

---

# Metadatenextraktion und Vorschlagssysteme im Visual SPARQL Builder, einer Webanwendung zur graphischen Modellierung von SPARQL-SELECT-Anfragen

Lukas Eipert<sup>1</sup>

**Abstract:** Die Forschung zum Speichern von Daten in semantischen Datenbanken ist in der Informatik weit fortgeschritten. Um jedoch die Nutzung semantischer Technologien in Digital Humanities oder der Industrie zu fördern, bedarf es Werkzeugen, die Nicht-Informatikern den Gebrauch des Semantic Web erleichtern. Der Visual SPARQL Builder ist ein solches Werkzeug und ermöglicht es, SPARQL-SELECT-Anfragen graphisch zu modellieren. Damit man mit dem Visual SPARQL Builder arbeiten kann, werden jedoch Metadaten über die Struktur der Daten einer semantischen Wissensbasis benötigt. In dieser Abhandlung wird beschrieben, wie der Visual SPARQL Builder die Strukturinformationen mit Hilfe von SPARQL-Anfragen ermittelt und dazu benutzt, durch geeignete Vorschlagssysteme die Eingaben für den Nutzer zu erleichtern.

**Keywords:** SPARQL, Semantic Web, Metadaten, Datenexploration, Visuelle Anfragesprache

## 1 Einleitung

Ziel des Projektes ist es, eine Anwendung zu entwickeln, die es ermöglicht, Anfragen für semantische Wissensdatenbanken einfach graphisch zu modellieren. Diese Anfragen-Modelle werden in die Anfragesprache SPARQL [Wo13a] übersetzt. Durch die Modellierung soll die Komplexität von SPARQL umgangen werden. Der Nutzer kann dadurch auf intuitive Weise Anfragen an semantische Wissensdatenbanken stellen. Im Allgemeinen dürfte eine graphische Darstellung komplexer Zusammenhänge in einem diagrammähnlichen Modell leichter zu erfassen und intuitiver sein als ihre textuelle Repräsentation.

Der Visual SPARQL Builder (kurz **VSB**) ist aus einem Softwaretechnikpraktikum, im Bachelorstudiengang Informatik an der Universität Leipzig, hervorgegangen. Zum Zeitpunkt des Praktikums wurde der VSB noch Graphical SPARQL Builder genannt. Nach dem Ende des Projektes führte ich die Entwicklung der Anwendung weiter mit dem Ziel, meine Bachelorarbeit über den VSB zu schreiben. Diese ist zur Zeit noch in Bearbeitung und wird innerhalb des Sommersemesters 2015 fertiggestellt.

Einige Anwendungsfälle für einen solchen visuellen Editor sind in Abschnitt 2.1 aufgeführt. Aus diesen Anwendungsfällen und den Vorgaben während des Softwaretechnikpraktikums ergeben sich die folgenden Anforderungen an den VSB, welche nach meinem Wissensstand zur Zeit von keinem anderen visuellen SPARQL Editor umgesetzt werden:

---

<sup>1</sup> Universität Leipzig, Bachelorstudiengang Informatik, lukas.eipert@studserv.uni-leipzig.de

- 
1. Clientseitiger Editor, welcher im Webbrowser des Anwenders läuft
  2. Extraktion der benötigten Metadaten vom genutzten SPARQL-Endpunkt während der Laufzeit der Anwendung
  3. Intuitive Benutzeroberfläche mit einem geeigneten Vorschlagssystem, um die Erstellung von Anfragen zu vereinfachen

Um diese Ziele zu erreichen, wurde für den VSB eine diagrammbasierte Modellierungssprache ähnlich visueller Programmiersprachen entwickelt. Auf diese wird im Abschnitt 2.2 genauer eingegangen. Die Anwendung ist als Javascript-Webapplikation umgesetzt und kann in jedem modernen Browser genutzt werden. Daraus ergeben sich gewisse Vorteile, der Administrator kann zum Beispiel durch eine Programmaktualisierung des VSB an zentraler Stelle eine neue Version an alle Nutzer verteilen. Genauere Informationen zur technischen Realisierung finden sich in Abschnitt 2.3.

