und verarbeiten können. Damit wird es möglich, Suchmaschinen durch semantische Informationen zu verbessern, oder auch aus den Ergebnissen von Arbeiten der Menschen weltweit Schlüsse zu ziehen, die man durch die Masse von Information als Mensch nicht finden könnte.

Durch die Entwicklung des Internets der Dinge können in Zukunft eventuell auch Geräte wie Fernseher, Handys etc. in das Semantic Web integriert werden [BL98, BLHL01].

2.2. RDF

Das Resource Description Framework (RDF) ist ein Framework, das verwendet werden kann um semantische Informationen im Internet darzustellen [BLHL01], [W3C14a].

Die Grundstruktur sind Tripel bestehend aus Subjekt, Prädikat und Objekt. Durch ein Tripel wird ausgedrückt, dass zwischen dem Element, das das Subjekt darstellt und dem Objekt eine Verbindung besteht, die durch das Prädikat beschrieben wird. Dies kann auch graphisch veranschaulicht werden, wie in Abbildung 2.1 dargestellt. Das Tripel beschreibt eine Aussage. Dies kann zum Beispiel eine Behauptung über einen Teil der Welt sein, oder ein logischer Ausdruck. Eine Menge von Tripeln bildet einen RDF-Graphen. Ein oder mehrere Graphen können in einem RDF-Dokument aufgeschrieben und zusammengefasst werden. Durch diese Dokumente kann man die Graphen zwischen Systemen austauschen. Eine geläufige Form der Darstellung für RDF-Graphen sind XML-Dokumente.

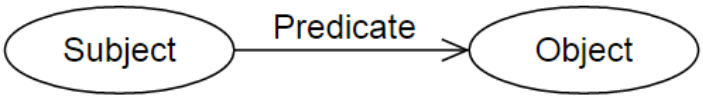


Abbildung 2.1.: Ein RDF-Graph mit zwei Knoten (Subjekt und Objekt), die durch ein Prädikat verbunden sind [W3C14a].

In einem RDF-Graphen können drei verschiedene Arten von Knoten vorkommen: IRIs (Internationalized Resource Identifier), Literale und blank nodes.

Eine IRI ist ein Unicode String, der eine Obermenge von URIs (Universal Resource Identifier) darstellt, die wiederum eine Obermenge der bekannteren URLs (Uniform Resource Locator) sind, die für Webseiten verwendet werden. IRIs sind in RDF-Graphen dafür verantwortlich, Ressourcen global eindeutig zu definieren.

Literale beschreiben einen Wert. Sie bestehen zusätzlich zum Wert aus einer IRI, die den Datentyp eindeutig identifiziert, und optional einem Sprachen Marker. Es wurden fast alle XSD Datentypen [W3C12] übernommen und um die Datentypen „HTML“ und „XML-Literal“ ergänzt.

IRIs und Literale beschreiben Elemente aus einer Miniwelt. Sie werden Ressourcen genannt. Ressourcen können alles darstellen: physische Gegenstände, abstrakte Konzepte, Nummern, Zeichenketten, etc. Literale und IRIs sind Konstanten. Sowohl der Wert eines Literals, als auch die IRI zu einer bestehenden Ressource sind unveränderbar.

Blank nodes stehen für unspezifizierte Ressourcen. Sie bedeuten, dass die Verbindung besteht ohne einen Partner explizit zu nennen. Blank nodes sind disjunkt von IRIs und Literalen und haben keinen Identifikator in der RDF-Syntax.

Subjekte, Prädikate und Objekte können IRIs sein, nur Subjekte und Objekte können eine blank node sein und lediglich Objekte können Literale sein. Es ist jedoch möglich, dass die IRI des Prädikates im einen Tripel das Subjekt oder Objekt eines anderen Tripels im selben Graphen ist.

2.3. RDF-Schema

Das RDF-Schema (RDFS) ist eine Erweiterung von RDF. Mithilfe von zusätzlichen Schlüsselwörtern können mit dem RDFS Gruppen von Ressourcen und Beziehungen zwischen diesen Gruppen definiert werden. Das RDF-Schema ist in RDF geschrieben [W3C14b].

Mit dem RDFS können Klassen definiert werden. Das Prädikat „rdf:type“ kann dann verwendet werden um auszudrücken, dass eine Ressource eine Instanz einer Klasse ist. Des Weiteren stellt das Schema Unterklassen, Unterprädikate und Möglichkeiten zur Eingrenzung der Datentypen zur Verfügung. Damit können weitere Aussagen über die RDF-Daten getroffen werden und die Vorgaben zur Verwendung des Schemas weiter spezifiziert werden.

2.4. SPARQL

SPARQL steht für SPARQL Protocol And RDF Query Language. Es ist eine Anfragesprache, die speziell für RDF-Datensätze erstellt wurde [W3C13].

In ihrem Aufbau ähneln SPARQL-Anfragen stark der SQL-Syntax. Die Syntax, Verwendung und Rückgabewerte von SPARQL-Anfragen, die auch in diesem Kapitel erklärt werden, wurden vom World Wide Web Consortium spezifiziert.

Eine SPARQL-Anfrage besteht aus fünf Teilen: „Prefix“ Klausel, je nach Anfrage-Art spezifische Informationen, „Dataset“ Klausel, „Where“ Klausel und „Solution Modifier“.

In der „Prefix“ Klausel können Präfixe für Namensräume definiert werden. Wie in RDF kann man so die IRIs der Ressourcen abkürzen und so Schreibaufwand sparen.

Es gibt vier verschiedene Arten von Anfragen und somit auch vier verschiedene Klauseln in SPARQL:

SELECT Mithilfe dieser Klausel kann man, ähnlich wie in SQL-Anfragen, Variablen bestimmen, deren Ergebnisse zurückgegeben werden sollen. Nach Auswertung der Anfrage die Ergebnis-Tupel zurück, die die Anfrage erfüllen. Mit „*SELECT* *“ kann man die Belegungen aller in der Abfrage verwendeten Variablen zurückgeben lassen.

CONSTRUCT Als Ergebnis erhält man einen RDF-Graphen. Das Template hierfür wird in der *CONSTRUCT* Klausel vorgegeben, anhand dem die einzelnen Ergebnistupel der Anfrage ersetzt und vereinigt werden.

ASK Diese Art von Anfragen wird dazu verwendet, um herauszufinden, ob es zu einer Anfrage überhaupt ein Ergebnis-Tupel gibt, das die angegebenen Bedingungen erfüllt. Es wird nur ein „True“ zurückgegeben, falls es Ergebnis-Tupel gibt oder andernfalls ein „False“ zurückgegeben.

DESCRIBE Gibt einen einzelnen RDF-Graph zurück. Je nach Endpoint werden von allen Ergebnis-Tupeln, die die Bedingungen der Anfrage erfüllen, Informationen zurückgegeben, die im Datensatz enthalten sind.

In der „Dataset“ Klausel kann man nun als nächstes optional mit den Stichworten *FROM* und *FROM NAMED* die spezifischen Datensätze referenzieren, die man für seine Anfrage verwenden möcht.

Anschließend gibt man in der „Where“ Klausel mithilfe von Tripel Patterns die Bedingungen für die Anfrage an. Diese bestehen wie RDF-Tripel aus Subjekt, Prädikat, Objekt. Allerdings können Elemente daraus durch Variablen ersetzt werden. Durch die selbe Variable kann so zum Beispiel ausgedrückt werden, dass das Subjekt von zwei Tripel Patterns das selbe sein soll. In Tabelle 2.1 finden sich die dafür zu verwendeten Schreibweisen der Elemente wie URIs, Variablen etc. Um die Bedingungen außerdem genauer zu spezifizieren gibt es in SPARQL Schlüsselwörter für zum Beispiel Optionale Teile, Filter, Aggregatsfunktionen. Eine Übersicht ist in Tabelle 2.3 zu finden. Außerdem ist es möglich, Subqueries zu verwenden, um die gewollte Anfrage auszudrücken.

Beim Auswerten der Anfrage werden Subgraph-Matching-Algorithmen verwendet. Der durch die Anfrage entstehende Subgraph wird über den RDF-Graphen gelegt. Wenn die Variablen durch Ressourcen ersetzt werden können und das dadurch entstehende Ergebnis RDF-graphäquivalent zum Subgraphen ist, bilden die eingesetzten RDF-Daten ein Ergebnistupel.

Die dadurch entstehende Menge von Ergebnissen ist zunächst unsortiert. Sie können aber durch verschiedene Sequenzmodifikatoren in eine gewünschte Reihenfolge gebracht werden. Die möglichen Modifikatoren und ihre Wirkungsweise finden sich ausführlicher in Tabelle 2.2.

Je nach gewählter Art der Anfrage werden diese Ergebnistupel dann in anderer Form an den Nutzer weitergegeben.

Element	Schreibweise
Variablen	Variablen werden mit einem voranstehenden „$“ oder „?“ gekennzeichnet. „$abc“ und „?abc“ stehen in einer Anfrage für die selbe Variable „abc“.
Gruppierungen	Mithilfe der geschweiften Klammern „{“ und „}“ kann man mehrere Tripel Pattern zu einem gruppieren und mit der gesamten Gruppe weiterarbeiten.
URIs	In den Tripel Patterns kann man URIs mittels „<“ und „>“ ausdrücken, wobei die spitzen Klammern nicht Bestandteil der URI sind. Um die URIs nicht ganz ausschreiben zu müssen, können die Präfixe verwendet werden, die in der Präfix Klausel definiert wurden. Dabei trennt man Präfix und die den restlichen Teil der URI mit einem „:“.
Blank nodes	Blank nodes können entweder als Label geschrieben werden (z.B. „_:abc“), oder in abgekürzter Form mit „[]“. Blank nodes sind in den Tripel Patterns dabei Variablen und stellen keine Referenzen zu den spezifischen blank nodes in den RDF-Datensätzen dar.
type of	Man kann das Schlüsselwort „a“ als Prädikat in einem Tripel Pattern verwenden. Es ist dabei eine Abkürzung für die IRI http://www.w3.org./1999/02/22-rdf-syntax-ns#type. Das Schlüsselwort ist case sensitive.

Tabelle 2.1.: Besondere Schreibweisen der Elemente der SPARQL-Syntax [W3C13].

Schlüsselwort	Erklärung
ORDER BY	Sortiert die Ergebnisse anhand einer Variable aufsteigend (*ASC*) oder absteigend (*DESC*). Es kann auch eine Reihe von Variablen und Sortierungen angegeben werden.
DISTINCT	Dieses Schlüsselwort muss am Anfang der Anfrage, zum Beispiel in der *SELECT* Klausel angegeben werden. Dadurch wird ausgeschlossen, dass die gleiche Lösung mehrmals in der Lösungsmenge vorkommt. Duplikate werden gelöscht.
REDUCED	Erlaubt, dass doppelte Lösungen nicht in die Ergebnismenge übernommen werden. Ist aber keine strikte Vorgabe wie *DISTINCT*. Es wird auch am Anfang der Anfrage angegeben.
OFFSET	Zusammen mit einem Zahlenwert werden die angegebenen ersten Ergebnisse der Lösungsmenge nicht zurückgegeben.
LIMIT	Damit werden die zurückgegebenen Ergebnisse auf eine vorgegebene Anzahl beschränkt.

Tabelle 2.2.: In SPARQL zur Verfügung stehende Modifikatoren [W3C13]

Schlüsselwort	Erklärung
FILTER	Mit diesem Schlüsselwort kann man die Lösungsmenge eingrenzen. Für Variablen des Datentyps String kann man reguläre Ausdrücke verwenden, oder für Zahlen, die als Variablen auftauchen arithmetische Ausdrücke.
OPTIONAL	In diesem Teil kann man optionale Tripel Patterns spezifizieren. Wenn für diese optionalen Patterns keine Lösungen gefunden werden, wird die Kombination trotzdem nicht aus der Lösung entfernt.
NOT EXISTS, *EXISTS*, *MINUS*	SPARQL bietet zwei verschiedene Wege um Negation in Anfragen zu integrieren. Mit *FILTER NOT EXISTS* kann getestet werden, ob ein gegebenes Tripel Pattern nicht auf eine Datenmenge anwendbar ist. *FILTER EXISTS* prüft im Gegensatz dazu, ob das Muster, das im Filter angegeben wurde, im Datensatz vorhanden ist. Die Alternative für eine Negation in einer SPARQL-Anfrage ist die Verwendung von *MINUS*. Alle Datenkombinationen, die die dort angegebenen Tripel Patterns erfüllen, werden aus den Lösungen gelöscht, die die Bedingungen im bisherigen Tripel Pattern erfüllt haben.
CONCAT	Kann verwendet werden, um mehrere Variablen miteinander zu verbinden.
BIND, *AS*	Wenn man Werte einer neuen Variable zuweisen möchte, kann man die Schlüsselwörter *BIND* und *AS* verwenden, um in weiteren Tripel oder im *SELECT* mit dieser Variable zu arbeiten.
UNION	Manchmal gibt es für eine Anfrage verschiedene Alternativen, die alle gleichzeitig gelten sollen. Dafür kann man das Stichwort *UNION* verwenden. Es kann auch Gruppen von Tripel Patterns miteinander verknüpfen.
COUNT, *SUM*, *MIN*, *MAX*, *AVG*, *GROUP_CONCAT*, *SAMPLE*	Seit SPARQL Version 1.1 kann man auch Aggregatsfunktionen verwenden. Sie können an solchen Stellen genutzt werden, wo der Anfragesteller das Resultat als Berechnung über eine Gruppe der Lösungen erfragen möchte. Zum Beispiel den maximalen Wert, den eine Variable annehmen kann, anstelle der einzelnen Werte, aus denen anschließend noch der maximale Wert berechnet werden müsste.
GROUP BY	In Verbindung mit den Aggregaten wurde auch das Schlüsselwort *GROUP BY* eingeführt. Damit kann man die Ergebnisse der Anfrage nach Variablen gruppieren.
HAVING	Dieses Schlüsselwort hat ähnliche Funktionalität wie der Ausdruck *FILTER*. Der Unterschied ist nur, dass *HAVING* auf Gruppen arbeitet, anstelle von individuellen Ergebnissen. Es wurde auch in der SPARQL Version 1.1 eingeführt.

Tabelle 2.3.: Schlüsselwörter für das Erstellen der Bedingungen der SPARQL-Syntax [W3C13].

3. Verwandte Arbeiten

Im Folgenden werden einige verwandte Forschungsarbeiten untersucht. Zunächst werden verschiedene Arbeiten zur Visuellen Filterung von Daten und der visuellen Aufbereitung und Erforschung von RDF-Daten diskutiert. Anschließend wird ein Konzept zur Suche in RDF-Daten ohne Visualisierung vorgestellt.

3.1. Visuelle Filterung von Daten

Es ist bekannt, dass graphische Benutzeroberflächen geeigneter sind, Informationen darzustellen und damit zu arbeiten als Textfelder und Tabellen. Mit Visualisierungen können Aufgaben meist schneller und exakter ausgeführt werden, wobei dem Nutzer zu Gute kommt, dass er nicht so schnell Frustration spürt und auf Dauer weniger Zeichen von Müdigkeit zeigt [YS98].

Nicht zuletzt deshalb hat Shneiderman mit dynamischen Queries [Shn94] SQL-Anfragen visuell aufbereitet. Sie geben dem Nutzer schnelle, zunehmende und reversible Kontrolle über die Daten, die Auswahl von Elementen erfolgt per Klicks, man muss nichts eintippen und vor allem sofortiges und kontinuierliches Feedback seiner Arbeit [YS98], [JGZ+11], [MAEGJ04]. Dadurch wird dem Nutzer das Gefühl gegeben Kontrolle über die Daten zu haben. Es gibt ihm die Möglichkeit schnell, sicher, sogar spielerisch die Datenbank zu erkunden sozusagen über die Daten zu fliegen.

Durch das Aufzeigen der Kontexte und Zusammenhänge der Daten kann man es Neulingen ermöglichen, ein besseres Verständnis der Daten zu erhalten und sogar erfahrenen Nutzern helfen eine besser Übersicht über komplexe Datenbanken zu erlangen [Shn94].