Das Hauptaugenmerk dieses Artikels soll jedoch auf der Metadatenextraktion aus SPARQL-Endpunkten während der Laufzeit einer Anwendung (Abschnitt 3) und des eingebauten Vorschlagssystems (Abschnitt 4) liegen.

## 2 Hintergrund

In diesem Abschnitt werden einige Hintergründe zum Visual SPARQL Builder für ein besseres Verständnis der späteren Abschnitte beleuchtet.

### 2.1 Anwendungsfälle

Im Wesentlichen sind zwei Anwendungsfälle für den VSB einschlägig. Zum einen die tägliche Anwendung durch Domänenexperten wie Historikern oder Bibliothekaren, die schnell auf Fragen antworten möchten, bezogen auf einen ihnen bekannten Datenbestand; zum anderen für Nutzer, die eine semantische Wissensdatenbank erforschen wollen, jedoch die Struktur und den Datenbestand derselbigen nicht oder wenig kennen. In beiden Fällen ist es von Nutzen, dass der VSB weder tiefgreifende SPARQL- noch Vokabularkennntnis erfordert, sondern durch geeignete Metadatenextraktion und Vorschlagssysteme eine intuitive Nutzung semantischer Daten möglich macht.

Der erste Anwendungsfall wird auch im Rahmen meiner Bachelorarbeit näher beleuchtet werden, da der VSB an der Universitätsbibliothek Leipzig benutzt werden soll, um auf Daten des semantischen Electronic Resource Management Systems AMSL<sup>2</sup> zuzugreifen .

Das vereinfachte Erforschen eines Endpunktes kann man anhand der Beispielinstantz des VSB<sup>3</sup> nachvollziehen; diese arbeitet mit dem SPARQL-Endpunkt der dbpedia<sup>4</sup>. Ähnliche Installationen sind für jeglichen SPARQL-Endpunkt denkbar, welcher der Öffentlichkeit Daten zur Verfügung stellt.

---

<sup>2</sup> <http://amsl.technology>

<sup>3</sup> <http://leipert.github.io/vsb/dbpedia>

<sup>4</sup> <http://dbpedia.org/sparql>

## 2.2 Modellierungssprache

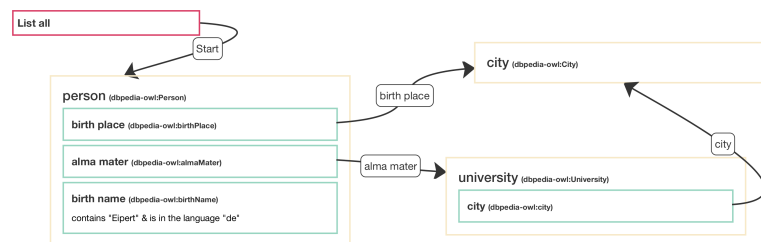


Abb. 1: Modell einer Anfrage an die dbpedia

Wie bereits in der Einleitung erwähnt, wurde für den VSB eine Modellierungssprache ähnlich zur UML [Ob11] entwickelt. In Abb. 1 ist eine Anfrage in dieser Modellierungssprache dargestellt. Es wird beispielhaft nach einer allen Personen in der dbpedia gesucht, deren Geburtsname “Eipert” beinhaltet und die in ihrer Geburtsstadt zur Universität gegangen sind. Wie leicht zu erkennen, ähnelt die Sprache einem Datenflussdiagramm.

Die Modellierungssprache wurde im Softwaretechnikpraktikum erstmals entworfen und während des Fortschreitens des Projektes weiterentwickelt. Im Wesentlichen wurden die Anfragesprache SPARQL 1.1 und die Empfehlungen RDF, RDFS und OWL des W3C während der Entwicklung des Sprachmodells berücksichtigt ([Wo14a], [Wo14b], [Wo04]). Daraus entstand diese klassenbasierte Diagrammsprache, in der Klassen durch Eigenschaften eingeschränkt und in Relation gesetzt werden können. Klassen sind als Boxen umgesetzt. Eigenschaften einer Klasse werden in der Klassenbox als Unterboxen dargestellt. Für die Abbildung von Relationen zwischen Klassen werden Pfeile benutzt.

Als besonderes Element der Modellierungssprache ist die Wurzel zu erwähnen. Dies ist dem Umstand geschuldet, dass ursprünglich die Übersetzung eines Anfragenmodells zu SPARQL rekursiv stattfand und dafür ein Startpunkt benötigt wurde. Die Wurzel ist erhalten geblieben, da es die Lesbarkeit deutlich erhöht, wenn ein Bezugspunkt für die Betrachtung der Anfrage gegeben ist. Perspektivisch sind auch andere Anfragen als SELECT-Queries denkbar, die dann durch unterschiedliche Wurzelbezeichner ausgedrückt werden können.

Anfragen in der Modellierungssprache des VSB können in JSON (JavaScript Object Notation) serialisiert und aus dieser wieder deserialisiert werden. Die Übersetzung nach SPARQL findet aus der serialisierten JSON-Variante eines Anfragenmodells statt.

## 2.3 Technische Realisierung

Der Visual SPARQL Builder ist als Javascript-Webapplikation umgesetzt. Dabei wurde das von Google entwickelte Javascript-Framework AngularJS<sup>5</sup> verwendet. Zum einen folgt die Entwicklung in AngularJS einem aus dem Studium bekannten Model-View-Controller Paradigma, was die Einstiegshürde für die Entwicklung senkte. Zum anderen lassen sich Komponenten leicht modularisieren, was die Weiterentwicklung, Erweiterung und den Austausch derselbigen vereinfacht. Auch die zahlreich existierenden AngularJS-Komponenten

<sup>5</sup> <https://angularjs.org/>

---

haben die Entwicklung beschleunigt. So greifen zum Beispiel die Übersetzung der Oberfläche, das Routing in der Anwendung oder die Dropdown-Menüs auf Open-Source-Projekte zurück.

Für die Kommunikation mit SPARQL-Endpunkten wird die Javascript-Bibliothek Jassa<sup>6</sup> (**J**avascript **S**uite for **S**parql **A**ccess) verwendet, eine Entwicklung von Claus Stadler an der Universität Leipzig, welche im Rahmen des GeoKnow-Projektes entsteht [St14].

Zahlreiche Teile des VSB sind konfigurierbar gestaltet, sodass nicht unbedingt Javascript- oder AngularJS-Kenntnisse notwendig sind, um den VSB für den Einsatz mit einer eigenen semantischen Wissensdatenbank anzupassen.

### 3 Metadatenextraktion

Für den Betrieb des VSB werden verschiedene Informationen über den Aufbau des Datensatzes benötigt. Es gibt verschiedene Möglichkeiten, diese Strukturdaten aus der semantischen Wissensbasis zu gewinnen und bereitzustellen.

#### 3.1 Ansätze der Metadatenextraktion

Im Wesentlichen gibt es drei mögliche Ansätze für die Bereitstellung von semantischen Strukturdaten.

1. Die Struktur wird statisch in Dateien bereitgestellt, zum Beispiel im JSON- oder XML-Format. Das birgt den Nachteil, dass diese statischen Dateien bei Änderungen der Struktur neu generiert werden müssen, um diese Änderungen zu reflektieren. Hat man eine wohldefinierte Struktur, die sich selten ändert, wäre dies die passende Lösung, da die Generierung der statischen Dateien selten stattfindet.
2. Die Struktur wird durch den VSB selbst mit Hilfe von SPARQL-Anfragen an den Endpunkt extrahiert. Dadurch wird gewährleistet, dass der Anwender immer mit dem aktuellen Stand der Metadaten arbeiten kann. Nachteilig wirkt sich dies nur aus, falls komplexe Anfragen sehr lange Berechnungszeiten in Anspruch nehmen und so die Performanz des VSB senken, da der Nutzer auf die Strukturdaten warten muss.
3. Die Vorteile der ersten beiden Arten der Metadatenextraktion ließe sich mit Hilfe eines gesonderten Cache-API kombinieren, welches die Strukturdaten bereitstellt. Dieses könnte in regelmäßigen Abständen komplexe SPARQL-Anfragen im Hintergrund ausführen und die Ergebnisse im JSON- oder XML-Format zur Verfügung stellen. Hierbei ist nachteilig, dass man diese API programmieren und warten muss.