SPARQL- und SQL-Anfragen bieten sich an visuell dargestellt zu werden, da sie zwar sehr mächtige Anfragen erstellen können, aber nur für professionelle Nutzer geeignet sind [JCCD11]. Zum einen, weil die Syntax der Anfragesprachen sehr komplex ist, zum andern, weil der Nutzer das Schema der (RDF) Datenbank wenigstens teilweise kennen muss, um die Anfragen schreiben zu können. Deshalb haben einige Forscher Visualisierungskonzepte entwickelt, um solche Anfragen leichter und schneller erstellen zu können. Im Folgenden werden vier verschiedene Konzepte vorgestellt, die mit unterschiedlichen Darstellungen Anfragen für verschiedene Datenbankarten und Anfragesprachen visualisieren.

3.1.1. FilterFlow

Da Forschungen gezeigt haben, dass Nutzer Schwierigkeiten damit haben boolsche Anfragen mit SQL zu spezifizieren, haben Young und Shneiderman [YS98] eine Visualisierung entwickelt, die die

Bedeutung der booleschen Operatoren AND, OR und Not übersetzt. Sie greifen dabei auf die Metapher von fließendem Wasser zurück, das durch Filter fließt. In ihrem Konzept werden die Daten durch Ströme dargestellt und die Bedingungen einer Anfrage durch Filter im Weg des Datenstroms.

Wie man in Abbildung 3.1 erkennen kann, untersuchen sie eine Tabellenspalte einer relationalen Datenbank, deren Daten von links nach rechts fließen. Alle andere Spalten sind durch Buttons im oberen Teil des Fensters verfügbar. Wenn der Nutzer sie als Filter in den Canvas zieht, kann er aus einer Liste aller möglichen Werte dieser Spalte die gewünschten auswählen, die die Daten erfüllen sollen.

Die gewünschten booleschen Operatoren werden durch die Platzierung der Attribut-Menüs im Canvas und den dadurch entstehenden Datenfluss erzeugt. Ein AND Operator kann zum Beispiel gebildet werden, indem man zwei Attribut-Menüs auf gleicher Höhe hintereinander setzt. Dies veranlasst den Datenfluss zuerst durch den einen, dann durch den anderen Filter zu fließen.

Mithilfe einer Studie konnten sie nachweisen, dass die Arbeit mit der graphischen Oberfläche als einfacher empfunden wurde, als die Erstellung von Anfragen mit einem textbasierten Interface. Vor allem das visuelle Feedback bei Änderungen der Anfrage und das Erstellen von Anfragen ohne Tippen wurden als gut empfunden. Allerdings stellten sie auch fest, dass es nicht günstig ist, die Werte der Attribute als Listen darzustellen. Für die Zukunft nahmen sie sich vor, dafür graphische Lösungen zu finden.

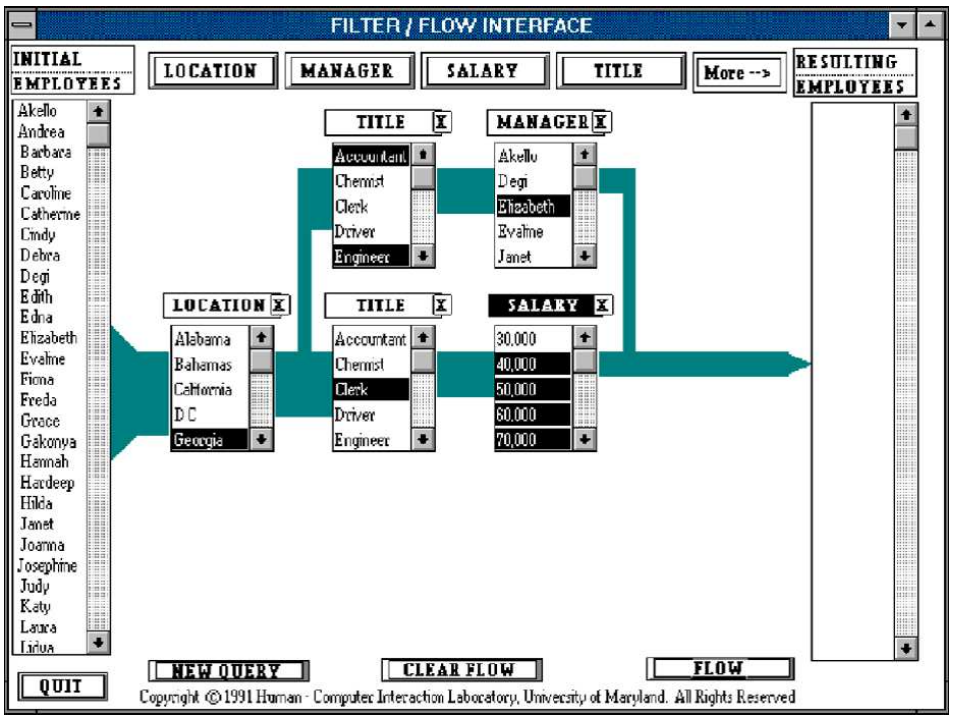


Abbildung 3.1.: Beispiel einer komplexen SQL-Anfrage, erstellt mit dem FilterFlow Prototyp [YS98]

3.1.2. Nitelight

Russell et al. entwickelten Nitelight - eine Applikation, mit der man SPARQL-Anfragen durch das Zusammenfügen von graphischen Elementen generieren kann [RSBS08].

Zuerst entwickelten sie vSPARQL – eine Menge von graphischen Elementen, die die gesamte Syntax von SPARQL abdecken soll. Dabei griffen sie, wie auch SPARQL auf das RDF-Konzept der Triple Patterns zurück. Subjekte und Objekte werden als Knoten, Prädikate als Kanten visualisiert. Durch Verknüpfungen, Gruppierungen, Ergänzungen der Knoten können sie die SPARQL-Syntax graphisch abbilden, wie in einem Beispiel in Abbildung 3.2 zu sehen. Als Vergleich ist hier im „SPARQL Example“ die textuelle SPARQL-Anfrage zu sehen.

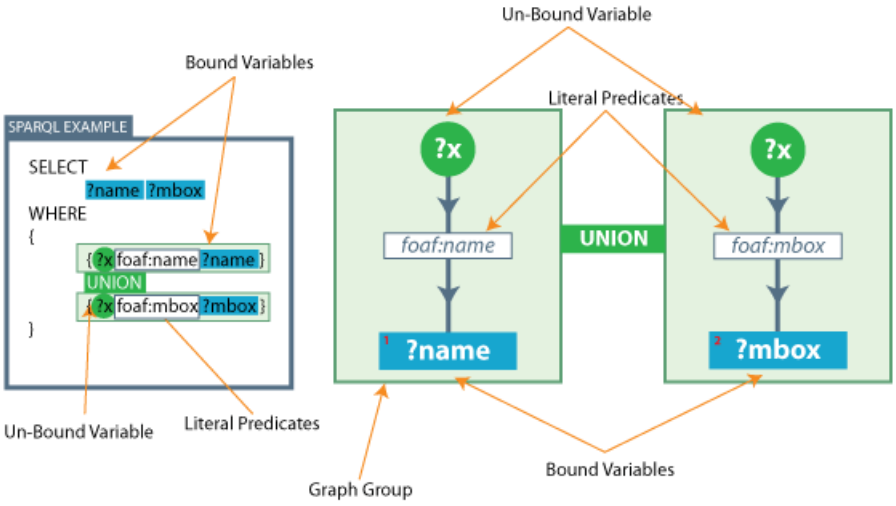


Abbildung 3.2.: Beispiel für die Verwendung von vSPARQL [RSBS08]

Um das Konzept von vSPARQL validieren zu können, entwickelten Russell et al. anschließend den Prototypen Nitelight. Wie in dem Fenster in Abbildung 3.3 zu sehen, wurde die graphische Darstellung von vSPARQL im query design canvas, in der Mitte eines Nitelight Fensters, übernommen. Auch die textuelle Version der SPARQL-Anfrage wird angezeigt. Zusätzliche Möglichkeiten von vSPARQL können über Menüs erreicht werden.

Damit der Nutzer keine URIs eintippen muss und um ihm einen Anhaltspunkt zu geben, wenn er in seiner SPARQL-Anfrage spezifische Ressourcen aus den RDF-Daten verwenden möchte, ist im oberen Drittel ein Ontology Browser integriert. Hier wird eine Liste der verfügbaren Ontologien, Klassen und Subklassen angeboten, die der Nutzer durchsuchen kann. Auch Eigenschaften von Klassen können über einen Klick erreicht werden.

Russell et al. führten noch keine Studie durch, stellten allerdings fest, dass das Verständnis der Ontologien, sowohl das Verständnis der Elemente darin von zentraler Bedeutung bei der Erstellung

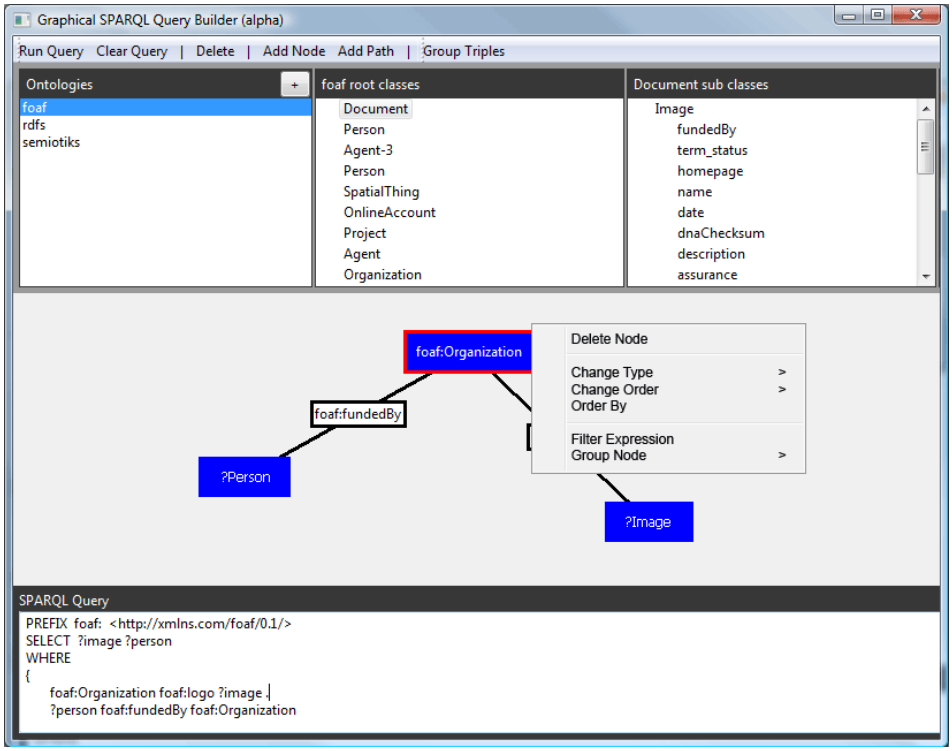


Abbildung 3.3.: Die Nitelight Benutzeroberfläche [RSBS08]

von SPARQL-Anfragen ist. Außerdem gaben sie an, für den Ontology Browser eine bessere Alternative entwickeln zu wollen, die ein Verständnis der Ontologie und ihrer Elemente für den Nutzer fördert.

3.1.3. FindFlow

Mit FindFlow entwickelten Hansaki et al. eine weitere Visualisierung für die Entwicklung von Anfragen [HSMT06]. Ähnlich wie FilterFlow basiert es auch auf den Konzept von Datenströmen, die durch Filter fließen. Allerdings wird die Metapher anders angewendet.

Wie in Abbildung 3.4 zu erkennen, werden in FindFlow die Filter als Kanten visualisiert, durch die die Daten fließen. Die Knoten stellen die Zwischenergebnisse dar. Dort werden die gesamten Daten aufgelistet, die nach dem durchlaufen der einzelnen Filter noch im Datenstrom vorhanden sind. Durch die Verwendung von Zwischenergebnissen sollen dem Nutzer mehr Möglichkeiten geboten werden. Es soll für ihn einfacher werden kleine Korrekturen zu testen, um zur gewünschten Anfrage zu kommen, ohne eine komplett neue Anfrage zu erstellen und die bisherigen Ergebnisse zu verlieren.

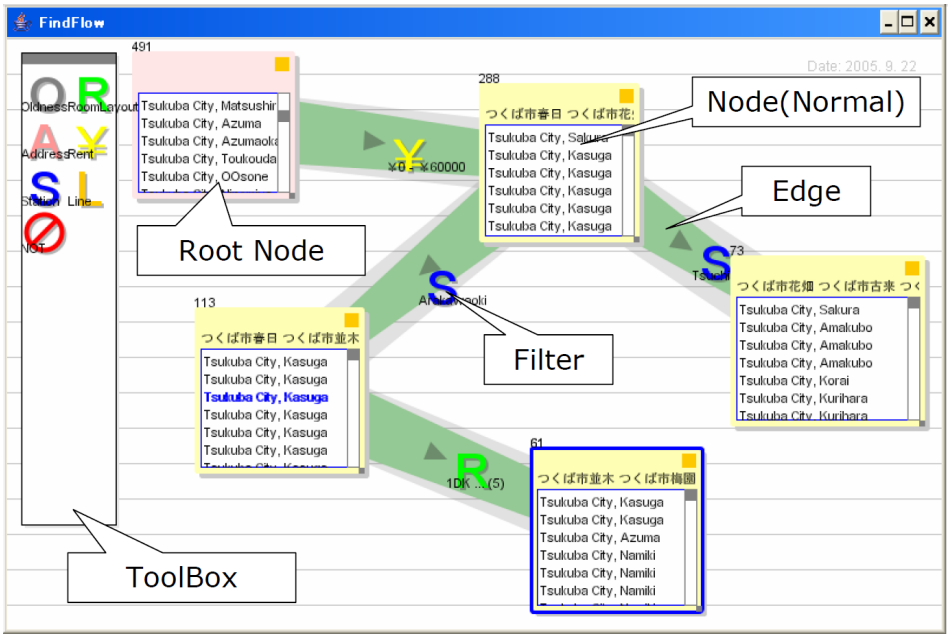


Abbildung 3.4.: Die Benutzeroberfläche von FindFlow [HSMT06].

3.1.4. Kaleidoquery

Murray et al. entwickelten mit Kaleidoquery eine visuelle Anfragesprache für Objektdatenbanken [MPG98]. Dabei greifen sie auch auf die Metapher von Datenflüssen, die durch Filter fließen, zurück.

Sie können das Schema verwenden, das bei Objektdatenbanken für die Objekte definiert wird und definieren für jede der Klassen beschriftete Symbole, die sie in ihrer Sprache verwenden. In den Knoten ihrer Darstellung werden die Zwischenergebnisse dargestellt, die für die Menge aller Instanzen einer Klasse, Ausgewählten Attributen dieser Instanzen, oder Gruppen von Elementen stehen können. Auf den Kanten werden die Filter eingetragen, die gelten sollen. Wie in Abbildung 3.5 zu sehen, haben sie auch einen Prototypen mit 3D-Visualisierung entwickelt, mit dem ein Nutzer Kaleidoquery verwenden kann.

3.2. Visualisierung von RDF-Daten

Nicht nur die Visualisierung von Anfragen an Datenbanken ist sinnvoll, auch die Visualisierung der Daten selbst ist nützlich für den Nutzer. Wie Russell et al. herausgearbeitet haben [RSBS08], ist es auch für Anfragen im Vorhinein wichtig, die Struktur der Daten zu kennen, um die Anfragen entwickeln zu können.