Für den VSB wurde der zweite Ansatz gewählt, da dies den geringsten Aufwand für einen Administrator darstellt, eine funktionierende Anwendung bereitzustellen und den Nutzen des VSB für einen vorhandenen SPARQL-Endpunkt zu evaluieren. In der Regel sind die

---

<sup>6</sup> <https://github.com/GeoKnow/Jassa-Core>

---

Strukturdaten im Endpunkt vorhanden und lassen sich leicht extrahieren. Die zur Metadatenextraktion dienenden SPARQL-Anfragen können leicht durch eine Konfigurationsdatei angepasst werden. Sollte dieser Ansatz für die Verwendung des VSB nicht die optimale Lösung darstellen, kann der Metadatenextraktionsservice leicht ausgetauscht werden, ermöglicht durch die modulare Gestaltung der Programmierung.

### 3.2 Metadatenextraktion im VSB

Im Folgenden sind die für den erfolgreichen Einsatz des VSB benötigten Metadaten beschrieben. Die Standardimplementierung arbeitet vor allem mit den Schemata RDFS und OWL, welche Empfehlungen vom W3C sind und daher als allgemeine Standards angesehen werden können. Die wichtigsten SPARQL-Queries sind die zur Bestimmung der vorhandenen Klassen und möglichen Eigenschaften einer Klasse.

```
SELECT DISTINCT * WHERE {  
  {?uri a rdfs:Class .} UNION {?uri a owl:Class .} .  
  OPTIONAL {  
    ?uri rdfs:label ?label .  
    BIND(LANG(?label) AS ?label_loc)  
  }  
  OPTIONAL {  
    ?uri rdfs:comment ?comment .  
    BIND(LANG(?comment) AS ?comment_loc)  
  }  
  FILTER (  
    IF(?comment_loc != "" && ?label_loc != "", ?comment_loc = ?label_loc, true)  
  )  
}
```

List. 1: Bestimmung der verfügbaren Klassen eines Endpunktes

Die verfügbaren Klassen werden mit Hilfe des Queries in List. 1 bestimmt. Es werden alle Klassen der Typen `rdfs:Class` und `owl:Class` gefunden. Insofern die Klassen Bezeichner `rdfs:label` oder Beschreibungen `rdfs:comment` haben, werden diese zusammen mit ihrer Sprache ausgelesen. Die Bezeichner und Beschreibungen werden für das Vorschlagssystem des VSB benötigt.

In List. 2 ist dargestellt, wie die Eigenschaften einer Klasse ausgelesen werden, im Beispiel die Klasse “Schauspieler” der `dbpedia`. Zuerst werden alle Überklassen und gleichgestellte Klassen der gewünschten Klasse bestimmt. Dann werden alle Eigenschaften geladen, deren `rdfs:domain` der Klasse, Überklasse oder gleichgestellten Klassen entspricht. Im Beispiel wären dies also nicht nur die Eigenschaften der Klasse “Schauspieler”, sondern auch die der Überklasse “Person”. Dies ist notwendig, da eine Unterklasse immer die Eigenschaften ihrer Überklasse erbt. Für das Vorschlagssystem werden weiterhin die möglichen Zielklassen `rdfs:range` und die Art `rdf:type` der Eigenschaften geladen. Wie auch bei den Klassen werden Bezeichner und Beschreibungen der Eigenschaften ausgelesen [vgl. List. 1].

---

```

SELECT DISTINCT
?uri ?range ?type ?label ?label_loc ?comment ?comment_loc
WHERE {
  dbpedia-owl:Actor
  \texttt{rdfs:subClassOf|(owl:equivalentClass|^owl:equivalentClass)}*
  ?class .
  ?uri rdfs:domain ?class .
  OPTIONAL { ?uri rdfs:range ?range }
  OPTIONAL { ?uri rdf:type ?type }
  [...]
}

```

List. 2: Bestimmung der Eigenschaften einer Klasse, hier `dbpedia-owl:Actor`

Zusätzlich zu der beschriebenen Methode, Eigenschaften ausgehend von der Klasse zu finden, gibt es eine ähnlich aufgebaute, welche nach inversen Relationseigenschaften sucht. So kann nicht nur Schauspieler  $\xrightarrow{\text{Kind}}$  Person, sondern auch Schauspieler  $\xleftarrow{\text{Elternteil}}$  Person abgebildet werden. Des Weiteren benutzt der VSB intern Anfragen um Unter-/Überklassen zu bestimmen, sowie mögliche Relationen zwischen zwei beliebigen Klassen. Diese Anfragen sind auch für das Vorschlagssystem von Nöten, welches im Folgenden beschrieben wird.