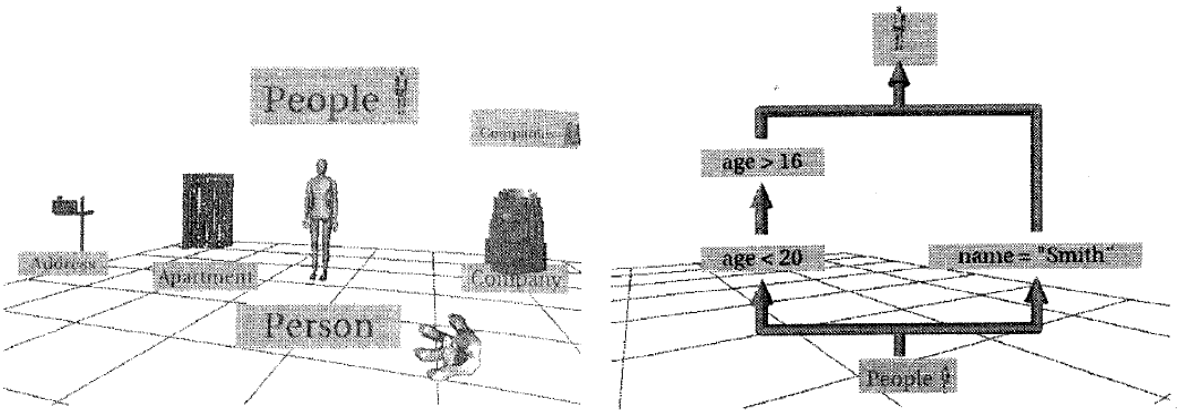


Abbildung 3.5.: Hier ist die Visualisierung der Anfragesprache Kaleidoquery zu sehen: links: Auswahl eines Elements, rechts: die erstellte Anfrage.

Dies stellt vor allem für RDF-Graphen eine besondere Herausforderung dar. Obwohl sie per Definition als Graph gebaut sind, können sie sehr komplex werden. Der ganze Graph ist dann aus mehreren Gründen nicht mehr sinnvoll auf einmal darzustellen:

- Die Darstellung der Knoten und Kanten benötigt viel Rechenkapazität.
- Ein visuelles Erforschen der Daten benötigt unter anderem viel Rechenkapazität und Zeit um die Veränderungen visuell darstellen zu können.
- Die Bildschirmgröße ist nicht ausreichend für viele Kanten, Knoten und Label.
- Einen Graphen mit vielen Kanten und Knoten kann ein Nutzer nicht mehr verstehen und begreifen.

Im Folgenden werden zwei Möglichkeiten aufgezeigt, wie man dennoch RDF-Daten visualisieren und sie visuell dynamisch erforschen kann.

3.2.1. Tabulator

Mit dem Tabulator haben Berners-Lee et al. einen generischen Browser für semantische Daten entwickelt [BLCC+06]. Dieser sollte für Nutzer eine leicht verständliche Visualisierung von RDF-Daten bieten und gleichzeitig Entwicklern die Möglichkeit offen halten, ihn leicht in ihre Anwendungen integrieren und anpassen zu können.

Der Tabulator hat zwei verschiedene, ineinandergreifende Modi: Entdecken und Analysieren. Wenn der Nutzer nicht weiß was für Daten in einem RDF-Dokument enthalten sind, kann er RDF-Graphen in einer Baum-Ansicht erforschen, die in Abbildung 3.6 zu sehen ist. Er kann zum einen verwandte Knoten entdecken, zum anderen weitere Informationen über gewünschte Knoten erhalten. Im zweiten Modus, Analysieren, kann der Nutzer Anfragen abschicken. Die Ergebnisse werden dann für ihn

in verschiedenen Ansichten visuell aufbereitet. Hierfür stehen bereits die Karten-Ansicht, Tabellen-Ansicht, Zeitstrahl-Ansicht und Kalender-Ansicht zur Verfügung. In Abbildung 3.7 ist beispielhaft die Kalender-Ansicht dargestellt. Der Browser ordnet selbstständig die Daten den einzelnen Elementen der Ansicht zu. Sollte der Nutzer dann mehr Informationen über eine Ressource erfahren wollen, kann er es wieder anklicken und in den Entdecken-Modus wechseln.

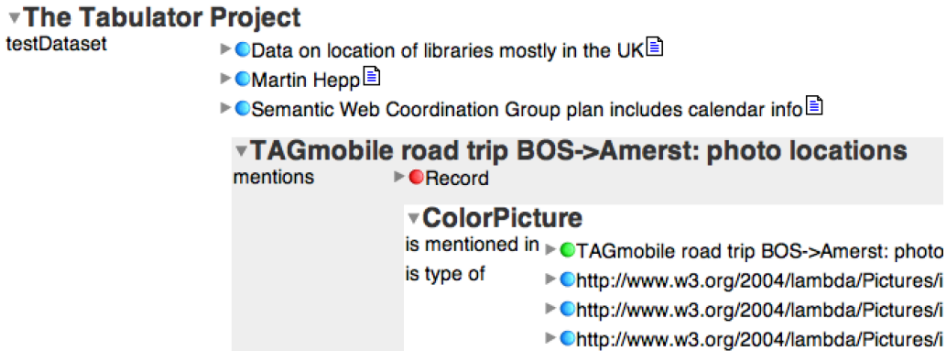


Abbildung 3.6.: Der Entdecken-Modus des Tabulator Browsers [BLCC+06].

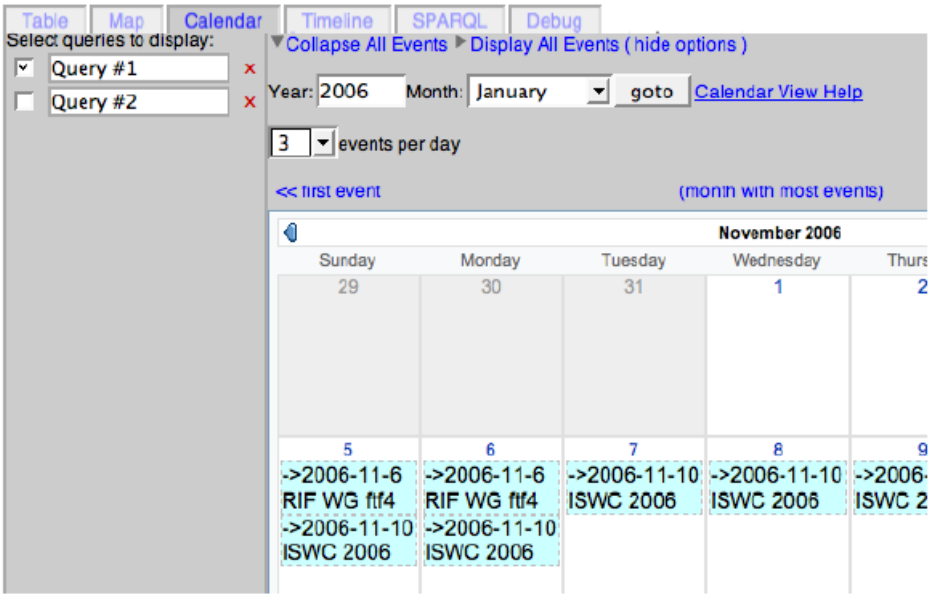


Abbildung 3.7.: Die Kalender-Ansicht im Analyse-Modus des Tabulator Browsers [BLCC+06].

Berners-Lee et al. haben sich aus diversen Gründen für eine Baum-Ansicht im Entdecken-Modus und Karten-, Tabellen-, Zeitstrahl- und Kalender-Ansicht im Analyse-Modus entschieden. Die Alternative wäre eine Graph-basierte Visualisierung mit Knoten und Kanten gewesen, die Vorteile haben, wenn man Cluster oder Zusammenhänge zwischen den Daten erkennen möchte. Allerdings sehen sie als

Verwendungszwecke Anwendungen für Finanz-, Musik- oder Kalender-Verwaltung und haben sich deswegen gegen eine Graph-Darstellung und für die oben beschriebenen Ansichten entschieden.

3.2.2. Paged Graph Visualization

Um Nutzern die Möglichkeit zu bieten, RDF-Daten über eine Visualisierung zu erforschen, und damit sie die Daten schneller und einfacher verstehen, entwickelten Deligiannidis et al. die Paged Graph Visualization (PGV) [DKS07]. Da große RDF-Ontologien detailliert betrachtet werden sollen, entschieden sie, mit einem kleinen Graphen zu beginnen und Werkzeuge bereitzustellen, um ihn inkrementell zu erforschen und die relevanten Daten zu visualisieren.

Die Visualisierungseinheit des PGV besteht aus zwei Elementen: Starter und Visualizer. Um einen Subgraphen zu finden, von dem aus der RDF-Graph erforscht werden soll, wird zunächst der Starter eingesetzt. Wie man in Abbildung 3.8 grob erkennen kann, wird zunächst das RDF-Schema angezeigt, aus der ein Laie eine Klasse wählen kann. Dadurch wird eine SPARQL-Anfrage generiert, in die der Nutzer noch das Label einer gewünschten Ressource einsetzen muss. Alternativ kann auch eine eigene Anfrage verwendet werden.

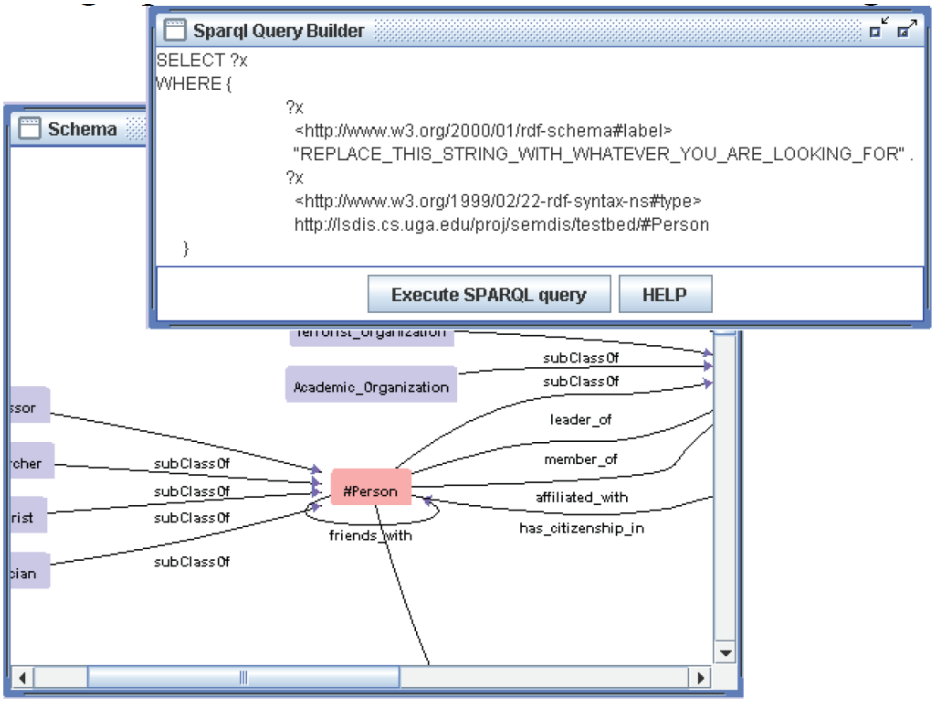


Abbildung 3.8.: Die Visualisierung des Starters in PGV [DKS07].

Von diesem Startpunkt ausgehend startet der Visualizer die Darstellung des Graphen, der erforscht werden kann. Sie unterscheiden Start-Knoten, direkte Nachbarn, Literale und schon untersuchte Knoten farblich voneinander. Der Startknoten wird mittig angezeigt, umrandet von allen Ressourcen, zu denen die Start-Ressource über Prädikate in Beziehung steht. Um weitere Nachbarknoten anzeigen

zu lassen, kann wie in Abbildung 3.9 zu sehen, ein Nachbarknoten gegriffen und im Kreis bewegt werden. Durch diesen „Ferris-Wheel“-Effekt werden dann weitere Nachbarknoten angezeigt. Der Nutzer kann außerdem Nachbarknoten anwählen um sie vergrößert darzustellen. Dann wird dieser als untersuchter Knoten an der Seite dargestellt und dessen Nachbarknoten angezeigt. Außerdem werden Kanten zwischen schon besuchten Knoten rot gefärbt.

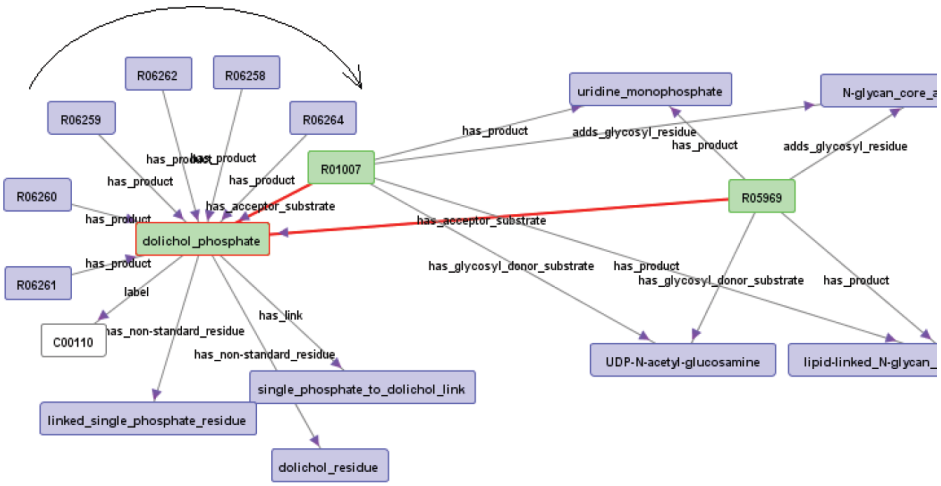


Abbildung 3.9.: Die Visualisierung des Visualizers in PGV [DKS07].

3.3. Stichwortsuche für RDF-Graphen

Die Menge von RDF-Datensätzen wächst schnell und die Suche in den RDF-Daten wird dadurch immer wichtiger. SPARQL ist dabei gut geeignet um RDF-Daten effizient zu durchsuchen, aber Nutzer müssen dafür sowohl das zugrundeliegende RDF-Schema, als auch die Syntax von SPARQL-Anfragen gut kennen. Wobei beides sehr komplex ist und deswegen nicht für Anfänger geeignet. Für die Suche von einzelnen Wörtern in den Daten bietet sich die Stichwortsuche dabei eher an als SPARQL-Anfragen zu schreiben, denn sie ist einfacher zu benutzen und schneller zu lernen.

Für solche Stichwortsuchen auf RDF-Graphen herkömmliche Graphen-Algorithmen für gerichtete, beschriftete Graphen zu verwenden ist aus mehreren Gründen nicht zielführend:

- Es sind in RDF-Graphen nicht nur Knoten, sondern auch Kanten beschriftet. Außerdem unterstützt das Format, das die RDF-Syntax für Prädikate definiert, die herkömmliche Stichwortsuche nicht ohne weiteres, sondern kann Probleme bereiten.
- Die Literale bestehen in herkömmlichen Graphen aus kurzen Labels. Bei RDF-Graphen können es jedoch ganze Sätze sein. Aus den vielen Wörtern würden herkömmliche Algorithmen jeweils einzelne Blatt-Knoten erzeugen, was nur einen Teil des eigentlichen RDF-Literals als Ergebnis zurückgibt.

- Semantische Daten sind deswegen so nützlich, weil sie semantische Zusammenhänge zwischen den einzelnen Ressourcen beinhalten. Dies wird in normalen Graph-Algorithmen überhaupt nicht bedacht und ausgenutzt. Ohne das Verwenden der Beziehungen zwischen den Ressourcen entsteht eine sehr große Ergebnismenge mit zum Teil mehrdeutigen Lösungen. Das Miteinbeziehen von Prädikaten und Beziehungen im Suchalgorithmus würde dem Nutzer viele Vorteile verschaffen. Zum Beispiel kann das Wort „Apple“ im Englischen für verschiedene Dinge stehen, wobei die Kombination aus „Apple“ und „hasCEO“ eindeutig angibt, dass es sich bei „Apple“ um eine Firma handelt.