## 4 Vorschlagssystem

Die wichtigste Komponente des VSB stellt das Vorschlagssystem dar, da durch dieses fehlerhafte Eingaben vermieden werden können. Auf diese Weise kann die Einstiegshürde zur Nutzung einer semantischen Datenbank gesenkt werden. Die Qualität des Vorschlagssystems hängt von der Qualität der Strukturdefinition und der richtigen Konfiguration der Metadatenextraktion des VSB ab. Sind beide gleichermaßen sichergestellt, ist die Nutzung die folgenden beschriebenen Teilkomponenten des Vorschlagssystems vollständig möglich.

### 4.1 Klassen- & Eigenschaftssuche

Die Suche nach Klassen und Eigenschaften ist im VSB durch Suchfelder mit Dropdown-Vorschlagsliste realisiert [vgl. Abb. 2]. Wenn man mit der Maus über die *i*-Symbole in der Vorschlagsliste geht, sieht man zusätzliche Informationen zu den Klassen/Eigenschaften.

Im Hintergrund wird hierfür die Liste der verfügbaren Klassen, beziehungsweise verfügbaren Eigenschaften einer Klasse mit Hilfe der beschriebenen Metadatenextraktion vom Endpunkt in den Arbeitsspeicher geladen. Es werden den Universal Resource Identifiers (URI) der zu durchsuchenden Elemente die Bezeichner und Kommentare in der eingestellten Sprache des VSB zugeordnet. Sind diese nicht in der gewünschten Sprache vorhanden, wird eine konfigurierbare Standardsprache wie zum Beispiel Englisch genommen. Ist kein

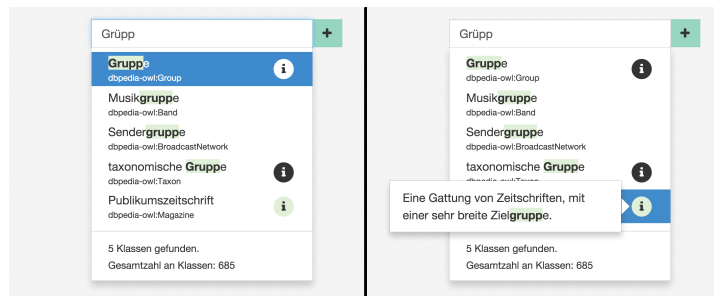


Abb. 2: Beispielhafte Klassensuche mit dem Begriff "Grüpp".

Bezeichner vorhanden, wird einer aus der URI der Ressource ermittelt. So würde der URI "http://dbpedia.org/ontology/Person" der Bezeichner "Person" zugeordnet werden.

Sobald der Nutzer einen Suchbegriff in das Eingabefeld eingibt, wird die Liste der Klassen/Eigenschaften durchsucht und auf alle Elemente reduziert, die einen Treffer in URI, Bezeichner oder Beschreibung aufweisen. Diese werden dann aufsteigend sortiert nach folgenden Kriterien:

1. Der Bezeichner fängt mit dem Suchbegriff an
2. Der Bezeichner enthält den Suchbegriff
3. Restliche Treffer

Jede dieser Gruppen ist in sich alphabetisch nach Bezeichnern sortiert. Die entstandene Liste wird auf eine gewisse Länge gekürzt und dem Nutzer in der Dropdownauswahl ausgegeben, in der die Suchbegriffe hervorgehoben werden.

Mehrere durch Leerzeichen getrennte Suchbegriffe werden als durch den logischen Operator "und" verknüpfte Suchbegriffe angesehen. Alle UTF-8 enkodierten Zeichen werden in der Suche berücksichtigt, bei lateinischen Buchstaben wird jedoch Groß-/Kleinschreibung ignoriert. Buchstaben mit Diakritika wie Akzenten (á), Tremata (ä) oder Tilden (ã) werden durch die Standardvariante (a) ersetzt. Dies soll die Trefferquote bei der Suche erhöhen und Nutzern, die zum Beispiel ein anderssprachiges Tastaturlayout nutzen, die Suche vereinfachen.