Aus diesen Gründen haben Jiang et al. [JCCD11] einen Algorithmus entworfen, der Prädikat-Vorschläge bei der Stichwortsuche in semantischen Daten integriert. Sie sehen außer den Nachteilen, die herkömmliche Verfahren mit sich bringen weitere Vorteile ihres Konzeptes:

- Nutzer, die Suchmaschinen verwenden kennen im Normalfall die Semantik, die hinter den von ihnen gesuchten Wörtern steckt. Jedoch haben sie keine Ahnung wie dies in RDF-Syntax umzusetzen ist und somit auch nicht welche Wörter sie in die Suchmaschine eingeben müssen.
- Durch die Eingabe von mehreren Suchbegriffen kann es passieren, dass sie Ergebnisse erzeugen, die gar nicht im Zusammenhang mit der Ressource stehen, die sie eigentlich gesucht haben. So können falsch negative Ergebnisse von vorneherein ausgeschlossen werden.
- Die Nähe der zu den Stichwörtern gehörenden Ressourcen im Graphen stellt gleichzeitig ein Maß für den semantischen Zusammenhang dar. Je näher sich zwei Knoten stehen, desto direkter ist ihre Beziehung. Deswegen kann man den Algorithmus durch diese Bedingung erweitern.
- Man kann die Menge der Knoten, die ein Ergebnis für eine Suchanfrage sein können auf die Literale im RDF-Graphen beschränken, da Jiang et al. davon ausgehen, dass Nutzer nicht nach URIs suchen.

3.3.1. Ablauf

Eine Suche mit dem Konzept von Jiang et al. läuft in folgenden Schritten ab (vgl. Abbildung 3.10):

1. Der Nutzer gibt eine Menge von Suchwörtern an.
2. Der Algorithmus zur Generierung von Prädikatvorschlägen wird gestartet.
   a) Es wird nach Übereinstimmungen der Suchwörtern mit Literalen im Graphen gesucht.
   b) Die damit verbundenen Prädikate werden nach vermuteter Relevanz sortiert.
   c) Dem Nutzer wird eine Liste mit möglichen Prädikaten für jedes der Suchwörter vorgeschlagen.
3. Aus der Liste mit Vorschlägen kann der Nutzer für eines der Suchwörter ein semantisch zugehöriges Prädikat auswählen.

4. Der von Jiang et al. entworfene Algorithmus zur Generierung von Vorschlägen wird wieder gestartet. Die Listen von Prädikaten für die Suchwörter, denen der Nutzer noch kein Prädikat zugeordnet hat, werden gegebenenfalls aktualisiert.
5. Wiederhole Schritte 4 und 5, bis allen Suchwörtern ein Prädikat zugeordnet wurde oder der Nutzer mit der eigentlichen Stichwortsuche fortfahren möchte.
6. Der Nutzer erhält als Ergebnis der Suche eine geordnete Liste der verbundenen Subgraphen, die alle Suchwörter enthalten.

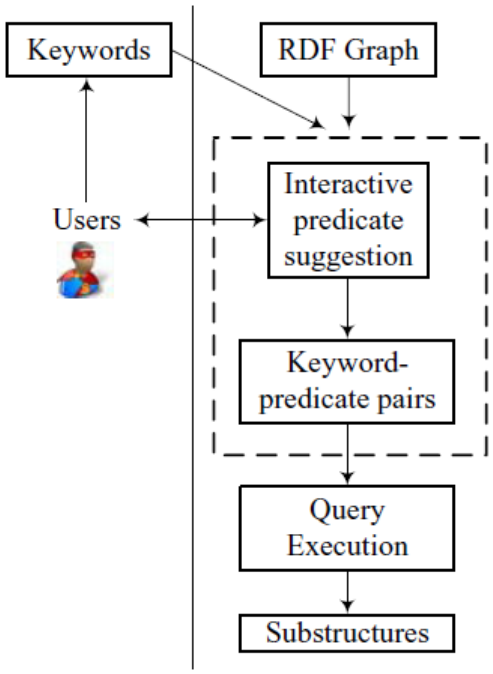


Abbildung 3.10.: Der Prozess der Stichwortsuche von Jiang et al. [JCCD11]

3.3.2. Algorithmus für Prädikat-Vorschläge

Das Hauptaugenmerk der Arbeit von Jiang et al. liegt jedoch nicht auf der Interaktion mit dem Nutzer, sondern dem Algorithmus für die Findung von passenden Prädikaten-Vorschlägen.

Sie bauten sich aus den RDF-Daten für den späteren Algorithmus Baumstrukturen. Gewöhnliche Suchen nach Stichwörtern für Graphen arbeiten mit Subbäumen, deren Blätter den Knoten entsprechen, die mit den Literal-Werten beschriftet sind. Diese werden hauptsächlich bei XML und relationalen Datenbanken verwendet, die auf gerichteten Graphen arbeiten. Für die Anwendung auf RDF-Graphen benötigt man die Richtung der Kanten jedoch nicht. Dies kann haben Jiang et al. an einem Beispiel illustriert: Anstatt „Alice, eats, Apple“ anzugeben, kann man auch „Apple, eatenBy, Alice“ als Tripel

verwenden. Also ist es für diese Aufgabe nicht schlimm die Tripel mit ungerichteten Kanten zu modellieren. In ihren ungerichteten Subbäumen verwenden sie zwei Arten von Knoten:

1. Stichwortknoten: Sie sind mit dem Wert der Literale beschriftet. In ihnen wird nach den Suchwörtern, die der Nutzer angegeben hat, gesucht.
2. Spannknoten: Stellen URIs dar und verknüpfen die Stichwortknoten.

Die Schwierigkeit an der Suche nach Prädikaten für die Suchwörter liegt nicht darin, die Suchwörter in den Subbäumen zu finden, sondern die relevanten Prädikate an den Nutzer weiterzugeben. Dafür verwenden sie drei Kriterien:

1. Die Ähnlichkeit zwischen den Suchwörtern und dem Literal, mit dem ein Stichwortknoten beschriftet ist.
2. Relevanz des Wurzelknotens des Subbaumes, in dem der Stichwortknoten liegt.
3. Die Entfernung vom Wurzelknoten des Subbaumes bis zum Stichwortknoten.

Um die Ähnlichkeit der Suchwörtern mit der Beschriftung der Stichwortknoten und die Entfernung von Knoten gleichzeitig in das Ranking der Ergebnisse einfließen zu lassen, verwenden sie einen match propagation and aggregation Algorithmus. Dieser ordnet zuerst den Knoten, die schon mit einem Prädikat vom Nutzer verknüpft worden sind, einen Wert für die Ähnlichkeit zu. Diese Knoten bezeichnen Jiang et al. als Verbreiter- (Propagator-) Knoten. Die anderen Stichwortknoten, die noch nicht mit einem Prädikat verknüpft sind, nennen sie Sammel- (Aggregator-) Knoten. Für die einzelnen Prädikate der Sammel-Knoten wird nun rekursiv ein Relevanz-Wert berechnet, der umso höher ist, je geringer der Abstand zu den Verbreiter-Knoten ist.

Da dieser Algorithmus für große Graphen und viele Suchwörter sehr zeitaufwändig werden kann, haben Jiang et al. ihn modifiziert und zu einem prioritized propagation and aggregation Algorithmus verändert. Dieser geht in der Rekursion nicht mehr alle möglichen Wege ab, sondern berechnet mittels einer stückweise linearen Funktion die wichtigen Wege.

Dieser Algorithmus wurde zusammen mit herkömmlichen Graphen-Such-Algorithmen implementiert und auf der RDF-Datenbank YAGO getestet, die zu dem damaligen Zeitpunkt einen Graph mit 10,796,073 Knoten und 92 verschiedenen Prädikaten gebildet hat. Sowohl im offline, als auch im online Modus wurde der neu entwickelte Algorithmus mit zufriedenstellender Performance getestet.

3.4. Zusammenfassung und Bewertung

Wie die in Unterkapitel 3.1 vorgestellten Modelle zur visuell unterstützten Filterung von Daten zeigen, ist es sinnvoll den Nutzer durch visuelle Elemente beim Erstellen von Anfragen an verschiedenste Datenbanken zu unterstützen [YS98], [RSBS08], [HSMT06], [MPG98]. Dadurch können Laien ohne Kenntnis der Syntax der Anfragesprachen Anfragen an Daten stellen. Allerings zeigen einige dieser Arbeiten auch auf, dass es noch nötig ist die Auswahl der Elemente, die für diese Anfragen verwendet werden, auch zu visualisieren. Denn wenn es eine Auswahl von Datenelementen gab, war sie in Form von Listen in die Visualisierung eingebunden [YS98], [RSBS08]. Russell et al. stellten außerdem fest,

dass es für die Arbeit mit RDF-Daten sinnvoll ist, dem Nutzer einen Überblick über den RDF-Graphen zu bieten, damit er Anfragen erstellen kann [RSBS08].

Die Visualisierung von RDF-Daten wurde im Unterkapitel 3.2 vorgestellt. Sowohl der Tabulator [BLCC+06], als auch die Paged Graph Visualization [DKS07] sind dafür geeignet, einen unbekannten RDF-Graphen zu erforschen. Allerdings ist es schlecht möglich, spezifische Ressourcen im Graphen zu finden um diese weiterzuverwenden. Um eine Ressource im Tabulator zu finden, muss der Nutzer diese Ressource kennen und wissen, wie sie im großen Zusammenhang des RDF-Graphen steht. Bei PGV ist dies schon etwas einfacher, allerdings muss der Nutzer dafür entweder deren Klasse im zugrundeliegenden RDF-Schema kennen, oder SPARQL-Anfragen erstellen können, da sich nur im Starter die Möglichkeit bietet zu suchen.

Schließlich wurde die von Jiang et al. entwickelte Stichwortsuche in RDF-Graphen vorgestellt [JCCD11]. Diese ist dafür geeignet eine durch die Semantik der Daten unterstützte Suche im Graphen durchzuführen. Allerdings ist es nicht möglich, davon ausgehend den Graphen weiter zu untersuchen, oder einen Überblick über die Daten zu erlangen.

Für eine Erweiterung der SAPRQL-Visualisierungen ist es also nötig ein neues Konzept zu erstellen, das eine Visualisierung des Graphen und eine Stichwortsuche in den RDF-Daten kombiniert. Außerdem sollte es dem Nutzer möglich sein, den Graphen selbstständig zu erforschen, um ein besseres Gefühl über die Beziehungen der Ressourcen in den Daten zu bekommen und es sollte die Möglichkeit bieten andere Elemente als die gesuchten auszuwählen.

4. Konzept

Dieses Kapitel beschreibt ein Konzept einer Visualisierung zur einfachen Auswahl von Ressourcen in RDF-Graphen. Es soll möglich sein, schnell einen Überblick über das dahinterstehende Schema und die Daten im Datensatz zu gewinnen. Dabei ist zu beachten, dass die Daten beliebig groß sein können. Die Visualisierung soll vor allem für unerfahrene Nutzer intuitiv verwendbar sein, damit sie mit RDF-Daten arbeiten können, ohne die RDF-Syntax oder Schreibweise der Label und URIs zu kennen. Aber auch erfahrene Nutzer sollen mithilfe dieser Visualisierung schnell und einfach Ressourcen finden und auswählen können.

4.1. Aspekte des Konzepts

Schon beim ersten Anzeigen der Visualisierung soll dem Nutzer ein Überblick über die Daten angeboten werden. Dadurch können sowohl die Art der Daten im Datensatz, als auch deren Beziehungen veranschaulicht werden. Russell et al. fanden heraus, dass es sehr wichtig ist, eine Übersicht über die Daten und das Schema zu bekommen, bevor man mit RDF-Daten arbeiten kann [RSBS08].

Bei der Darstellung der Übersicht muss berücksichtigt werden, dass die Datenmenge hinter der Visualisierung sehr groß sein können. Für die Darstellung gibt es schon einige Methoden, die mehr oder weniger aufwändig oder nützlich sind. Gemeinsam haben sie, dass nicht alle Daten auf einmal dargestellt werden, sondern nur Ausschnitte, damit der Nutzer in der Lage ist die Informationen zu verarbeiten [DKS07], [HMM00].

Eine Möglichkeit sind hyperbolische Ansichten des Graphen. In Abbildung 4.1 ist solch ein Baum hyperbolisch in 3D dargestellt. Diese Art der Visualisierung entspricht dem Fischaugeneffekt im dreidimensionalen Raum [HMM00]. Dies ist sehr nützlich, um die gesamten Zusammenhänge bei Bäumen zu erkennen. Allerdings ist es nicht möglich, Label zu lesen oder Semantiken anhand der Graphik abzulesen. Außerdem sind RDF-Graphen keine Bäume und besitzen deshalb viel mehr Kanten. Dies würde in einer hyperbolischen Darstellung unübersichtlich werden.

Eine andere Möglichkeit ist die Zusammenfassung der Elemente des Graphen in Cluster. Diese können dann weiter hierarchisch in Clustern zusammengefasst werden. Abello et al. zum Beispiel sind in der Lage hierarchische Cluster auf einem Graphen bis 200 000 Knoten zu berechnen. Sie möchten in weiteren Arbeiten auch Labels zu den Clustern automatisch berechnen können [AHK06].

Eine Implementierung eines solchen Algorithmus für das automatische hierarchische Clustering von semantischen Daten ist zu aufwändig für diese Arbeit. Deswegen wird der Überblick mithilfe der am häufigsten verwendeten Klassen dargestellt. Da diese die meisten Objekte des Graphen beinhalten wird dies als ein ausreichendes Clustering angenommen.

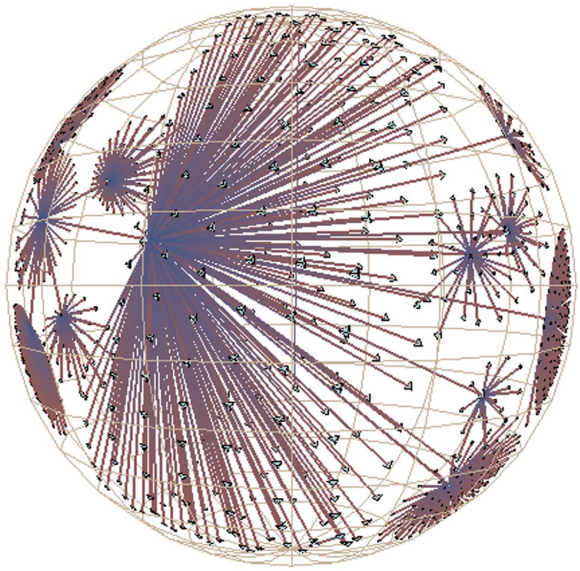


Abbildung 4.1.: Hyperbolische Ansicht eines Baums in 3D [HMM00]

Des Weiteren soll eine Stichwortsuche, wie sie Jiang et al. entwickelten [JCCD11], integriert werden. Sie soll dabei helfen, Schreibfehler zu verhindern und einen direkten Einstieg in die Daten zu ermöglichen. Vor allem für erfahrene Nutzer ist dies wichtig, damit sie nicht lange nach der gewünschten Ressource suchen müssen, sondern sie gleich als Ergebnis durch die Stichwortsuche erhalten können. Als Ergebnis der Suche soll die Semantik der Ergebnisse mit angezeigt werden, um Mehrdeutigkeiten gleich zu Beginn aufzulösen. Zusätzlich zu der von Jiang et al. entwickelten Stichwortsuche wäre es möglich, gleich bei Eingabe der Wörter nach ähnlichen Wörtern oder Phrasen zu suchen, um einen Vorschlag zu Vervollständigung anzubieten, oder semantisch ähnliche Begriffe vorzuschlagen [CZX11]

Zusätzlich zur Klassenübersicht, können dann auch vergangene Suchanfragen in die Anfangsdarstellung integriert werden. Dafür kommen mehrere Historien in Frage [CZX11]:

- die letzten Suchanfragen des Nutzers
- die letzten Suchanfragen aller Nutzer
- die letzten gewählten Elemente des Nutzers
- die letzten gewählten Elemente aller Nutzer

Somit soll es möglich sein, schnell kürzlich verwendete Ressourcen wieder auszuwählen ohne sie explizit suchen zu müssen.

Um dem Nutzer schließlich ein besseres Gefühl für die Nutzung des Prototypen zu geben, soll es möglich sein, mit der Visualisierung zu interagieren und so die Daten zu erforschen [Shn94].

4.2. Visuelle Darstellung

Russell et al. haben mit vSPARQL [RSBS08] gezeigt, dass es sich anbietet bei der Visualisierung von RDF-Daten auf die Struktur dieser Daten – einen Graphen mit Kanten und Knoten – zurückzugreifen.

Dabei soll das Prinzip der inkrementellen Entdeckung angewendet werden. Dies bedeutet, dass zunächst ein Subgraph der Daten angezeigt wird, der dann durch den Nutzer in die gewünschte Richtung interaktiv erweitert werden kann [HMM00], [DKS07].

4.2.1. Darstellung des Graphen

Russell et al. visualisierten bei vSPARQL beim Verwenden derselben Ressource in verschiedenen Tripeln die Ressource doppelt, als zwei verschiedene Knoten. Bei dem Konzept, das im Rahmen dieser Arbeit entwickelt wurde, steht es jedoch im Vordergrund die Zusammenhänge im RDF-Graphen aufzuzeigen, weshalb jede Ressource nur einmal als Knoten auftauchen soll. Außerdem kann es vorkommen, dass die URI des Prädikats des einen Tripels in einem anderen Tripel als Subjekt oder Objekt auftreten kann. Deshalb werden, anders als bei Deligiannidis et al. [DKS07], in diesem Konzept nicht nur Subjekt und Objekt als Knoten modelliert, sondern alle Ressourcen. Wie in Abbildung 4.2 zu sehen, bleibt die Richtung der Kante aus dem Verständnis von RDF-Tripeln erhalten [DKS07], sie ist jedoch nicht länger mit dem Prädikat beschriftet, da diese Information in einem zwischengeschalteten Knoten dargestellt wird.

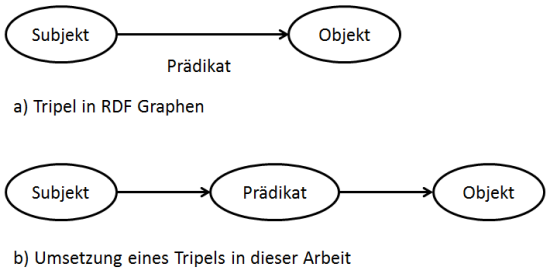


Abbildung 4.2.: Vergleich von Tripeln im Konzept von RDF-Graphen [W3C14a] und der Visualisierung dieser Arbeit.

Deligiannidis et al. verwendeten in ihrer Visualisierung optisch verschiedene Knotentypen [DKS07]. Auch hier sollen die Kanten verschiedene Knotentypen verbinden, deren Übersicht in Abbildung 4.3 zu sehen ist. Die in der Abbildung verwendeten Formen und Farben sind nur ein Beispiel. Es sollte jedoch visuell zu erkennen sein, dass es sich bei den Knoten um vier verschiedene Arten handelt. Die Richtung der Kanten, die die Knoten verbinden, gibt dabei die Stellungen der Ressourcen im Tripel an, vergleiche Abbildung 4.2. Durch die verschiedenen Knotentypen soll zwischen verschiedenartigen Ressourcen unterschieden werden. Literal-Knoten entsprechen den Literal-Ressourcen des RDF-Graphen, damit Werte auf den ersten Blick zu erkennen sind [DKS07]. Da das RDF-Schema

nicht aufgezeigt wird [DKS07], aber trotzdem eine gewisse Struktur der Knoten aufgezeigt werden soll, werden Klassen-Knoten als eigene Art von Knoten eingeführt. Als Klassen werden dabei alle Ressourcen bezeichnet, zu denen es eine „typeof"-Beziehung gibt. Damit ist es möglich, die Struktur, die Russell et al. im Ontology-Browser ihres Nitelight Prototypen verwenden, nachzubauen [RSBS08]. Außerdem wird eine Differenzierung von verschiedenen Klassen-Knoten aufgrund des Vorkommens definiert. Die Häufigkeit der Verwendung einer Klasse wird mit der Häufigkeit der Verwendung aller anderen Klassen in Relation gesetzt und dementsprechend größer oder kleiner dargestellt. Die restlichen Ressourcen werden schließlich durch Objekt-Knoten visualisiert.

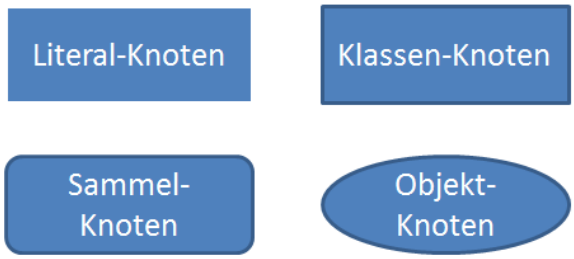


Abbildung 4.3.: Übersicht über die verschiedenen verwendeten Knotentypen

Bei der Initiierung des Subgraphen, der inkrementell entdeckt und erweitert werden soll, steht eine Ressource im Mittelpunkt. Sie wird dargestellt durch einen der drei schon beschriebenen Knoten-Typen – Literal-, Klassen- oder Objekt-Knoten. Davon ausgehend sollen nun alle Beziehungen angezeigt werden, die diese Ressource zu anderen im RDF-Graphen hat. Dies können bei einem großen Graphen viele angrenzende Knoten und Kanten sein, die dargestellt werden müssten. Da die Visualisierung übersichtlich gehalten werden soll [HMM00], wird diese nun, anders als bei Deligiannidis et al. [DKS07], vereinfacht. Ein Vorschlag ist, alle ein- und ausgehenden Kanten zur selben Ressource zusammenzufassen. Diese Ressource wird deswegen durch einen Aggregat-Knoten dargestellt. Zum Anderen soll nur eine begrenzte Anzahl der Aggregat-Knoten angezeigt werden. Die Reihenfolge in der die Aggregat-Knoten angezeigt werden, wird im Unterkapitel 4.2.3 beschrieben. Es soll mittels Pfeilbuttons, Scrollbalken oder ähnlichem möglich sein, die Liste der Aggregat-Knoten zu durchsuchen [DKS07]. Um die Wirkung von Aggregat-Knoten zu zeigen, ist in Abbildung 4.4 ein Beispiel eines Subgraphen ohne Aggregat-Knoten dargestellt. Durch die vielen Knoten und Kanten, die für den Nutzer möglicherweise gar nicht von Interesse sind, wirkt der Graph recht unübersichtlich. In Abbildung 4.5 ist der selbe Ausgangsknoten mit Aggregat-Knoten dargestellt. Diese sind, wie der „type of"-Aggregat-Knoten, durch die andere Form besser von den Objekt-Knoten unterscheidbar. Hier ist es nun für den Nutzer möglich, den Graphen an den Aggregat-Knoten in gewünschter Richtung zu erweitern. So wird die Visualisierung nicht so unübersichtlich und es ist möglich den RDF-Subgraphen darzustellen und den gesamten RDF-Graphen zu erreichen, obwohl nicht alle Knoten auf einmal dargestellt werden.

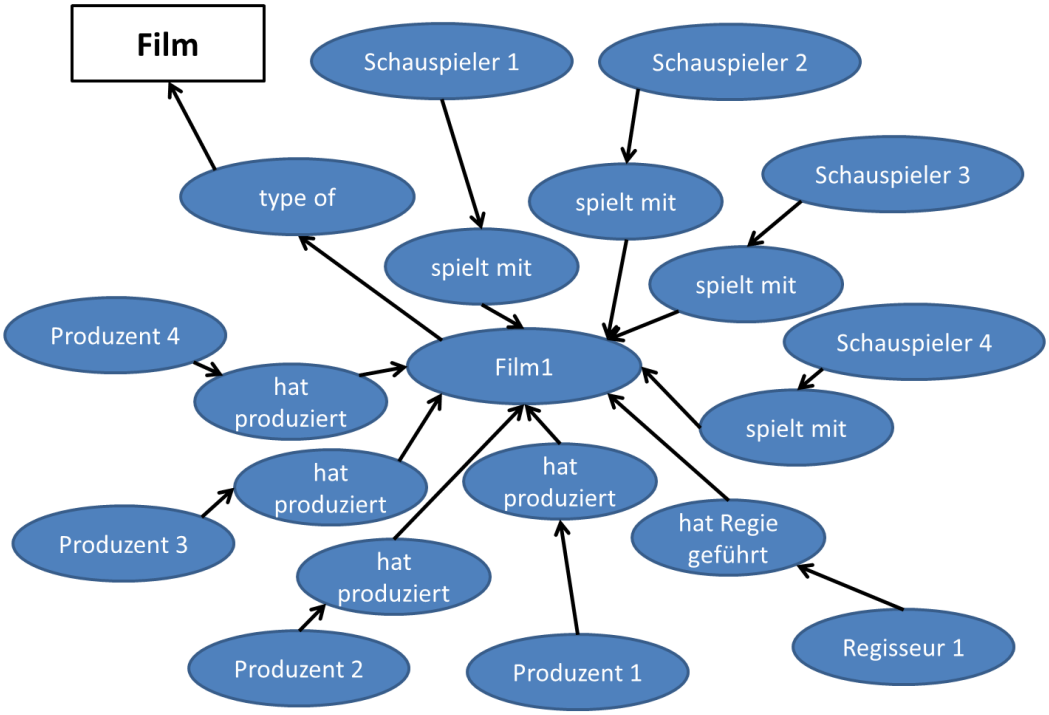


Abbildung 4.4.: Beispiel eines Subgraphen ohne Aggregatknoten.

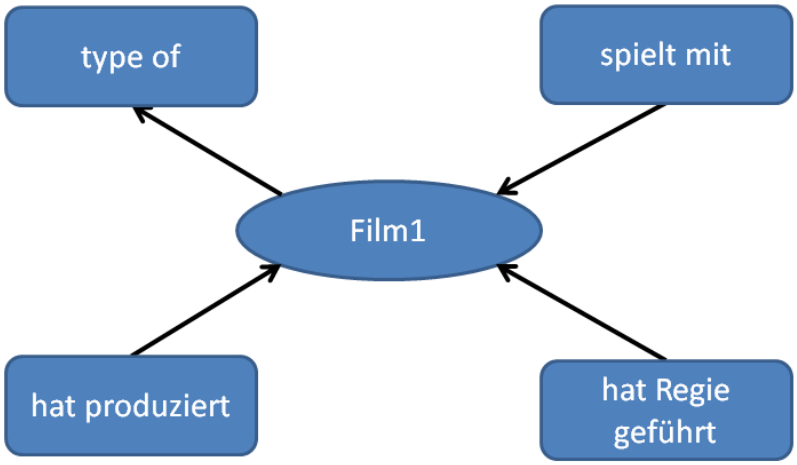


Abbildung 4.5.: Beispiel eines Subgraphen mit Aggregatknoten.

4.2.2. Navigation im Graphen

Um dem Nutzer die Möglichkeit zu bieten, den Subgraphen interaktiv zu erkunden, ist es möglich, ihn wie bei Deligiannidis et al. [DKS07] an den gewünschten Stellen zu erweitern. Wenn man einen Aggregat-Knoten anwählt, wird er als Klassen- oder Objekt-Knoten zum neuen Mittelpunkt des Fensters. Dabei soll beachtet werden, dass der Graph weiterhin übersichtlich und lesbar bleibt. Graph Ästhetik Richtlinien schlagen vor [HMM00], [Pur02]:

- wenig Kantenüberschneidungen
- symmetrische Graphstruktur
- Kanten als gerade Linien
- kontinuierliche Wege durch Kanten
- Knoten und Kanten gleichmäßig verteilt
- Kanten mit gleicher Länge
- Nachbarknoten beieinander platzieren

Zusätzlich sollen die weiterführenden Sammelknoten angezeigt werden, damit der Nutzer dort den Graph weiter erweitern kann.

Da nur die Gesamtheit der Beziehungen erkennbar ist und nicht die Tripel soll es für jeden Klassen- und Objekt-Knoten einen Marker-Button geben. Dieser soll die Kanten der zugehörigen Tripel farbig hervorheben. Zunächst soll es nur möglich sein, einen Knoten gleichzeitig zu markieren.

4.2.3. Metriken

Um die Aggregat-Knoten anzuordnen können verschiedene Metriken eingesetzt werden. Diese Metriken lassen sich in zwei Klassen einteilen: statische und dynamische Metriken. Statische Metriken sind beispielsweise die Anordnung der Aggrgat-Knoten nach dem Alphabet oder die Anordnung nach ihrer Häufigkeit. Bei dynamischen Metriken kann sich im Gegensatz zu statischen die Reihung im Laufe der Zeit ändern. Eine mögliche dynamische Metrik wäre die Anordnung nach der Häufigkeit der Auswahl. Um diese Metrik umzusetzen, könnte beispielsweise eine History implementiert werden, die aufzeichnet welche Knoten durch den Benutzer ausgewählt wurden. Wenn diese History lokal mitloggt, kann sie dazu verwendet werden, den Benutzer, der seine Ontologie kennt, zu unterstützen. Eine globale History könnte dazu verwendet werden Benutzern, für die die Ontologie neu ist, zu zeigen, welche Bestandteile für die Mehrheit der Benutzer wichtig ist und dadurch die Einstiegshürde zu senken. Als Startwert für eine dynamische Metrik könnten statische Metriken eingesetzt werden, bis für die dynamischen Metriken genügend Daten vorliegen

4.3. Beispiel-Ablauf

Um die Navigation durch einen Subgraphen mit dem Prototypen zu veranschaulichen, wird sie im Folgenden anhand einer kleinen, fiktiven semantischen Datenbank erläutert. Diese enthält Daten zu ausgewählten Werken der Schriftsteller Wolfgang Hohlbein, Agatha Christie und Cecilia Ahern.

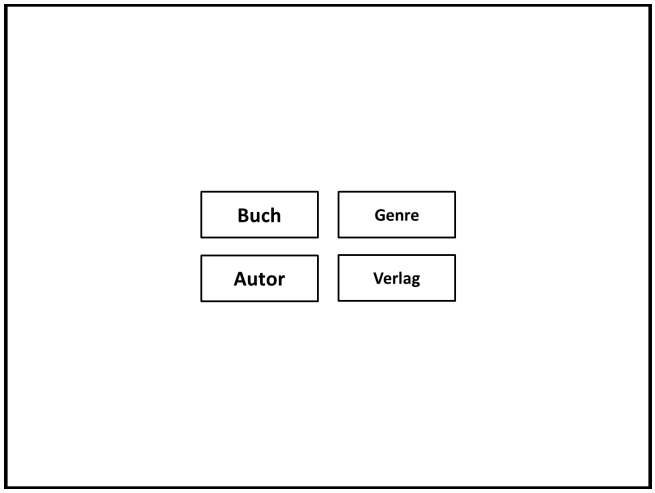
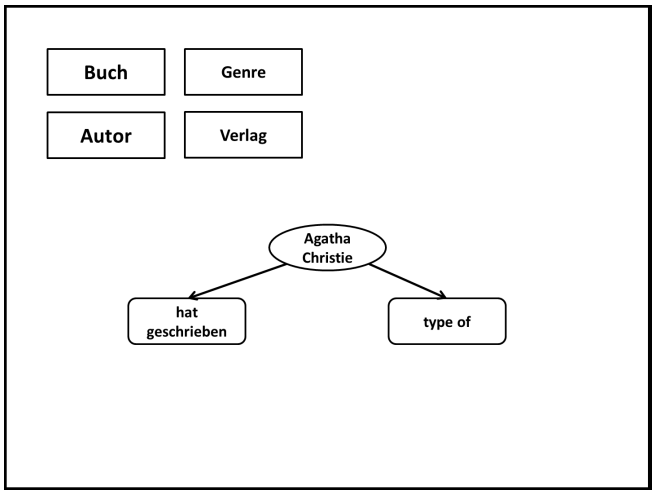


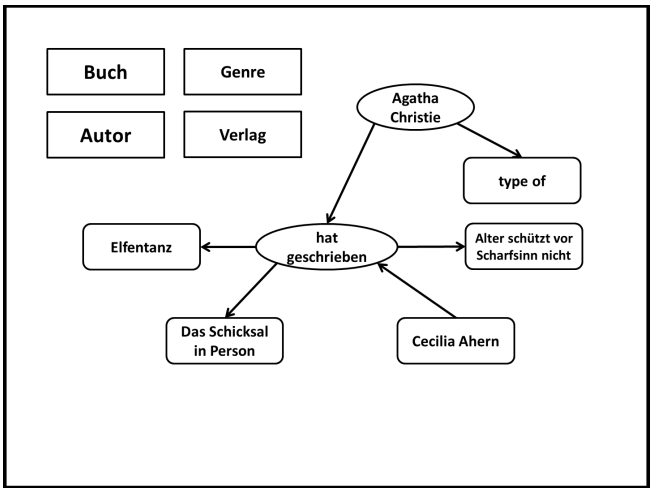
Abbildung 4.6 zeigt die Übersicht über die Daten, die zu Beginn dem Nutzer angezeigt wird. Darin werden die am häufigsten vorkommenden Klassen dargestellt. Sie sollen dem Nutzer helfen einen ersten Eindruck über die Daten zu erlangen.
In dieser Datenbank sind dies „Buch“, „Autor“, „Genre“ und „Verlag“. Die Schriftgrößen der Labels der Klassen-Knoten zeigen dem Nutzer, dass es mehr Buch- und Autor-Instanzen als Genre- oder Verlag-Instanzen gibt.