In Abb. 2 ist die Arbeitsweise der Suche deutlich zu erkennen. Der Suchbegriff "Grüpp" wurde intern als "grupp" behandelt, und so einige Klassen gefunden. Als erstes ist das Resultat "Gruppe" aufgelistet, da es mit dem Suchbegriff anfängt, gefolgt von einigen Klassen, die den Suchbegriff im Bezeichner enthalten. Als letzten Treffer ist "Publikumszeitschrift" zu finden, da diese Klasse das Wort "Zielgruppe" nur in der Beschreibung enthält.

Die Eigenschaftssuche hat im Unterschied zur Klassensuche die Möglichkeit nach Eigenschaftstypen zu filtern. Außerdem werden die Suchergebnisse nach den Eigenschaftstypen gruppiert. Um zum Beispiel alle Eigenschaften zu finden, die vom Typ *Zeichenkette* sind, kann der Nutzer einfach ":string" eingeben. Die verfügbaren Kategorien entsprechen den englischen Bezeichnungen der unterstützten Eigenschaftstypen des VSB. Diese werden in Abschnitt 4.2 an den passenden Stellen benannt.

## 4.2 Eigenschaftsfilter

In SPARQL ist es möglich, Filter auf Werte anzuwenden. Diese Filter unterstützen logische Operatoren, Vergleiche und je nach Datentyp bestimmte Funktionen [Wo13c]. Um jedoch den Nutzern die Eingabe solcher Filter zu vereinfachen und Fehler zu verhindern, bietet der VSB Eingabe- und Auswahlfelder, die semantisch korrekte Eingaben erfordern.

Jeder Eigenschaft im VSB kann der Filter NOT EXIST zugewiesen werden, damit man nach allen Instanzen einer Klasse mit Nichtvorhandensein dieser Eigenschaft suchen kann. Des Weiteren kann jede Eigenschaft als optional markiert werden, was kein Filter im Sinne von [Wo13c] darstellt, sondern die komplette Eigenschaft mit Einschränkung gemäß [Wo13b] in einem OPTIONAL-Block einfasst.

Der VSB unterscheidet zwischen zwei Eigenschaftstypen. Erstens die Datentypeigenschaften owl:DatatypeProperty, welche eine Zielklasse rdfs:range) eines elementaren Datentyps wie *Zeichenkette* oder *Zahl* aufweisen. Zweitens die Relationseigenschaften owl:ObjectProperty, welche Relationen zwischen Klassen beschreiben. Die Zuordnung von Eigenschaftstypen ist konfigurierbar durch den Administrator des VSB. Zum einen können Eigenschaften nach Typ rdf:type, zum anderen nach Zielklasse rdfs:range kategorisiert werden. Allen nicht zugeordneten Eigenschaften wird der Typ *Zeichenkette* zugewiesen.

### Datentypeigenschaften

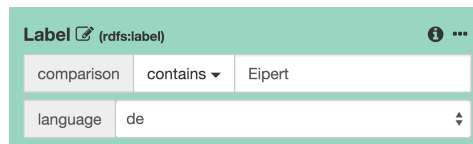
The image shows a user interface for configuring a filter for a property named 'Label' (rdf:type: label). The interface has a light green header bar with the property name and a dropdown menu. Below the header, there are two input fields: 'comparison' with a dropdown menu showing 'contains', and 'value' with the text 'Eipert'. Below these, there is a 'language' field with a dropdown menu showing 'de'. The interface is clean and modern, with a light green color scheme.