Abbildung 4.6.: Beispielhafte Übersicht über die Daten.



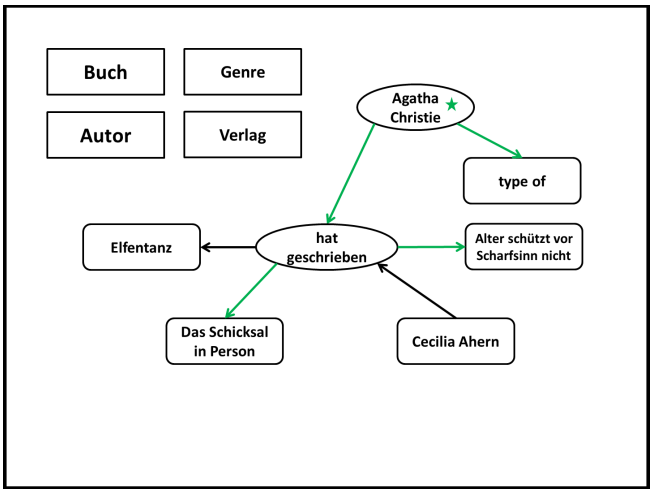
Über das Suchfeld hat der Nutzer die Ressource „Agatha Christie“ ausgewählt. Sie erscheint nun im Fokus des Fensters, die zuletzt gezeigten Knoten sind noch sichtbar, wie in Abbildung 4.7 zu sehen ist. Außerdem wurden alle Ressourcen, die über RDF-Tripel mit der gewählten Ressource in Zusammenhang stehen, berechnet und in Aggregat-Knoten zusammengefasst angezeigt. Dies ist zum einen „type of“, da in den RDF-Daten eine Klasse zugewiesen wurde und die Sammel-Ressource „hat geschrieben“ als Zeichen, dass es Tripel der Form „Agatha Christie hat geschrieben y“ gibt.

Abbildung 4.7.: Beispielhaftes Auswählen eines Elements.



Der Nutzer in diesem Beispiel ist daran interessiert, welche Bücher Agatha Christie geschrieben hat. Deswegen erweitert er den angezeigten Subgraphen in diese Richtung. Der Aggregat-Knoten wird geöffnet, wie in Abbildung 4.8 zu sehen. Da „hat geschrieben“ kein Literal ist und in den Daten keine „type of“-Beziehung existiert, was implizieren würde, dass diese Ressource eine Klasse ist, wird sie als Objekt-Knoten dargestellt. Um die Ressource herum wird der Anfang der Liste von Aggregat-Knoten angezeigt, die zusammen mit der Ressource mit dem Label „hat geschrieben“ in Verbindung stehen. Die Liste wurden beispielhaft in alphabetischer Reihenfolge sortiert. Es sind also sowohl Bücher, als auch Autoren, die als Aggregat-Knoten angezeigt werden.

Abbildung 4.8.: Beispielhaftes Erweitern des Subgraphen durch auflösen eines Aggregatknotens.



Um zu erkennen, welche der angezeigten Ressourcen nun in Verbindung zur ursprünglich gewählten Ressource mit dem Label „Agatha Christie“ stehen, hat der Nutzer diesen Knoten markiert, wie in Abbildung 4.9 zu erkennen ist. Nun werden die Tripel mit Hilfe von farbigen Kanten sichtbar und man kann jetzt erkennen, dass Agatha Christie unter anderem die Werke „Alter schützt vor Scharfsinn nicht“ und „Das Schicksal in Person“ geschrieben hat.

Abbildung 4.9.: Beispielhaftes Markieren einer Ressource.

5. Implementierung

In diesem Kapitel wird die Implementierung eines Protoypen vorgestellt, der das Konzept aus Kapitel 4 umsetzt. Zunächst wird das Framework beschrieben, mit dem er programmiert wurde. Anschließend wird ein Überblick über das Fenster der Anwendung gegeben und die einzelnen Elemente der Implementierung erläutert.

5.1. Windows Presentation Foundation

Windows Presentation Foundation – oder kurz WPF – ist ein Grafik Framework und Teil des .NET Frameworks von Microsoft [Mica]. Mit WPF kann man Anwendungen mit visuell ansprechenden Benutzeroberflächen erstellen. Durch den Kern von WPF, einem vektorbasierten Renderingmodul, können graphisch aufwändige Benutzeroberflächen schnell und unabhängig von der Bildschirmauflösung dargestellt werden. WPF wurde gewählt, weil es schon einige WPF-Prototypen im Institut für Visualisierung und Interaktive Systeme der Universität Stuttgart gibt, in die der hier entstehende Prototyp eingebunden werden soll.

Das Grundprinzip beim Erstellen von Anwendungen mit WPF ist die Trennung von Aussehen und Verhalten in der Entwicklung des Programmes. Mit der auf XML basierenden Extensible Application Markup Language (XAML) können die Fenster aus verschiedensten Elementen zusammengesetzt werden. Dies können eigene Steuerelemente oder Controls sein, oder Elemente aus Bibliotheken des .NET Frameworks. Im davon getrennten Code-Behind wird nur das Verhalten beschrieben, das beim Benutzen der Anwendung auftreten soll. Dieser kann in einer beliebigen Programmiersprache aus dem .NET Framework entwickelt werden, zum Beispiel C# oder Visual Basic. Durch Aufrufe von Methoden aus dem XAML Dokument, Events oder Datenbindungen können Verhalten und Aussehen des Programms verknüpft werden.

5.2. Übersicht über den Prototypen

In Abbildung 5.1 ist das Fenster des Prototypen zu sehen. Am oberen Rand befindet sich die Suchleiste, mit der Ressourcen gezielt gesucht werden sollen. Dazu gehören auch die Checkboxen zur Filterung der Suchergebnissen nach Klassen, Literalen und Prädikaten. Sie wird in Kapitel 5.4 beschrieben. Darunter befindet sich der große Canvas, in dem der Subgraph dargestellt wird, durch den der Nutzer die Daten genauer erforschen kann. Genaueres findet sich in Kapitel 5.3. Um die Darstellung des Graphen zurückzusetzen kann der „Refresh“- Button über dem Canvas verwendet werden. Der „Options“-Button öffnet ein weiteres Fenster, in dem Optionen wie der SPARQL-Endpoint eingestellt und geändert werden können.

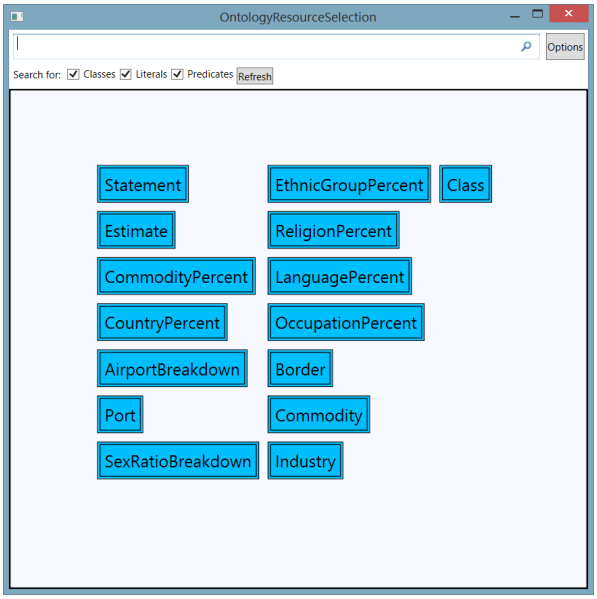


Abbildung 5.1.: Das Fenster des Prototypen zu Begin der Nutzung.

5.3. Canvas

Den größten Teil des Fensters nimmt der Canvas ein. Auf ihm wird der Subgraph mithilfe von VisUtils.NodeLink.Display eingezeichnet, damit der Nutzer ihn erforschen kann. Im Folgenden wird zunächst die Implementierung der Knoten, die Übersicht über die Ressourcen zu Beginn der Anwendung und anschließend die Verwendung von EventHandlern für die Navigation erläutert.

5.3.1. Knoten

Wie im Konzept definiert, wurden vier verschiedene Knotentypen implementiert: Literal-, Klassen-, Aggregat- und Objekt-Knoten. Dem Prinzip von WPF bezüglich Trennung von Verhalten und Darstellung der Elemente folgend, wurde die visuelle Darstellung der Knoten in einer XAML- und deren Funktionalität in einer C#-Datei implementiert.

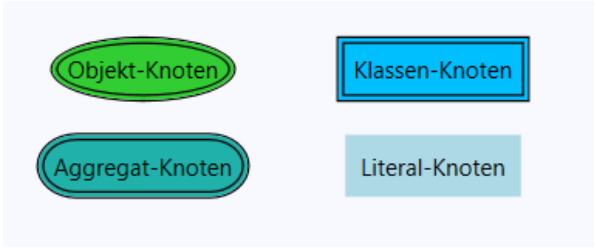


Abbildung 5.2.: Die Visualisierung der verschiedenen Knotentypen im Prototyp.

Zunächst wurde die Darstellung der minimierten Knoten in der XAML-Datei definiert. Anschließend wurden DataTrigger eingesetzt. Diese verändern abhängig von einem Attribut des jeweiligen Knotens dessen Aussehen. Abbildung 5.2 zeigt die vier verschiedenen Knotentypen im minimierten Zustand. Um die Farben später im Code-Behind ändern zu können, wurde Data Binding eingesetzt, das das Attribut im XAML-Dokument mit einem Attribut des C#-Codes verbindet.

Da es nicht möglich war, die Werte von ActualHeight und ActualWidth der Knoten per Data Binding mit dem Code-Behind zu verknüpfen, wurde ein zusätzliches Element verwendet: das SizeMeasurementContainer Control. Es wurde im XAML-Dokument um die Darstellung der Knoten gesetzt. Durch dessen Implementierung war es nun möglich, die Größe des Controls und damit auch des Knotens abzugreifen und mit dem C#-Code zu verknüpfen. Dies wurde unter anderem verwendet um die Knoten so einzeichnen zu können, dass sie sich nicht überlappen.

Im maximierten Knoten werden fünf Buttons sichtbar, wie in Abbildung 5.3 zu sehen:

- Blättern nach links
- Blättern nach rechts
- Sortieren alphabetisch
- Sortieren nach Häufigkeit
- Knoten markieren

Abbildung 5.3.: Ein maximierter Klassenknoten im Prototyp.

In WPF ist es nicht möglich in dynamischen XAML-Dokumenten Buttons direkt Methoden zuzuweisen. Deswegen wurde ein Umweg über ICommands gewählt. Diese sind eigentlich dafür gedacht, Methoden zu implementieren, die von verschiedensten Elementen verwendet werden können, wie zum Beispiel „ausschneiden“ oder „einsetzen“. Für ICommands müssen die Methoden CanExecute und Execute implementiert werden. Die erste gibt zurück, ob es zum jetzigen Zeitpunkt möglich ist, den Command auszuführen, die zweite enthält die Implementierung dessen, was passieren soll, wenn der Command ausgeführt wird.

Für jeden Button wurde ein eigener ICommand mit eigener Klasse implementiert. Deshalb wurde in der Definition der Knoten im XAML Dokument außer dem Command ein CommandParameter angegeben – den Button selbst. Dieser wird den Methoden von ICommand bei ihrem Aufruf übergeben.

Mithilfe der Methode FindVisualParent aus der VisUtils.WPF Bibliothek ist es kann möglich den Knoten selbst zu ermitteln und auf ihm Methoden auszuführen.

5.3.2. Übersicht durch initiale Knoten

Für die Übersicht über den RDF Graphen durch initiale Knoten wurden im Konzept die am häufigsten vorkommenden Klassen gewählt. Es wird angenommen, dass sie die wichtigsten Klassen des Datensatzes darstellen und somit einen guten Überblick über die Daten gewährleisten. Außerdem sollte die Möglichkeit angeboten werden, zuletzt gewählte Elemente ohne großen Suchaufwand wiederzuverwenden. Deshalb wurden auch diese gespeichert.

Da die Berechnung der am häufigsten vorkommenden Klassen zeitaufwändig ist und die Nutzer die zuletzt verwendeten Knoten eventuell auch beim erneuten Öffnen der Anwendung verwenden möchten, werden diese Elemente beim Beenden der Anwendung in ein XML-Dokument auf dem Rechner abgelegt.

Am Anfang des XML-Dokuments sind allgemeine Daten, sowie die Komponente gespeichert, die das Dokument erstellt hat. Danach steht ein XML-Element „endpoint“ in ihm ist sowohl die URL, als auch die Anzahl der Klassen-Ressourcen im Endpoint gespeichert. Beim Laden des Dokuments kann somit herausgefunden werden, ob Daten zum eingestellten Endpoint gespeichert wurden. Anschließend kann anhand der Berechnung der Anzahl der Klassen-Ressourcen festgestellt werden, ob sich der Endpoint so weit verändert hat, dass die am häufigsten vorkommenden Klassen und Quartile neu berechnet werden müssen.

Anschließend werden in jedem Endpoint zunächst die Werte der Quartile, die Liste der häufigsten Klassen und schließlich die zuletzt gewählten Knoten gespeichert. Dabei sieht ein XML-Element für einen Klassen-Knoten wie folgt aus:

<classNode
quartile=“1“ uri=“http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement“ label=“Statement“/>

Andere Knotenarten werden durch die Art des Elements gekennzeichnet. Für Objekt-Knoten wird nur die URI, bei Literalen nur der Wert gespeichert.

5.3.3. EventHandler für die Navigation

Um die Navigation zu implementieren wurden verschiedene EventHandler verwendet, die auf einfache Klicks, Doppelklicks und Bewegungen der Maus reagieren. Außerdem gab es EventHandler, die im NodeLinkDisplay schon definiert waren:

- Doppelklick auf einen Knoten: der Knoten wird größer oder kleiner.
- Klick auf einen Button innerhalb eines Knotens: der dazugehörige ICommand wird aufgerufen.
- Klick auf einen Punkt auf dem Canvas: speichert den Punkt, um später den Canvas zu verschieben.

- Die Bewegung der Maus bei gedrückter linker Maustaste: der Canvas wird verschoben.
- NodeLinkDisplay reagiert auf Mausklick: ein Knoten kann ausgewählt werden.
- NodeLinkDisplay reagiert auf Bewegung der Maus bei gedrückter linker Maustaste: wenn ein Knoten ausgewählt wurde, wird dieser bewegt.

Diese durch Mausklick getriggerten Events sind in WPF Routed Events 5.4. Das bedeutet, dass es für die verschiedenen Schichten in der Benutzeroberfläche EventHandler gibt, zu denen man weitere hinzufügen kann. Das Event wird dann durch die verschiedenen Schichten und Handler geleitet und alle ausgeführt. Es gibt die Möglichkeit durch ein Attribut das Event auf „bearbeitet“ zu setzen, sodass das Event zwar zu den weiteren EventHandlern geleitet, aber nicht mehr verarbeitet wird.

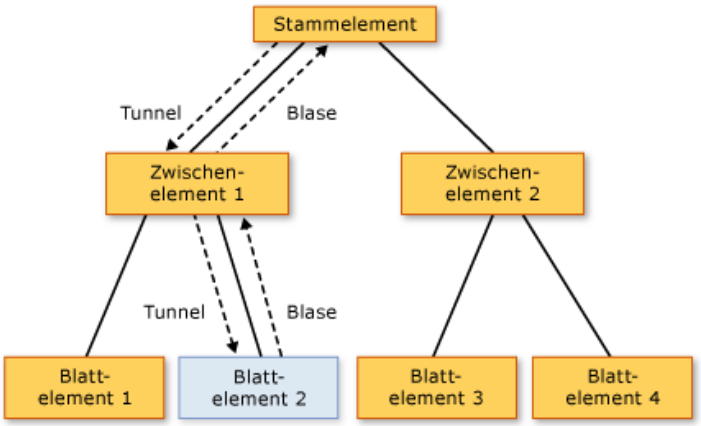


Abbildung 5.4.: Ereignisverarbeitung bei Routed Events in WPF [Micb]

In Abbildung 5.4 ist ein beispielhafter abstrakter Aufbau einer Benutzeroberfläche aufgezeigt. Das Stammelement stellt das Fenster als solches dar, das Zwischenelement zum Beispiel ein Canvas, das Blattelement zum Beispiel einen Button. Zunächst wird vom Stammelement aus ein Event Richtung Blattelement geroutet, anschließend wieder vom Blattelement zum Stammelement. Diese Vorgänge nennt man in WPF Tunneling und Bubbling. Es werden also die Eventhandler in folgender Reihenfolge angesprochen:

1. PreviewMouseDown auf dem Stammelement
2. PreviewMouseDown auf dem Zwischenelement 1
3. PreviewMouseDown auf dem Blattelement 2
4. MouseDown auf dem Blattelement 2
5. MouseDown auf dem Zwischenelement 1
6. MouseDown auf dem Stammelement

Um also das nicht gewünschte Verschieben von einzelnen Knoten und weitere ungewünschte Konflikte zu verhindern, wurde der allgemeine Klick auf den Canvas als MouseDown Eventhandler und der Eventhandler für die Bewegung der Maus bei gedrückter Taste als PreviewMouseMove Eventhandler implementiert. Außerdem musste in Letztem geprüft werden, ob auf einen Button geklickt wurde, da sonst bei jedem Klick auf einen Button der Canvas verschoben wurde.

In Abbildung 5.5 ist der Protoyp zu sehen, der durch das Öffnen der „type“ Ressource entstand. Als Endpoint wurde zu diesem Zeitpunkt das CIA World FactBook [Dea03] verwendet. Die türkisen Knoten, die im Kreis angeordnet sind, sind wie aus Abbildung 5.2 abgeleitet werden kann, Aggregat-Knoten. Wenn der Nutzer den Graphen erweitern möchte, kann er diese Knoten doppelklicken, damit er sich öffnet.

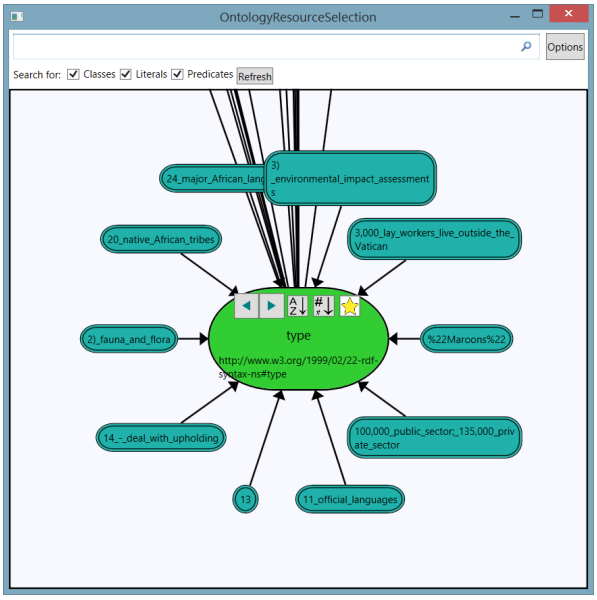


Abbildung 5.5.: Der Protoyp während der Nutzung.

5.4. Suchleiste

Die Suchleiste soll mehrere Aufgaben erfüllen. Zum einen muss ein Nutzer Text eingeben können, nach dem gesucht werden soll. Außerdem muss sie Möglichkeiten für das Starten und den Abbruch der Suche bieten und schließlich auch eine Möglichkeit die Suchergebnisse an den Nutzer zurückzugeben. Da kein bereits existierendes WPF Steuerelement die benötigte Kombination aus Textfeld, Button, Events und Ergebnisliste zur Verfügung stellt, wurde ein eigenes Control implementiert, das in 5.4.1 vorgestellt wird. Darüber hinaus wurden Methoden entwickelt, die nach Start der Suche die RDF Daten nach passenden Ressourcen durchsuchen. Deren Implementierungen werden in 5.4.2 aufgezeigt.

5.4.1. Das Control *Suchleiste*

Ausgangspunkt für das in dieser Arbeit implementierte Control stellt die „WPF Search Text Box“ von David Owens II [Owe09] dar. Er entwickelte ein Textfeld mit Button für das Abbrechen und Ausführen der Suche. Außerdem integrierte er ein Such-Event, das entweder beim Betätigen der Return-Taste oder durch eine Pause des Nutzers beim Eintippen des Suchbegriffs gestartet wurde. Zudem fügte er Attribute dem Suchfeld hinzu, mit denen andere Entwickler, die das Suchefeld integrieren, Farben und Schriftarten einstellen können.

Le Duc Anh entwickelte in [Anh10] eine Erweiterung des Suchfeldes von Owens. Er fügte mehr Buttons hinzu, die ein Popup unter dem Textfeld öffnen, das je nach Button eine Liste von vordefinierten Elementen anzeigt. Durch die Auswahl eines der Elemente wird entweder ein Attribut des Suchfeldes oder der Text des Textfeldes geändert.

Da keines der eben erwähnten Entwicklungen die Anforderungen an ein Suchfeld für diesen Prototypen ganz erfüllt, wurde auf ihrer Grundlage ein neues entwickelt.

Nun besteht die Suchleiste aus einem Textfeld und einem Button. Die Suche kann über das Betätigen der Return-Taste oder des integrierten Buttons gestartet werden. Dadurch wird ein „Search“-Event getriggert, dem beim Einbinden des Suchfeldes in eine XAML Datei eine Methode übergeben kann, die durch das Anstoßen des Events ausgeführt wird. Außerdem können in einer ObservableCollection die Suchergebnisse zurückgespeichert werden. Nach Ausführen der angegebenen Methode wird das Popup geöffnet, in dem die Ergebnisse der Suche zu sehen und auswählbar sind. Durch das Auswählen eines Elements wird außerdem ein weiteres Event getriggert, für das beim Einbinden des Controls auch eine Methode zugewiesen werden kann. Für eine bessere Verwendbarkeit wurden verschiedene Ergänzungen eingefügt, die das Popup verstecken, auf Tastendrücke reagieren etc.

5.4.2. Suche in RDF Daten

Zur Suche in RDF Daten wurde eine Methode implementiert, die beim Eintreten eines „Search“-Events aufgerufen wird. Sie sucht mittels SPARQL Anfragen nach Subjekten, Prädikaten und Objekten, die den Suchbegriff im Namen tragen, der zugehörigen Klasse und Label. Aus diesen Informationen wird pro gefundener Ressource ein Ergebnisstring der Form <Label>|<letzterTeilDerUri> ’(’<Klasse>’)’ gebildet. Diese werden in verschiedene Listen für Klassen, Prädikate, Literale uns sonstige Ergebnisse gespeichert. Dadurch können durch Checkboxen aktivierte Filter für eben jene Arten von Objekten – Klassen, Prädikate und Literale – schneller die Ergebnismenge ändern. Anschließend werden je nach aktivierten Checkboxen die verschiedenen Listen vereint oder Teile daraus entfernt und die Ergebnisse sortiert. Als Sortier-Kriterium wurde die Stelle, an dem der Suchstring auftritt, gewählt. Die dann verbleibenden Ergebnisstrings werden in die ObservableCollection Variable der Suchleiste gespeichert. Dadurch wird das Popup geöffnet.

5.5. Probleme bei der Implementierung

Während der Implementierung des Prototypen traten einige Probleme in Verbindung mit WPF und verwendeten Assemblies auf:

„%“ in SPARQL Anfragen Für SPARQL-Anfragen wurde die Assembly dotNetRDF verwendet. Das hat im Allgemeinen auch gut funktioniert, allerdings werden Exceptions geworfen, wenn man Anfragen mit dem Zeichen „%“ darin abschicken möchte. Auch im Update auf die neueste Version war dieser Fehler nicht behoben.

VisUtils.NodeLink.Graph Zu Beginnn dieser Arbeit gab es vermehrt Probleme mit der verwendeten Komponente Graph aus der Assembly VisUtils.NodeLink. Sie wurde genutzt um die Knoten und Kanten zu halten, die gezeichnet wurden. Allerdings waren alle Variablen private gehalten und keine Methoden implementiert, mit denen man die Listen der Knoten und Kanten löschen und somit auch die grafischen Elemente selbst löschen konnte. Dies wurde aber durch die Updates auf neuere Versionen gelöst.

NodeHeight und NodeWidth In WPF werden keine Events ausgelöst, sobald ein Element in ein Fenster gezeichnet wird. Dadurch erhält der Entwickler auch keine Rückmeldung, wann die Größe der Elemente feststeht, was für die Anordnung der Knoten in diesem Prototypen von Bedeutung war. Die initialen Knoten konnten mithilfe des Events ContentRendered, das vom Window selbst angestoßen wurde, passend gesetzt werden. Für die spätere Anordnung der Knoten wurde jedem Knoten eine maximale Größe zugeordnet und anhand dieser definierten Werte die Knoten mit entsprechendem Abstand in das Display eingezeichnet.

6. Studie

Unter Verwendung des Prototypen, der das in dieser Arbeit beschriebene Konzept wie in Kapitel 5 umsetzt, wurde eine kleine Studie durchgeführt, um einen Eindruck von der Tauglichkeit und Intuitivität des Konzepts zu bekommen.

6.1. Vorbereitung

Für diese Studie wurde ein SPARQL-Endpoint gewählt, der Zugang zu den RDF-Daten des CIA World Factbook [Dea03] ermöglicht. Es sollte untersucht werden, ob es möglich ist, mithilfe des Prototypen einen Überblick über die Daten zu gewinnen und Elemente in dem RDF-Graphen zu finden. Dafür wurden folgende Aufgaben vorbereitet:

1. Finde die am Häufigsten in den Daten auftretenden natürlichen Ressourcen.
2. Zu welcher Ressource gehört die Sprache Isländisch?
3. Wie viele Objekte gibt es, die mit Deutschland in Verbindung stehen?
4. Was sind die häufigsten Importe?

6.2. Durchführung

Jedem Probanden wurde als erstes erklärt, worüber diese Studie geht, und wie sie ablaufen soll:

- Consent Form lesen und unterzeichnen
- Fragebogen zu Angaben der Person ausfüllen
- genauere Erklärungen zu Semantischen Daten und dem Prototypen
- kleines Beispiel anhand dem die Funktionalität und Einschränkungen des Prototypen aufgezeigt werden
- Aufgaben bearbeiten – Kommentare werden mitgeschrieben
- Fragebogen zur Visualisierung des Konzeptes ausfüllen

6.2.1. Probanden

Durch einen ersten Fragebogen (siehe Anhang) wurden Angaben zur Person der fünf Probanden aufgenommen:

- Die Teilnehmer der Studie waren 24 - 27 Jahre alt.
- Die Teilnehmer bestanden aus einer Frau und vier Männern.
- Ein Teilnehmer war Diplom Informatiker, die anderen Teilnehmer Studenten der Informatik.
- Drei der Teilnehmer hatten schon vom Semantic Web gehört, aber noch keine genaueren Kenntnisse, zwei Teilnehmer hatten noch nie davon gehört.
- Ein Teilnehmer hatte schon von RDF gehört, die anderen noch nie.
- Keiner der Teilnehmer kannte das CIA World Factbook.

6.3. Ergebnisse

Aus der Studie, dem Verhalten der Probanden und ihren Antworten auf den Fragebögen lassen sich einige Ergebnisse schließen. Diese sind in Tabellen zusammengefasst im Unterkapitel 6.3.1 zu finden.

6.3.1. Antworten der Benutzer

Tabellen 6.1, 6.2, 6.3 6.4, 6.5, 6.6, 6.7 und 6.8 stellen die Ergebnisse durch den zweiten Fragebogen (siehe Anhang) dar, den die Probanden ausfüllen mussten. In Tabelle 6.8 sind außerdem auch die Kommentare, die während der Studie von den Probanden gefallen sind mit aufgenommen.

6.3.2. Beobachtungen

Der Graph wurde im Allgemeinen als hilfreich, aber auch unübersichtlich bewertet. Vor allem die Menge der Kanten wurde schnell zu groß und führte zu Verwirrung. Die Teilnehmer, die noch nie etwas vom Semantic Web gehört hatten, verstanden auch den Hintergrund der gerichteten Kanten nicht richtig. Außerdem merkte ein Teilnehmer an, die Möglichkeit Kanten und Knoten wieder auszublenden würde ihm sehr weiterhelfen. Dabei halfen die verschiedenen Arten von Knoten dreien der fünf Teilnehmern beim Verwenden des Prototyps. Zwei Probanden waren sie egal, sie hatten sie nicht beachtet. Kein Proband hat sie nicht verstanden. Außerdem war es schwierig den Graphen ohne Kenntnisse der Datenbank zu deuten, da die Namen unverständlich waren, manche mehrfach vorhanden (einmal groß- und einmal kleingeschrieben). Auch die Verknüpfung der Ressourcen war für die Teilnehmer nicht aussagekräftig. Sie gaben aber an, das auch dies an den Beziehungen der Semantischen Daten lag.

Die Navigation durch den Graphen war für zwei der Teilnehmer einfach, für zwei andere Teilnehmer kompliziert, drei Teilnehmer fanden sie intuitiv. Das Scrollen wurde kritisch betrachtet. Zum einen dauerte es lange zu einem bestimmten Buchstaben zu blättern, wenn die Daten alphabetisch geordnet waren, zum anderen wurde es als unvorhersehbar betrachtet, da es Knoten gab, die nicht verschwanden beim Blättern. Es wurde erwartet, dass die Buttons auch jeweils in die andere Richtung sortieren können. Für die Navigation wurde die Möglichkeit zu zoomen vermisst.

Nützlich fanden die Probanden die Suche mit Hilfe der Suchleiste. Drei der Probanden fanden die Elemente dadurch leicht, zwei fanden sie nur zum Teil leicht. Die Teilnehmer bemängelten allerdings, dass die Suche trotz Filter viele Ergebnisse präsentierte, die zum Großteil semantisch nichts mit ihrer Suche zu tun hatten. Außerdem waren ihnen die Bedienungsmöglichkeiten der Suchleiste zu eingeschränkt, da die Pfeiltasten nicht verwendet werden konnten und ihnen fehlte ein visuelles Feedback in Form eines veränderten Cursors während die Anwendung die Suche durchführt.