Abb. 3: Zeichenkette mit Filterfeldern

Der VSB unterstützt standardmäßig Datentypeigenschaften der Typen Zeichenkette, Datum und Zahl. Zukünftig sollen auch Uhrzeiten unterstützt werden. Jeder der Datentypen hat eigene Eingabefelder für Filter, die eine Eingabe von syntaktisch falschen Werten verhindern soll:

- **Zeichenketten** haben eine Vergleichssuche mit verschiedenen Optionen wie “enthält” oder “startet mit”. Reguläre Ausdrücke sind auch möglich. Zusätzlich lassen sich die Zeichenketten nach ihrer Sprache filtern. In Abb. 3 ist beispielhaft eine String-Eigenschaft mit ausgefülltem Filter dargestellt. Der Kategoriefilter für Zeichenketten ist “:string”.
- **Zahlen** können arithmetisch durch Addition, Multiplikation, Subtraktion und Division verändert werden. So ist es ein Leichtes, sich eine Bevölkerungszahl in Tausendern ausgeben zu lassen. Des Weiteren lassen sich die Zahlwerte gemäß der SPARQL-Definition [Wo13c] mit Vergleichsoperatoren wie “<” oder “=” vergleichen. Der Kategoriefilter für Zahlen ist “:number”.



- **Daten** lassen sich mit den Operationen “gleich”, “ungleich”, “vor dem” und “nach dem” vergleichen. Zur Eingabe des Vergleichsdatums wird ein Eingabefeld bereitgestellt, welches das Datum automatisch richtig formatiert und ein Kalender-Widget zur Datumsauswahl bereitstellt. Eine Eingabemöglichkeit für Datenbereiche ist zukünftig ebenso vorgesehen. Der Kategoriefilter für Daten ist “:date”.

Datentypeneigenschaften können, wenn nötig, von einem Nutzer des VSB in einen anderen Typ überführt werden, damit der passende Filter gesetzt werden kann. Dies ist erforderlich, falls eine Datentypeneigenschaft wegen inkorrektter Konfiguration des VSB oder mangelnder Deklaration von `rdfs:range` einer Eigenschaft nicht richtig erkannt wurde. Standardmäßig sind den elementaren Datentypen die entsprechenden Typen der W3C-Empfehlung XSD [Wo12] zugeordnet:

- String: `xsd:string`, `xsd:literal`
- Zahl: `xsd:integer`, `xsd:float`, `xsd:double`, `xsd:decimal`
- Datum: `xsd:date`

## Relationseigenschaften

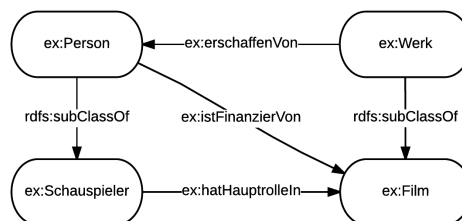


Abb. 4: Graph einer Beispielontologie

Um Relationen zwischen Klassen auszudrücken, sind im VSB Relationseigenschaften implementiert. Es gibt zwei Arten dieser Eigenschaften, die direkten (Kategoriefilter “:object”) und die inversen (Kategoriefilter “:inverse”). Anhand von Abb. 4 lässt sich am besten der Unterschied zwischen beiden erklären. In diesem Beispiel kann man der Klasse `ex:Person` eine direkte Eigenschaft `ex:istFinanzierVon` und eine inverse Eigenschaft `ex:erschaffenVon-1` aus der Beispielontologie “ex:” zuweisen.

Der VSB schränkt die Zuweisung von Relationen auf wohldefinierte mögliche Verbindungen ein. Ist die Beispielontologie aus Abb. 4 im VSB geladen, wäre es nicht möglich die Relation `ex:Schauspieler`  $\xrightarrow{\text{ex:hatHauptrolleIn}}$  `ex:Werk` zu erstellen, da dies in der Struktur der Ontologie nicht vorgesehen ist. Wie jedoch in Abschnitt 3.2 bereits erwähnt, beachtet der VSB Klassenhierarchien. So wäre es möglich die Relation `ex:Film`  $\xrightarrow{\text{ex:erschaffenVon}}$  `ex:Schauspieler`, aber auch ihre Inverse `ex:Schauspieler`  $\xleftarrow{\text{ex:erschaffenVon}}$  `ex:Film` zu modellieren.

Possible alma mater of person:

Filter available Classes

person → alma mater → ???

college (dbpedia-owl:College)
educational institution (dbpedia-owl:EducationalInstitution)
library (dbpedia-owl:Library)
scuola (dbpedia-owl:School)
university (dbpedia-owl:University)

Create Class

Cancel

Abb. 5: Zielklassen der Eigenschaft `dbpedia-owl:almaMater`

Um den Nutzern des VSB das Erstellen von Relationseigenschaften zu erleichtern, bietet der VSB zwei Arten von Relationssuchen. Zum einen kann einer existierenden Relationseigenschaft eine passende Klasse hinzugefügt werden, zum anderen kann nach möglichen Relationen zwischen zwei Klassen gesucht werden. In beiden Fällen öffnet sich ein modales Fenster [vgl. Abb. 5], welche dem Nutzer die möglichen Klassen beziehungsweise Eigenschaften vorschlägt. Die Listen in der möglichen Relationen beachtet auch Klassenhierarchien.

## 5 Fazit und Ausblick

Der VSB wurde derzeit noch nicht in Produktivumgebungen evaluiert. Somit sind auch die Metadatenextraktion als auch das Vorschlagssystem noch nicht ausgiebig getestet. Versuche mit den Ontologien der dbpedia und von AMSL während der VSB-Entwicklung zeigten jedoch vielversprechende Ergebnisse. Im Rahmen meiner Bachelorarbeit wird die Tauglichkeit des VSB für die Ontologie des AMSL-Projektes evaluiert werden. Die Reaktionen auf Vorführungen des Visual SPARQL Builder waren meist positiv.

Das größte während der Entwicklung festgestellte Problem ist die Unverträglichkeit des VSB für mangelnde beziehungsweise implizite Definition der Ontologie. Wenn einer Eigenschaft eine explizite Zuordnung zu einer Klasse durch `rdfs:domain` fehlt, kann diese nicht der entsprechenden Klasse zugeordnet werden und somit auch nicht im VSB genutzt werden. Zur Zeit ist im VSB der Umgang mit Interferenzen auch schwierig gestaltet. Liegen zum Beispiel in einer als *Zahl* erkannten Eigenschaft auch *Datums*-Werte vor, ist es nicht möglich Filter für beide Fälle zu definieren. Des Weiteren könnte eine teilweise Übersetzung der Bezeichner und Beschreibungen für die Nutzer verwirrend sein, da die Elemente in Ergebnislisten eventuell in unterschiedlichen Sprachen angezeigt werden.

---

Da der Visual SPARQL Builder als allgemeines Werkzeug programmiert ist, kann er theoretisch mit jeder semantischen Wissensbasis benutzt werden, die einen SPARQL-Endpunkt für den Datenzugriff anbietet. Der einfachste Weg die Eignung des VSB für die Anwendung mit einer eigenen Wissensbasis ist, ihn zu evaluieren und die Open-Source-Entwicklung des Werkzeugs voranzutreiben.

## Literaturverzeichnis

- [Ob11] Object Management Group, Unified Modeling Language Infrastructure Version 2.4.1, <http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF>, Stand: 13.04.2015.
- [St14] Stadler, Claus und Westphal, Patrick und Lehmann Jens: Jassa - A JavaScript Suite for SPARQL-based Faceted Search. In (Verborgh, Ruben und Mannens, Erik, Hrsg.): Proceedings of the ISWC Developers Workshop 2014. Riva del Garda, S. 31–36, 2014.
- [Wo04] World Wide Web Consortium, OWL Web Ontology Language, <http://www.w3.org/TR/2004/REC-owl-semantic-20040210/>, Stand: 13.04.2015.
- [Wo12] World Wide Web Consortium, XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, Abschnitt Primitive Datatypes, <http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/#built-in-primitive-datatypes>, Stand: 13.04.2015.
- [Wo13a] World Wide Web Consortium, SPARQL 1.1 Query Language, <http://www.w3.org/TR/2013/REC-sparql11-query-20130321>, Stand: 13.04.2015.
- [Wo13b] World Wide Web Consortium, SPARQL 1.1 Query Language, Abschnitt Constraints in Optional Pattern Matching, <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/#OptionalAndConstraints>, Stand: 13.04.2015.
- [Wo13c] World Wide Web Consortium, SPARQL 1.1 Query Language, Abschnitt Expressions and Testing Values, <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/#expressions>, Stand: 13.04.2015.
- [Wo14a] World Wide Web Consortium, RDF 1.1 Semantics, <http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/>, Stand: 13.04.2015.
- [Wo14b] World Wide Web Consortium, RDF Schema 1.1, <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>, Stand: 13.04.2015.