6.3.3. Schlussfolgerungen

Im Allgemeinen wurde der Prototyp als nützlich empfunden. Einige der Probanden meinten während der Studie, dass sie die Elemente ohne ihn nicht gefunden hätten. Da sie alle nur wenig bis keine Ahnung von RDF und semantischen Daten hatten, taten sie sich mit dem Konzept von Tripeln schwer und auch die Markier-Funktion half ihnen nicht dabei, sie zu verstehen. Unter anderem deshalb fiel es ihnen schwer den Graphen zu verstehen. Außerdem machte es für sie schwerer die Navigation durch den Graphen nachvollziehen zu können. Viele Kanten, die sie durch viele geöffnete Ressourcen erzeugten, machten den Graphen zusätzlich unübersichtlich. Zudem war die verwendete Datenbank nicht so gestaltet, dass unerfahrene Nutzer sie leicht verstehen konnten. Die Ressourcen waren unverständlich benannt und erst über mehrere Tripel ersichtlich, welche Ressourcen in Beziehung zueinander standen. Man kann nur beschränkt Einfluss auf die Datensätze nehmen, der Graph sollte jedoch verschlankt werden und es sollte eine Möglichkeit geben, das Label je nach Datensatz vom Nutzer neu definieren zu lassen.

Die Suchleiste, als weiteren Teil des Prototypen, sollte zum einen im Hinblick auf die Auswahl der gefundenen Ressourcen verbessert werden und dem Nutzer sollte mehr Feedback bei der Nutzung zurückgegeben werden. Allerdings konnten sich die Nutzer einen besseren Eindruck vom Datensatz durch die zusätzlichen Ergebnisse verschaffen, was ein Teilnehmer „Wikipedia-Effekt“ nannte. Für den Überblick über den Graphen fanden die Probanden die am Häufigsten vorkommenden Klassen nur wenig hilfreich, was sie aber auch auf den Datensatz zurückführten. Die zuletzt besuchten Knoten, die nach dem Klick auf den Refresh-Button auch aktualisiert wurden, wurden jedoch häufig benutzt.

Der Graph war...	
... hilfreich	3x
... verständlich aufgebaut	1x
... unübersichtlich	4x
... unnötig	0x

Tabelle 6.1.: Antworten zu Frage 1, Teil 1

Das war gut:	**Das war schlecht:**
der Prototyp erleichtert die Arbeit	mochte die Farben nicht, Text manchmal zu groß für die Knoten, viele Pfeile wirken verwirrend
farbliche Highlights	
Vorschläge	kein Feedback bei Suche (Ladesymbol)
Man erfährt durch Zufall Dinge, die man noch nicht wusste. Zusammenhänge können hervorgehoben werden.	lange Antwortzeiten, stellenweise nicht deterministisches Verhalten (Kanten)
Visuelle Darstellung hilft beim Erfassen des Themenkomplexes	meist zuviele Kanten, darunter leidet die Übersicht

Tabelle 6.2.: Antworten zu Frage 1, Teil 2

Die verschiedenen Knotentypen...	
... waren nützlich	3x
... habe ich nicht beachtet	2x
... habe ich nicht verstanden	0x

Tabelle 6.3.: Antworten zu Frage 2

Die Navigation durch den Graphen war...	
... einfach	2x
... intuitiv	3x
... kompliziert	2x
... unverständlich	0x

Tabelle 6.4.: Antworten zu Frage 3, Teil 1

Das war gut:	**Das war schlecht:**
intuitiv -> man muss nicht viel denken	Man muss manchmal viel durchblättern, bis man zum gewünschten Element kommt
Solange man weiß was man sieht, ist es einfach	Ohne Kenntnisse der Datenbank schwierig zu deuten
automatisches Öffnen von neuen Knoten	kein Zoom
Sortierung	dubiose Namen, Graph schnell unübersichtlich, keine Möglichkeit Kanten auszublenden
Doppelklick auf Knoten min/-max Knoten, Scrollen stellt Subknoten übersichtlich dar	Ich fand es verwirrend, dass beim scrollen bestimmte Knoten stehen bleiben.. -> irgendeine Art Unterscheidung der Knoten, die beim nächsten "Klick"weitergescrollt werden, von denen die durch das scrollen nicht beeinflusst werden.

Tabelle 6.5.: Antworten zu Frage 3, Teil 2

Ich war in der Lage Elemente in den Daten leicht zu finden...	
... ja	3x
... nein	0x
... manchmal	2x

Tabelle 6.6.: Antworten zu Frage 4, Teil 1

Das war gut:	Das war schlecht:
Suche findet viele Ergebnisse	Suche findet viele Ergebnisse
Suche findet bei korrekter Nutzung zielsicher zum Element	Ohne genaue Kenntnisse schwer Suchbegriff zu finden
Öffnen von assoziativen Kanten	
Wikipedia-Effekt: ich finde Dinge, die ich nicht finden wollte	ich hab gefühlt ewig gebraucht um das zu finden, was ich finden wollte

Tabelle 6.7.: Antworten zu Frage 4, Teil 2

Weitere Anmerkungen und Kommentare
Meldung wenn nichts gefunden
Pfeiltasten bei Suchfeld
tolles Konzept, mit Buttons auch rückwärts sortieren, Graph wirkt sehr verwirrend
ich klick mich mal durch bis ich bei „I.“ bin, Knoten wieder wegklicken

Tabelle 6.8.: Antworten zu Frage 5 und Kommentare

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Visualisierungskonzept entwickelt, mit dem Ressourcen in RDF-Daten einfach gefunden werden können sollen. Dies besteht aus einer Suchleiste um Elemente direkt zu finden und einem Subgraphen der RDF-Daten, den man inkrementell in beliebige Richtungen erweitern kann, um den zugrundeliegenden RDF-Graphen zu untersuchen. Es wurden verschiedene Möglichkeiten vorgestellt, mit denen man einen Überblick über den Datensatz verschaffen kann, der dem Nutzer beim Suchen und Arbeiten mit den Daten helfen soll. Außerdem wurden verschiedene Knotentypen definiert und ein Navigationskonzept um sich im Subgraphen intuitiv zu bewegen.

Dieses Konzept wurde anschließend mit WPF prototypisch umgesetzt. Für die Suchleiste wurde ein eigenes User Control entwickelt. Die verschiedenen Knotentypen sind in einer maximierten und minimierten Form verfügbar. In der vergrößerten Form sind sowohl Label, als auch URI ersichtlich und fünf verschiedene Buttons. Diese sind zum Blättern in den Aggregatknoten, Sortieren der Aggregatknoten und Markieren des Knotens um die Tripel hervorzuheben. Es wurden verschiedene EventHandler implementiert, um die Navigation durch Doppelklicks und ein Panning zu implementieren. Dabei wurde immer wieder auf die Bibliotheken VisUtils, VisUtils.WPF und SparqlUtilities zurückgegriffen.

Der Prototyp wurde in einer Studie mit fünf Teilnehmern ausgewertet. Sie hatten noch keinerlei Erfahrung mit semantischen Daten und RDF und taten sich etwas schwer das Konzept der Tripel und des RDF-Graphen zu verstehen. Sie fanden das Konzept dieser Arbeit verständlich und gut, allerdings wurde der dadurch entstehende Graph durch viele Kanten sehr unübersichtlich und für sie schwer zu verwenden. Sie schlagen weitere Interaktionsmöglichkeiten vor um die Navigation zu vereinfachen. Außerdem befanden sie die Ergebnisse der Suchleiste für zu umfangreich. Es wurden viele lange Texte angezeigt, die nicht gesucht waren und viel Platz einnahmen, aber sie konnten dadurch auch einen guten Überblick über die Daten gewinnen. Im verwendeten Datensatz waren die am häufigsten verwendeten Klassen nicht ausschlaggebend für eine Übersicht über die Daten.

Ausblick

Das Konzept der Verwendung eines Suchfeldes und eines Subgraphen zu Visuellen Suche in RDF-Daten wurde in der Suche bestätigt. Allerdings muss es vor allem für unerfahrene Nutzer vereinfacht werden. Zum Beispiel könnten mehrere Subgraphen, die sich teilweise überschneiden oder gleiche Ressourcen in verschiedenen Graphen mehrfach anzeigen, verwendet werden. Da vor allem die „type“-Ressource viele unübersichtliche Kanten verursacht hat, könnte sie aus dem Graphen herausgenommen und in die Knoten integriert werden. Selbiges gilt für das Label. Da nicht alle Endpoints die üblichen

rdf-Stichworte verwenden, könnte danach gesucht werden oder dies vom Nutzer eingestellt und anschließend gespeichert werden.

Die integrierte Suchleiste wurde von den Studienteilnehmern als sehr nützlich empfunden, allerdings zeigte sie nicht immer nur die gewünschten Ergebnisse an. Dies könnte über die Verwendung von mehr Semantik beim Suchvorgang ermöglicht werden. Außerdem könnten zu lange Literale aus den Ergebnissen gelöscht werden.

Die Übersicht über den Graphen durch die Darstellung der am häufigsten vorkommenden Klassen im Datensatz war nicht sehr erfolgreich. Das Clustering der RDF-Daten und dessen Präsentation könnte eine bessere Übersicht bieten.

A. Fragebogen der Studie

Den Probanden der Studie, die in Kapitel 6 beschrieben ist, wurde der folgende Fragebogen auf Papier ausgehändigt. Teil 1 füllten sie zu Beginn der Studie aus. Er dient dazu Angaben über die Teilnehmer aufzunehmen, um ihre Vorkenntnisse einschätzen zu können. Den Teil 2 des Fragebogens bekamen sie erst zum Ende der Studie zu Gesicht. Dort sind Fragen zu den verschiedenen Teilen der Visualisierung, die in der durchgeführten Studie evaluiert wurden.

Fragebogen – Teil 1
Angaben zur Person:

1. **Alter:**

   _____

2. **Geschlecht:**

   [ ] männlich [ ] weiblich

3. **Beruf:**

   _____

4. **Berufsfeld:**

   _____

5. **Kenntnisse über:**

   a) Semantic Web:

[ ] gute Kenntnisse [ ] schon davon gehört [ ] noch nie gehört

   b) RDF:

[ ] gute Kenntnisse [ ] schon davon gehört [ ] noch nie gehört

   c) CIA World Factbook

[ ] gute Kenntnisse [ ] schon davon gehört [ ] noch nie gehört

Fragebogen – Teil 2
Visuelle Darstellung:

6. **Der Graph war…**

[ ] … hilfreich [ ] …verständlich aufgebaut

[ ] … unübersichtlich [ ] … unnötig

Das war gut: _____

_____

_____

Das war schlecht: _____

_____

_____

7. **Die verschiedenen Knotentypen …**

[ ] … waren nützlich [ ] … habe ich nicht beachtet

[ ] … habe ich nicht verstanden

8. **Die Navigation durch den Graphen war …**

[ ] … einfach [ ] … intuitiv [ ] … kompliziert [ ] … unverständlich

Das war gut: _____

_____

_____

Das war schlecht: _____

_____

_____

9. Ich war in der Lage Elemente in den Daten leicht zu finden:

[ ] ja [ ] nein [ ] manchmal

Das war gut: _____

_____

_____

Das war schlecht: _____

_____

_____

10. Weitere Anmerkungen:

_____

_____

_____

_____

_____

_____

Vielen Dank für Ihre Zeit!

Literaturverzeichnis

[AHK06] J. Abello, F. van Ham, N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. *TVCG*, 12(5):669–676, 2006. (Zitiert auf Seite 31)

[Anh10] L. D. Anh. Building a Search Text Box Control with WPF, 2010. URL http://www.codeproject.com/Articles/101975/Building-a-Search-Text-Box-Control-with-WPF. (Zitiert auf Seite 45)

[BL98] T. Berners-Lee. Semantic Web Road map, 1998. (Zitiert auf den Seiten 9 und 11)

[BLCC+06] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, D. Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *SWUI*, Band 2006. 2006. (Zitiert auf den Seiten 7, 22, 23 und 29)

[BLHL01] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. *SCI AM*, 284(5):34–43, 2001. (Zitiert auf den Seiten 11 und 12)

[CZX11] L. Chao, Y. Zhang, C. Xing. Improving Query Suggestion for Digital Libraries. In *COMPSACW '11*, S. 428–434. IEEE, 2011. (Zitiert auf Seite 32)

[Dea03] M. Dean. CIA World Fact Book in DAML, 2003. URL http://www.daml.org/2001/12/factbook/. (Zitiert auf den Seiten 44 und 47)

[DKS07] L. Deligiannidis, K. J. Kochut, A. P. Sheth. RDF Data Exploration and Visualization. In *CIMS '07*, S. 39–46. ACM, 2007. (Zitiert auf den Seiten 7, 24, 25, 29, 31, 33, 34 und 36)

[HMM00] I. Herman, G. Melançon, M. S. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *TVCG*, 6(1):24–43, 2000. (Zitiert auf den Seiten 7, 31, 32, 33, 34 und 36)

[HSMT06] T. Hansaki, B. Shizuki, K. Misue, J. Tanaka. FindFlow: Visual Interface for Information Search based on Intermediate Results. In *Proc. PacificVis '06*, S. 147–152. Australian Computer Society, Inc., 2006. (Zitiert auf den Seiten 7, 20, 21 und 28)

[JCCD11] M. Jiang, Y. Chen, J. Chen, X. Du. Interactive Predicate Suggestion for Keyword Search on RDF Graphs. In *ADMA*, S. 96–109. Springer, 2011. (Zitiert auf den Seiten 7, 17, 26, 27, 29 und 32)

[JGZ+11] H.-C. Jetter, J. Gerken, M. Zöllner, H. Reiterer, N. Milic-Frayling. Materializing the Query with Facet-Streams - A Hybrid Surface for Collaborative Search on Tabletops. In *CHI*, S. 3013–3022. ACM, 2011. (Zitiert auf Seite 17)

[MAEGJ04] A. J. Morris, A. I. Abdelmoty, B. A. El-Geresy, C. B. Jones. A Filter Flow Visual Querying Language and Interface for Spatial Databases. *GeoInformatica*, 8(2):107–141, 2004. (Zitiert auf Seite 17)

[Mica] Microsoft Corporation. Introduction to WPF. URL http://msdn.microsoft.com/de-de/library/aa970268(v=vs.110).aspx. (Zitiert auf Seite 39)

[Micb] Microsoft Corporation. Routed Events Overview. URL http://msdn.microsoft.com/de-de/library/ms742806(v=vs.110).aspx. (Zitiert auf den Seiten 7 und 43)

[MPG98] N. Murray, N. Paton, C. Goble. Kaleidoquery: A Visual Query Language for Object Databases. In *Proc. AVI '98*, S. 247–257. ACM, 1998. (Zitiert auf den Seiten 7, 21 und 28)

[Owe09] D. Owens II. WPF Search Text Box, 2009. URL http://davidowens.wordpress.com/2009/02/18/wpf-search-text-box/. (Zitiert auf Seite 44)

[Pur02] H. C. Purchase. Metrics for Graph Drawing Aesthetics. *J VISUAL LANG COMPUT*, 13(5):501–516, 2002. (Zitiert auf Seite 36)

[RSBS08] A. Russell, P. R. Smart, D. Braines, N. R. Shadbolt. NITELIGHT: A Graphical Tool for Semantic Query Construction. 2008. (Zitiert auf den Seiten 7, 19, 20, 21, 28, 29, 31, 33 und 34)

[Shn94] B. Shneiderman. Dynamic Queries for Visual Information Seeking. *IEEE SOFTWARE*, 11(6):70–77, 1994. (Zitiert auf den Seiten 17 und 32)

[W3C12] W3C. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, 2012. URL http://www.w3.org/TR/xmlschema11-2/. (Zitiert auf Seite 12)

[W3C13] W3C. SPARQL 1.1 Query Language, 2013. URL http://www.w3.org/TR/sparql11-query/. (Zitiert auf den Seiten 8, 13, 15 und 16)

[W3C14a] W3C. RDF 1.1 Concepts and Abstract Syntax, 2014. URL http://www.w3.org/TR/rdf11-concepts/. (Zitiert auf den Seiten 7, 12 und 33)

[W3C14b] W3C. RDF Schema 1.1, 2014. URL http://www.w3.org/TR/rdf-schema/. (Zitiert auf Seite 13)

[YS98] D. Young, B. Shneiderman. A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. 1998. (Zitiert auf den Seiten 7, 17, 18 und 28)

Alle URLs wurden zuletzt am 14. 11. 2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift