

# **Navigation und Selektion in großen semantischen Datensätzen**

Großer Beleg  
Technische Universität Dresden  
Mai 2012

Nikolaus Piccolotto

Betreuer: Dipl.-Medieninf. Martin Voigt,  
Dipl.-Medieninf. Vincent Tietz  
Hochschullehrer: Prof. Dr.-Ing. Klaus Meißner

Fakultät Informatik  
Institut für Software- und Multimediatechnik  
Professur für Multimediatechnik



# Aufgabenstellung

Diese Seite muss vor dem Binden der gedruckten Fassung der Arbeit durch die von Herrn Meißner und dem Studenten eigenhändig unterschriebene originale Aufgabenstellung ersetzt werden. Das zweite abzugebende gebundene Exemplar soll stattdessen eine Kopie dieser originalen Aufgabenstellung enthalten.

# Erklärung

Hiermit erkläre ich, Nikolaus Piccolotto, den vorliegenden Großen Beleg zum Thema

## **Navigation und Selektion in großen semantischen Datensätzen**

selbstständig und ausschließlich unter Verwendung der im Quellenverzeichnis aufgeführten Literatur- und sonstigen Informationsquellen verfasst zu haben.

Dresden, 15. Mai 2012

Unterschrift

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problemstellung und Zielsetzung . . . . .	2
1.3 Aufbau der Arbeit . . . . .	3
<b>2 Stand der Forschung und Technik</b>	<b>4</b>
2.1 Grundlagen . . . . .	4
2.1.1 Ontologien . . . . .	4
2.1.2 Visualisierungskonzepte . . . . .	6
2.1.3 Konzepte für Navigation und Interaktion . . . . .	14
2.2 Szenario . . . . .	20
2.3 Clusteranalyse . . . . .	22
2.3.1 Allgemeine Algorithmen . . . . .	22
2.3.2 Key Concept Extraction . . . . .	25
2.3.3 Profiling Linked Open Data . . . . .	26
2.4 Visualisierung semantischer Daten . . . . .	27
2.4.1 Untersuchungskriterien . . . . .	27
2.4.2 TopBraid Composer . . . . .	28
2.4.3 Jambalaya . . . . .	32
2.4.4 Knoocks . . . . .	38
2.5 Zusammenfassung . . . . .	43
<b>3 Nutzerstudie</b>	<b>45</b>
3.1 Grundlagen . . . . .	45
3.1.1 Effektivität & Effizienz . . . . .	45
3.1.2 Arten der Usabilitybewertung . . . . .	45
3.1.3 Task Load Index . . . . .	47
3.2 Objekt der Studie . . . . .	48
3.3 Methode . . . . .	49
3.4 Durchführung . . . . .	49
3.5 Aufgaben . . . . .	49
3.6 Fragebogen . . . . .	51
3.7 Teilnehmer . . . . .	52
3.8 Diskussion der Ergebnisse . . . . .	52
3.8.1 Effektivität . . . . .	52
3.8.2 Task Load Index . . . . .	53
3.8.3 Aufgaben . . . . .	54
3.8.4 Feedback . . . . .	57
3.9 Zusammenfassung . . . . .	58

<b>4 Konzeption</b>	<b>60</b>
4.1 Anforderungsanalyse . . . . .	60
4.1.1 Overview . . . . .	60
4.1.2 Zoom . . . . .	61
4.1.3 Filter . . . . .	62
4.1.4 Details on demand . . . . .	62
4.1.5 Relate . . . . .	63
4.1.6 History . . . . .	63
4.1.7 Navigation . . . . .	63
4.1.8 CRUISe-spezifische Anforderungen . . . . .	64
4.1.9 Nicht-funktionale Anforderungen . . . . .	65
4.2 Frontend . . . . .	65
4.2.1 Überblick . . . . .	65
4.2.2 Main View . . . . .	67
4.2.3 Buttonleiste . . . . .	71
4.2.4 Hierarchieansicht . . . . .	72
4.2.5 Sidebar . . . . .	73
4.2.6 Navigationsleiste . . . . .	78
4.2.7 Farbgebung & Schrift . . . . .	79
4.2.8 Interaktion . . . . .	80
4.3 Backend . . . . .	81
4.3.1 Funktionen . . . . .	82
4.3.2 Integration in das DaRe . . . . .	83
4.3.3 Schnittstellen . . . . .	84
<b>5 Implementation</b>	<b>88</b>
5.1 Frontend . . . . .	88
5.1.1 Architektur . . . . .	89
5.1.2 Frameworks & Bibliotheken . . . . .	90
5.2 Backend . . . . .	90
5.2.1 Architektur . . . . .	91
5.2.2 Frameworks & Bibliotheken . . . . .	92
5.2.3 Analysephase . . . . .	92
5.2.4 Annotationsphase . . . . .	96
5.2.5 Laufzeit . . . . .	98
5.3 Evaluation . . . . .	101
5.3.1 Nutzerstudie . . . . .	102
5.3.2 Performance . . . . .	104
<b>6 Zusammenfassung und Ausblick</b>	<b>106</b>
6.1 Zusammenfassung . . . . .	106
6.2 Ausblick . . . . .	108
<b>A Anhang</b>	<b>i</b>
A.1 Erste Nutzerstudie . . . . .	i
A.1.1 Teilnehmer . . . . .	i
A.1.2 Lösungen der Aufgaben . . . . .	i
A.1.3 Ergebnisse der Nutzerstudie . . . . .	i

A.2 Zweite Nutzerstudie . . . . .	i
<b>Literaturverzeichnis</b>	<b>vii</b>

# Abbildungsverzeichnis

2.1	Kompromiss zwischen Stärke der Semantik und dem Aufwand, sie zu erstellen . . . . .	5
2.2	Ein Liste in TopBraid Composer . . . . .	7
2.3	Ein TreeView in TopBraid Composer . . . . .	8
2.4	Eine Tabelle in TopBraid Composer . . . . .	8
2.5	Das Netzwerk von Linked Open Datasets . . . . .	9
2.6	Eine Klassenhierarchie mit horizontalem Layout in Jambalaya . . . . .	10
2.7	Eine Klassenhierarchie mit Spring-Layout in Jambalaya . . . . .	10
2.8	Eine Klassenhierarchie mit Radial-Layout in Jambalaya . . . . .	10
2.9	Eine beispielhafte Cluster Map . . . . .	11
2.10	Eine zum damaligen Zeitpunkt aktuelle Treemap-Ansicht des Ordners für diese Belegarbeit . . . . .	12
2.11	CropCircles . . . . .	12
2.12	Power Graphs . . . . .	13
2.13	OntoSphere . . . . .	13
2.14	Google Maps . . . . .	15
2.15	Das Dock aus MacOS X mit Fisheye-Effekt . . . . .	16
2.16	Figuren, die das weiße Pferd bedrohen, hervorgehoben durch Semantic Depth of Field . . . . .	17
2.17	Breadcrumb-Navigation im Windows 7 Explorer . . . . .	17
2.18	Eine Overview & Detail Interaktion mit Protovis . . . . .	18
2.19	Faceted Search bei IEEEexplore . . . . .	19
2.20	Ein Screenshot aus Visor. Es wurde mit den Sets <i>Basketball players</i> , <i>Basketball teams</i> und <i>Popes</i> gestartet, <i>Administrative regions</i> wurde automatisch hinzugefügt. . . . .	20
2.21	Verbindungen zwischen Justin Timberlake und Britney Spears . . . . .	21
2.22	Der k-Means-Algorithmus . . . . .	23
2.23	Mögliche Membership Funktionen von zwei Elementen x und y . . . . .	24
2.24	Hierarchisches Clustering . . . . .	25
2.25	TopBraid Composer mit QUDT . . . . .	29
2.26	Jambalaya mit QUDT . . . . .	33
2.27	Klassenhierarchie als Treemap in Jambalaya . . . . .	33
2.28	Klassenhierarchie als Node-Link mit verschiedenen Layouts in Jambalaya: Radial, Spring, Baum, Baum ohne Labels (v. l. o. n. r. u.) . . . . .	33
2.29	Object Propertys der QUDT in Jambalaya mit vollständig expandierter Hierarchie . . . . .	35
2.30	Klassenhierarchie als Treemap in Jambalaya . . . . .	36
2.31	Datatype Propertys der Ressource nist:PhysicalConstant in Jambalaya . . . . .	37
2.32	Property Editor für nist:ExactConstant in Jambalaya . . . . .	38
2.33	Die Travel Ontologie in Knoocks . . . . .	39

2.34 Eine Infobox für Object Propertys in Knoocks . . . . .	41
2.35 Instanzen und deren Details in Knoocks . . . . .	42
3.1 Aktuelle Datenvorauswahlkomponente in VizBoard mit QUDT . . . . .	48
3.2 Median und Standardabweichung pro TLX Demand und Aufgabe . .	53
4.1 Layout des Frontends . . . . .	66
4.2 Mockup des MainViews . . . . .	67
4.3 Mögliche Varianten in der Darstellung von Knoten . . . . .	68
4.4 Platzierung der Beschriftung einer Kante . . . . .	69
4.5 Overview/Detail Ansicht des Main View . . . . .	70
4.6 Assistent für den Vorgang des Clustering . . . . .	70
4.7 Clustering . . . . .	71
4.8 Verbindungen finden . . . . .	71
4.9 Mockup der Buttonleiste . . . . .	72
4.10 Mockup der Hierarchieansicht . . . . .	72
4.11 Mögliche Layouts für die Hierarchie-Ansicht . . . . .	73
4.12 Mockup der Sidebar . . . . .	74
4.13 Assistent für das Hinzufügen/Bearbeiten von Filtern . . . . .	75
4.14 Mögliche Darstellungen der Filter . . . . .	76
4.15 Die verschiedenen Facets . . . . .	77
4.16 Mögliche Darstellungen der Facets . . . . .	77
4.17 Mockup der Navigationsleiste . . . . .	78
4.18 Mögliche Darstellungen der Zeitleiste . . . . .	78
4.19 Mögliche Darstellung der Pivoting-Funktion . . . . .	79
4.20 Farbschema mit einem Pangramm in PT Sans . . . . .	80
4.21 Eingabe in das Relate-Textfeld . . . . .	81
4.22 Die Aufgaben des Backends . . . . .	82
4.23 Der Workflow von VizBoard . . . . .	83
4.24 Die Struktur des Data Repository . . . . .	84
4.25 Integration des Backends in das DaRe . . . . .	85
5.1 Ein Screenshot des Frontends . . . . .	88
5.2 Die wichtigsten Klassen des Frontends . . . . .	89
5.3 Der Programmablauf, wenn der Benutzer einen Knoten verschiebt .	89
5.4 Integration des Backends in das Data Repository . . . . .	90
5.5 Die zwei Vorverarbeitungsphasen des Backends . . . . .	91
5.6 Die REST API des Data Repository . . . . .	91
5.7 Verkettung der RMonto-Operatoren in RapidMiner zu einem Prozess	93
5.8 Tiefensuche vs. Breitensuche . . . . .	95
A.1 Gelöste Aufgaben . . . . .	iii
A.2 Durchschnitt der Effizienzmetriken pro Aufgabe . . . . .	iii
A.3 Durchschnitt der Effizienzmetriken pro Aufgabe und Nutzergruppe .	iv
A.4 Durchschnitt der Effizienzmetriken pro Aufgabe und Testperson .	iv

# Tabellenverzeichnis

2.1	Größen von Ontologien . . . . .	5
2.2	Bestandteile einer Ontologie . . . . .	22
2.3	Anwendbarkeit von K-Means für Ontologiebestandteile . . . . .	23
2.4	Anwendbarkeit von Fuzzy C-Means für Ontologiebestandteile . . . . .	24
2.5	Anwendbarkeit von hierarchischem Clustering für Ontologiebestandteile	25
2.6	Anwendbarkeit von KCE für Ontologiebestandteile . . . . .	26
2.7	Anwendbarkeit von ProLOD für Ontologiebestandteile . . . . .	27
2.8	Anforderungen . . . . .	29
2.9	Bewertungen für KH von TopBraid Composer . . . . .	30
2.10	Bewertungen für OP und DP von TopBraid Composer . . . . .	31
2.11	Bewertungen für IN von TopBraid Composer . . . . .	31
2.12	Bewertungen von TopBraid Composer . . . . .	32
2.13	Bewertungen für KH von Jambalaya . . . . .	34
2.14	Bewertungen für OP von Jambalaya . . . . .	35
2.15	Bewertungen für IN von Jambalaya . . . . .	36
2.16	Bewertungen für DP von Jambalaya . . . . .	37
2.17	Bewertungen von Jambalaya . . . . .	38
2.18	Bewertungen für KH von Knoocks . . . . .	40
2.19	Bewertungen für OP von Knoocks . . . . .	41
2.20	Bewertungen für IN von Knoocks . . . . .	42
2.21	Bewertungen für DP von Knoocks . . . . .	43
2.22	Bewertungen von Knoocks . . . . .	43
3.1	Fragebogen, allgemeiner Teil . . . . .	51
3.2	Fragebogen, NASA-TLX . . . . .	52
3.3	Fragebogen, freies Feedback . . . . .	52
5.1	Beispiel einer Unähnlichkeitsmatrix von vier Ressourcen . . . . .	93
5.2	Beispiel eines Clusteringmodells . . . . .	94
5.3	Beispiel eines Clusterings . . . . .	94
5.4	Verfügbarkeit der Pivoting-Faktoren nach Ontologiebestandteil . . . . .	101
5.5	Vergleich der Performance . . . . .	105
A.1	Testpersonen und ihre Facetten . . . . .	ii
A.2	Schwierige Teilaufgaben . . . . .	iv
A.3	Einfache Teilaufgaben . . . . .	v
A.4	Schlecht gelöste Dinge . . . . .	v
A.5	Gut gelöste Dinge . . . . .	v
A.6	Ergebnisse der zweiten Nutzerstudie . . . . .	vi



# 1 Einleitung

Semantische Daten sind maschinenlesbare Datenformate, die eingeführt wurden, damit Computer die Semantik von Daten verstehen können. Menschen können das intuitiv: Wenn sie ein Bild von einem Baum gezeigt bekommen, erkennen sie den Baum. Ein Computer hingegen sieht nur eine Menge an eingefärbten Pixeln. Um domänenspezifisches Wissen aus einer Ontologie in semantischen Daten darstellen zu können, wird auf Konzepte wie Begriffe, Typen, Instanzen, Relationen, Vererbung und Axiome zurückgegriffen. So lassen sich auch sehr abstrakte Sachverhalte abbilden.

## 1.1 Motivation

Im Zuge der Entwicklung des Semantic Webs und durch verschiedene Initiativen wie zum Beispiel Open Government wurden im Internet viele verschiedene und in der Regel sehr große semantische Datensätze verfügbar. Ein Beispiel dafür ist die DBpedia<sup>1</sup>, ein Projekt welches die Wikipedia in semantische Daten überführt und Informationen über mehr als 364 Millionen »Dinge« bietet [DBp11] oder die Gene Ontology<sup>2</sup>, welche einheitliche Bezeichnungen für Gene enthält und besonders im Forschungsbereich der Biologie verwendet wird. Auch im E-Commerce werden semantische Daten eingesetzt, z. B. die GoodRelations Ontologie<sup>3</sup>, welche Produkteigenschaften für Suchmaschinen lesbar darstellt.

Durch die vielen unterschiedlichen Einsatzzwecke für Ontologien können sowohl Maschinen und Experten als auch alle anderen Menschen davon profitieren. Die GoodRelations Ontologie optimiert die Darstellung von Produkten für Suchmaschinen, was dazu führt, dass bessere Suchergebnisse erzielt werden können. Die Käufer sind dadurch zufriedener und die Verkäufer verdienen mehr Geld. Die Gene Ontology auf der anderen Seite erlaubt es Biologen, ein einheitliches dynamisches Vokabular für Gene zu benutzen, obwohl sich deren Klassifikation und das Wissen über sie im Laufe der Zeit ändert. Das macht es für die Wissenschaftler einfacher, sich auszutauschen. Die DBpedia hingegen ist sozusagen ein allgemeiner Wissensspeicher. Ihre Daten können alle Menschen nutzen, die Fragen haben, deren Antworten sie selbst sonst entweder nicht recherchieren könnten oder deren manuelle Recherche zu lange dauern würde.

Semantische Daten werden als formatierter Text abgespeichert. Damit sind sie für Maschinen problemlos lesbar, Menschen tun sich naturgemäß schwer, große Mengen an nicht für sie optimierten Text zu verarbeiten. Eine visuelle Repräsentation ist

---

<sup>1</sup><http://dbpedia.org>

<sup>2</sup><http://www.geneontology.org/>

<sup>3</sup><http://www.heppnetz.de/projects/goodrelations/>

für sie besser geeignet. Sie kann helfen, sich einen Überblick über große Daten zu verschaffen, Patterns zu erkennen und Zusammenhänge zu verstehen. Dabei müssen einige Problemstellungen besonders beachtet werden.

## 1.2 Problemstellung und Zielsetzung

Bei einem Datensatz mit Informationen über 100.000 Wissensobjekte und mehr ist es nicht ungewöhnlich, wenn mehrere zehntausend Elemente gleichzeitig dargestellt werden müssen. Ohne weitere Hilfe kann ein Mensch das nicht verarbeiten und es kommt zum »Information Overload«. Wie verschafft die Software dem Benutzer aber einen Überblick über zehntausende Wissensobjekte? Wie kann der Benutzer eine für ihn relevante Teilmenge davon identifizieren und eingrenzen? Bei so vielen dargestellten Elementen gehen deren Verbindungen untereinander leicht unter, aber gerade diese sind oft von Interesse. Es muss ein Weg gefunden werden, um sie sichtbar zu machen. Auch die Anordnung der Objekte ist von Bedeutung, denn von ihr hängt ab, ob Patterns sichtbar werden oder nicht. Ein weiteres Problem im Umgang mit einem so großen Datensatz ist die Navigation. Wie kommt der Benutzer am Besten von einem Objekt oder einer relevanten Teilmenge zur nächsten? Wie kann sichergestellt werden, dass er immer weiß, an welcher Stelle er sich im Datensatz befindet und wie er dahin gekommen ist? Eine visuelle Repräsentation so vieler Daten wird sich zwangsläufig auf das Nötigste beschränken müssen, um dem »Information Overload« vorzubeugen. Wie kommt der Benutzer aber an die Daten, die nicht sichtbar sind und wie kann er sie durchsuchen? Wie kann er überhaupt herausfinden, dass noch Daten vorhanden sind, wenn er sie nicht sehen kann?

Das Ziel dieser Arbeit ist die Entwicklung eines Konzepts für eine Software, welche es Anfängern und Experten gleichermaßen ermöglicht, relevante Informationen aus großen semantischen Datensätzen zu extrahieren. Dabei soll der Fokus besonders auf einer Lösung für die zuvor beschriebenen, für große Datensätze spezifischen Probleme liegen. Die visuelle Präsentation soll sich auf das Wesentliche beschränken und das Bedienungskonzept muss den Benutzer bei der Navigation in dem großen Datensatz unterstützen. Das erarbeitete Konzept soll in einem Prototypen als CRUISe-Komponente umgesetzt werden. CRUISe ist ein Framework für die Erstellung von Mashup-Anwendungen [Pie+09] und dient als Basis für verschiedene andere Forschungsprojekte wie VizBoard [VPM12] und DEMISA [Tie+11].

Daraus leiten sich die folgenden Teilziele der Arbeit ab: Da der Fokus auf großen Datensätzen liegt, müssen zuerst geeignete Visualisierungskonzepte für große Ontologien identifiziert werden. Auch bestehende Konzepte für Navigation und Interaktion müssen auf ihre Tauglichkeit und Einsetzbarkeit in großen Datensätzen untersucht werden. Zudem sollten verschiedene Möglichkeiten zur automatischen Extraktion von relevanten Daten oder nutzbaren Metadaten untersucht werden. Die Probleme, welche sich bei großen Datensätzen stellen, sollten anhand bestehender Visualisierungstools noch einmal verdeutlicht und daraus Ansätze zur Lösung der selben entwickelt werden. Weiterer Input für das zu entwickelnde Konzept kann aus einer Nutzerstudie von einem bereits bestehenden Mockup bezogen werden.

## 1.3 Aufbau der Arbeit

Im folgenden Kapitel wird zunächst auf die Grundlagen von Ontologien, Visualisierungs-, Navigations- und Interaktionskonzepten sowie der Clusteranalyse eingegangen. Außerdem werden drei Visualisierungstools auf ihre Nutzbarkeit mit großen semantischen Datensätzen untersucht. Kapitel 3 befasst sich mit der Nutzerstudie des Mockups, wo dessen Probleme identifiziert werden. Anschließend erfolgt die Entwicklung eines Konzepts einer Software, welche für den Umgang mit großen semantischen Datensätzen geeignet ist. In Kapitel 5 wird auf die Umsetzung des Prototypen eingegangen und dessen Evaluation diskutiert. Im letzten Kapitel wird die Arbeit mit einer Zusammenfassung und einem Ausblick abgeschlossen.

## 2 Stand der Forschung und Technik

Der folgende Abschnitt gliedert sich in vier Teile. Zuerst werden die Grundlagen von Ontologien, semantischen Datenformaten wie RDF oder OWL, Visualisierungs- und Interaktionskonzepten erklärt. Im nächsten Abschnitt wird das Szenario, auf das sich die darauf folgenden Abschnitte beziehen werden, eingeführt. Danach werden Algorithmen zur Clusteranalyse eingeführt. Zum Schluss werden verfügbare Visualisierungen für semantische Daten vorgestellt und es folgt eine Zusammenfassung.

### 2.1 Grundlagen

In den folgenden Abschnitten wird auf die Grundlagen von ontologischen Datenformaten sowie Visualisierungs- und Bedienungskonzepten eingegangen.

#### 2.1.1 Ontologien

Nach Gruber [RG95] ist eine Ontologie »an explicit specification of a conceptualization«, also eine konkrete Spezifikation einer Begriffsbildung. Der Begriff stammt aus der Philosophie, dort bezeichnet er die Theorie von grundlegenden Strukturen der Realität. In der Informatik werden Ontologien zur Wissensbeschreibung verwendet und helfen u. a. bei Sprachdatenverarbeitung, Datenbankdesign, Information Retrieval, Übersetzung von Sprachen sowie in der Medizin, Biologie und geografischen Informationssystemen [Gua98].

Eine Ontologie besteht aus mehreren Elementen.

- Eine **Klasse** repräsentiert eine Entität, ein Etwas (z. B. Tier oder Fluss).
- Eine **Instanz** ist ein konkretes Objekt einer Klasse (z.B. Aal oder Donau).
- **Attribute** beschreiben eine Instanz näher (z.B. Name oder Länge).
- **Relationen** verbinden Klassen und deren Instanzen (z. B. Tier »schwimmt in« Fluss). Eine spezielle Relation ist »is-a«, sie beschreibt Vererbung und damit eine Klassenhierarchie. Z. B. Fisch »ist ein« Tier.
- Außerdem existieren noch **Einschränkungen**, **Axiome**, **Regeln** und **Funktionen**. Diese beschreiben die Logik einer Ontologie.

Die Größe einer Ontologie kann als Summe der enthaltenen Ressourcen beschrieben werden. Ernst & Storey [ES03] verwendeten Klassen und Instanzen als Metrik und identifizierten fünf Größen, die hier mit einem sprachlichen Attribut versehen wurden um Missverständnisse zu vermeiden (Tabelle 2.1).

0 - 100	»sehr klein«
101 - 1000	»klein«
1001 - 10000	»mittel«
ab 10001	»groß«

Tabelle 2.1: Größen von Ontologien

Um eine Ontologie maschinenlesbar darzustellen, hat das *World Wide Web Consortium* (W3C) verschiedene Beschreibungssprachen eingeführt. Die bekanntesten sind *Resource Description Framework Schema* (RDFS) und *Web Ontology Language* (OWL). Mit diesen Sprachen lässt sich unterschiedlich viel Semantik ausdrücken, die Komplexität der Dokumente und damit der Aufwand, sie zu erstellen, verhalten sich allerdings direkt proportional (siehe Abb. 2.1) [Ber07].

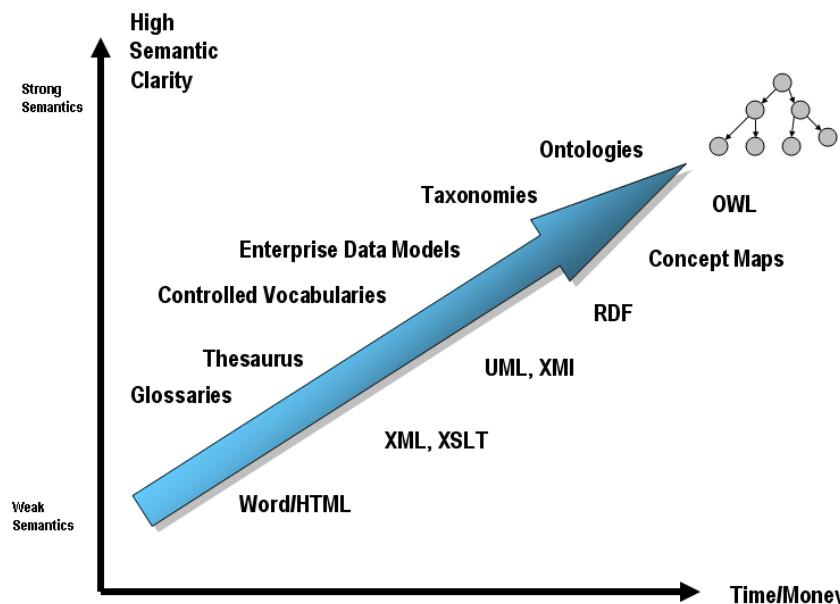


Abbildung 2.1: Kompromiss zwischen Stärke der Semantik und dem Aufwand, sie zu erstellen

## RDF

Das *Resource Description Framework* (RDF) ist seit 2004 eine Recommendation des W3C [W3C04c] und damit ein Web-Standard. Ursprünglich war es als Beschreibungssprache für Metadaten gedacht. RDF kann in verschiedenen Formaten serialisiert werden, z. B. XML, N3 oder Turtle.

Zentraler Bestandteil ist die Ressource, welche mit einem eindeutigen *Uniform Resource Identifier* (URI) identifiziert wird. Diese URI kann, aber muss keine gültige Web-Adresse sein. Mit RDF können nun Beziehungen zwischen Ressourcen in der Form (Subjekt, Prädikat, Objekt) repräsentiert werden, z. B. (<http://inf.tu-dresden.de/alice>, <http://purl.org/dc/elements/1.1/creator>, <http://alice-bob.com>) um

auszusagen, dass die Entität »Alice« »Ersteller der Ressource« »Homepage von Alice & Bob« ist. Allerdings können Ressourcen noch nicht in Klassen, Propertys o. ä. unterteilt oder Logiken beschrieben werden. Das ist erst mit RDFS bzw. OWL möglich.

## RDFS

Das RDF Schema (abgekürzt durch RDFS, RDF(S), RDF-S oder RDF/S) ist wie RDF seit 2004 eine W3C Recommendation [[W3C04b](#)]. RDFS erweitert das Vokabular von RDF um

- Klassen und Subklassen,
- Datatype Propertys (Attribute), Object Propertys (Relationen) und SubPropertys,
- Instanzen von Klassen.

RDFS ist damit noch nicht so mächtig wie OWL, aber es können Ontologien beschrieben werden.

## OWL

Die Web Ontology Language (OWL) [[W3C04a](#)] wurde wie RDF und RDFS am 10. Februar 2004 zur W3C Recommendation erklärt. Sie basiert auf RDFS und erweitert es noch um diverse Eigenschaften. Zum Beispiel können Axiome definiert werden, Propertys können transitiv oder symmetrisch sein, Kardinalitäten und Werte können eingeschränkt werden und es gibt Mengenoperationen. Modularisierung bietet viele Vorteile wie z. B. bessere Strukturierung komplexer Ontologien, Wiederverwendbarkeit oder unabhängiges Arbeiten an Teilen. Deswegen sind in OWL auch Importe möglich, mit denen Ontologien eingebunden werden können. Da meistens nicht der volle Umfang von OWL benötigt wird, hat das W3C die Untersprachen OWL Lite und *OWL Description Language* (OLWL-DL) eingeführt. OWL Lite ist für Anwendungen gedacht, wo nur eine Klassenhierarchie und wenige Einschränkungen gebraucht werden. OWL-DL fokussiert sich auf die Logik einer Ontologie. Deswegen wird es besonders für Reasoner eingesetzt. Ein Reasoner ist eine Software, die selbstständig Schlüsse in einem gegebenen formalen System ziehen kann. Diese können zum Beispiel Tumore in Röntgenbildern automatisch erkennen [[LOR09](#)].

Seit Oktober 2009 existiert der OWL 2 Standard [[W3C09](#)]. Dieser erweitert wiederum OWL von 2004 um z. B. asymmetrische, reflexive und disjunkte Propertys, mehr Cardinality Restrictions oder verbesserte Annotationsmöglichkeiten.

### 2.1.2 Visualisierungskonzepte

Dieser Abschnitt soll einen Überblick über Visualisierungskonzepte von semantischen Daten geben. Ein Visualisierungskonzept soll hier die visuelle Aufbereitung semantischer Daten sein. Die Auswahl der Konzepte orientiert sich an Katifori [[Kat+07](#)]. Im Gegensatz zu den später beschriebenen Visualisierungen (Abschnitt

2.4), die üblicherweise mehrere Visualisierungskonzepte umsetzen und eine ganze Ontologie darstellen sollen, kann sich ein Konzept auf einen Teil davon beschränken. Außerdem wird die Interaktion mit dem Benutzer hier nicht betrachtet.

Zuerst wird das Konzept allgemein erläutert, dann die Eigenschaften desselben aufgezeigt und zum Schluss mögliche Anwendungsfälle für semantische Datensätze genannt.

## Tabellen

Tabellen bestehen aus einer Menge an Zeilen und Spalten, die in der Regel formatierten Text beinhalten. Listen und TreeViews sind Spezialfälle einer Tabelle. Eine Liste ist eine Tabelle mit einer Spalte und ein TreeView eine Liste mit versteckten Zeilen, die vom Benutzer ein- und ausgeblendet werden können (Expand/Collapse). Alle drei Ausprägungen von Tabellen haben ein klares Layout ohne Überlappungen von Elementen oder Schrift gemein, allerdings nutzen sie den Platz am Bildschirm nicht optimal aus, da immer nur ein Teil der Daten sichtbar ist.

Listen (Abb. 2.2) sind für kurze Aufzählungen besonders geeignet, z. B. für die letzten 10 selektierten Elemente in einer Visualisierung oder für Lesezeichen. Längere Aufzählungen sind auch möglich, dann müssen aber bestimmte Hilfestellungen vorhanden sein. Ein Beispiel dafür ist die Volltextsuche, die Elemente ausblenden oder die Liste dynamisch neu sortieren kann. Ein andere Möglichkeit ist die Liste in kleinere Listen mit einer festen Anzahl von Elementen zu unterteilen (Pagination).

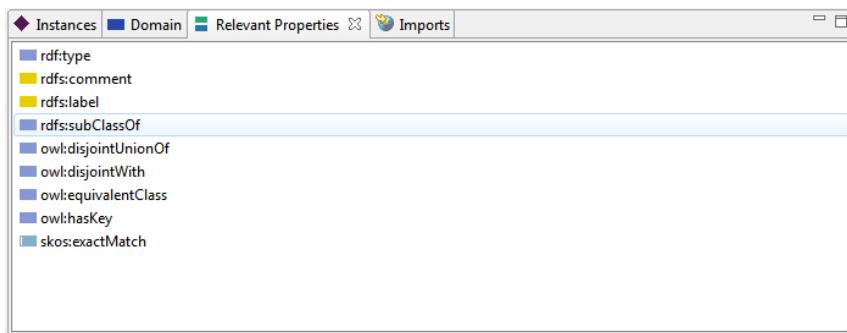


Abbildung 2.2: Ein Liste in TopBraid Composer

TreeViews (Abb. 2.3) werden besonders für die Darstellung von Hierarchien verwendet, da diese wegen der eindeutigen Anordnung und Einrückung von Elementen intuitiv klar ist. Außerdem ist dieses Konzept vielen Menschen aus dem Windows Explorer und anderen Dateibrowsern bekannt. Die Eigenschaften eines TreeViews sind ähnlich wie die einer Liste, bei vielen Elementen ist eine Volltextsuche für den Benutzer nützlich. Die angezeigten Elemente durch Pagination zu reduzieren ist bei TreeViews unüblich, weil durch Pagination die Zeilenanzahl fest sein soll, TreeViews diese aber dynamisch verändern.

Tabellen eignen sich ähnlich wie Listen für kurze Aufzählungen, aber mit zusätzlichen Informationen. Überblicke, Vergleiche oder Statistiken lassen sich so realisieren, besonders wenn die Zeilen nach Werten in Spalten sortiert werden können. Zum

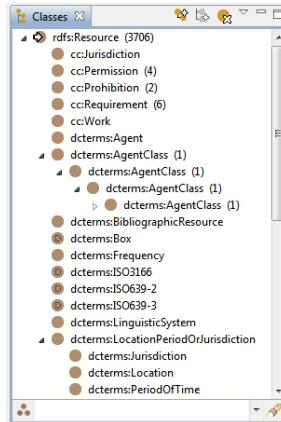


Abbildung 2.3: Ein TreeView in TopBraid Composer

Beispiel einer Übersicht über Instanzen im semantischen Datensatz (siehe Abb. 2.4) Tabellen haben die zusätzliche Anforderung, dass sie auch in die Breite nicht zu groß werden dürfen. Volltextsuche und Pagination können bei zu großer Länge dem Nutzer helfen.

[Resource]	rdf:type	rdfs:label	rdfs:comment
cc:Attribution	cc:Requirement	Attribution	credit be given to copyright...
cc:CommercialUse	cc:Prohibition	Commercial Use	exercising rights for commercial...
cc:Copyleft	cc:Requirement	Copyleft	derivative and combined works...
cc:DerivativeWorks	cc:Permission	Derivative Works	distribution of derivative works...
cc:Distribution	cc:Permission	Distribution	distribution, public display...
cc:HighIncomeNationUse	cc:Prohibition	High Income Nation Use	use in a non-developing country...
cc:Jurisdiction	rdfs:Class	Jurisdiction	the legal jurisdiction of a country...
cc:LesserCopyleft	cc:Requirement	Lesser Copyleft	derivative works must be freely distributed...
cc:License	rdfs:Class, owl:Class	License	a set of requests/permissions...
cc:Notice	cc:Requirement	Notice	copyright and license notice...
cc:Permission	rdfs:Class	Permission	an action that may or must be taken...
cc:Prohibition	rdfs:Class	Prohibition	something you may be allowed to do...
cc:Reproduction	cc:Permission	Reproduction	making multiple copies...
cc:Requirement	rdfs:Class	Requirement	an action that may or must be taken...
cc:ShareAlike	cc:Requirement	Share Alike	derivative works be licensed...

Abbildung 2.4: Eine Tabelle in TopBraid Composer

## Node-Link

Graphen und Bäume werden unter dem Begriff »Node-Link Diagramm« zusammengefasst und für die Darstellung von Netzwerkdaten benutzt. Ein Graph [Bol98]  $G = (V, E)$  besteht aus einer disjunkten Menge von Knoten  $V$  (Vertices, Nodes) und Kanten  $E = \{(u, v) | u, v \in V, u \neq v\}$  (Edges), die diese verbinden. Seien im Folgenden  $u, v \in V$  Knoten und durch eine Kante  $(u, v) \in E$  verbunden. Man unterscheidet unter anderem zwischen gerichteten und ungerichteten Graphen. Bei ungerichteten Graphen beschreibt eine Kante eine symmetrische Verbindung, d. h.  $u$  ist mit  $v$  verbunden und umgekehrt. Bei gerichteten Graphen hingegen verbindet  $(u, v)$  nur  $u$  mit  $v$ . Man nennt  $u$  den »Vorgänger« von  $v$  und  $v$  den »Nachfolger« von  $u$ . Ein Baum ist nun ein Spezialfall eines gerichteten Graphen, wo jeder Knoten nur einen Vorgänger hat. Es existieren außerdem noch Multigraphen, in denen zwei Knoten durch mehrere Kanten verbunden sein können.

Graphen eignen sich naturgemäß sehr gut für Daten, die ein Netzwerk beschreiben (Abb. 2.5, [CJ11]). Ein großes Problem von Graphen ist die schlechte Skalierbarkeit. In großen Datensätzen mit vielen Knoten und Kanten überdecken sich diese gegenseitig. Eine mögliche Lösung ist eine geschicktere Anordnung der Knoten (Layoutalgorithmus), sodass die Anzahl an sich überschneidenden Kanten minimiert wird. Außerdem kann der Graph eventuell mit einer reduzierten Version dargestellt werden, ohne dass viel Information verloren geht, aber der Nutzer sich besser zurechtfindet. Dieser kann eventuell mit Minimum Spanning Trees [GR69] oder Pathfinder Networks [Sch90] berechnet werden. Weitere Varianten wären nur Knoten ab einem bestimmten Gewicht anzusehen oder Knoten mittels Clusteringalgorithmen zusammenzufassen.

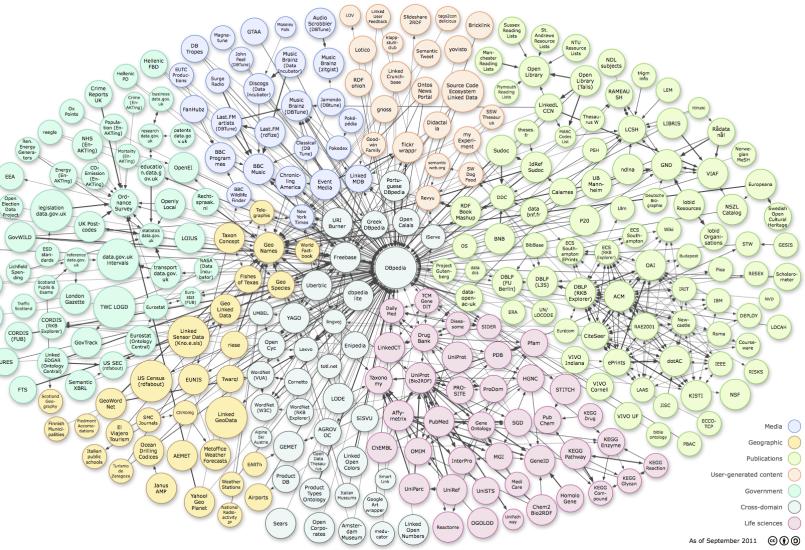


Abbildung 2.5: Das Netzwerk von Linked Open Datasets

Bäume eignen sich besonders für Hierarchien<sup>1</sup>, da diese ähnlich wie bei TreeViews durch die eindeutige Richtung der Pfeile intuitiv klar ist. Allerdings trägt hier wie bei allen Graphen der verwendete Layoutalgorithmus viel zur Wahrnehmung bei (vgl. Abb. 2.6 und 2.7). Auch über die Platzausnutzung entscheidet der Layoutalgorithmus: Bei vertikalem oder horizontalem Layout ist der Bereich um die Wurzel üblicherweise recht leer und schlecht genutzt, radiales Layout kann das verbessern (Abb. 2.8).

## Cluster Maps

Cluster Maps wurden von Fluit et al. [FSH05] vorgeschlagen. Sie clustern die Instanzen nach Klassen und hinterlegen sie mit unterschiedlichen Farben. Dabei stellen sie Mehrfachvererbung besonders gut dar. Abb. 2.9 zeigt eine Klassenhierarchie. *Job Vacancies* ist die Superklasse, *IT*, *Technology* und *Management* sind Subklassen davon. Die gelben Kugeln sind die Instanzen der jeweiligen Klassen. Es ist gut zu

<sup>1</sup>Bäume können nur Hierarchien darstellen, bei denen ein Element maximal einen Vorgänger haben kann. Ansonsten repräsentiert die Hierarchie einen Graphen.

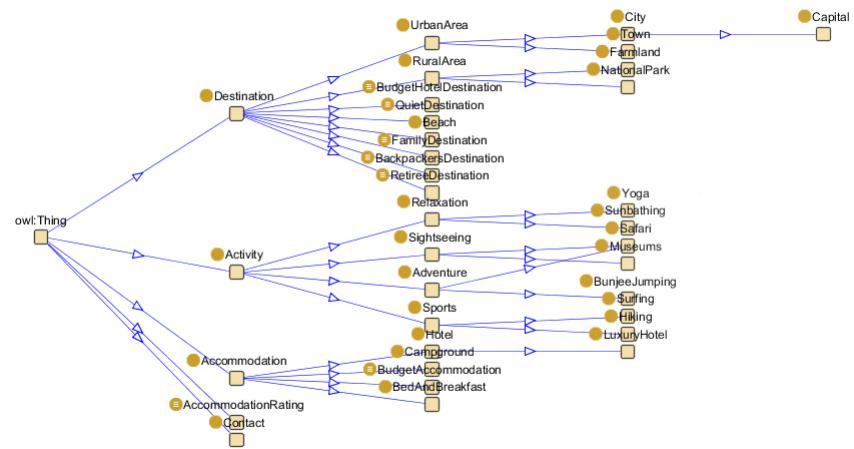


Abbildung 2.6: Eine Klassenhierarchie mit horizontalem Layout in Jambalaya

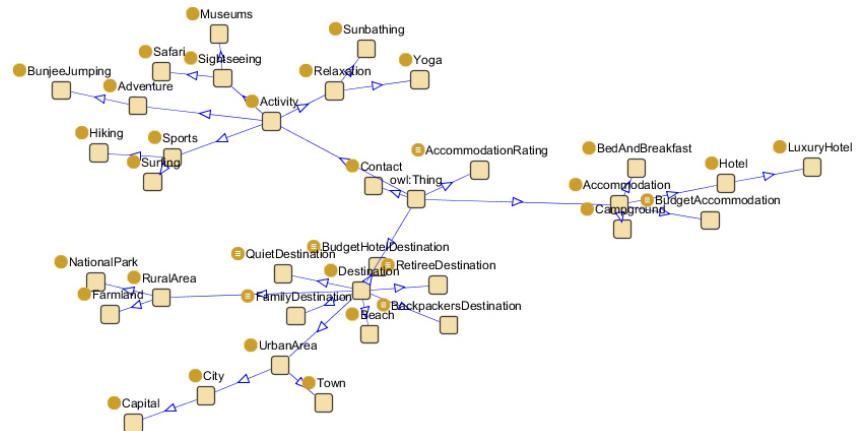


Abbildung 2.7: Eine Klassenhierarchie mit Spring-Layout in Jambalaya

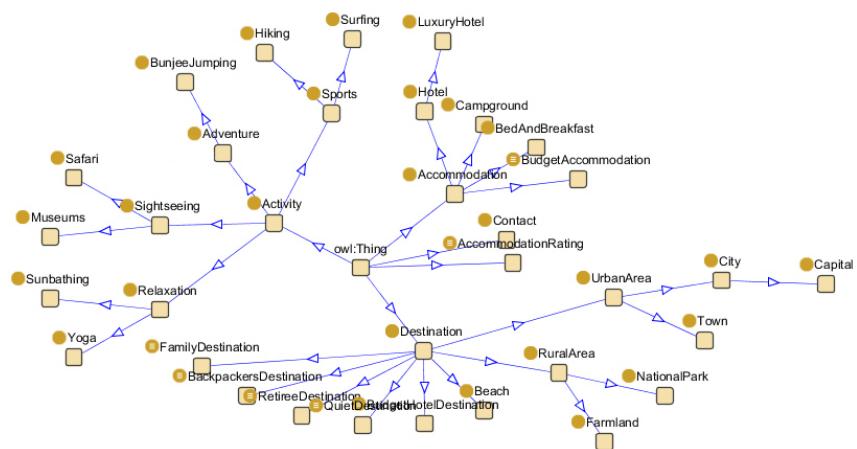


Abbildung 2.8: Eine Klassenhierarchie mit Radial-Layout in Jambalaya

sehen und sofort verständlich, dass es mehr Jobs aus dem Bereich IT und Technology als Technology und Management gibt. Diese Instanzen sind Ausprägungen von zwei Klassen (Mehrfachvererbung).

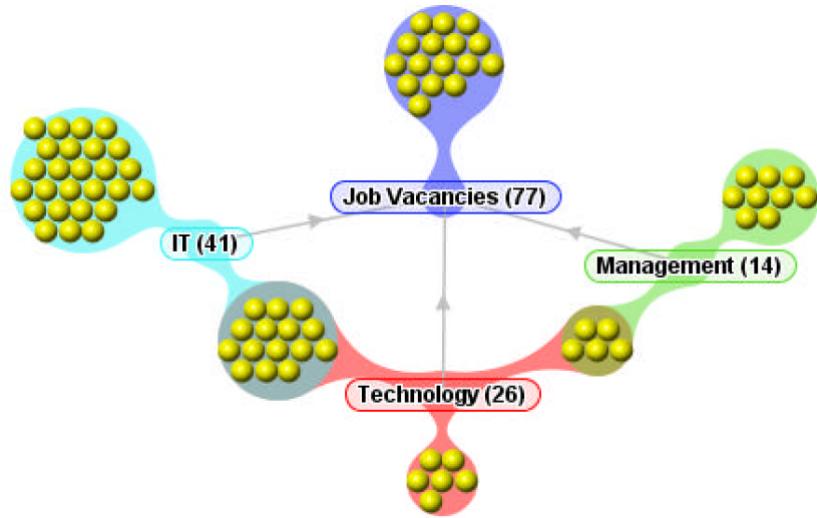


Abbildung 2.9: Eine beispielhafte Cluster Map

In ihrem Paper sagen Fluit et al. selbst, dass Cluster Maps für »light-weight ontologies«, also kleine Ontologien, konzipiert wurde. Ein Layout, welches die Überlappung von Verbindungen minimieren und gleichzeitig die Übersichtlichkeit für den Benutzer gewährleisten kann, ist bei großen Datensätzen vermutlich sehr schwer zu finden.

### Nested Views

Verschachtelte Darstellungen von hierarchischen Daten werden als Nested Views bezeichnet. Dabei werden die Kinder eines Elements im Element selbst dargestellt.

Die Treemap (Abb. 2.10) wurde von Ben Shneiderman entwickelt [Shn92]. Die Kinder eines Elements werden hier proportional zur Größe des Elternelements gezeichnet. Dieses Konzept bietet eine optimale Platzausnutzung, weil alle Elemente auf dem am Bildschirm verfügbaren Rechteck dargestellt werden. Nachteile sind die eventuelle Unlesbarkeit von Labels, wenn zu viele Kinder dargestellt werden, und der teilweise Verlust der Reihenfolge durch die Bedingung der Platzausnutzung. Da dieses Visualisierungskonzept von einer Baumstruktur ausgeht<sup>2</sup>, müssten bei Mehrfachvererbung im semantischen Datensatz betroffene Knoten zweimal gezeichnet werden.

CropCircles [WP06] ist wie eine Treemap ein Nested View, bei dem Kinder innerhalb ihrer Eltern dargestellt werden. Allerdings werden hier Kreise statt Rechtecke benutzt (Abb. 2.11). Die Anordnung der Elemente ist hier egal, weil der Layoutalgorithmus nicht platzfüllend arbeitet, deswegen kann die Reihenfolge der Elemente

<sup>2</sup>Shneiderman suchte nach einem Algorithmus, um seine gefüllte Festplatte übersichtlich darzustellen, sodass große Verzeichnisse sofort erkannt werden können [Shn98].

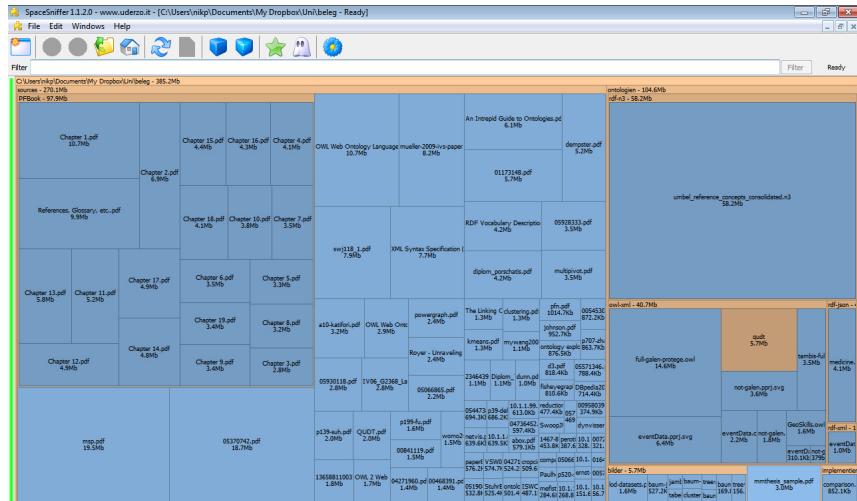


Abbildung 2.10: Eine zum damaligen Zeitpunkt aktuelle Treemap-Ansicht des Ordners für diese Belegarbeit

erhalten werden. Allerdings verbraucht diese Darstellung im Vergleich zur Treemap relativ viel Platz. Das Konzept an sich ist einfach verständlich, wenn auch anfangs vielleicht gewöhnungsbedürftig.

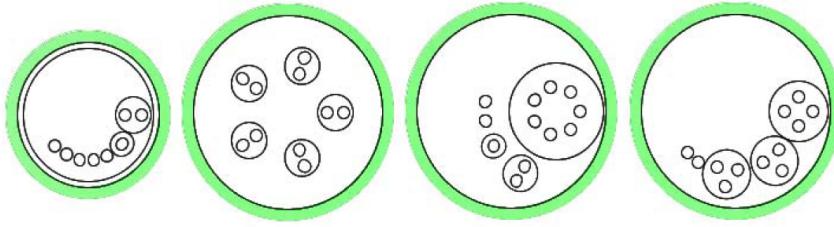


Abbildung 2.11: CropCircles

Power Graphs [Roy+08] (Abb. 2.12) wurden entwickelt, um Proteinnetzwerke besser visualisieren zu können. Genau genommen sind sie ein Hybrid aus Node-Link und Nested Views. Für Power Graphs wird die Definition eines Graphen um sogenannte Power Nodes und Power Edges erweitert. Power Nodes enthalten mehrere normale Knoten und Power Edges verbinden Power Nodes. Die Semantik von Power Edges ist, dass jeder normale Knoten im ersten Power Node mit allen normalen Knoten im zweiten Power Node verbunden ist. So lässt sich die visuelle Komplexität eines Graphen deutlich reduzieren und Verbindungsmuster werden besser sichtbar.

Power Graphs könnten bei semantischen Daten zur Visualisierung einer Hierarchie mit Mehrfachvererbung eingesetzt werden.

### 3D

Dreidimensionale Visualisierungen bieten viele Möglichkeiten, weil Daten auf einer zusätzlichen Achse dargestellt werden können. Außerdem wirken Größenverhältnisse in 3D für den Benutzer realistischer, weil 3D der alltäglichen Wahrnehmung entspricht. Leider werden 3D Visualisierungen nicht so realistisch wahrgenommen, wenn

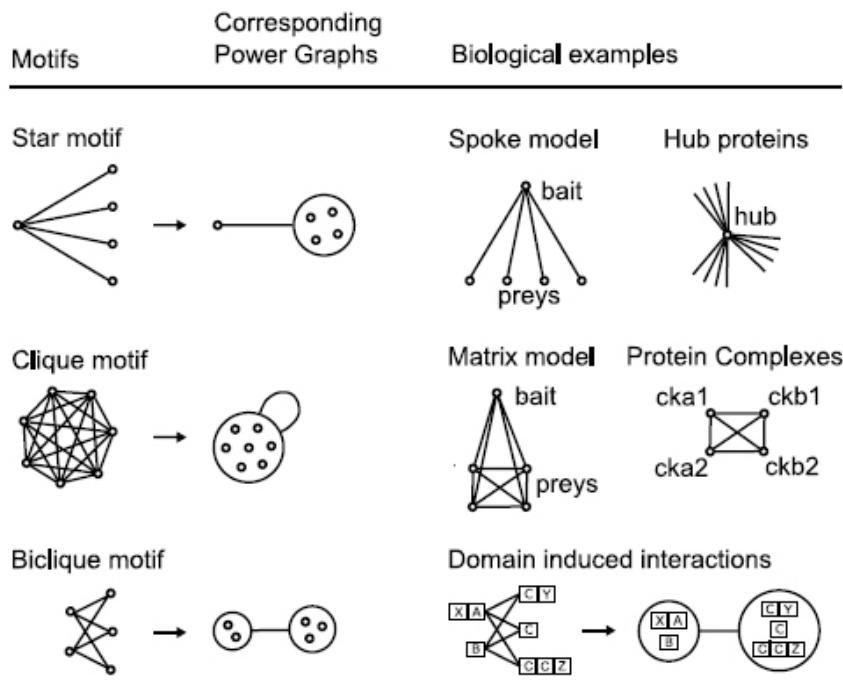


Abbildung 2.12: Power Graphs

sie auf zwei Dimensionen projiziert werden. Auch die Navigation in drei Dimensionen ist schwieriger, wenn sie mit Interaktionen in zwei Dimensionen gesteuert werden soll.

Ein Ansatz semantische Daten in 3D darzustellen ist OntoSphere [BBP05]. OntoSphere stellt verschiedene Ansichten zur Visualisierung der Ontologie zur Verfügung. Abb. 2.13 zeigt die Übersicht (a) in der Konzepte an der Oberfläche der Kugel verteilt werden. In der Detailansicht (b) kann der Benutzer eine selektierte Klasse und dessen Kinder genauer ansehen. Katifori et al. [Kat+07] merkten in ihrer Studie

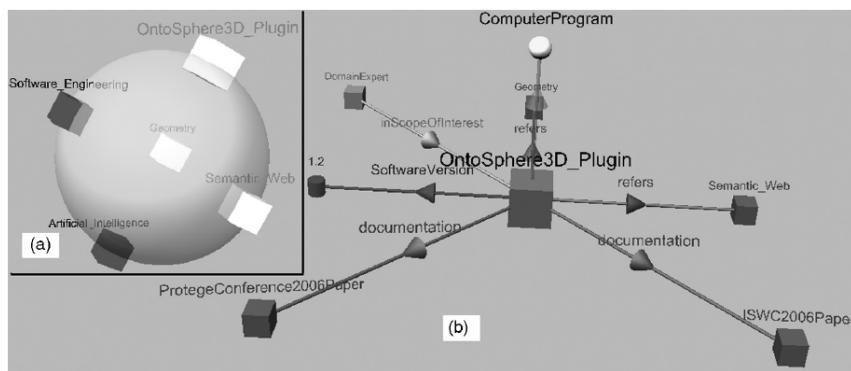


Abbildung 2.13: OntoSphere

an, dass OntoSphere mit größeren Ontologien (mehr als 1000 Knoten) nicht gut umgehen kann.

### 2.1.3 Konzepte für Navigation und Interaktion

Nachdem mögliche visuelle Darstellungen betrachtet wurden, werden in diesem Abschnitt Konzepte für die Navigation in und Interaktion mit der Visualisierung betrachtet. Der Fokus dieser Arbeit liegt auf großen semantischen Datensätzen. Unabhängig vom gewählten Visualisierungskonzept wird der Benutzer wegen der Menge an Daten Probleme haben, sich zurechtzufinden. Deswegen ist es nötig ihn bei der Navigation und dem Identifizieren von relevanten Daten zu unterstützen.

Herman [HMM00] identifizierte drei Gruppen dieser Konzepte: Zoom & Pan, Focus & Context und Incremental Exploration and Navigation. Die erste beschreibt das im Folgenden vorgestellte Pan & Zoom<sup>3</sup>. Focus & Context ist ebenfalls ein eigener Abschnitt in diesem Kapitel. Hermans Incremental Exploration and Navigation entspricht sehr genau dem später vorgestellten Overview & Detail. Verallgemeinert man die Beschreibung allerdings soweit, dass das »Fenster« nicht sichtbar sein muss, sondern eine logische Teilansicht des Datensatzes sein kann, fallen auch die Konzepte Faceted Navigation, Magnet, Breadcrumbs und Multi-Pivoting in diese Kategorie. Die einzelnen Konzepte werden zuerst allgemein beschrieben, dann folgt ein Beispiel und zum Schluss wird ein möglicher Einsatz im Bereich semantischer Daten genannt.

Um zu erklären, wie eine Interaktion definiert werden kann und wo die Navigation darin einzuordnen ist, wird kurz der »Human Interaction Cycle« (auch »Seven Stages of Action«) von Norman [Nor86] vorgestellt. Eine Interaktion besteht demnach aus sieben Schritten des Nutzers.

1. **Formulieren des Ziels:** Der Nutzer bestimmt den Endzustand des Systems.  
Zum Beispiel die Tür am Ende des Raumes ist offen.
2. **Festlegen der Absicht:** Der Nutzer bestimmt, wie er sein Ziel erreichen will.  
Zum Beispiel selbst die Tür öffnen oder jemanden darum bitten.
3. **Festlegen der Aktionssequenz:** Der Nutzer unterteilt seine Absicht in einzelne Aktionen. Zum Beispiel ans Ende des Raumes gehen, die Klinke drücken und die Tür ziehen.
4. **Ausführen der Aktionssequenz:** Der Nutzer führt seine festgelegten Aktionen aus.
5. **Überprüfen des Systemzustands:** Der Nutzer überprüft wie das System auf seine Aktionen reagiert hat. Ist die Tür offen oder nicht?
6. **Interpretieren des Systemzustands:** Der Nutzer sucht einen Grund für den Zustand des Systems. Zum Beispiel wenn die Tür noch geschlossen ist, hat er in die falsche Richtung gedrückt oder ist sie abgeschlossen?
7. **Evaluieren des Systemzustands:** Der Nutzer entscheidet, ob der Systemzustand seinen Zielen und Absichten genügt und fängt gegebenenfalls noch einmal bei Schritt 1 an.

---

<sup>3</sup>Der Autor findet diese Kombination leichter auszusprechen.

Norman identifizierte außerdem zwei gängige Fälle, welche die sieben Schritte unterbrechen und deswegen den Benutzer frustrieren.

- Der **gulf of execution** tritt ein, wenn der Benutzer seine Aktionssequenz nicht vollständig ausführen kann. Um bei obigem Beispiel zu bleiben, z. B. wenn auf dem Weg zur Tür ein Wachmann das Weitergehen verbietet.
- Der **gulf of evaluation** ist gegeben, wenn der Benutzer den Systemzustand nicht überprüfen kann, weil das System sich nicht sichtbar verändert hat. Dann ist nicht klar, ob die Aktionssequenz richtig war oder nicht oder vielleicht braucht das System ja nur länger...?

Die im Folgenden vorgestellten Navigationskonzepte verringern die nötige Anzahl der auszuführenden Aktionen. Zum Beispiel kann der Nutzer auf Google Maps schneller von Tokyo nach New York navigieren, wenn er den Zoomfaktor einstellen kann. Dadurch erreicht der Benutzer schneller sein Ziel.

## Pan & Zoom

Bildlich gesprochen ist Pan & Zoom mit einem Mikroskop vergleichbar. *Panning* beschreibt dabei das Bewegen der Probe unter dem Mikroskop, das Verändern des sichtbaren Bildausschnitts. Durch *Zooming* kann der Benutzer die Vergrößerungsstufe und damit den Detailgrad selbst bestimmen. Das populärste Beispiel für eine Anwendung dieses Konzepts ist wahrscheinlich Google Maps<sup>4</sup> (Abb. 2.14). Die Bedienungselemente für Pan & Zoom befinden sich dort links oben. Der Zoom muss allerdings nicht rein geografisch sein, also vergrößernd wirken. Die Technik, bei höherem Detailgrad auch tatsächlich mehr Informationen anzuzeigen, wird »Semantic Zoom« genannt.

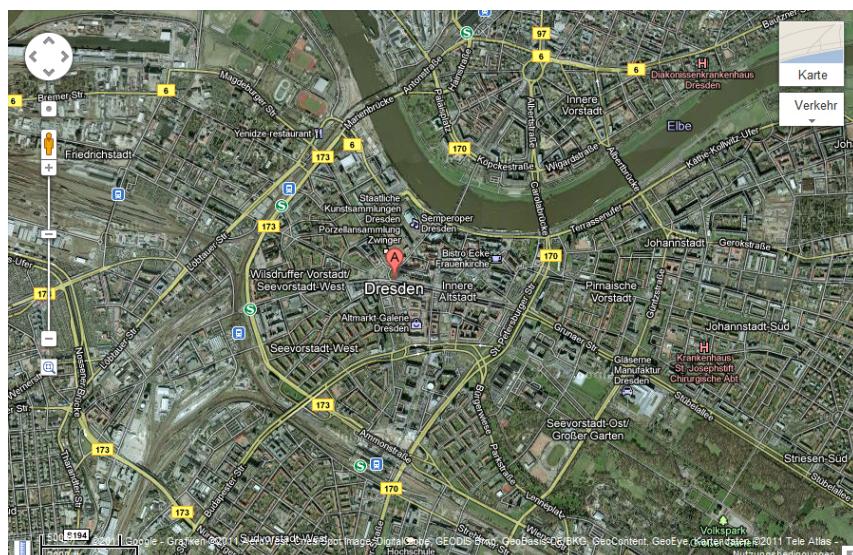


Abbildung 2.14: Google Maps

<sup>4</sup><http://maps.google.com>

Eine Graphvisualisierung eines semantischen Datensatzes sollte Pan & Zoom unterstützen. Der Zoom könnte semantisch sein und zuerst die Klassen, bei höherem Detailgrad auch die Instanzen anzeigen.

## Focus & Context

Die Idee hinter Focus & Context ist folgende: Gewählte Regionen detaillreicher anzeigen, den Rest unverändert lassen und keine Bereiche verdecken. Der populärste Vertreter ist wahrscheinlich das Dock von MacOS X (Abb. 2.15<sup>5</sup>), welches dies durch einen Fisheye-Effekt erreicht. Dieser wurde auch schon bei Tabellen (Table Lens [RC94]), Menüs [Bed00], Kalendern (DateLens [Bed+04]) und Textdokumenten (DocumentLens [RM93]) angewandt.



Abbildung 2.15: Das Dock aus MacOS X mit Fisheye-Effekt

Abseits des Fisheye-Effekts ist es auch möglich, sich der Tiefenschärfe zu bedienen (Abb. 2.16, [KMH01]). Das Konzept nennt sich Semantic Depth of Field [KMH01]. Mit dieser Technik können z. B. von einem Filter betroffene Elemente in einer ungeordneten Menge hervorgehoben werden.

## Breadcrumbs

Die Brotkrümel-Navigation zeigt dem Benutzer, welchen Pfad er bisher in der Hierarchie gegangen ist (Abb. 2.17). Laut Nielsen [Nie07] haben Breadcrumbs viele Vorteile:

- Sie helfen dem Benutzer sich zurechtzufinden.
- Mit nur einem Klick kann der Benutzer wieder auf höhere Level zurück navigieren.
- Breadcrumbs brauchen wenig Platz.

Bei Interaktion mit semantischen Daten könnte z. B. die Klassenhierarchie eines ausgewählten Elements als Breadcrumb dargestellt werden.

<sup>5</sup>Quelle: <http://crossgeared.com/stuff/wp-content/uploads/2007/10/dock.jpg>

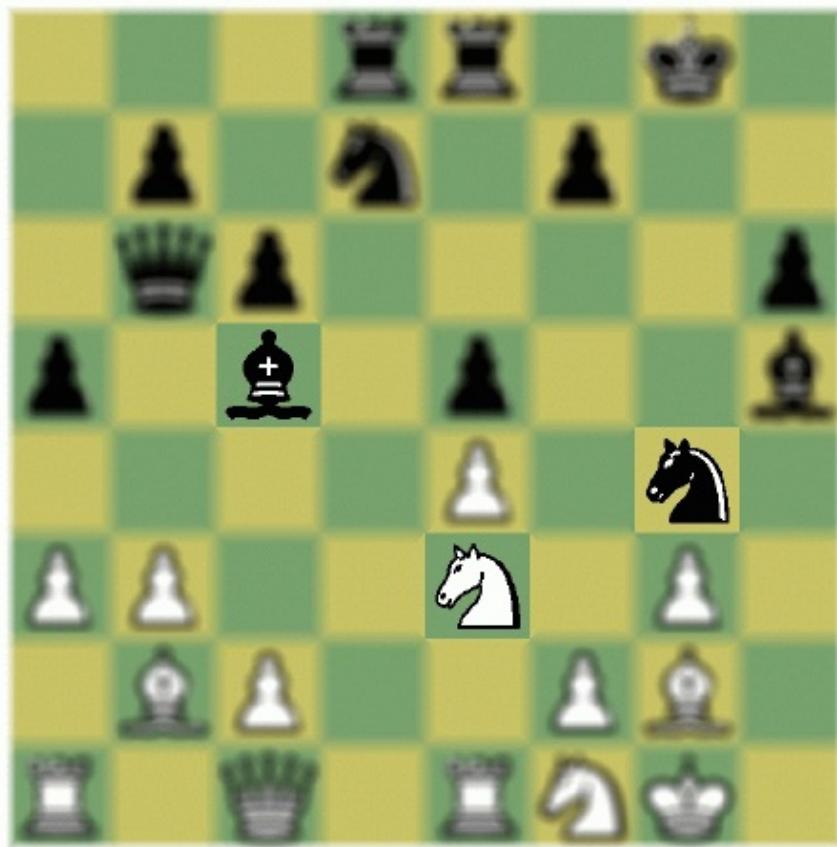


Abbildung 2.16: Figuren, die das weiße Pferd bedrohen, hervorgehoben durch Semantic Depth of Field

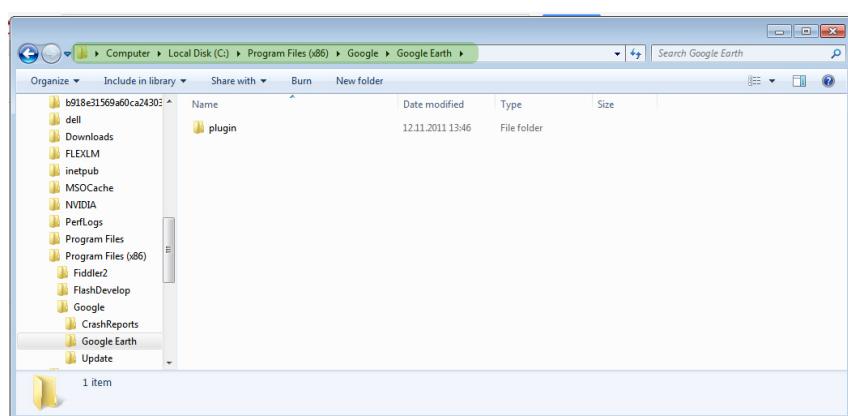


Abbildung 2.17: Breadcrumb-Navigation im Windows 7 Explorer

## Overview & Detail

Nach Cockburn [CKB06] umfasst der Begriff »Overview & Detail« mehrere Interaktionskonzepte, hier soll damit der im Paper betrachtete Typ »Gestalt/Radar« gemeint sein. Dieser beschreibt zwei synchronisierte Views, wovon einer für die Darstellung der Daten (Detail) und einer für Überblick bzw. Navigation zuständig ist (Overview). Abbildung 2.18 [Bos] zeigt ein solches Konzept, in dem die Detailansicht oben und der Overview darunter ist. Bei semantischen Datensätzen könnte damit z. B. die Navigation im Datensatz realisiert werden.



Abbildung 2.18: Eine Overview & Detail Interaktion mit Protovis

## Faceted Navigation

Eine Facettenklassifikation (auch analytisch-synthetische Klassifikation) entstand aus der Colon Classification von Ranganathan [Ran64]. Sie ist ein polyhierarchisches Klassifikationssystem. Dabei werden Objekte des Wissensbereichs nicht in eine starre Baumstruktur gegliedert, sondern durch mehrere unabhängige Einfachklassen beschrieben. Facetten bestehen dann aus mehreren Einfachklassen und ein Objekt wird für jede Facette klassifiziert. Zum Beispiel wären mögliche Facetten für das Wissensobjekt »Kamera« Hersteller, Größe, Megapixel, Sensortyp, Sensorgröße, Kameratyp und noch viele andere.

Faceted Navigation [Yee+03] beschreibt ein Interaktionskonzept, wonach der Benutzer die Menge der angezeigten Objekte immer weiter einschränkt. Im Initialzustand werden alle Objekte angezeigt. Wenn der Nutzer eine Einschränkung durch eine Facette definiert, werden die *möglichen* Einschränkungen aller Facetten und die angezeigten Objekte aktualisiert. Sind keine weiteren Einschränkungen möglich, werden dem Benutzer auch keine mehr präsentiert. Deswegen kann die Ergebnismenge nie leer sein.

Führt der Nutzer zuerst eine Volltextsuche durch und kann dann die Ergebnismenge mit Faceted Navigation immer weiter einschränken, wird das Faceted Search genannt (Abb. 2.19).

Bei einem semantischen Datensatz könnten Instanzen ihrer Klasse sein oder Klassen auf Grund von topologischen Facetten (Anzahl Instanzen, Anzahl Properties o. ä.) gefiltert werden.

Abbildung 2.19: Faceted Search bei IEEEexplore

## Magnet

Spritzer und Freitas stellten eine interessante Metapher zur Filterung von Knoten in einem Graphen vor [SF11]. Basierend auf dem Force-Layoutalgorithmus [FR91] des Graphen, in dem Knoten sich voneinander abstoßen, entwickelten sie eine Magnet-Metapher für Filter. Sie ziehen Knoten an, die bestimmten Kriterien entsprechen. Außerdem definierten sie Hierarchien und Mengenoperationen von Magneten. Spritzer und Freitas zeigten in ihrer Evaluation, dass die Magnet-Metapher leicht verständlich und zu benutzen ist. Allerdings war es für die Probanden nicht einfach mit mehreren Magneten ein sinnvolles Layout zu produzieren. Mit diesem Prinzip könnten Instanzen in einem semantischen Datensatz einfach verständlich gefiltert werden.

## Multi-Pivoting

Pivoting [Pop+11] bezeichnet ein Interaktionskonzept, wo dem Nutzer dem aktuell betrachteten Element Ähnliches zur weiteren Navigation vorgeschlagen wird. Das soll den Serendipitätsfaktor<sup>6</sup> erhöhen. Zum Beispiel könnte sich der Benutzer die Ressource »Basketball player« ansehen und vom System die Ressource »Basketball team« vorgeschlagen bekommen, weil sie über die Object Propertys »team« und »former team« von »Basketball player« verbunden sind. Eine andere Variante für Vorschläge wären solche, die auf der Topologie der Klassenhierarchie des Datensatzes beruhen, also z. B. Nachbarn, Eltern und Kinder. Eine weitere Möglichkeit wären Vorschläge auf Basis der Navigationshistorie. Wenn das System erkennt, dass der Benutzer viel im Bereich der Einheiten navigiert, könnte es ihm Währungen als wei-

<sup>6</sup>Serendipität bezeichnet das Finden von etwas Relevantem, aber ursprünglich nicht Gesuchten.

tere Einheit, die er noch nicht besucht hat, vorschlagen. Bei vielen RDF-Browsern (z. B. Tabulator [BL+06]), die dieses Interaktionskonzept unterstützen, startet der Benutzer bei einer Ressource und navigiert von dort aus immer in eine Richtung weiter.

Ein fortgeschritten Ansatz zu diesem Konzept ist Visor [Pop+11]. Der Nutzer arbeitet hier mit sogenannten Sets aus der DBpedia. Ein Set fasst gleichartige Instanzen zusammen, z. B. »Michael Jordan« und »Magic Johnson« zu »Basketball players« (aber auch »Athletes«, »NBA Players« etc.). Der Benutzer wählt zuerst beliebige Sets aus, die für ihn von Interesse sind. Visor sucht dann nach Verbindungen beliebiger Richtung zwischen diesen und fügt gegebenenfalls Sets hinzu. Abbildung 2.20 zeigt Visor, nachdem mit den Sets »Basketball players«, »Basketball teams« und »Popes« gestartet wurde. Der Nutzer kann dann die Sets, enthaltene Instanzen und Relationen genauer ansehen und dann wenn nötig weitere Sets zum Graphen hinzufügen.

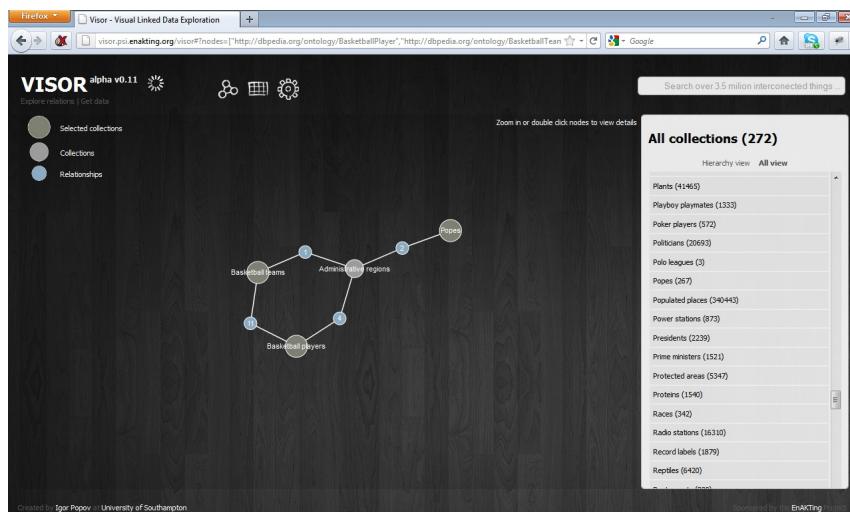


Abbildung 2.20: Ein Screenshot aus Visor. Es wurde mit den Sets *Basketball players*, *Basketball teams* und *Popes* gestartet, *Administrative regions* wurde automatisch hinzugefügt.

Ein ähnliches Tool ist RelFinder<sup>7</sup> [Loh+10] (Abb. 2.21). RelFinder nimmt als Startknoten Ressourcen aus der DBpedia und findet sämtliche Verbindungen zwischen diesen. Knoten lassen sich dann nach bestimmten Kriterien hervorheben (z. B. nur Links über Personen) oder verstecken (z. B. nur Verbindungen über mindestens zwei Ressourcen).

## 2.2 Szenario

Um die grundlegende Problematik noch einmal zu verdeutlichen, wird hier beispielhaft ein mögliches Auswahlszenario beschrieben.

<sup>7</sup><http://www.visualdataweb.org/>

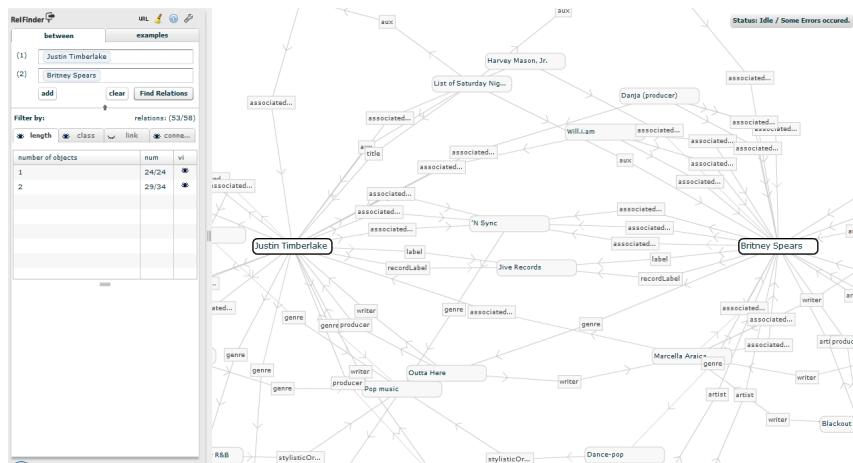


Abbildung 2.21: Verbindungen zwischen Justin Timberlake und Britney Spears

Ein Nutzer A loggt sich in VizBoard ein und wählt als seine semantische Datenbasis die QUDT Ontologie [HK11] (im Folgenden nur noch QUDT). Diese stellt ein einheitliches Modell von Mengen, Einheiten, Dimensionen und deren Datentypen zur Verfügung.

Nutzer A möchte die Datenbasis erst erkunden, da er noch nicht so richtig weiß, was für ihn interessant sein könnte. Er sieht sich erst Teile der Klassenhierarchie, dargestellt durch einen Graphen, an. Die Auswahl erfolgt mehr oder weniger zufällig. A verschafft sich erst einen Überblick über die Datenstruktur und identifiziert dann Regionen, in die er hineinzoomt. Nach wenigen Iterationen des Pan & Zoom findet er die Ressource PhysicalUnit. Jetzt fängt A an, zielgerichteter zu suchen und interessiert sich für die Nachbarn, Kinder und Eltern der Ressource. Dabei entdeckt er über Umwege die Ressource PerSecond und sieht, dass sie über verschiedene Object Propertys mit anderen Ressourcen verbunden ist. Diese sind exactMatch und quantityKind. A möchte jetzt gern wissen, welche Ressourcen die Property exactMatch noch verbindet. Darunter sind Joule und NewtonMeter. Außerdem interessieren ihn Propertys, die zu einem großen Teil dieselben Instanzen verbinden wie exactMatch, weswegen er ein Clustering durchführt. Er will jetzt wieder zu PhysicalUnit navigieren und die anderen Subklassen ansehen. Gleichzeitig soll SIDerivedUnit, das Konzept von NewtonMeter, für späteren Zugriff bereit stehen, da es ein möglicher Auswahlkandidat ist. Wieder startet er von PhysicalUnit eine zielgerichtete Suche auf topologischer Basis und landet über ScienceAndEngineeringUnit, Unit und ResourceUnit bei FinancialUnit. A lässt sich nun alle Ausprägungen von CurrencyUnit anzeigen, interessiert sich aber nur für jene mit einem  $\text{CurrencyExponent} \neq 2$ . Dabei stößt er auf Forint, eine Währung die im Gegensatz zum Euro nicht weiter unterteilt wird, also einen Currency Exponent von Null hat. Er durchsucht die gefundenen Währungen noch weiter, nimmt Vorschläge des Systems, was ihn noch interessieren könnte, an. Im Laufe der Zeit merkt A, dass die eingeschlagene Richtung nicht zielführend ist und navigiert ein paar Schritte zurück und Hierarchieebenen nach oben, um sich daran zu erinnern, wonach er eigentlich gesucht hatte. Er bemerkt, dass er sich schon einige Ressourcen als mögliche Auswahl zurechtgelegt hat. A validiert diese noch einmal gegen seine Interessen und trifft schließlich seine finale Auswahl.

## 2.3 Clusteranalyse

Bisher wurden Konzepte zur Reduktion der Komplexität von Visualisierungen auf visueller und interaktiver Basis betrachtet. Eine weitere Möglichkeit stellt das Identifizieren von ähnlichen Objekten und deren Zusammenfassen in Clustern dar. So ist es zum Beispiel möglich, einen Cluster im dargestellten Graphen als einen einzelnen Knoten darzustellen oder das Layout entsprechend der Clusteranalyse anzupassen. So können Visualisierungen semantischer Daten mittels Clusteranalyse noch weiter verbessert werden.

Eine allgemein akzeptierte Definition von Clusteranalyse existiert nicht. Eine mögliche wäre folgende: Clusteranalyse beschreibt den Vorgang, eine Menge von Objekten in mehrere Partitionen (Cluster) zu unterteilen [XW05]. Einander ähnliche Objekte sollen sich danach im selben Cluster befinden. Xu [XW05] unterscheidet zwei Arten von Clusteralgorithmen: Partitionierendes und hierarchisches Clustering. Im Abschnitt 2.3.1 werden drei Algorithmen vorgestellt: K-Means (hart partitionierend), Fuzzy C-Means (weich partitionierend) und die allgemeine Funktionsweise von hierarchischen Clusteringalgorithmen. Die Abschnitte danach beschreiben zwei speziell für semantische Daten entwickelte Clusteringalgorithmen, die auf den allgemeinen Algorithmen aufbauen. Für jeden Algorithmus wird zuerst die Funktionsweise erklärt und danach die prinzipielle Anwendbarkeit auf die in Abschnitt 2.1.1 beschriebenen Bestandteile von Ontologien betrachtet (Tabelle 2.2).

---

KH	Klassen und ihre Hierarchie
OP	Object Propertys
IN	Instanzen
DP	Datatype Propertys

---

Tabelle 2.2: Bestandteile einer Ontologie

### 2.3.1 Allgemeine Algorithmen

Hier wird kurz die Funktionsweise von drei allgemeinen Clusteringalgorithmen erläutert.

#### k-Means

Der 1967 von MacQueen vorgestellte k-Means-Algorithmus [Mac67; KJ10] unterteilt eine Datenmenge in  $k$  Cluster. Der Algorithmus geht davon aus, dass ein Objekt von zwei Dimensionen beschrieben wird, sodass es als Punkt im zweidimensionalen Raum aufgefasst werden kann. Ein Maß für die »Nähe« (Ähnlichkeit) von Paaren von Objekten wird durch eine Distanzfunktion  $dist$  modelliert. Das kann die Distanz im euklidischen Raum sein, aber z. B. auch der Dice-Koeffizient, das Cosinus-Maß oder der Jaccard-Koeffizient. Das Ziel des Algorithmus ist nun, einander nahe Objekte in Clustern zu gruppieren. Der Algorithmus (Abb. 2.22) läuft wie folgt:

1. Wähle zufällig  $k$  Objekte als Centroid eines Clusters.

2. Weise alle Objekte  $x$  dem Cluster  $S_x$  mit dem nächsten Centroid zu und berechne das Centroid eines Clusters neu, sobald ein Objekt seine Cluster-Zugehörigkeit ändert.
3. Wiederhole Schritt 2 bis sich nichts mehr ändert.

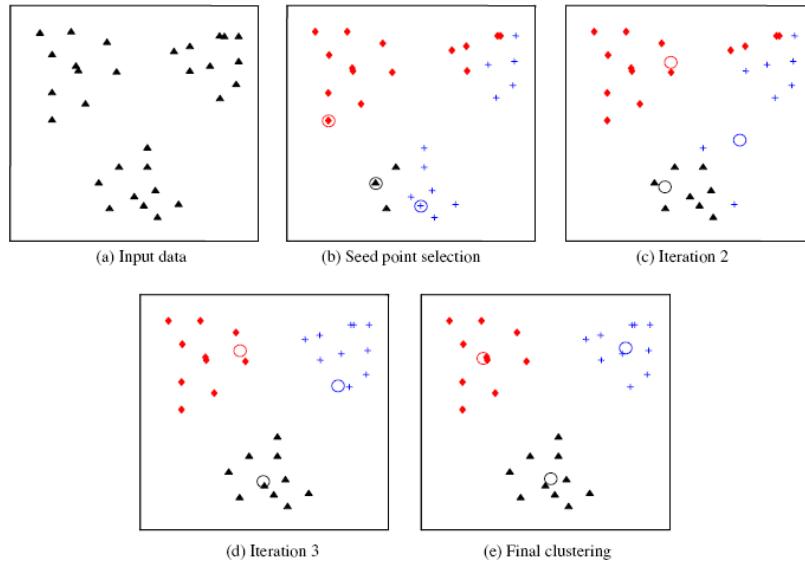


Abbildung 2.22: Der k-Means-Algorithmus

K-Means konvergiert gegen ein (meistens lokales) Minimum und findet daher selten eine optimale Lösung. Das Ergebnis des k-Means-Algorithmus hängt stark vom Input-Parameter  $k$ , von der Reihenfolge in der die Daten gelesen werden und von den im ersten Schritt gewählten Elementen ab, was als sein größter Nachteil betrachtet wird.

Ob K-Means mit den Bestandteilen von Ontologien arbeiten kann (Tabelle 2.3), hängt von der Distanzfunktion ab. Die Klassenhierarchie könnte durch topologische Kriterien (z. B. Anzahl Kinder, Tiefe im Baum) geclustert werden. Eine mögliche Ähnlichkeitsmetrik von Object Properties wären die Instanzen, die sie verbinden oder ihre Eigenschaften (z. B. transitiv, symmetrisch, reflexiv). Instanzen könnten auf Basis der Werte ihrer Properties geclustert werden. Ähnlich wie Object Properties könnte die Ähnlichkeit von Datatype Properties anhand der Instanzen, die sie implementieren, gemessen werden. Eine andere Möglichkeit wäre eine Gruppierung nach Datentyp (rdfs:range), was nach der Definition von Xu auch eine Clusteranalyse ist.

	KH	OP	IN	DP
K-Means	J	J	J	J

Tabelle 2.3: Anwendbarkeit von K-Means für Ontologiebestandteile

## Fuzzy C-means

Der Fuzzy C-means-Algorithmus wurde 1973 von J. C. Dunn vorgestellt [Dun73] und von James Bezdek 1981 verbessert [Bez81]. Er erweitert den k-Means-Algorithmus um eine sogenannte *membership* Variable, die angibt, wie sehr ein Datum einem Cluster angehört. So können Elemente nicht nur binär einem Cluster zugeordnet werden oder nicht, sondern »fuzzy« mehreren Clustern (siehe Abb. 2.23).

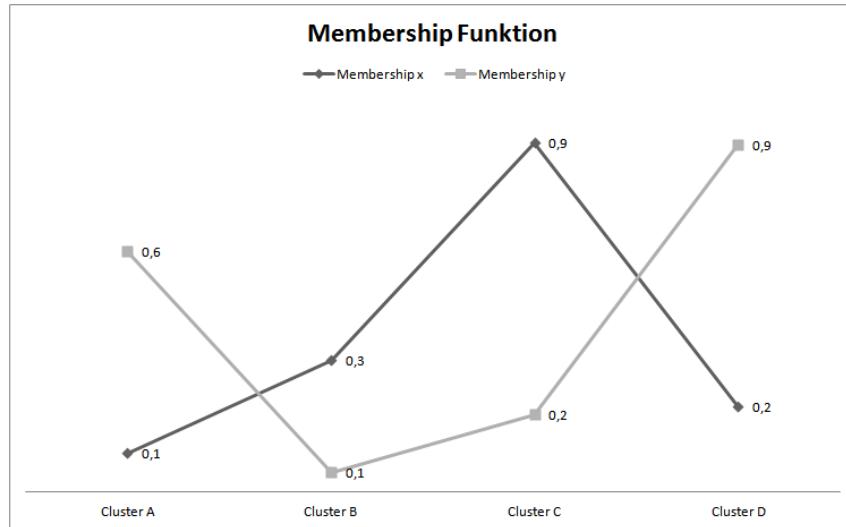


Abbildung 2.23: Mögliche Membership Funktionen von zwei Elementen x und y

Da Fuzzy C-Means im Prinzip nur eine Erweiterung von K-Means ist, ergibt sich hier dieselbe Anwendbarkeit (Tabelle 2.4).

	KH	OP	IN	DP
Fuzzy C-Means	J	J	J	J

Tabelle 2.4: Anwendbarkeit von Fuzzy C-Means für Ontologiebestandteile

## Hierarchisches Clustering

Dieser Algorithmus wurde 1967 von Stephen Johnson vorgestellt [Joh67]. Voraussetzungen sind  $n$  Objekte und eine  $n \times n$  Distanz- oder Ähnlichkeitsmatrix. Dann werden schrittweise die zwei ähnlichsten Objekte (oder Cluster) in einem Cluster vereint und die Distanzmatrix neu berechnet, bis nur noch ein Cluster übrig ist. Für die Berechnung der Distanz zwischen Clustern gibt es verschiedene Möglichkeiten:

- Single-Linkage (Minimum Method): Die Distanz entspricht der kürzesten Distanz zwischen einem Objekt im Cluster und Objekt im Vergleichscluster.
- Complete-Linkage (Maximum Method): Die Distanz entspricht der größten Distanz zwischen einem Objekt im Cluster und Objekt im Vergleichscluster.
- Average-Linkage: Die Distanz entspricht dem Median oder Durchschnitt der Distanz zwischen den Elementen im Cluster und denen im Vergleichscluster.

Der Algorithmus (Abb. 2.24, [SIM]) arbeitet also wie folgt:

1. Jedes Objekt ist ein Cluster.
2. Finde zwei die ähnlichsten Objekte und vereine sie zu einem Cluster.
3. Berechne die Distanzmatrix neu.
4. Wenn noch Objekte vereint werden können, fahre bei Schritt 2 fort.

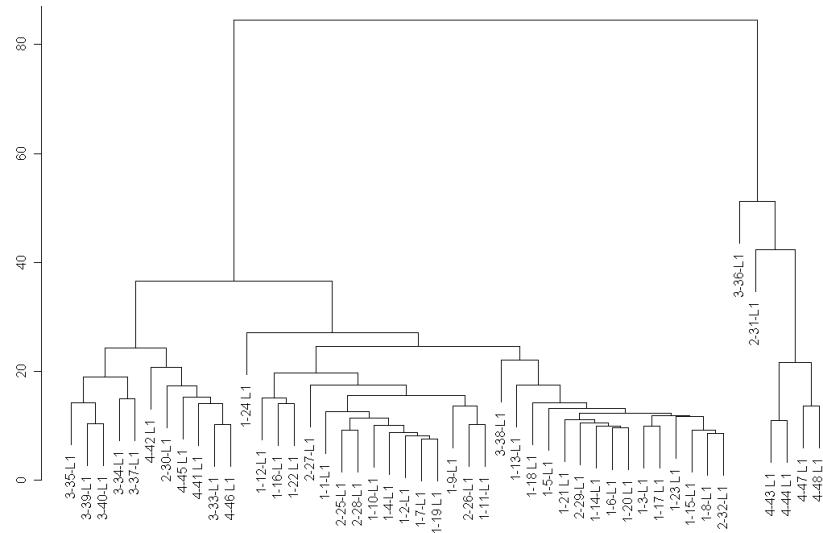


Abbildung 2.24: Hierarchisches Clustering

Der Nachteil dieses Algorithmus' ist die Komplexität von mindestens  $O(n^2)$ .

Wie K-Means ist hierarchisches Clustering hart partitionierend und arbeitet mit einer Distanzfunktion, deswegen ergeben sich keine Unterschiede in der Anwendbarkeit.

	KH	OP	IN	DP
Hierarchisches Clustering	J	J	J	J

Tabelle 2.5: Anwendbarkeit von hierarchischem Clustering für Ontologiebestandteile

### 2.3.2 Key Concept Extraction

Der Algorithmus des *Key Concept Extraction* (KCE) wurde von Peroni et al. entwickelt [PMd08] und findet im Plugin KC-Viz [Mot+11] für das NeOn-Toolkit bereits Anwendung. Die Idee hinter KCE ist, dass der Benutzer eine ganze Zahl  $n$  eingibt und das System ihm die  $n$  wichtigsten Konzepte (Key Concepts) der Ontologie präsentiert. Dabei greifen sie auf verschiedene Kennzahlen, u. a. aus der Psycholinguistik zurück:

- Natürliche Kategorien [RL78]: Wenn Menschen Objekte kategorisieren sollen, benutzen sie einfache Begriffe, sogenannte *basic objects*, z. B. Stuhl oder Auto. Diese sind in der Begriffshierarchie sehr zentral gelegen und erlauben so die größte Abgrenzung gegenüber anderen Begriffen. Abstraktere Bezeichnungen wie z. B. Einrichtung oder Fortbewegungsmittel beschreiben das Objekt zu wenig, Konkretere wie z. B. Küchenstuhl oder Sportwagen sind wiederum zu genau, da ein Küchenstuhl sich nur in Details von anderen Stühlen unterscheidet. Diese natürlichen Kategorien werden durch *name simplicity* (zusammengesetzte Worte werden im Algorithmus berücksichtigt), *basic level* (ein Maß für die Zentralität eines Konzepts in der Klassenhierarchie) und *popularity* (Anzahl Treffer bei Yahoo) identifiziert.
- Topologische Kriterien: Weil natürliche Kategorien nicht ausreichen, werden auch *density* (wie genau ein Konzept beschrieben ist) und *coverage* (um zu verhindern, dass große Teile der Ontologie übersehen werden) betrachtet.

Bei der Evaluation stimmten die von KCE identifizierten Konzepte im Schnitt zu 72 % mit denen von Menschen ausgewählten überein. Dieser Algorithmus könnte z. B. für eine Art »Startseite« in der Navigation benutzt werden.

Da KCE nur für Konzepte definiert ist, ist er auch nicht in dieser Form für Object Propertys, Instanzen oder Datatype Propertys anwendbar.

	KH	OP	IN	DP
KCE	J	N	N	N

Tabelle 2.6: Anwendbarkeit von KCE für Ontologiebestandteile

### 2.3.3 Profiling Linked Open Data

Böhm et al. [Boh+10] stellten einen Algorithmus vor, mit dem Linked Open Datasets (wie z. B. die DBpedia) analysiert werden können (ProLOD). Das Problem hierbei ist die Heterogenität der Daten: Analyse beruht auf verschiedenen statistischen Verfahren und ein Attribut *length* kommt in vielen verschiedenen Kontexten vor, weswegen es nicht immer sinnvoll ist, z. B. die Standardabweichung über alle Werte zu berechnen.

Zuallererst wird ein hierarchisches Clustering über die Daten gemacht. Danach folgt ein erweitertes K-Means Clustering mit einer schemabasierten Distanzfunktion, dass zur Laufzeit verschiedene Heuristiken einsetzt, um zu entscheiden, ob ein Cluster noch weiter unterteilt werden soll. Für jeden der identifizierten Cluster wird dann eine Art »kleinstes gemeinsames Schema« mit den häufigsten Prädikaten berechnet und der Cluster wird automatisch beschriftet. Das wird durch ein tf-idf-Verfahren über die zuvor identifizierten häufigsten Prädikate erreicht.

Der Algorithmus beinhaltet noch weitere Schritte, nach denen dann die gängigen

statistischen Verfahren eingesetzt werden können. Für eine Visualisierung semantischer Daten erscheint der Schritt mit der automatischen Beschriftung am Relevanstenen, da so z. B. durch eine Facette gebildete Cluster von Instanzen beschriftet werden können.

ProLOD clustert Instanzen, weil der Algorithmus entwickelt wurde, um konkrete Daten zu analysieren. Da Ausprägungen von Propertyklassen auch Instanzen sind, lässt er sich wahrscheinlich prinzipiell auf Object und Datatype Propertys übertragen. Diese enthalten aber typischerweise abseits von rdfs:label und rdfs:comment keine konkreten Ausprägungen von Datentypen, also ist die Anwendbarkeit von Fall zu Fall unterschiedlich.

	KH	OP	IN	DP
ProLOD	N	?	J	?

Tabelle 2.7: Anwendbarkeit von ProLOD für Ontologiebestandteile

## 2.4 Visualisierung semantischer Daten

Bis jetzt wurden Visualisierungskonzepte, Navigations- und Interaktionskonzepte sowie das Szenario und Clusteringalgorithmen vorgestellt. In diesem Abschnitt soll nun alles zusammenkommen und Visualisierungen semantischer Daten betrachtet werden, die eines oder mehrere der zuvor vorgestellten Konzepte umsetzen. Es werden drei Visualisierungen untersucht. Zuerst wird ein Überblick über die Funktionen gegeben, dann wird die Visualisierung anhand der Untersuchungskriterien bewertet und zum Schluss ein Fazit gezogen.

Die Auswahlkriterien für Visualisierungen waren Verbreitung, Aktualität, Neuartigkeit des Visualisierungskonzepts und die native Unterstützung von semantischen Daten. Stellvertretend ausgewählt wurden Jambalaya, weil es ein Plugin für das verbreitete Open-Source Tool Protégé ist. TopBraid Composer, ein verbreiteter kommerzieller Editor, und Knoocks als Beispiel für eine aktuelle Entwicklung mit ungewöhnlichem Visualisierungskonzept. Durch diese Auswahlkriterien wurden einige Visualisierungen ausgeschlossen, z. B. TGVizTab [Ala] (schon etwas älter), Tulip [Aub04] (keine native Unterstützung semantischer Daten) oder RDFGravity [GW] (gängiges Visualisierungskonzept).

### 2.4.1 Untersuchungskriterien

Als Referenz für den Funktionsumfang dienen die in den Grundlagen identifizierten Bestandteile einer Ontologie (siehe Tabelle 2.2). Diese werden an Hand verschiedener Kriterien bewertet (Tabelle 2.8).

- **Sichtbarkeit von Elementen:** Sie sind ein Indikator für die Übersichtlichkeit der Darstellung. Elemente dürfen nicht zu klein sein, wenn doch, muss der Nutzer zoomen können. Außerdem sollten sie sich nicht mit anderen Visualisierungselementen überschneiden.

- **Unterscheidbarkeit von Elementen:** Der Nutzer sollte zumindest grob unterscheiden können, welcher Klasse die dargestellten Elemente angehören. Das bedeutet z. B. Konzepte und Instanzen sind erkennbar zwei unterschiedliche Elemente und farblich, mit Symbolen oder anders gekennzeichnet. Weiters sollten bei kleinen Ontologien zwei unterschiedliche Object Propertys auch als solche wahrgenommen werden, z. B. durch zwei verschiedene Kanten im Graph dargestellt. Im Idealfall sind auch Elemente selbst klar voneinander zu unterscheiden, z. B. indem sie genug Abstand halten.
- **Lesbarkeit von Schrift:** Meistens hat eine Ressource ein Label, eine für den Menschen verständliche Bezeichnung. Es ist wichtig, dass diese (wenn verfügbar) einfach lesbar ist.
- **Platzausnutzung:** Wird der Platz schlecht genutzt, bedeutet das zusätzlichen physischen Aufwand für den Benutzer, wenn er die aktuelle Ansicht verändern will, z. B. mit Scrollen, Pan & Zoom o. ä.
- **Navigationshilfen:** Sie verkürzen die Ausführungszeit bestimmter Aktionen. Das können z. B. Lesezeichen sein, Tastaturkürzel, eine Art History ausgewählter Elemente, ein Zurück-Button oder Pivoting (vgl. Abschnitt 2.1.3).
- **Filtermöglichkeiten:** Diese schränken die Anzahl an angezeigten Elementen auf Grund bestimmter Kriterien ein. Das erhöht die Übersicht und der Benutzer findet schneller zu seinem Ziel.
- **Layoutflexibilität:** Die Software soll keinen statischen Graphen bieten, sondern verschiedene Layoutalgorithmen zur Verfügung stellen und die Elemente auf Grund von Ähnlichkeitskriterien automatisch anordnen können (Clustering).
- **Interaktionsmöglichkeiten:** Elemente könnten auf Klick oder Hover reagieren, mit Drag & Drop verschoben werden oder in ihrer Größe verändert werden. Interaktionen können sowohl über Maus als auch über die Tastatur ausgelöst werden. Gezielt eingesetzte Interaktionen erhöhen die Übersichtlichkeit und verbessern die Navigation.
- **Bedienungshilfen:** Zum Beispiel Tooltips, Hilfswerkzeuge oder Assistenten. Sie helfen unerfahrenen Benutzern eine Aktionssequenz festzulegen.

Die QUDT (vgl. Szenario, Abschnitt 2.2) kommt als Datenbasis zum Einsatz, wenn eine lauffähige Version der Visualisierung vorhanden ist. Wenn nicht, wird das Verhalten auf Grund des Papers eingeschätzt. Für jede Visualisierung wird die Darstellung und Bedienung der Ontologiebestandteile an Hand der eben festgelegten Kriterien auf einer Skala von eins bis fünf bewertet, wobei eine Eins der niedrigste Wert ist. So ergibt sich eine sehr differenzierte Einschätzung der Visualisierung.

## 2.4.2 TopBraid Composer

TopBraid Composer ist ein Eclipse-basierter, kommerzieller Ontologie-Editor von TopQuadrant [Top]. Er wird in drei Versionen vertrieben: Free (eingeschränkte Funktion), Standard (mit Datenbank-Backends) und Maestro (für Web-Development).

- 
- K1 Sichtbarkeit der Elemente
  - K2 Unterscheidbarkeit von Elementtypen
  - K3 Lesbarkeit von Schrift
  - K4 Platzausnutzung
  - K5 Navigationshilfen
  - K6 Filtermöglichkeiten
  - K7 Layoutflexibilität
  - K8 Interaktionsmöglichkeiten
  - K9 Bedienungshilfen
- 

Tabelle 2.8: Anforderungen

Standard und Maestro sind kostenpflichtig, in diesem Abschnitt wird die Free-Version betrachtet.

## Überblick

Die Benutzeroberfläche von TopBraid Composer (Abb. 2.25) ist gut strukturiert. In der Mitte befindet sich ein Formular (Alternative: Source Code in verschiedenen Formaten) in dem die aktuell ausgewählte Ressource angezeigt wird. Links davon befindet sich ein TreeView mit der Klassenhierarchie, rechts werden alle Properties aufgelistet. Die Elemente dieser Ansichten lassen sich nach Namespace gruppieren. Unter dem Formular in der Mitte werden die Instanzen der ausgewählten Klasse angezeigt, außerdem kann auf andere Views umgeschaltet werden. Rechts daneben befindet sich der »Basket«, der Lesezeichen auf Ressourcen für schnelleren Zugriff ermöglicht. Links neben den Instanzen ist ein Navigator für das Dateisystem. Das Navigationskonzept von TopBraid Composer ist dem eines Webbrowsers ähnlich: Es wird immer nur eine Ressource im Formular dargestellt, der Benutzer kann aber mit der Kombination STRG und Klick auf eine andere Ressource sofort zu dieser navigieren.

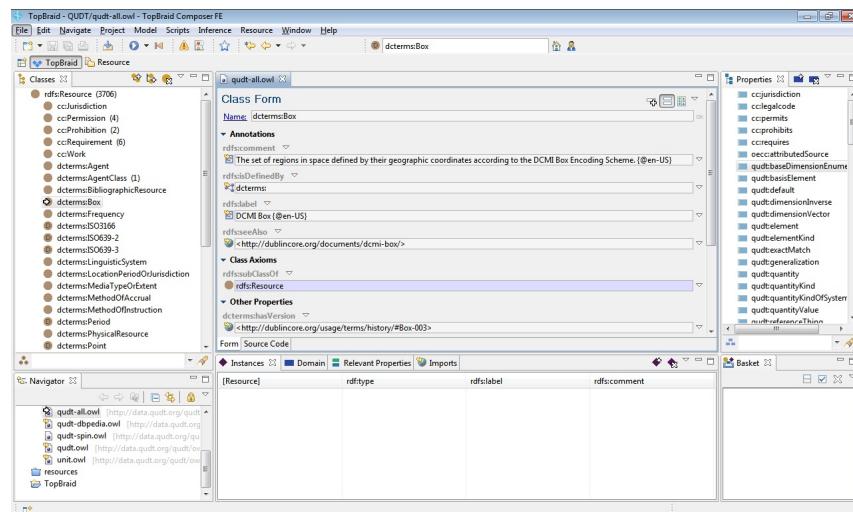


Abbildung 2.25: TopBraid Composer mit QUDT

## Klassenhierarchie

Für das Kriterium *Sichtbarkeit* wird die maximale Anzahl an Punkten vergeben, weil die Größe stimmt und sich nichts überlappt.

Bezüglich *Unterscheidbarkeit* werden auch 5 Punkte vergeben, da sich innerhalb des Views nur gleichartige Elemente befinden und die Icons es erlauben, importierte, RDFS- und OWL-Ressourcen zu unterscheiden.

Die *Lesbarkeit von Schrift* ist sehr gut, auch hier 5 Punkte.

Der TreeView hat leider Nachteile bezüglich *Platzausnutzung*, deswegen nur 3 Punkte.

TopBraid Composer bietet als *Navigationshilfen* Lesezeichen, Volltextsuche, einen Zurück-Button, viele Tastaturkürzel, und durch die STRG+Klick Navigation auch ein bisschen Pivoting, weswegen 4 Punkte vergeben werden.

Die Konzepte können nur nach ID *gefiltert* werden, aber TopBraid Composer unterstützt auch fortgeschrittene Queries wie "*\*Unit*". Hier werden 3 Punkte vergeben.

Das *Layout* des TreeViews ist starr und unflexibel, deswegen sollte es hier nur einen Punkt geben. Die Möglichkeit nach Namespace zu gruppieren kann aber als eine Art Clustering aufgefasst werden, deswegen werden 2 Punkte vergeben.

Die *Interaktionsmöglichkeiten* mit Konzepten beschränken sich auf Klick und Drag & Drop. Das ist etwas wenig, weswegen 3 Punkte vergeben werden.

Falls ein Nutzer Schwierigkeiten beim Festlegen seiner Aktionssequenz hat, helfen eine umfangreiche statische Hilfe und eine dynamische Hilfe bei der *Bedienung*. Das ist Standard, deswegen werden hierfür 3 Punkte vergeben. Daraus ergibt sich folgende Bewertung (Tabelle 2.9):

	K1	K2	K3	K4	K5	K6	K7	K8	K9
KH	5	5	5	3	4	3	2	3	3

Tabelle 2.9: Bewertungen für KH von TopBraid Composer

## Properties

TopBraid Composer trennt Object und Datatype Properties nicht in unterschiedliche Views, weswegen diese Elemente dieselbe Bewertung bekommen. Properties werden außerdem in einem gleichartigen TreeView angezeigt wie die Klassenhierarchie. Aus diesem Grund werden sie, wenn nicht anders erwähnt, auch gleich bewertet. Geänderte Werte sind in der Tabelle 2.10 fett markiert.

Die *Unterscheidbarkeit* der Elemente ist gleich gut wie bei der Klassenhierarchie, TopBraid Composer färbt das Icon abhängig von der Art der Property (Datatype, Annotation, Object, Data...) ein. Bezuglich des Kriteriums *Filtermöglichkeiten* wird aber 1 Punkt abgezogen, da es trotz der unterschiedlichen Typen keine Möglichkeit gibt, die Anzeige auf einen davon zu beschränken.

## Instanzen

Im Gegensatz zu den anderen beiden Views werden Instanzen nicht in einem TreeView, sondern in einer Tabelle angezeigt. Die Spalten sind standardmäßig die ID der

	K1	K2	K3	K4	K5	<b>K6</b>	K7	K8	K9
OP	5	5	5	3	3	<b>2</b>	2	3	3
DP	5	5	5	3	3	<b>2</b>	2	3	3

Tabelle 2.10: Bewertungen für OP und DP von TopBraid Composer

Ressource, die Superklasse (`rdfs:type`), ein menschenfreundlicher String (`rdfs:label`) und ein Kommentar (`rdfs:comment`).

Die *Sichtbarkeit der Elemente* ist gut, hier werden 5 Punkte vergeben.

Manchmal fällt es schwer, ein gerade noch fokussiertes Element in der Menge an Zeilen wieder zu finden (*Unterscheidbarkeit*), deswegen scheinen 4 Punkte angemessen.

Die *Lesbarkeit von Schrift* ist gut, aber unter der getesteten Bildschirmauflösung von  $1280 \times 800$  sind die Spalten oft zu schmal - 4 Punkte.

Die Tabelle nutzt den *Platz* sehr gut aus. Es gibt kaum Freiräume, aber es wird immer nur eine Teilmenge der Daten dargestellt. Deswegen werden bei diesem Kriterium 4 Punkte vergeben.

Leider gibt es im Vergleich zu den TreeViews hier weniger *Navigationshilfen*: Es fehlen die Volltextsuche und Tastaturkürzel. Weil gerade diese sehr nützlich gewesen wären und man sich nur mit dem Sortieren von Spalten behelfen kann, wird hier 1 Punkt vergeben.

*Filtermöglichkeiten* sind nicht vorhanden, deswegen nur 1 Punkt.

Das *Layout* kann nicht verändert und Instanzen nicht geclustert werden, aus diesem Grund wird auch hier nur 1 Punkt vergeben.

Es ist wie bei den TreeViews möglich, mit den Instanzen mittels Klick und Drag & Drop zu *interagieren* - 3 Punkte.

Das Spektrum an *Bedienungshilfen* erweitert sich um Tooltips, wenn eine Spalte zu schmal ist, deswegen werden 4 Punkte vergeben. Daraus ergibt sich folgende Bewertung (Tabelle 2.11):

	K1	K2	K3	K4	K5	K6	K7	K8	K9
IN	5	4	4	4	1	1	1	3	4

Tabelle 2.11: Bewertungen für IN von TopBraid Composer

## Fazit

Wie sich an der Tabelle 2.12 ablesen lässt, ist TopBraid Composer ein übersichtliches Tool mit sinnvollen Interaktionsmöglichkeiten. Die Darstellung von Instanzen wird allerdings etwas vernachlässigt. An der Menge an Text und dem Interaktionskonzept, nachdem immer eine Ressource in Detailansicht sichtbar ist, und der Abwesenheit visueller Darstellungen wird auch deutlich, dass es sich um ein Authoring-Tool für IT Experten handelt.

	K1	K2	K3	K4	K5	K6	K7	K8	K9	avg
KH	5	5	5	3	4	3	2	3	3	3.55
OP	5	5	5	3	3	2	2	3	3	3.44
IN	5	4	4	4	3	1	1	3	4	3.22
DP	5	5	5	3	1	2	2	3	3	3.22
avg	5	4.75	4.75	3.25	2.75	2	1.75	3	3.25	3.35

Tabelle 2.12: Bewertungen von TopBraid Composer

### 2.4.3 Jambalaya

Jambalaya [Sto+01] ist ein Plugin für Protégé [NFM01], ein verbreiteter Open-Source Ontologie-Editor, der von der Stanford University zusammen mit der University of Manchester entwickelt wurde.

#### Überblick

Der Startbildschirm von Jambalaya (Abb. 2.26) zeigt in der Mitte die Klassenhierarchie und Object Propertys der Konzepte in einem Nested View. Subklassen werden in ihrem Elternelement dargestellt. In der Toolbar links davon hat der Nutzer die Möglichkeit, auf andere Views (z. B. Node-Link der Klassenhierarchie mit oder ohne Instanzen, Treemap oder Domain/Range) umzuschalten. Darüber sind eine Volltextsuche, die auch Regular Expressions unterstützt, Navigationshilfen wie der Zurück-Button, Filter und verschiedene Layoutalgorithmen vorhanden. Unter dem View kann der Nutzer unterschiedliche Einschränkungen vornehmen, wie z. B. eine andere Wurzel zu definieren, Labels auszublenden und anderes. Zusätzlich befindet sich links der Hauptansicht je ein Klassen- und Instanzbrowser als TreeView und Liste mit Suche. Ein besonderes Feature von Jambalaya sind die drei unterschiedlichen Zooms: Normal, Magnify und Fisheye. Der normale Zoom vergrößert wie eine Lupe. Magnify zeigt das ausgewählte Konzept bildschirmfüllend an und macht im Nested View die nächste Ebene sichtbar. Die Transitionen werden animiert, sodass der Benutzer der Navigation folgen kann. Der Fisheye-Zoom navigiert nicht, vergrößert das gewählte Konzept aber mittels Fisheye-Effekt.

#### Klassenhierarchie

Die *Sichtbarkeit der Elemente* hängt davon ab, welche Form der Darstellung benutzt wird. Im Nested View sind die Elemente einer Ebene durch die rechteckige Form und einfache Anordnung leicht zu erkennen. Wählt man die Treemap, können Konzepte mit wenig Subklassen sehr klein werden und untergehen (Abb. 2.27, ganz unten). Bei der Ansicht als Node-Link spielt wiederum der verwendete Layoutalgorithmus eine Rolle. Allen ist aber gemein, dass bei großen Datensätzen nur mit ausgeblendeten Labels ansatzweise etwas erkannt werden kann (Abb. 2.28). Trotzdem überlappen sich dann viele Elemente. Weil aber auch einfach der Klassenbrowser benutzt werden kann, werden hier 4 Punkte vergeben.

Die Elemente sind durch unterschiedliche Einfärbung gut voneinander zu unterscheiden, wegen einheitlicher Darstellung in allen Views waren Formen bzw. Symbole wahrscheinlich keine Option, um das noch weiter zu verbessern. Hier werden 4 Punk-

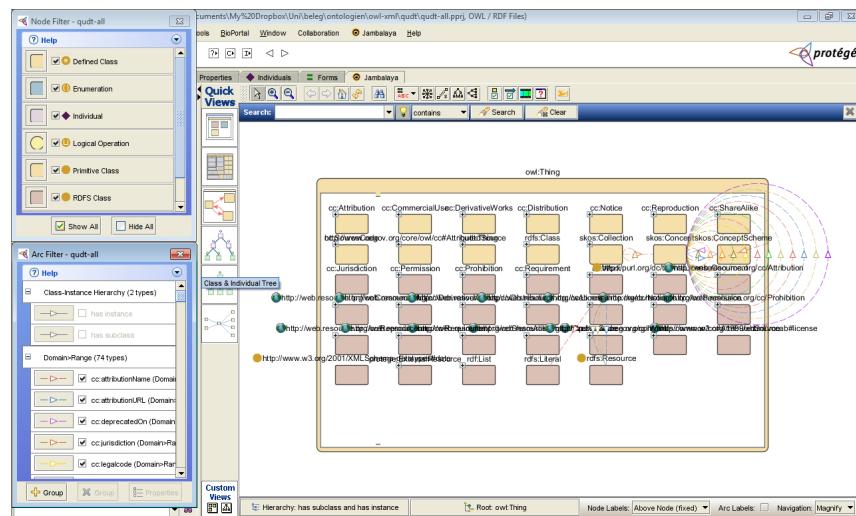


Abbildung 2.26: Jambalaya mit QUDT

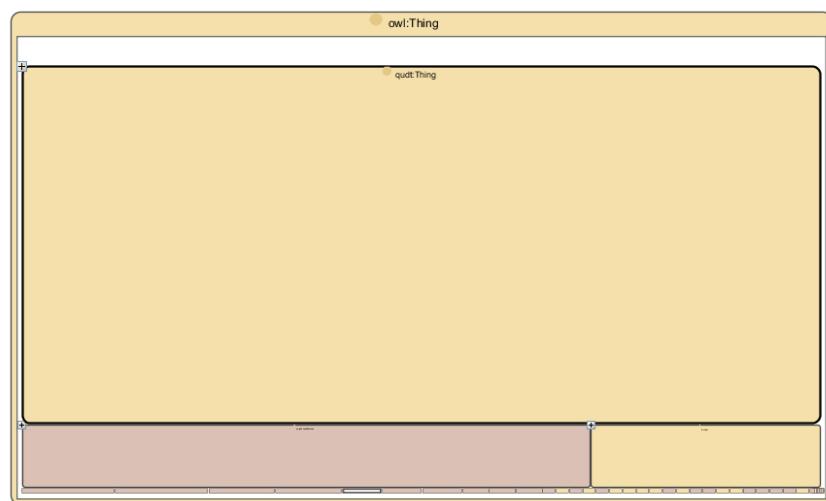


Abbildung 2.27: Klassenhierarchie als Treemap in Jambalaya

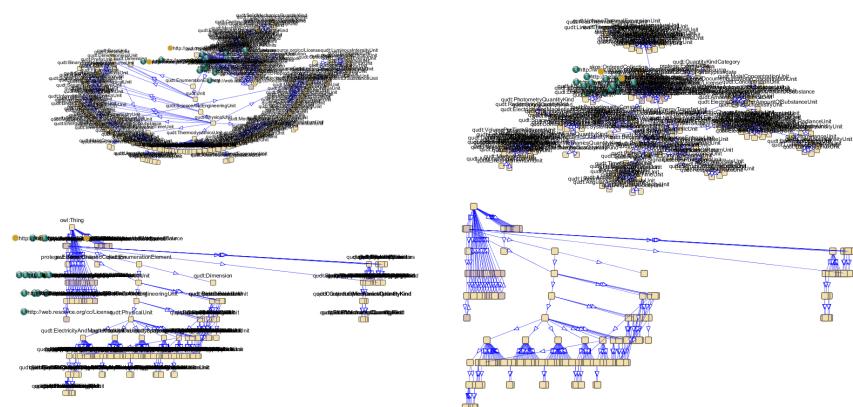


Abbildung 2.28: Klassenhierarchie als Node-Link mit verschiedenen Layouts in Jambalaya: Radial, Spring, Baum, Baum ohne Labels (v. l. o. n. r. u.)

te vergeben.

Die *Lesbarkeit von Schrift* hängt wieder von der ausgewählten Ansicht ab. Im Überblick von Node-Link Views ist sie gar nicht lesbar (Abb. 2.28), bei der Treemap eventuell zu klein (Abb. 2.27) in einer bildschirmfüllenden Ansicht im Nested View hingegen gut zu lesen. Weil sie bei großen Daten wegen Überlappung aber meistens nicht zu lesen ist, aber als Alternative der Klassenbrowser benutzt werden kann, werden hier 3 Punkte vergeben.

Auch die *Platzausnutzung* ist je nach View und Layoutalgorithmus unterschiedlich. Weil hier viele Variationen möglich sind, sodass eine ungünstige Situation immer rückgängig gemacht werden kann, scheinen 4 Punkte angemessen.

Jambalaya bietet als *Navigationshilfen* Vor/Zurück-Buttons, einen Home-Button (zoomt ganz heraus) und Tastaturkürzel. Dafür fehlen Bookmarks, eine History, Pivoting oder ähnliches. Das erscheint ein bisschen wenig, weswegen 2 Punkte vergeben werden.

Deutlich besser unterstützt werden *Filtermöglichkeiten*. So gibt es eine Volltextsuche, die Regular Expressions versteht und angezeigte Ressourcen können nach Typ (Enumeration, Restriction, Individual, RDFS Class...) gefiltert werden. Für die maximale Punktzahl fehlen hier topologische Kriterien wie z. B. Anzahl an Subklassen, Anzahl an Instanzen o. ä., darum 4 Punkte.

Das *Layout* von Jambalaya ist wie schon gezeigt sehr flexibel, bietet aber keine Möglichkeit zu clustern, darum werden hier 4 Punkte vergeben.

Es ist möglich, mit Elementen über Klick (links, rechts und Mitte), Doppelklick links und Drag & Drop zu interagieren. Damit haben sie die Maus voll ausgenutzt, aber nutzen die Tastatur überhaupt nicht, deswegen werden hier 4 Punkte vergeben.

*Bedienungshilfen* sind eine Online-Hilfe, ein Quick Start Tutorial und Tooltips. Das ist Standard, darum 3 Punkte. Daraus ergibt sich folgende Bewertung (Tabelle 2.13):

	K1	K2	K3	K4	K5	K6	K7	K8	K9
KH	4	4	3	4	2	4	4	4	3

Tabelle 2.13: Bewertungen für KH von Jambalaya

## Object Propertys

Die *Sichtbarkeit der Elemente* ist gut, solange man sich nur auf einer Ebene bewegt. Natürlich können nicht alle Object Propertys des (großen) Datensatzes auf einmal zufriedenstellend angezeigt werden (Abb. 2.29). Weil die Sichtbarkeit auch nicht überragend ist, werden hier 3 Punkte vergeben.

*Unterscheidbar* sind die Object Propertys so lange, wie ihre gezeichneten Linien unterschiedliche Winkel haben und sie insgesamt relativ wenige sind. Bei vielen Elementen ist die Unterscheidung nach Farbe nicht mehr möglich. Auch hier gibt es 3 Punkte.

Die Beschriftung der Elemente ist unterschiedlich gut *lesbar*, das hängt von Hintergrund und gewählter Schriftfarbe ab. Weil sich bei vielen Object Propertys auch einiges überlappt, werden hier 2 Punkte vergeben.

Die *Platzausnutzung* ist denkbar gut, da eine gerade Linie über den kürzesten Weg

gezeichnet wird, deswegen 5 Punkte. Jambalaya bietet leider keinerlei *Navigationshilfen* für Object Propertys, hier gibt es nur 1 Punkt.

Dafür können sie gruppiert oder einzeln *gefiltert* werden. Das muss allerdings manuell gemacht werden, automatisch alle Object Propertys eines Namespaces hervorzuheben oder eine Textsuche fehlen leider. Bei vielen Elementen ist das manuelle Ausblenden und Scrollen mühsam, deswegen werden in dieser Kategorie nur 2 Punkte vergeben.

Bezüglich *Layoutflexibilität* ist zu sagen, dass zwar die jeweiligen Endpunkte einer Object Property verschoben werden können. Aber die Linien einzeln zu transformieren und z. B. Bezierkurven daraus zu machen, ist nicht möglich. Insofern ist das Layout recht unflexibel und Jambalaya bekommt hier 2 Punkte.

Die Object Propertys reagieren auf *Interaktion* mittels Hover oder Rechtsklick. Das ist relativ wenig, deswegen gibt es auch hier nur 2 Punkte.

Die *Bedienungshilfen* sind im ganzen Programm dieselben, darum wird hier gleich gewertet wie im vorigen Abschnitt. Daraus ergibt sich folgende Bewertung (Tabelle 2.14).

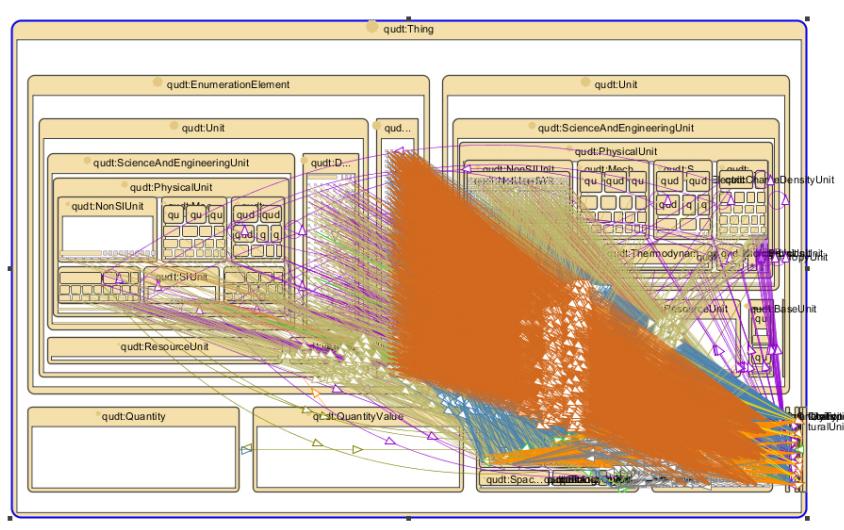


Abbildung 2.29: Object Propertys der QUDT in Jambalaya mit vollständig expandierter Hierarchie

	K1	K2	K3	K4	K5	K6	K7	K8	K9
OP	3	3	2	5	1	2	2	2	3

Tabelle 2.14: Bewertungen für OP von Jambalaya

## Instanzen

Ausprägungen von Konzepten sind in Jambalaya relativ schwer *sichtbar*. In den Nested Views sieht man sie nur auf unterster Ebene, also erst wenn hinreichend lange navigiert wurde. Dort sind sie dann auch recht klein, wenn es viele sind (Abb. 2.30). Bei der Ansicht als Node-Link müssen erst die Konzepte herausgefiltert werden und dann fehlen die Object Propertys, weil Jambalaya diese anscheinend

nur auf Konzept-Ebene verarbeitet. Das ist 2 Punkte wert.

Instanzen sind durch ihre Position *unterscheidbar*, das funktioniert bei Anordnung im Raster gut, als Node-Link mit Spring-Layout mal mehr, mal weniger. Hier werden 3 Punkte vergeben.

Da Instanzen nach demselben Prinzip dargestellt werden wie Konzepte, sind die Wertungen der restlichen Kategorien deckungsgleich. Daraus ergibt sich folgende Bewertung (Abb. 2.15).

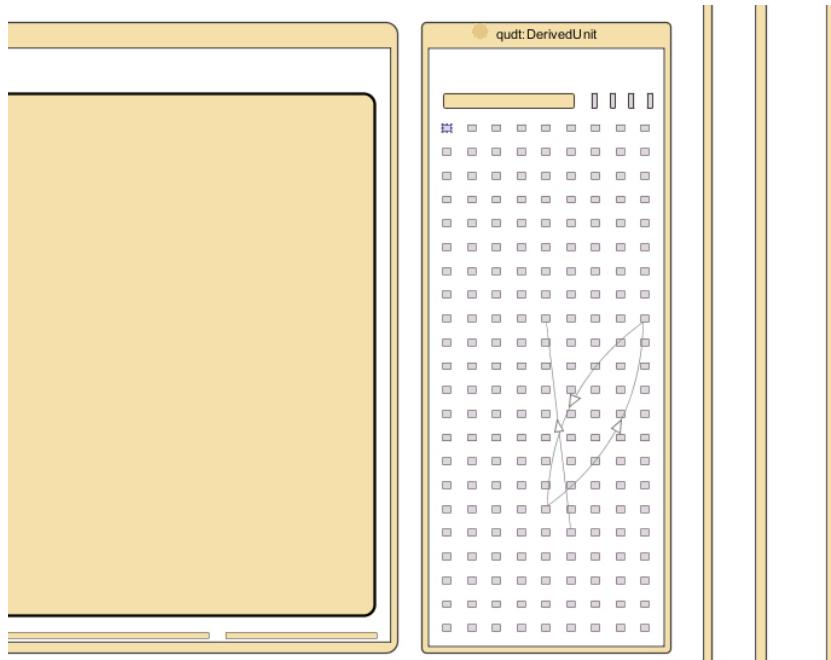


Abbildung 2.30: Klassenhierarchie als Treemap in Jambalaya

	K1	K2	K3	K4	K5	K6	K7	K8	K9
IN	2	3	3	4	2	4	4	4	3

Tabelle 2.15: Bewertungen für IN von Jambalaya

## Datatype Propertys

Datatype Propertys werden nach Doppelklick auf eine Instanz oder im Kontextmenü der Klassen unter »Propertys« innerhalb des Elements, zu dem sie gehören, *sichtbar* (Abb. 2.31). Dort werden sie in einer Tabelle bzw. in zwei unterschiedlichen Views (Logic und Propertys) angezeigt, wo sie nach einem weiteren Doppelklick im Property Editor editiert werden können (Abb. 2.32). Durch die Tabelle bzw. Listen ist alles sehr übersichtlich, 5 Punkte in dieser Kategorie.

Dadurch und wegen den je nach Typ der Property unterschiedlichen Icons sind die Elemente auch gut voneinander zu *unterscheiden*, 5 Punkte.

Die *Lesbarkeit von Schrift* ist in dieser Darstellung auch optimal, 5 Punkte.

Wie für Tabellen und Listen üblich, wird der *Platz* recht gut genutzt, wofür es 4 Punkte gibt.

Leider gibt es in dieser Ansicht keinerlei *Navigationshilfen*, darum bekommt Jambalaya in dieser Kategorie nur 1 Punkt.

Auch *Filtermöglichkeiten* wie z. B. nach Typ oder Label (Volltextsuche) fehlen, deswegen nur 1 Punkt.

Da hier mit Tabellen und Listen gearbeitet wird, ist auch das *Layout* entsprechend unflexibel, 1 Punkt.

Die *Interaktionmöglichkeiten* beschränken sich auf Doppelklick links und Einfachklick rechts (Kontextmenü). Dafür gibt es 3 Punkte, weil es zwar wenig ist, aber eigentlich nicht viel mehr benötigt wird.

Die *Bedienungshilfen* sind dieselben wie in den anderen Kategorien auch, deswegen ergibt sich folgende Bewertung (Tabelle 2.16).

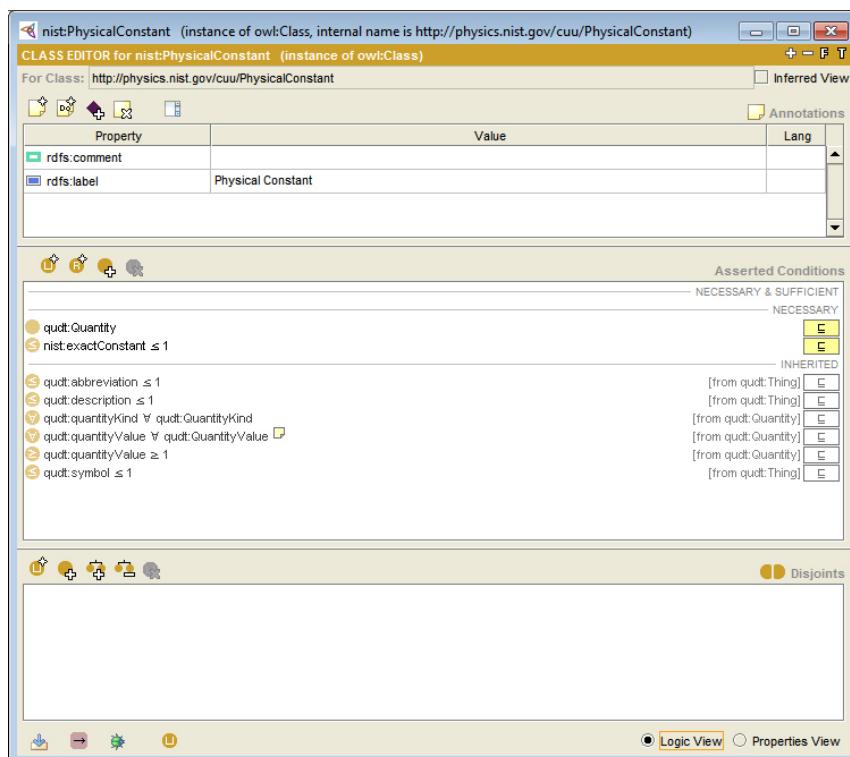


Abbildung 2.31: Datatype Propertys der Ressource nist:PhysicalConstant in Jambalaya

	K1	K2	K3	K4	K5	K6	K7	K8	K9
DP	5	5	5	4	1	1	1	3	3

Tabelle 2.16: Bewertungen für DP von Jambalaya

## Fazit

Die Bewertung (Tabelle 2.17) zeigt, dass Jambalaya die Klassenhierarchie gut darstellt, aber dem Nutzer nur wenig Navigationshilfen bietet. Die Propertys werden sehr übersichtlich angezeigt, können aber nicht gefiltert werden. Alles in allem kann

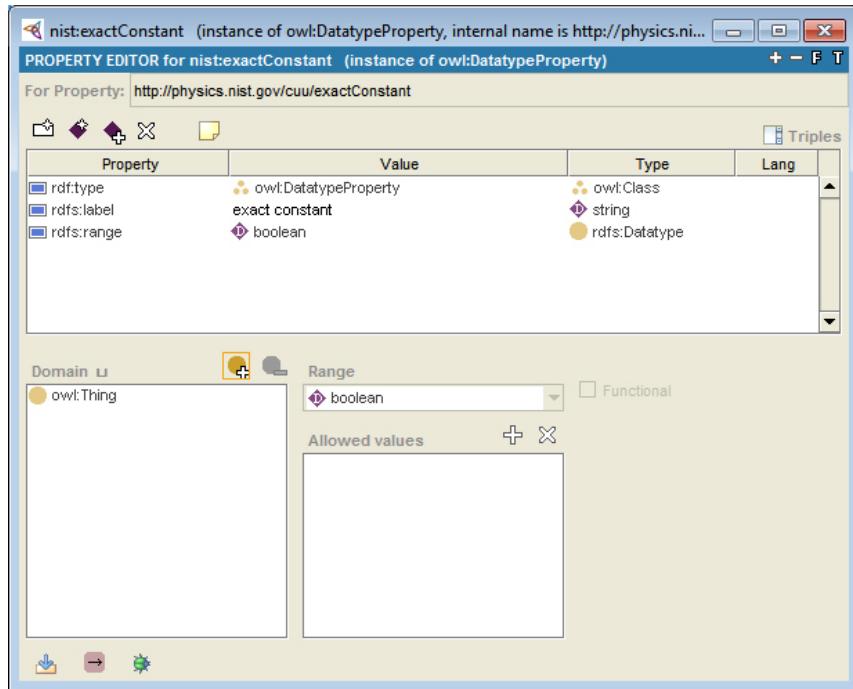


Abbildung 2.32: Property Editor für nist:ExactConstant in Jambalaya

die Visualisierung als überdurchschnittlich gut betrachtet werden. Nach kurzer Einarbeitungszeit könnten wahrscheinlich auch unbedarfe Nutzer damit umgehen.

	K1	K2	K3	K4	K5	K6	K7	K8	K9	avg
KH	4	4	3	4	2	4	4	4	3	3.55
OP	3	3	2	5	1	2	2	2	3	2.55
IN	2	3	3	4	2	4	4	4	3	3.22
DP	5	5	5	4	1	1	1	3	3	3.11
avg	3.5	3	3.25	4.25	1.5	2.75	2.75	3.25	3	3.10

Tabelle 2.17: Bewertungen von Jambalaya

## 2.4.4 Knoocks

Knowledge Blocks [KW10] ist ein an der TU Wien entwickeltes Visualisierungstool für semantische Daten. Der Fokus liegt auf Instanzen und deren Struktur. Knoocks ist noch ein Prototyp und kann aktuell nur mit Ontologien im OWL Lite Format umgehen. Deswegen wird für den Test statt der QUDT die von den Autoren mitgelieferte, wesentlich kleinere Travel Ontologie verwendet, weil Knoocks diese öffnen kann. Das Verhalten bei großen Ontologien wird geschätzt.

### Überblick

Die Benutzeroberfläche von Knoocks besteht hauptsächlich aus zwei Views, die das Overview & Detail Konzept umsetzen. Knoocks stellt die Hierarchie als von links nach rechts ausgerichtete Rechtecke dar, sodass sich für jede direkte Subklasse von

owl:Thing ein Block ergibt. Diese werden im Overview-Teil zusammen mit allen Object Propertys der Instanzen angezeigt. Ein Block kann im Detail-View näher unter die Lupe genommen werden. Instanzen einer Klasse werden im Block selbst mit Pagination aufgelistet. (vgl. Nested Views)

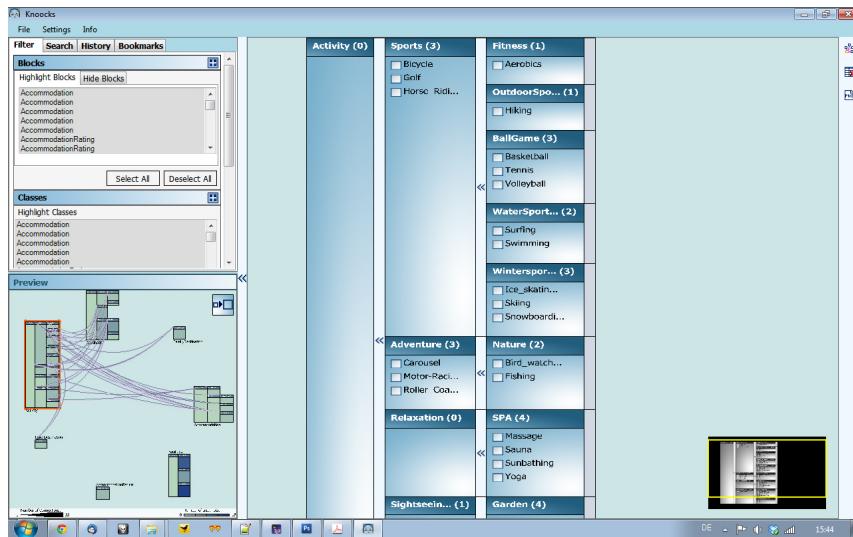


Abbildung 2.33: Die Travel Ontologie in Knoocks

S

## Klassenhierarchie

Die *Sichtbarkeit* der Konzepte ist bei einer kleinen Ontologie sehr gut, da genug Platz vorhanden ist. Bei sehr vielen Konzepten hängt es davon ab, ob diese direkt von owl:Thing erben oder eine tiefe Hierarchie haben. Im ersten Fall gibt es sehr viele Blocks, die sich überlappen könnten, im zweiten Fall könnte links ein sehr breiter Block sein, der Blöcke rechts überdeckt. Es hängt sehr vom Zeichenalgorithmus ab, deswegen wird hier mit 3 Punkten gewertet.

Durch die strukturierte Position innerhalb eines Blocks sind die Konzepte gut von einander zu unterscheiden und zudem noch im Detail-View sichtbar. Bei Blöcken an sich könnte es schwieriger werden, da diese nur im Overview zu sehen sind und bei einem großen Datensatz entsprechend viele davon vorhanden sind. Auch hier wird mit 3 Punkten gewertet.

Die *Lesbarkeit von Schrift* ist mit der Travel Ontologie sehr gut, das kann sich bei einem großen Datensatz aber schnell ändern. Hier bekommt Knoocks 3 Punkte.

Durch das Konzept der Blocks erzielt Knoocks innerhalb eines Teils der Klassenhierarchie eine sehr gute *Platzausnutzung*, die radiale Anordnung der Blocks wirkt aber entgegengesetzt. Bei sehr vielen Blöcken muss dieser Kreis theoretisch immer größer werden. Damit würden die Blöcke immer kleiner gezeichnet werden müssen, weil Pan & Zoom nicht unterstützt wird, deswegen werden in dieser Kategorie 2 Punkte vergeben.

Als *Navigationshilfen* bietet Knoocks eine History, allerdings keinen Zurück-Button oder Tastaturkürzel (außer STRG+F für die erweiterte Suche). Dafür bekommt Knoocks 2 Punkte.

Mit einer Volltextsuche können Konzepte hervorgehoben, aber nicht *gefiltert* werden, deswegen werden hier 2 Punkte vergeben.

Die Blöcke können per Drag & Drop verschoben werden. Automatisiert kann das *Layout* aber nicht verändert werden. Innerhalb der Blöcke können Konzepte nicht neu angeordnet werden. Da dieses Verhalten bei einem großen Datensatz sehr nachteilig sein kann, bekommt Knoocks 1 Punkt in dieser Kategorie.

Konzepte und Blöcke reagieren in Knoocks auf *Interaktion* mit Einfachklick links oder rechts und Blöcke können mit Drag & Drop verschoben werden. Das ist nicht viel, reicht aber aus, weswegen hier 3 Punkte vergeben werden.

Abgesehen von Tooltips bietet Knoocks keine *Bedienungshilfe*, das Tool ist aber auch noch ein Prototyp. Im weiteren Entwicklungsverlauf könnte sich noch manches ändern, darum ist es verständlich, dass noch keine Hilfe vorhanden ist. Aus diesem Grund wird diese Kategorie hier auch nicht gewertet.

Es ergibt sich folgende Bewertung (Tabelle 2.18):

	K1	K2	K3	K4	K5	K6	K7	K8	K9
KH	3	3	3	2	2	2	1	3	-

Tabelle 2.18: Bewertungen für KH von Knoocks

## Object Properties

Die *Sichtbarkeit* der Object Propertys ist sehr gut, da alle mehr oder weniger durch das Zentrum des Kreises gezeichnet werden, an dem die Blocks ausgerichtet sind. Ob das bei sehr vielen Blöcken auch noch so ist, hängt vom Algorithmus ab, deswegen werden hier 3 Punkte vergeben.

Durch die farbliche Kennzeichnung und die unterschiedlichen Winkel der Linien sind relativ wenige Object Propertys sehr gut voneinander zu unterscheiden. Das ändert sich allerdings bei sehr vielen Elementen, da so viele Farben vom Menschen nicht mehr sinnvoll zugeordnet werden können und Quelle bzw. Ziel einer Verbindung nicht mehr erkannt wird. Weil der Fokus in dieser Arbeit auf großen Ontologien liegt und hier die Nachteile überwiegen, werden 2 Punkte vergeben.

Die *Lesbarkeit von Schrift* ist gut, weil sie erst nach einem Klick auf eine Verbindung in einer Infobox angezeigt wird (Abb. 2.34). Diese Box kann aber nicht gescrollt und nur entlang der Verbindung verschoben werden. Bei langen Einträgen wird ein Teil davon nicht lesbar sein. Die Infobox an sich ist eine gute Lösung, aber nicht durchdacht umgesetzt. Deswegen bekommt Knoocks in dieser Kategorie 3 Punkte.

Der *Platz* wird von den Verbindungen sonst gut genutzt, durch die radiale Anordnung der Blocks bleiben die Wege kurz. Hier wird mit 5 Punkten gewertet.

*Navigationshilfen* für Object Propertys gibt es bei Knoocks leider nicht, auch keine Suche. Deswegen wird hier 1 Punkt vergeben.

Verbindungen können einzeln *gefiltert* werden, das muss aber manuell passieren und funktioniert nicht automatisiert auf Grund von Eigenschaften der Object Property (z. B. Anzahl der Instanzen, die sie verbindet o. ä.). Bei großen Datensätzen ist das unter Umständen eine langwierige Arbeit, vor allem weil es auch keine Suche gibt. Darum wird auch hier mit 1 Punkt gewertet.

Das *Layout* der Verbindungen ist fest, ändert sich aber durch Verschieben der Blö-

cke. Dafür werden 3 Punkte vergeben.

Die *Interaktion* mit Object Propertys funktioniert mit Hover und Einfachklick links bzw. rechts. Das ist ganz gut und 3 Punkte wert.

Weil die *Bedienungshilfen* nicht gewertet werden, ergibt sich folgende Bewertung (Tabelle 2.19).

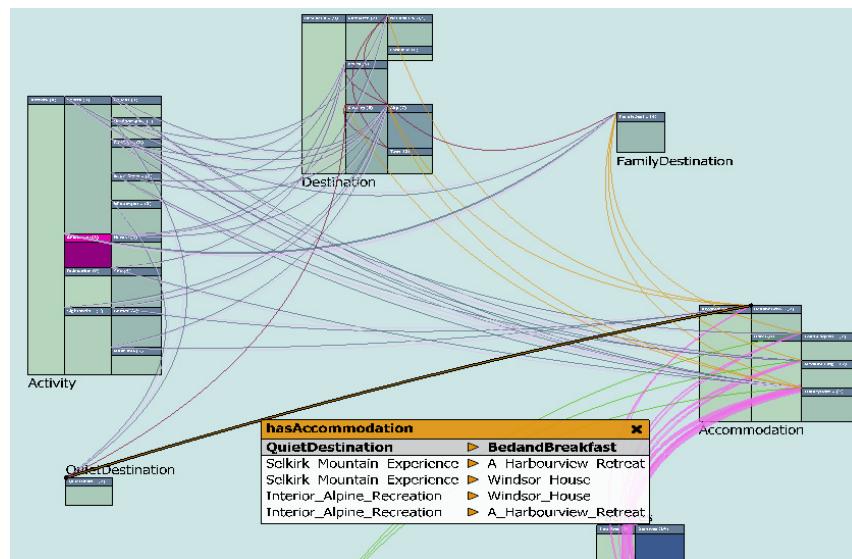


Abbildung 2.34: Eine Infobox für Object Propertys in Knoocks

	K1	K2	K3	K4	K5	K6	K7	K8	K9
OP	3	2	3	5	1	1	3	3	-

Tabelle 2.19: Bewertungen für OP von Knoocks

## Instanzen

Die *Sichtbarkeit* von Instanzen ist gut, wenn eine Klasse im Detail-View angezeigt wird (Abb. 2.35). In der Übersicht sind sie nicht sichtbar. Hier bekommt Knoocks 4 Punkte.

Durch die Visualisierung als Liste sind die Instanzen auch gut voneinander zu unterscheiden. Instanzen von verschiedenen Klassen werden auch in verschiedenen Listen angezeigt. In dieser Kategorie wird mit 5 Punkten gewertet.

Die *Lesbarkeit von Schrift* ist gut, aber die Blöcke waren im Test oft zu schmal. Deswegen werden hier 4 Punkte vergeben.

Wegen der Listendarstellung mit Pagination wird der *Platz* relativ gut genutzt, dafür bekommt Knoocks 4 Punkte.

Als *Navigationshilfe* für Instanzen gibt es Lesezeichen, die History funktioniert nur für Konzepte. Weil es für die Klassenhierarchie wiederum keine Lesezeichen gibt und diese Kategorie dort mit 2 Punkten gewertet wurde, erscheint hier dasselbe angemessen.

Instanzen können durch die Volltextsuche hervorgehoben, aber nicht *gefiltert*, also ausgeblendet werden. Darum werden hier wie bei der Klassenhierarchie 2 Punkte

vergeben.

Das *Layout* der Instanzen ist fest, sie können nicht neu angeordnet oder geclustert werden. Dafür bekommt Knoocks 1 Punkt in dieser Kategorie.

*Interaktionsmöglichkeiten* mit Instanzen beschränken sich auf Einfachklick links und rechts. Für eine höhere Wertung fehlt das Drag & Drop, deswegen werden hier 2 Punkte vergeben.

Auch in diesem Abschnitt wird die Hilfe nicht weiter betrachtet und es ergibt sich folgende Bewertung (Tabelle 2.20):

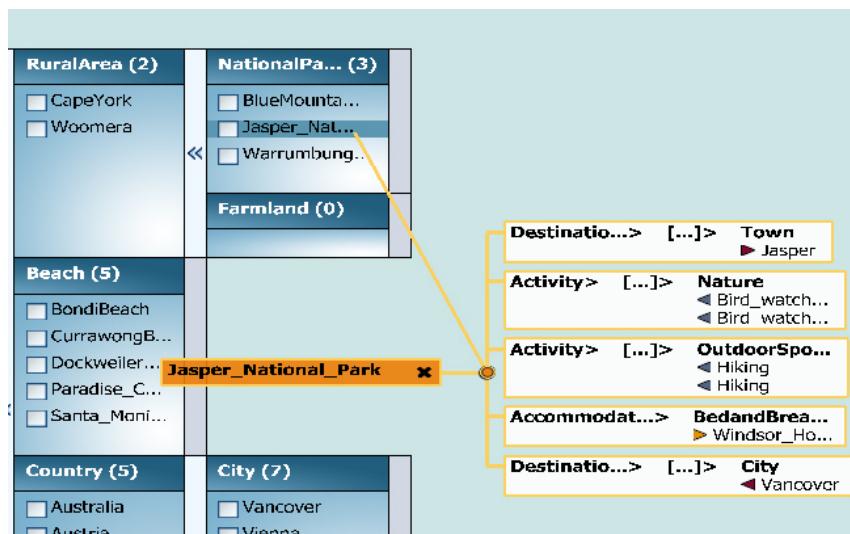


Abbildung 2.35: Instanzen und deren Details in Knoocks

	K1	K2	K3	K4	K5	K6	K7	K8	K9
IN	4	5	4	4	2	2	1	2	-

Tabelle 2.20: Bewertungen für IN von Knoocks

## Datatype Propertys

Die *Sichtbarkeit* von Datatype Propertys ist, wenn geöffnet, sehr gut (Abb. 2.34). 5 Punkte.

Sie sind durch die tabellarische Darstellung auch gut zu unterscheiden, Object Propertys werden in anderen Boxen angezeigt. 5 Punkte.

Die *Lesbarkeit von Schrift* ist in Ordnung, längere Texte (Kommentare mit 100 bis 200 Wörtern) dürften aber schwierig werden. Deswegen werden hier nur 3 Punkte vergeben.

Der *Platz* wird durch die Tabelle gut genutzt, die Infobox ist aber nur umständlich scrollbar. Weil ein paar Hundert Propertys deswegen kein Spaß sein werden, gibt es in dieser Kategorie nur 2 Punkte.

*Navigationshilfen* gibt es für Datatype Propertys nicht, deswegen bekommt Knoocks hier nur 1 Punkt.

Jeweils 1 Punkt wird in den Kategorien Filtermöglichkeiten, Layoutflexibilität und Interaktionsmöglichkeiten vergeben, da diese nicht unterstützt werden.

Wie in den anderen Abschnitten wird die *Hilfe* nicht betrachtet. Daraus ergibt sich folgende Bewertung (Tabelle 2.21):

	K1	K2	K3	K4	K5	K6	K7	K8	K9
DP	5	5	3	2	1	1	1	1	-

Tabelle 2.21: Bewertungen für DP von Knoocks

## Fazit

Leider skaliert das Visualisierungs- und Interaktionskonzept von Knoocks nicht: Es ist gut geeignet um kleine Ontologien zu erforschen, bietet aber keine der für große Ontologien wichtigen Filter- oder Layout/Clustermöglichkeiten. Das spiegelt sich auch in Tabelle 2.22 wider, wo tendenziell unterdurchschnittliche Werte vergeben wurden.

	K1	K2	K3	K4	K5	K6	K7	K8	K9	avg
KH	3	3	3	2	2	2	1	3	-	2.37
OP	3	2	3	5	1	1	3	3	-	2.63
IN	4	5	4	4	2	2	1	2	-	3
DP	5	5	3	2	1	1	1	1	-	2.37
avg	3.75	3.75	3.25	3.25	1.5	1.5	1.5	2.25	-	2.59

Tabelle 2.22: Bewertungen von Knoocks

## 2.5 Zusammenfassung

In diesem Kapitel wurde zuerst in die Grundlagen von Ontologien und semantischen Daten eingeführt. Dort wurde gezeigt, dass RDF das Serialisierungsformat von RDFS und OWL ist, mit denen Ontologien beschrieben werden können. Der Wartungsaufwand für eine Ontologie steigt jedoch direkt proportional mit dem enthaltenen Grad an Semantik. Danach wurden mögliche Visualisierungskonzepte für semantische Daten betrachtet. Graphen sind die verbreitetste Art, wie eine Ontologie dargestellt wird. Andere Konzepte mit ihren spezifischen Vor- und Nachteilen wurden vorgestellt. Die im dritten Teilabschnitt der Grundlagen gezeigten Navigations- und Interaktionskonzepte können dem Nutzer viel Frustration und Zeit ersparen. Vom Bereich der Webentwicklung übertragbare Konzepte sind Breadcrumbs, Pan & Zoom und Faceted Navigation. Fortgeschrittenere Möglichkeiten wie Multi-Pivoting sind dem Nutzer wahrscheinlich nicht so vertraut, können aber leicht erlernt werden.

Im zweiten Abschnitt wurde ein beispielhaftes Szenario beschrieben, um die Probleme, auf die Nutzer bei der Navigation in einem großen semantischen Datensatz stoßen kann, zu verdeutlichen. Hier spielen viele verschiedene Faktoren eine Rolle, z. B. die Erfahrung des Benutzers mit Computern und Ontologien, die Vertrautheit mit dem Datensatz, unterschiedliche Arten zu Suchen (explorativ und zielgerichtet) und verschiedene Anforderungen an die Software, die der Nutzer erst im Laufe des

Auswahlprozesses identifiziert.

Es wurden drei grundlegende Clusteranalyseverfahren vorgestellt, die - eine entsprechende Distanzfunktion vorausgesetzt - mit allen Ontologiebestandteilen arbeiten können. Zwei weitere Algorithmen wurden betrachtet, die zwar sehr fortgeschritten sind, aber nur mit bestimmten Teilen der Ontologie funktionieren.

Zum Schluss wurden drei verschiedene Visualisierungen semantischer Daten betrachtet, die eines oder mehrere der zuvor vorgestellten Konzepte umsetzen. Keine war vollständig zufriedenstellend, was vermutlich mit den vielen unterschiedlichen Anforderungen (siehe Szenario) zusammenhängt. Auffallend ist, dass alle drei Tools bei den Kriterien Navigationshilfen (Jambalaya, Knoocks), Filtermöglichkeiten (TopBraid, Knoocks) und Layoutflexibilität (TopBraid, Knoocks) eine schlechte Bewertung (2 oder darunter) bekamen. Das legt nahe, dass bei der Konzeption der Fokus auf diesen Punkten liegen sollte.

# 3 Nutzerstudie

Ziel der Arbeit ist es, eine Komponente zur semantischen Datenvorauswahl zu erstellen. Zu diesem Zweck wird die vorhandene Komponente auf Bedienbarkeit, Effektivität und Effizienz überprüft, um Erkenntnisse daraus in der Konzeption verwenden zu können. Dies deckt sich mit der Definition von Usability im ISO 9241 Standard. Es soll also die Usability festgestellt werden.

»Usability (Gebrauchstauglichkeit) ist das Ausmaß, in dem ein technisches System durch bestimmte Benutzer in einem bestimmten Nutzungskontext verwendet werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.«

## 3.1 Grundlagen

Die verschiedenen Arten von Teilnehmerbefragung und Usabilitybewertung werden nachfolgend eingeführt.

### 3.1.1 Effektivität & Effizienz

Da Effektivität und Effizienz oft verwechselt oder synonym verwendet werden, werden die beiden Begriffe kurz voneinander abgegrenzt. Mit Effektivität ist gemeint, ob der Nutzer mit der Software ein Ziel (z.B. Details zu Instanzen ansehen) *überhaupt* erreichen kann. Die Effizienz hingegen ist ein Maß dafür, *mit wie viel Aufwand* (Mausklicks, Zeit...) der Nutzer dieses Ziel erreicht.

### 3.1.2 Arten der Usabilitybewertung

Für die Bewertung der Usability werden verschiedene Methoden eingesetzt. Diese können nach Merkmalen der Ausführung (user-centered, expert-based, model-driven...) [Sch04], nach Zeitpunkt der Durchführung (formativ vs. summativ) oder nach Fragestellung kategorisiert werden [Dix11]. Einige Beispiele für Methoden der User Interface Evaluation sind:

- **Usabilitytest** (user-centered): Dieser Begriff wird hier allgemein verwendet, um eine Studie zu beschreiben, in der repräsentative Nutzer mit der Software selbst in Kontakt kommen und Daten in verschiedenen Formen (Audio, Video, Maustracking...) gesammelt und ausgewertet werden.
- Eine **formative Evaluation** ist eine Evaluation, die üblicherweise während Designprozesses durchgeführt wird, um schnell Ergebnisse zu bekommen, mit denen das Design verbessert werden kann.

- Eine **summative Evaluation** wird in der Regel am Ende des Designprozesses durchgeführt. Damit werden Fragen wie »Ist die Software gut genug für einen Release?« beantwortet.
- **Heuristische Evaluation** (expert-based) [NM90]: Bei dieser Methode untersucht eine kleine Gruppe von Usabilityexperten die Software und identifiziert Probleme an Hand von zuvor festgelegten Richtlinien (Heuristiken), die z. B. die Form »Jeder Dialog ist nicht-modal« haben können.
- **Cognitive Walkthrough** (expert-based) [Wha+94]: Im Gegensatz zur heuristischen Evaluation ist bei dieser Methode nur ein Usabilityexperte involviert, der sich in die Rolle des Nutzers versetzt und so Usabilityprobleme findet.
- **GOMS** (model-driven) [JK96] verfolgt einen ganz anderen Ansatz als die zuvor beschriebenen Methoden. Mit dem »Goals, Operators, Methods and Selection rules« Ansatz wird ein Modell der Software erstellt (z. B. mit CogTool<sup>1</sup>), in dem mögliche Interaktionen des Nutzers beschrieben sind. Am Ende wird für jede Aktion des Nutzers (z. B. Nachdenken, Taste drücken, Maus bewegen...) ein Zeitlimit definiert und für verschiedene Testfälle ausgerechnet, wie lange der Nutzer dafür benötigt. Usabilityprobleme sind wahrscheinlich vorhanden, wenn sehr gängige Anwendungsfälle eine lange Ausführungszeit aufweisen.

Welche dieser Methoden eingesetzt wird, richtet sich auch nach dem Ziel der Studie [Dix11] (übersetzt vom Autor):

1. Ein vorhandenes User Interface verbessern (üblicherweise formative Evaluation).
2. Ein User Interface überprüfen, ob es gut genug ist (üblicherweise summative Evaluation).
3. Quantitative Fragen wie »Wie viel Prozent der Nutzer sind in der Lage, eine Aufgabe auszuführen?« beantworten.
4. Eine Hypothese bestätigen oder widerlegen. Zum Beispiel »Folgen die Augen Fitt's Law?«.
5. Qualitative Fragen wie »Was hält ältere Menschen von der Nutzung des Internets ab?« beantworten.
6. Interessante Fragen für weitere Forschung finden (explorative Studie).

Die Frage ist nun, ob das Ziel dieser Nutzerstudie eher Ziel 1 oder Ziel 6 entspricht. Da in dieser Arbeit ein Prototyp entwickelt werden soll, sind Erkenntnisse darüber, wie Nutzer mit der vorhandenen Datenvorauswahlkomponente interagieren, sehr interessant. Diese Art der Studie wäre durchaus explorativ. Andererseits kann diese Arbeit als Teil eines iterativen Designprozesses gesehen werden: Die vorhandene Komponente war ein umgesetzter Mockup, der nun auf mögliche Verbesserungen evaluiert wird, die dann umgesetzt und erneut evaluiert werden. Ein Gegenargument

---

<sup>1</sup><http://cogtool.hci.cs.cmu.edu/>

wäre, dass ein iterativer Designprozess typischerweise eine »Abbruchbedingung« hat, entweder nach  $n$  Iterationen oder wenn die Evaluationsergebnisse zufriedenstellend sind. Das ist bei dieser Arbeit nicht gegeben. Auf der anderen Seite muss es das vielleicht gar nicht, schließlich können mit dem Prototypen weitere Iterationen durchgeführt werden. Davon abgesehen wird hier ein Prototyp entwickelt und kein fertiges Produkt. Deswegen ist diese Nutzerstudie mehr Ziel 1 zuzuordnen als Ziel 6.

Basierend auf den zuvor beschriebenen Überlegungen wird ein formativer Usabilitytest durchgeführt, bei dem die Teilnehmer zur Beurteilung der Effektivität verschiedene, dem Szenario entnommene Aufgaben innerhalb eines Zeitlimits lösen müssen. Dabei werden ihre Maus- und Tastatureingaben aufgezeichnet, um ein Maß für die Effizienz einzuführen. Um ihre Zufriedenheit festzustellen müssen die Testpersonen einen Task Load Index in Kombination mit einem allgemein und kurz gehaltenen Feedback-Fragebogen ausfüllen.

### 3.1.3 Task Load Index

Bei einem Task Load Index [G.88] gibt der Benutzer an, wie anstrengend eine Aufgabe bezogen auf verschiedene Formen der Aktivität war (Tabelle 3.2). Der 1988 vorgestellte Fragebogen schlägt vor, dass am Ende ein Zahlenwert gebildet wird (Workload Index) und die einzelnen Aktivitäten dafür gewichtet werden. Es existieren zahlreiche Modifikationen davon, die verbreitetste ist aber, dass auf Berechnung des Workload Index und Gewichtung verzichtet wird [Har06]. So erhält man ein differenzierteres Bild der Belastung des Nutzers. Die Fragestellungen nach den einzelnen Aktivitäten stammen aus dem Appendix des Papers von Hart [Har06], übersetzt wurden Sie vom Autor.

- **Mental Demand (MD):** Wie viel gedankliche/wahrnehmende Arbeit war nötig? Zum Beispiel denken, rechnen, erinnern, nach Funktionen suchen, entscheiden...
- **Physical Demand (PD):** Wie viel physische Aktivität war nötig um die Aufgabe zu lösen? Zum Beispiel Maus bewegen, scrollen, Text eingeben...
- **Temporal Demand (TD):** Wie viel Zeitdruck verspürten Sie beim Lösen der Aufgabe?
- **Effort (ED):** Wie viel (physische und mentale) Arbeit war nötig, um Ihre Leistung zu erreichen? Wie sehr mussten Sie sich bemühen bzw. anstrengen?
- **Performance (PD):** Wie erfolgreich haben Sie Ihrer Meinung nach die gestellten Aufgaben gelöst? Wie zufrieden sind Sie mit Ihrer Leistung?
- **Frustration Level (FL):** Wie unsicher, entmutigt, irritiert, gestresst oder verärgert waren Sie während der Aufgabe? Wie sicher, entspannt, zufrieden?

Eine alternative Befragungsmethode zum Task Load Index wäre Thinking Aloud [Jor90]. Dabei wird die Testperson aufgefordert, während des Usabilitytests alles

auszusprechen, was ihr durch den Kopf geht. Falls sie aufhört zu sprechen, hakt der Studienleiter nach. Ein Vorteil dieser Methode ist, dass die Person die Möglichkeit bekommt, Probleme genauer zu adressieren, als dies beim Ausfüllen eines Task Load Index möglich ist. Leider hat Thinking Aloud auch einige Nachteile. Zum Beispiel hat der Studienleiter mehr Einfluss auf das Ergebnis der Studie als bei einem Fragebogen, da Erfahrung, Stimmung, Gestik und andere zwischenmenschliche Komponenten eine Rolle spielen. Des Weiteren ist die Aufzeichnung einer Thinking Aloud Session aufwändiger auszuwerten als ein Fragebogen. Als Ausgleich wurde der Fragebogen eingeführt, der Möglichkeit zum freien Feedback bieten soll.

## 3.2 Objekt der Studie

Die vorhandene Datenvorauswahlkomponente »Datenauswahlzauberer« (wörtlich aus dem engl. *data selection wizard*) ist ausschließlich in JavaScript mit Hilfe der Frameworks ExtJS<sup>2</sup> (Version 3.3) und Protovis<sup>3</sup> (Version 3.2) geschrieben.

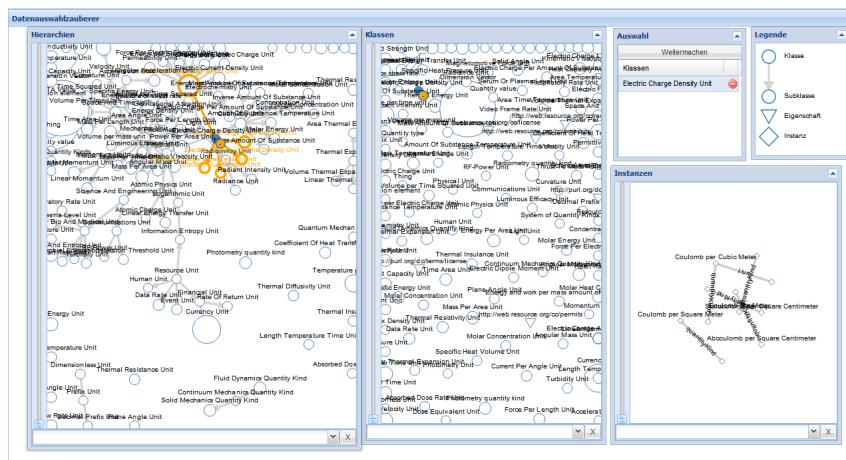


Abbildung 3.1: Aktuelle Datenvorauswahlkomponente in VizBoard mit QUDT

Die Komponente besteht aus fünf verschiedenen Fenstern (Abb. 3.1):

- **Hierarchien:** Hier wird die Klassenhierarchie dargestellt. Die Richtung des Pfeils beschreibt dabei die Property »ist Generalisierung (Superklasse) von«.
- **Klassen:** Hier werden die Object Propertys auf Schema-Ebene dargestellt. Die QUDT enthält bis auf Klassen keine weiteren Schemainformationen, weswegen hier keine Pfeile zu sehen sind.
- **Instanzen:** Hier werden die Instanzen der aktuell ausgewählten Klasse und deren Subklassen dargestellt. Zusätzlich sind andere Instanzen, die über Object Propertys mit den Instanzen der Klassen verbunden sind, sichtbar. Datatype Propertys werden im Tooltip aufgelistet.
- **Auswahl:** Hier werden die ausgewählten Klassen abgelegt. Über den Button wird die Komponente verlassen.

<sup>2</sup><http://www.sencha.com/products/extjs/>

<sup>3</sup><http://mbostock.github.com/protovis/>

- **Legende:** Hier wird eine kurze Übersicht über die Symbole dargestellt.

In den drei Fenstern mit Node-Link Views können die angezeigten Knoten über die Textbox am unteren Ende mit Hilfe einer Volltextsuche eingeschränkt werden. Wenn der Nutzer auf einen Knoten klickt, werden der Knoten sowie seine Eltern und Kinder hervorgehoben und die Instanzansicht aktualisiert. Die Größe der Knoten repräsentiert die Anzahl an Instanzen dieser Klasse im Datensatz. Über den +-Button wird die Ressource zur Auswahl hinzugefügt.

### 3.3 Methode

Die Teilnehmer füllten zuerst die allgemeinen Informationen im Fragebogen aus. Basierend auf den Angaben bei der Erfahrung mit Visualisierungstools und semantischen Daten bekamen sie je ein Informationsblatt über Graphen und Ontologien. Sie konnten währenddessen so viele Fragen stellen, wie sie wollten. Danach wurden die möglichen Interaktionen mit der vorhandenen Komponente kurz erklärt und die Testpersonen bekamen zwei Minuten frei verfügbare Zeit, um die Software kennenzulernen. Auch hier wurden alle Fragen beantwortet. Nach Ablauf der zwei Minuten mussten sie verschiedene Aufgaben in einem gegebenen Zeitlimit ohne Hilfe lösen. Dabei wurden die Anzahl der Mausklicks, der mit der Maus zurückgelegte Weg auf dem Bildschirm, Tastatureingaben sowie die benötigte Zeit gemessen. Für das Aufzeichnen des User Inputs wurde eine einfache Chrome-Erweiterung geschrieben, die dies für eine gegebene Anzahl an Sekunden durchführt. Am Schluss wurde ein Task Load Index (TLX) für jede Aufgabe erhoben, um die anstrengendsten Aufgaben identifizieren zu können und es musste der restliche Fragebogen ausgefüllt werden.

### 3.4 Durchführung

Die Studie wurde auf einem Dell XPS M1330 mit 13.3" Bildschirmdiagonale ( $1280 \times 800$  Pixel Auflösung) und Maus durchgeführt. Die Maussensitivität konnten die Teilnehmer vor Beginn selbst bestimmen. Als Browser kam Google Chrome in der Version 15.0.874.121m zum Einsatz. Die Studie wurde an keinem festen Ort durchgeführt. Die Datenbasis war die bereits im Szenario (Abschnitt 2.2) beschriebene QUDT. Die Teilnehmer arbeiteten mit der vorhandenen Datenvorauswahlkomponente von Viz-Board. Die Anordnung der Views war dabei am Anfang immer gleich.

### 3.5 Aufgaben

Die den Teilnehmern gestellten Aufgaben 3 - 10 wurden dem Szenario entnommen. Die ersten Zwei wurden gewählt, um den Overview-Task nach Shneiderman abzudecken. Shneiderman [Shn96] identifizierte in seinem »Information-seeking Mantra« sieben Tasks, die immer wiederkehren (Übersetzung vom Autor):

- **Overview:** Sich einen Überblick über den ganzen Datenbestand verschaffen.
- **Zoom:** In einen Bereich hineinzoomen.

- **Filter:** Uninteressante Objekte verstecken bzw. herausfiltern.
- **Details-on-demand:** Ein oder mehrere Objekte selektieren und bei Bedarf nähere Informationen bekommen.
- **Relate:** Verbindungen zwischen Objekten herausfinden.
- **History:** Eine History der Aktionen halten, sodass Undo/Redo und iterative Verfeinerung der Suche möglich ist.
- **Extract:** Eine Teilmenge der Daten extrahieren können bzw. Suchparameter für aktuelle Auswahl ansehen können.

Den ersten sechs davon wurden Aufgaben zugeordnet. Der Task Extract wurde ausgelassen, weil die Komponente genau diese Funktion für das übergeordnete System VizBoard umsetzen soll. Für jede Aufgabe wurde ein Zeitlimit von zwei Minuten festgelegt, innerhalb dessen sie bewältigt werden musste, um gewertet zu werden. Wurde das Zeitlimit überschritten oder gab die Person auf, zählte die Aufgabe als nicht gelöst. Die Teilnehmer wurden aufgeklärt, dass nicht alle Aufgaben lösbar sind und in diesem Fall eine derartige Antwort als richtig gewertet würde.

Im Folgenden werden die Aufgaben aufgelistet, in Klammern befindet sich der korrelierende Task nach Shneiderman.

1. Von welchem Konzept gibt es die meisten Instanzen?  
(Task Overview)
2. Welche Klasse hat die meisten direkten Subklassen?  
(Task Overview)
3. Finden Sie das Konzept *PhysicalUnit*. Nennen Sie Beispiele für Eltern, Kinder und Geschwister.  
(Task Zoom)
4. Nennen Sie Beispiele für Propertys der Ressource *PerSecond*.  
(Task Details-on-demand)
5. Welche Ressourcen verbindet die Property *exactMatch*? Nennen Sie 3 Beispiele.  
(Task Relate)
6. Über welche Konzepte sind *CurrencyUnit* und *MechanicsUnit* verbunden? Geben Sie den vollständigen Pfad an.  
(Task Relate)
7. Nennen Sie Beispiele für Ausprägungen der Klasse *CurrencyUnit*, wo die Eigenschaft *CurrencyExponent*  $\neq 2$  ist.  
(Task Filter)
8. Blenden Sie alle Instanzen mit »unit« im Namen aus.  
(Task Filter)

9. Legen Sie eine Art Lesezeichen für *SIDerivedUnit* für einen späteren Zugriff an.  
(Task History)
10. Welche Ressource hatten Sie 5 Navigationsschritten zuvor ausgewählt?  
(Task History)

Die richtigen Antworten auf die Fragen werden im Anhang (Abschnitt A.1.2) aufgelistet.

Das Zeitlimit von zwei Minuten hat verschiedene Gründe. Zum einen ist die Datenvorauswahlkomponente Teil eines größeren Auswahlprozesses, an dessen Ende eine Mashup-Visualisierung steht. Deswegen ist es notwendig, dass die Benutzer ihr Ziel möglichst schnell erreichen (effizientes Arbeiten). Außerdem wird in der Nutzerstudie die Kernfunktionalität der Komponente betrachtet. Zwei Minuten scheinen ein passendes Zeitlimit für zentrale Aufgaben zu sein. Es ist nicht unrealistisch kurz, sodass der Nutzer gar nicht erst seine Aktionssequenz erarbeiten könnte. Es ist aber auch nicht zu lang, denn wiederkehrende Aufgaben sollten schnell gelöst werden können, um den Nutzer zufriedenzustellen. Zu guter Letzt war das geplante Zeitlimit für eine Evaluationssession 30 Minuten. Im Worst Case werden 20 Minuten für die Aufgaben verbraucht und es bleibt noch genug Zeit für Task Load Index und Fragebogen.

## 3.6 Fragebogen

Es wurden allgemeine Informationen über den Nutzer, ein Task Load Index für jede Aufgabe und Feedback erhoben. Im allgemeinen Teil (Tabelle 3.1) wurden neben Alter und Geschlecht noch weitere Fragen gestellt, um die Facetten des Benutzers bezogen auf Visualisierungen, semantischen Daten und Computer zu identifizieren. Für jede Aufgabe wurde ein Task Load Index erhoben um besonders anstrengende,

Fragestellung	Antwort
Wie viel Stunden pro Tag benutzen Sie Ihren Computer durchschnittlich?	offen
Wie schätzen Sie Ihr Erfahrungslevel mit semantischen Daten ein?	Likert-Skala 1-5
Wie schätzen Sie Ihr Erfahrungslevel mit Visualisierungstools ein?	Likert-Skala 1-5
Was ist ihr Fachgebiet (Studiengang, Branche o. ä.)?	offen
Haben Sie schon einmal mit der QUDT Ontologie gearbeitet?	Binär (ja/nein)

Tabelle 3.1: Fragebogen, allgemeiner Teil

fordernde Aufgaben zu identifizieren und diesen auf den Grund zu gehen.

Fragestellung	Antwort
Mental Demand	Likert-Skala 1-9
Physical Demand	Likert-Skala 1-9
Temporal Demand	Likert-Skala 1-9
Effort	Likert-Skala 1-9
Performance	Likert-Skala 1-9
Mental Demand	Likert-Skala 1-9

Tabelle 3.2: Fragebogen, NASA-TLX

Abschließend wurden die Teilnehmer nach Verbesserungsvorschlägen und freiem Feedback befragt, um grobe Designfehler genauer und schneller zu finden als es mit einem TLX möglich ist (Tabelle 3.3).

Fragestellung	Antwort
War etwas besonders schwer zu lösen und warum?	offen
War etwas einfach und problemlos zu lösen und warum?	offen
Hat Ihnen etwas besonders gut gefallen und warum?	offen
War etwas überhaupt nicht gut gelöst und warum?	offen

Tabelle 3.3: Fragebogen, freies Feedback

## 3.7 Teilnehmer

Mehreren wissenschaftlichen Arbeiten [Vir92; NL93] zufolge reichen in einem iterativen Designprozess fünf bis sieben Personen pro Iteration aus, da so mehr als 80 % der Usability-Probleme gefunden werden. Für die Studie wurden zehn Personen ausgewählt, wobei darauf geachtet wurde, dass die Verteilung der Benutzergruppen ungefähr gleich ist. Die Teilnehmer wurden nicht in starre Benutzergruppen eingeordnet, sondern mit Hilfe von Facetten bezogen auf ihr Vorwissen mit Computern, Visualisierungstools und semantischen Daten im Allgemeinen sowie des verwendeten Datensatzes im Speziellen beschrieben (Tabelle A.1). Die Teilnehmer waren durchschnittlich 25,6 Jahre alt, wobei der Älteste 30 und der jüngste 22 Jahre alt war. Es nahmen 2 Frauen und 8 Männer an der Nutzerstudie teil.

## 3.8 Diskussion der Ergebnisse

In den folgenden Abschnitten werden die Ergebnisse der Nutzerstudie vorgestellt und diskutiert.

### 3.8.1 Effektivität

Abbildung A.1 zeigt, wie viele Testpersonen eine Aufgabe gelöst hatten (orange). Auffällig ist, dass Aufgabe 9 von allen und Aufgabe 10 von keinem gelöst wurde.

Aufgabe 3 war außerdem als relativ einfache Aufgabe gedacht, konnte aber trotzdem nur von einer Person gelöst werden. Was die Nutzer hier und bei den anderen Aufgaben behinderte, wird im Folgenden untersucht.

### 3.8.2 Task Load Index

Abbildung 3.2 zeigt Median und Standardabweichung der angegebenen Werte beim Task Load Index pro Aufgabe. Die Werte des TLX reichen von eins bis neun. Angaben bis inklusive drei entsprechen einer »niedrigen« Anstrengung, Angaben größer als sechs deuten auf eine »große« Anstrengung hin. Werte dazwischen werden als »mittlere« Anstrengung interpretiert. In den Median-Spalten wurden niedrige Werte grün und große Werte rot eingefärbt, um wichtige Angaben besser sichtbar zu machen. Werte der Standardabweichung bei TLX Angaben verteilen sich zwischen 0 (alle Nutzer gaben denselben Wert an) und 4,21 (eine Hälfte mit dem niedrigsten Wert, die andere mit dem höchsten). Analog zur Interpretation der Werte des Median wird eine Standardabweichung größer als 2,81 als »hoch« interpretiert, kleiner oder gleich 1,40 als »niedrig«. Wie auch in den Spalten des Median wurden hohe und niedrige Werte bei der Standardabweichung rot bzw. grün gekennzeichnet.

Aufgabe	Mental Demand		Physical Demand		Temporal Demand		Performance		Effort		Frustration	
	Median	STDEV	Median	STDEV	Median	STDEV	Median	STDEV	Median	STDEV	Median	STDEV
1	4,50	1,90	3,00	1,25	2,50	2,37	3,00	1,76	3,50	2,79	2,50	2,31
2	5,00	2,10	3,50	1,62	4,00	2,38	4,50	1,89	5,00	3,27	3,00	2,32
3	4,50	1,72	5,50	2,30	6,00	3,31	5,00	1,70	4,00	3,37	6,00	2,54
4	5,50	1,94	5,50	2,92	6,50	2,83	7,00	2,87	7,00	2,84	7,00	2,67
5	6,00	2,25	6,50	2,15	7,50	2,26	7,00	1,87	7,50	3,13	7,50	2,33
6	5,50	1,85	5,00	2,18	6,00	2,31	5,00	1,91	6,00	3,02	5,50	2,74
7	6,50	2,73	6,00	2,46	6,00	2,40	7,00	2,08	3,50	2,80	5,00	2,26
8	6,00	3,12	2,50	2,30	3,00	3,14	5,50	2,99	3,50	3,40	5,50	3,27
9	2,00	0,82	3,00	0,67	1,00	0,97	2,00	0,99	1,00	2,49	1,50	0,70
10	6,50	2,32	1,00	2,42	3,00	2,37	8,00	2,20	8,00	2,94	5,00	2,46

Abbildung 3.2: Median und Standardabweichung pro TLX Demand und Aufgabe

Die Kombinationen aus schwarz, rot und grün werden wie folgt interpretiert (die erste Angabe entspricht dabei dem Median, die zweite der Standardabweichung):

1. **schwarz schwarz**: Die Aufgabe konnte mittig gelöst werden. Weder der Median noch die Standardabweichung sind auffällig. Im Mittel war die Anstrengung mittelgroß und die Werte verteilen sich relativ gleichmäßig zwischen beiden Extremen. Hier kann keine eindeutige Schlussfolgerung gezogen werden.
2. **schwarz rot**: Der Median ist in der Mitte der Skala, die Standardabweichung aber hoch. Das bedeutet, dass eine Hälfte der Nutzer niedrige Werte angegeben hat, die andere aber hohe. Bei dieser Kombination muss untersucht werden, was für ein Problem die Hälfte mit hohen Angaben gehabt haben könnte.
3. **schwarz grün**: Der Median ist unauffällig, die Standardabweichung gering. Die Aufgabe war mittelmäßig anstrengend und die Teilnehmer waren sich da ziemlich einig. Wenn der Prototyp verbessert wird, können diese Fälle betrachtet werden, sie sollten aber nicht oberste Priorität sein.
4. **rot schwarz**: Im Mittel war es eine anstrengende Aufgabe, die Standardabweichung deutet auf eine gleichmäßige Verteilung der Werte hin. Diese Fälle sollten genauer untersucht werden.

5. **rot** **grün**: Die Aufgabe war anstrengend und die Teilnehmer haben ungefähr denselben Wert angegeben. Hier ist eindeutig ein Problem, welches gelöst werden muss.
6. **rot** **rot**: Die Aufgabe war im Mittel anstrengend, es gibt aber wenige extreme Ausreisser. In diesem Fall sollte herausgefunden werden, was diese Ausreisser besser oder zumindest anders gemacht haben, um weitere Schlüsse ziehen zu können.
7. **grün** **schwarz**: Im Mittel wurde die Aufgabe als wenig anstrengend empfunden, die Werte sind aber etwas gestreut. Diese Fälle sollten erst untersucht werden, wenn alle anderen behandelt wurden.
8. **grün** **rot**: Die Aufgabe war für den Großteil der Teilnehmer einfach, wenige denken aber ganz anders darüber. Es sollte herausgefunden werden, was mit den Wenigen los war, um weitere Schlüsse ziehen zu können.
9. **grün** **grün**: Die Teilnehmer empfanden die Aufgabe durchgehend als wenig anstrengend. Solche Fälle können als Positivbeispiel gesehen und sollten wenn möglich beibehalten werden.

Es ist also ratsam, alle Kombinationen mit einer roten Zelle und grün/grün Paarungen genauer anzusehen, was im Folgenden getan wird.

### 3.8.3 Aufgaben

#### Aufgabe 1

*Von welchem Konzept gibt es die meisten Instanzen?*

Beim **Physical Demand** ist eine grün/grün Kombination vorhanden (3/1,25). Das bedeutet, dass die Testpersonen wenig physische Anstrengung empfanden. Da die Frustration im Mittel niedrig war (2,5/2,31) können die Ergebnisse so interpretiert werden, dass das Force-Layout für eine solche Aufgabe übersichtlich genug ist, denn die Teilnehmer mussten nicht viel zoomen oder navigieren. Die anderen Werte sind jedoch nicht so gut.

**Schlussfolgerung:** Da diese als relativ einfach eingeschätzte Aufgabe nicht durchweg gut gelöst wurde, muss diese Anforderung in der neuen Komponente verbessert werden. Prinzipiell scheint die Aufgabe für Maschinen besser geeignet zu sein als für Menschen, weswegen eine Automatisierung in irgendeiner Form in Betracht gezogen werden sollte.

#### Aufgabe 2

*Welche Klasse hat die meisten direkten Subklassen?*

Eine schwarz/rot Kombination bei **Effort** (5/3,27) lässt darauf schließen, dass die User hier eindeutig geteilter Meinung sind, da der Median auch genau in der Mitte

liegt. In dieser Aufgabe mussten sie die Klasse mit den meisten direkten Subklassen finden. Dies konnte am Besten bewältigt werden, wenn zunächst das niedrigste Zoomlevel eingestellt und dann auffällige Ansammlungen von Klassen identifiziert wurden, die danach genauer untersucht werden mussten, indem die Subklassen manuell abgezählt wurden. Für eine einfache Aufgabe ist das definitiv zu viel Aufwand.

**Schlussfolgerung:** Wie Aufgabe 1 scheint es besser, die Lösung einem Computer zu überlassen und den Vorgang zu automatisieren.

### Aufgabe 3

*Finden Sie das Konzept PhysicalUnit. Nennen Sie Beispiele für Eltern, Kinder und Geschwister.*

Beim **Temporal Demand** und **Effort** ist eine schwarz/rot Paarung zu sehen. Das bedeutet, dass die gefühlte Anstrengung bzw. der empfundene Schwierigkeitsgrad der Aufgabe und der Zeitdruck von einer Hälfte als hoch und von der anderen als niedrig eingeschätzt wurde. Die Aufgabe konnte durch Nutzung der Volltextsuche und/oder manuelle Suche gelöst werden. Die Musterlösung beinhaltet das Suchen des entsprechenden Konzepts, Hinzufügen desselben zu den Lesezeichen, Löschen des Suchfilters und danach manuelles Untersuchen der Topologie. Während der Studie fiel auf, dass einige Teilnehmer vom Verhalten der Volltextsuche, die ausschließlich Knoten anzeigt und deren Kanten ausblendet, sehr überrascht wurden. Diese probierten dann andere Lösungsmöglichkeiten aus, obwohl sie eigentlich auf dem richtigen Weg waren. Alle Teilnehmer hatten das Problem, dass sie von anderen bewegten Objekten gestört wurden, während sie die Topologie untersuchten.

**Schlussfolgerung:** Dieser Prozess muss in der neuen Komponente einfacher gestaltet werden. Direkt anbieten würde sich eine überarbeitete Suche und/oder ein statisches Layout.

### Aufgabe 4

*Nennen Sie Beispiele für Propertys der Ressource PerSecond.*

Durch einen Bug in der Software wurde die gesuchte Ressource nicht angezeigt, weswegen hier keine Schlüsse gezogen werden können.

### Aufgabe 5

*Welche Ressourcen verbindet die Property exactMatch? Nennen Sie 3 Beispiele.*

Hier ergaben sich rote Felder bei allen Anstrengungstypen außer dem Mental Demand, was darauf schließen lässt, dass die Testpersonen an dieser Stelle bereits wussten, wo sie welche Funktion finden. Bei dieser Aufgabe hatten viele Teilnehmer Schwierigkeiten, eine Aktionssequenz festzulegen. Aber selbst wenn sie eine korrekte Abfolge gefunden hätten, wäre der Lösungsweg so umständlich gewesen, dass sie es unmöglich innerhalb des Zeitlimits schaffen hätten können. Während der Studie fiel

auf, dass die Volltextsuche nicht optimal arbeitet, da sie bei keinem Ergebnis einfach nichts anzeigt anstatt den Benutzer z. B. über ein Popup zu informieren, dass nichts gefunden wurde.

**Schlussfolgerung:** Die Funktionalität, die nötig ist, um diese Aufgabe zu lösen, muss in der neuen Komponente vorhanden sein. Außerdem sollte besser darauf geachtet werden, dass der Benutzer adäquat über den Systemzustand informiert wird.

## Aufgabe 6

*Über welche Konzepte sind CurrencyUnit und MechanicsUnit verbunden? Geben Sie den vollständigen Pfad an.*

Nur der **Effort** zeigt eine schwarz/rot Kombination (6/3,02), was darauf hindeutet, dass die Nutzer uneins waren, weil sie stark unterschiedliche Werte angaben. Eigentlich sollte diese Angabe im Mittel hoch sein, da nur eine Person die Aufgabe gelöst hat. Trotzdem sind drei niedrige Werte (3, 2, 2) zu finden. Durch einen Softwarefehler kam es wenige Male vor, dass die beiden Konzepte tatsächlich nicht verbunden angezeigt wurden. Diese drei Teilnehmer machten richtige Angaben basierend auf einer falschen Anzeige und waren zu Recht der Meinung, dass sie die Aufgabe gut gelöst hätten. Aus den anderen Typen von Anstrengung kann direkt keine Schlussfolgerung gezogen werden, in der Nutzerstudie wurde aber deutlich, dass die Probleme auf mehreren Ebenen lagen: Die Volltextsuche, das dynamische Layout und Festlegen der Aktionssequenz bereiteten einigen Nutzern Schwierigkeiten. Außerdem benötigten die Teilnehmer bei dieser Aufgabe im Schnitt am längsten Zeit.

**Schlussfolgerung:** Dieser Anwendungsfall sollte in der neuen Komponente schneller vonstatten gehen, damit sich Benutzer dafür nicht so lange aufhalten müssen. Außerdem muss besser ersichtlich sein, wie ein solches Ziel zu erreichen ist.

## Aufgabe 7

*Nennen Sie Beispiele für Ausprägungen der Klasse CurrencyUnit, wo die Eigenschaft CurrencyExponent ungleich Zwei ist.*

Diese Aufgabe war bezüglich **Mental Demand** und **Performance** anstrengend. Sie konnte gelöst werden, indem die Klasse CurrencyUnit markiert und dann jede Instanz einzeln inspiziert wurde. Eine mögliche Erklärung für den hohen Wert beim Mental Demand ist die Aufgabenstellung, wo zum ersten und einzigen Mal das Wort »Ausprägung« fällt und auch alle Begriffe verwendet werden. Einige Teilnehmer taten sich mit den neuen, unbekannten Begriffen schwer. Es wäre auch möglich, dass es auf das manuelle Suchen und Inspizieren der Instanzen zurückzuführen ist. Das würde bei denen, die Probleme mit den Begriffen hatten, noch dazu kommen.

Für die hohe Anstrengung bei **Performance** ist wohl definitiv der umständliche Vorgang verantwortlich, mit dem Propertys einer Instanz angesehen werden können. Durch das Force-Layout bewegen sich die Knoten im Graphen ständig, der

Nutzer muss aber eine Zeit lang mit der Maus über einem Knoten bleiben, damit der Tooltip erscheint, in dem dann letztendlich die Datatype Propertys aufgelistet werden.

**Schlussfolgerung:** Der Nutzer sollte die Propertys in Ruhe durchsuchen können, ohne sich auf andere Dinge konzentrieren zu müssen. Die Darstellung sollte des Weiteren übersichtlich genug sein.

### Aufgabe 8

*Blenden Sie alle Klassen mit »unit« im Namen aus.*

Diese Aufgabe war mit der vorhandenen Komponente nicht lösbar, weswegen auch keine Schlüsse gezogen werden können. Zwar versuchten es einige Teilnehmer mit der Volltextsuche und Ausdrücken wie »!unit« oder »-unit«, aber nicht alle. Darum kann kein populärster Kandidat identifiziert werden. Speziell für Informatiker können aber zum Beispiel Regular Expressions in Betracht gezogen werden.

### Aufgabe 9

*Legen Sie eine Art Lesezeichen für SIDerivedUnit für einen späteren Zugriff an.*

Bei dieser Aufgabe ist jedes Feld grün bis auf die Standardabweichung bei den Angaben zum **Effort**, das schwarz ist. Diese Statistiken decken sich mit der Beobachtung der Testpersonen. Sie verstanden schnell, was zu tun ist, wenige hatten es auch vorher schon systematisch angewendet. Auch die Ausführung ging schnell vonstatten, diese Funktion konnte problemlos benutzt werden.

**Schlussfolgerung:** Diese Funktion hat gut funktioniert und kann so in die neue Komponente übernommen werden.

### Aufgabe 10

*Welche Ressource hatten Sie 5 Navigationsschritte zuvor ausgewählt?*

Keine der Testpersonen konnte sich korrekt daran erinnern, welche Ressource sie als fünftletztes angewählt hatte. Das es viele trotzdem sehr engagiert probierten, zeigen die hohen Anstrengungen bei **Mental Demand**, **Performance** und **Effort**. Die Streuung der Werte bei letzterem ist darauf zurückzuführen, dass wenige zwar eine Angabe machten und sich sicher fühlten, diese jedoch falsch war.

**Schlussfolgerung:** Der Benutzer sollte mit der neuen Komponente in der Lage sein, seine Navigation nachvollziehen zu können.

### 3.8.4 Feedback

Zusätzlich zum Task Load Index bekamen die Testpersonen noch die Möglichkeit, ihre Kritik in eigene Worte zu fassen. Diese wurde dann vom Autor zusammen mit

den mündlichen Anmerkungen in eine diskrete Anzahl an Problemen überführt.

Tabelle A.3 zeigt, was den Testpersonen zufolge einfach zu lösen war. Das Hinzufügen von Lesezeichen ist hier Spitzenreiter (6/10 Stimmen), das deckt sich auch mit den Ergebnissen des Task Load Index von Aufgabe 9. Auf Platz zwei und drei sind das Suchen von Klassen (4/10), also die Textsuche, und das Zählen von Instanzen (3/10). Es war einfach verständlich, dass große Klassen mehr Instanzen beinhalten. Eine Testperson merkte aber auch an, dass man daraus nicht ableiten könne, wie viele Instanzen *genau* vorhanden sind.

Gut gefiel einem Drittel der Testpersonen die parallele Darstellung der unterschiedlichen Bestandteile (Tabelle A.5). Im Gespräch mit den Testpersonen entstand aber der Eindruck, dass eher zu viel Information dargestellt wurde. Punktegleich auf Platz 2 kamen dann noch die Suche und die physikalischen Eigenschaften des Layouts.

Besonders schwer fanden 40 % der Testpersonen das Finden von Verbindungen zwischen Ressourcen (Tabelle A.2). Je ein Drittel nannte hier außerdem das Filtern eines Views und das Nachvollziehen der eigenen Navigation. Eine weitere Nennung war das Inspizieren von Propertys (20 %).

40 % der Testpersonen gaben an, dass die Software sehr träge auf Eingaben reagierte (Tabelle A.4). Weitere Designfehler waren die (Nicht-)Lesbarkeit der Beschriftungen, unübersichtliches Layout, das Fehlen einer Reset-Funktion und die verwendeten Begriffe mit je 2/10 Stimmen. Jeweils einer Person wünschte explizit eine Vollbildfunktion und Regular Expressions.

## 3.9 Zusammenfassung

Die bestehende Datenvorauswahlkomponente wurde in einer Nutzerstudie mit 10 Teilnehmern evaluiert. Dabei musste jeder Teilnehmer 10 Aufgaben lösen, einen Task Load Index für jede davon ausfüllen und konnte außerdem noch frei formuliertes Feedback angeben. Durch Beobachtung der Testpersonen und die aufgezeichneten Daten wurde die Komponente auf ihre Usability untersucht.

Es wurden verschiedene Aspekte der Studie diskutiert. Bezuglich der Effektivität der Teilnehmer wurde deutlich, dass die Aufgaben 3 bis 7 am schwierigsten zu lösen waren. Um herauszufinden, warum, wurde zusätzlich zu den Beobachtungen der Task Load Index ausgewertet. Dabei zeigten sich viele Verbesserungsmöglichkeiten, unter anderem Automatisierung von Aufgaben, Vereinfachung der Bedienung, bessere Rückmeldungen an den Benutzer (*gulf of evaluation*) und Funktionen klar kennzeichnen.

Zuletzt wurde das Feedback der Studienteilnehmer zusammengetragen. Diese fanden das Auffinden von Verbindungen und Suchen/Filtern der Daten am schwierigsten und die Trägheit der Software am störendsten. Positive Spitzenreiter waren die Lesezeichen und die parallele Darstellung von verschiedenen Bestandteilen der Ontologie.

Diese Erkenntnisse werden im nachfolgenden Kapitel, der Konzeption der neuen Komponente, besondere Beachtung finden.

# 4 Konzeption

In diesem Kapitel wird basierend auf dem Information Seeking Mantra von Schneiderman [Shn96], auf den Ergebnissen der Nutzerstudie (Abschnitt 3.8) und auf den Problemen anderer Visualisierungen (Abschnitt 2.4) eine Anforderungsanalyse durchgeführt und in weiterer Folge das Konzept für eine Datenvorauswahlkomponente entwickelt.

## 4.1 Anforderungsanalyse

In den letzten Kapiteln sind an vielen verschiedenen Stellen unterschiedliche Anforderungen an eine Software, die Navigation in großen semantischen Datensätzen ermöglichen soll, aufgetaucht. Diese werden hier nach dem Information Seeking Mantra klassifizierend gesammelt, wobei wenn nötig neue Klassen eingeführt werden, wenn diese im Mantra nicht vorkommen (z. B. Interaktion, nicht-funktionale Anforderungen).

### 4.1.1 Overview

Der Benutzer soll sich einen Überblick über den Datenbestand verschaffen können. Von den Untersuchungskriterien aus Kapitel 2 sind folgende für den Überblick relevant:

- Sichtbarkeit der Elemente
- Unterscheidbarkeit der Elemente
- Lesbarkeit von Schrift
- Platzausnutzung
- Layoutflexibilität
- Clustering (als Teil der Layoutflexibilität)

In den korrelierenden Aufgaben aus der Nutzerstudie (1 und 2) wurde deutlich, dass quantitative Aufgaben besser automatisiert werden sollten. Die Teilnehmer schlugen im Feedback die Verbesserung von Beschriftungen und des Layouts vor. Während der Studie kam es vor, dass Teilnehmer auf Funktionen der Komponente vergaßen, weil sie eine zehnminütige Einführung bekamen und man sich da naturgemäß nicht alles merkt. Um dem beizukommen sollte die Komponente ein kurzes Tutorial und eine Hilfe-Funktion zur Verfügung stellen.

Daraus lassen sich die folgenden Anforderungen ableiten:

- O1. **Darstellung der Ontologiebestandteile:** Damit der Benutzer sich einen Überblick verschaffen kann, müssen als ersten Schritt die einzelnen Ontologiebestandteile angezeigt werden.
- O2. **Sichtbarkeit der Elemente:** Elemente der Visualisierung dürfen nicht zu klein sein und sollten sich nicht überschneiden.
- O3. **Unterscheidbarkeit der Elemente:** Elemente der Visualisierung, die unterschiedlichen Typs sind, müssen als solche zu identifizieren sein.
- O4. **Lesbarkeit der Beschriftung**
- O5. **Platzausnutzung:** Platz auf dem Bildschirm ist kostbar und sollte bestmöglich genutzt werden.
- O6. **Automatisierung der Aufgaben:** Potenziell langwierige Aufgaben (z. B. Suchen, Abzählen) sollen automatisch ausgeführt werden können.
- O7. **Layout:** Es muss den Benutzer unterstützen und nicht, wie es in der Nutzerstudie vorkam, behindern.
- O8. **Layoutflexibilität:** Die Komponente soll verschiedene Layoutalgorithmen zur Verfügung stellen, damit der Benutzer das aktuell am Besten geeignete verwenden kann.
- O9. **Clusteranalyse:** Die Software soll Clusteranalyse unterstützen, auf Basis derer die Visualisierungselemente neu angeordnet werden können. Das hilft dem Benutzer zusammengehörende bzw. ähnliche Elemente zu identifizieren.
- O10. **Deterministischer Clusteringalgorithmus:** Wenn möglich, soll der Clusteringalgorithmus für dieselbe Eingabe auch immer dasselbe Ergebnis liefern. Das erhöht die Sicherheit des Benutzers, wenn er Aktionen wiederholt.
- O11. **Orientierung in der Visualisierung:** Wenn eine Visualisierung große Datensätze darstellt, ist sie oft selbst sehr umfangreich. Deswegen sollte der Benutzer immer wissen, an welcher Stelle er sich befindet.
- O12. **Tutorial:** Die Komponente soll eine kurze Führung durch die Funktionen anbieten.
- O13. **Hilfe:** Die Komponente soll ein Hilfe-Werkzeug zur Verfügung stellen.

### 4.1.2 Zoom

Der Benutzer muss die Möglichkeit haben, interessante Bereiche zu vergrößern. Hier denkt man zuerst an den geografischen Zoom, es besteht aber auch die Möglichkeit semantisch zu zoomen. Daraus können folgende Anforderungen abgeleitet werden:

- Z1. **Pan & Zoom:** Ein durch Google Maps bekanntes Prinzip, mit dem der Benutzer sich beliebig in der Visualisierung bewegen kann.

- Z2. Bewegliche Visualisierungselemente:** Da Layoutalgorithmen nicht vollständig korrekt arbeiten bzw. der Benutzer trotzdem etwas an der Anordnung auszusetzen haben kann, müssen sich die Elemente der Visualisierung verschieben lassen. So kann der Benutzer interessante Bereiche freilegen.
- Z3. Semantic Zoom:** Eine Variante von Semantic Zoom sollte in Betracht gezogen werden. Es könnte zum Beispiel sinnvoll sein, die Beschriftungen erst ab einem gewissen Zoomlevel anzuzeigen, da sie vorher sowieso zu klein sind.

### 4.1.3 Filter

Uninteressante Elemente müssen herausgefiltert werden können. Solche Filtermöglichkeiten waren auch Bestandteil der Untersuchungskriterien in Kapitel 2. In der vorhandenen Komponente ist so etwas nur eingeschränkt möglich, da keine inversen Filter unterstützt werden (siehe Aufgabe 8 der Nutzerstudie, »zeige keine Elemente mit unit im Namen an«). Im Feedback der Nutzerstudie wurde angemerkt, dass das Filtern des Datensatzes nicht gut funktioniert. Eine Person wünschte sich Regular Expressions. Zusätzlich zu vom Benutzer festgelegten Filtern sind auch vom System generierte Filter denkbar. So wie die in Abschnitt 2.1.3 vorgestellte Faceted Navigation, die u. a. im Information Retrieval viel Verwendung finden.

- F1. Umfangreichere Filter:** Der Benutzer soll nicht nur nach dem Label einer Ressource filtern können (Suche), sondern nach verschiedenen Kriterien: Klassen mit Property XY, Instanzen von Klasse AB, Instanzen mit einer Property mit Label CD, Klassen mit Propertys vom Datentyp EF und ähnliche.
- F2. Facets:** Die Software soll Facets generieren, nach denen der Benutzer die angezeigten Elemente weiter einschränken kann. Zwar ergibt sich das Problem, dass eine Property »Länge« in verschiedenen Kontexten verschiedene Bedeutungen hat (Länge eines Flusses? Länge eines Meetings? Länge als Maß für die Größe eines algebraischen Moduls?). Dieses wurde aber bereits in Abschnitt 2.3.3 behandelt und ist grundsätzlich lösbar.
- F3. Regex:** Für Benutzer mit entsprechenden Fähigkeiten sicher ein praktisches Feature, die anderen müssen es nicht benutzen.

### 4.1.4 Details on demand

Der Benutzer muss Details zu einem Element aufrufen können, wenn es nötig ist. Zum Beispiel müssen nähere Informationen einer Klasse wie Anzahl der Instanzen oder implementierte Propertys angezeigt werden. Diese Funktionalität wurde im Feedback der Nutzerstudie als zu umständlich beurteilt, da die Details durch einen Tooltip über bewegenden Objekten angezeigt wurden. Folgende Anforderungen können festgemacht werden:

- D1. Details auf Anfrage:** Wenn der Benutzer es benötigt, müssen Details zu einem Element angezeigt werden können.
- D2. Durchsuchbarkeit:** Da in einem großen semantischen Datensatz sehr viele Details vorhanden sein können, müssen diese durchsuchbar sein.

D3. **Übersichtlichkeit:** Aus demselben Grund müssen die Details übersichtlich präsentiert werden.

#### 4.1.5 Relate

Wenn es Verbindungen zwischen Elementen gibt - egal über wie viele »Hops« - muss der Benutzer diese einfach finden können. Die korrelierenden Aufgaben in der Nutzerstudie konnten kaum gelöst werden. Bei der einen war die Funktionalität nicht vorhanden, bei der anderen war der Vorgang zu umständlich.

R1. **Relationen finden:** Der Benutzer muss Verbindungen zwischen mehreren Ressourcen des Datensatzes finden können. Diese Aufgabe muss mit möglichst wenig Aufwand verbunden sein, da die Teilnehmer in der Nutzerstudie für diese Aufgabe am längsten benötigten.

#### 4.1.6 History

Die Aktionen des Nutzers müssen gespeichert werden, damit Undo/Redo unterstützt werden kann. Wenn die Aktionen auch noch visuell dargestellt werden, kann der Benutzer seine Aktionen später auch besser nachvollziehen, was in der Nutzerstudie kein Teilnehmer korrekt schaffte. Daraus ergeben sich folgende zwei Anforderungen:

H1. **Navigation/Aktionen nachvollziehen:** In der Nutzerstudie wurde festgestellt, dass sich die Testpersonen vielleicht an die letzten zwei bis drei, aber nicht die fünfletzte Aktion erinnern können. Funktionalität, um die eigene Navigations- bzw. Aktionsvergangenheit nachvollziehen zu können, muss in der neuen Komponente vorhanden sein.

H2. **Undo/Redo:** Bei der Navigation in großen Datensätzen ist es leicht möglich, dass man in eine Sackgasse läuft. Dann muss der Benutzer zu einem bekannten Punkt in der Vergangenheit zurückkehren können, von wo aus er eine neue Erkundung starten kann.

#### 4.1.7 Navigation

Anforderungen an die Navigation wurden bereits in Kapitel 2 gestellt, wo Navigationshilfen (Pivoting, Lesezeichen, Tastaturkürzel, Zurück-Button, für History siehe oben) Teil der Untersuchungskriterien waren. In der Nutzerstudie wurde die Nützlichkeit von Lesezeichen bestätigt und im Feedback ein Reset-Button gewünscht, der alles zurücksetzt. Ein weiterer Bestandteil der Navigation ist die Suche, die in der Nutzerstudie mehr schlecht als recht abschnitt. Daraus ergeben sich folgende Anforderungen:

N1. **Pivoting:** Der Benutzer soll Ressourcen, zu denen er noch nicht navigiert hat, die ihn aber interessieren könnten, vorgeschlagen bekommen. Das ist vergleichbar mit einem Empfehlungssystem, TopBraid implementierte aber auch eine simplere Version davon, wo mit der Tastenkombination STRG+Klick direkt zu jeder Ressource navigiert werden konnte.

- N2. **Lesezeichen:** Diese funktionierten in der Nutzerstudie sehr gut und können von der alten Komponente übernommen werden.
- N3. **Tastaturkürzel:** Prinzipiell ist es immer besser, verschiedene Interaktionen zum selben Ziel zu unterstützen, weswegen Tastaturkürzel für Vielbenutzer sinnvoll sind, auch wenn Einsteiger sie nicht benötigen.
- N4. **Zurück-Button:** Diese Anforderung ist deckungsgleich mit Undo/Redo aus dem vorigen Abschnitt. Die eine Funktion ist nötig, um die andere zur Verfügung zu stellen.
- N5. **Reset-Button:** Auch diese Anforderung ist im Prinzip durch Undo/Redo bereits abgedeckt, denn Reset ist nichts anderes als Undo bis zum Anfang. Trotzdem wird sie noch einmal explizit festgehalten.
- N6. **Textsuche:** In der Nutzerstudie zeigte sich, dass deren Verhalten sich nicht mit den Erwartungen der Nutzer deckte, weswegen sie überarbeitet werden muss. Die grundlegende Funktionalität wurde als sinnvoll empfunden.

#### 4.1.8 CRUISe-spezifische Anforderungen

Die Software soll eine funktionierende Komponente zur Datenvorauswahl in der CRUISe Laufzeitumgebung sein. Deswegen müssen Schnittstellen für Datenein- und ausgabe angepasst, obligatorische Methoden implementiert und eine Funktion zur Auswahl von Ressourcen zur Verfügung gestellt werden.

- C1. **Schnittstellen zur Dateneingabe:** Unterstütztes Format (RDF, OWL), Syntax (JSON, XML) und Herkunft (Web URL, DaRe) müssen festgelegt werden.
- C2. **Schnittstellen zur Datenausgabe:** Es muss festgelegt werden, in welchem Format und in welcher Syntax die Daten an die nächste Komponente weitergereicht werden.
- C3. **Obligatorische Methoden:** Damit die Komponente grundlegende Events und Operationen wie Laden, Öffnen, Schließen etc. unterstützt, müssen von CRUISe vorgegebene obligatorische Methoden implementiert werden.
- C4. **Komponentenbeschreibung:** Die Komponente muss mittels SMCML (Semantic Mashup Component Description Language) beschrieben werden.
- C5. **Funktion zur Auswahl von Ressourcen:** Die Komponente soll zur Datenvorauswahl verwendet werden, weswegen die entsprechende Funktionalität dafür vorhanden sein muss.
- C6. **Funktion zum Bestätigen der Auswahl:** Der Benutzer muss der Komponente mitteilen können, dass sie die ausgewählten Daten an die Nächste weiterreichen soll.

### 4.1.9 Nicht-funktionale Anforderungen

Aus Beobachtungen während der Nutzerstudie und dem Feedback derselben, sowie weiteren Überlegungen ergeben sich folgende nicht-funktionale Anforderungen:

- NF1. **Unmittelbare Rückmeldungen:** Im Feedback der Nutzerstudie wurde das träge Verhalten der Software bemängelt. In der neuen Komponente soll sich der Zustand des Systems unmittelbar nach einer Aktion des Nutzers verändern, auch wenn es nur die Anzeige eines Ladebalkens ist. So wird der gulf of evalution überbrückt.
- NF2. **Mögliche Funktionen sichtbar machen:** Während der Nutzerstudie kam es vor, dass die Teilnehmer entweder keine Aktionssequenz festlegen konnten oder auf Funktionen vergaßen. Um dem beizukommen sollen Funktionen der Software gut sichtbar gemacht werden.
- NF3. **Verständlichkeit:** Die Darstellung sollte so einfach wie möglich und so detailliert wie nötig sein, damit verschiedene Benutzergruppen damit zurechtkommen. In der Nutzerstudie taten sich besonders die Personen, welche gar nichts mit Informatik zu tun haben, mit den ontologiespezifischen Begriffen schwer. Diese sollten weitestmöglich reduziert werden, ohne das Ontologieexperten davon verwirrt werden. Zu dieser Anforderung gehört auch, dass ein komplexer Vorgang in mehrere einfache aufgeteilt wird und ähnliches.
- NF4. **Fehlerrobustheit:** Im Falle eines Fehlers darf die Software nicht unbenutzbar werden.
- NF5. **Korrektheit:** Die Software muss die verwendeten Daten der Wahrheit entsprechend darstellen.

## 4.2 Frontend

In den folgenden Abschnitten wird auf Basis der eben bestimmten Anforderungen ein User Interface konzipiert.

### 4.2.1 Überblick

Die Komponente besteht aus einem Frontend für die Nutzerinteraktion und einem Backend, welches für speicher- und rechenintensive Aufgaben wie Clustering zuständig ist.

Der wichtigste Bestandteil des Frontends ist der **Main View** (Abb. 4.1). Hier werden die Daten angezeigt, aber immer nur ein Bestandteil der Ontologie (Anforderungen O1, NF3). Es wird kein dynamisches Layout mehr verwendet und die Anzahl der verschiedenen Views auf zwei reduziert (Anforderungen O7, NF3). Da in der Visualisierung potentiell sehr viele Elemente angezeigt werden können, wird der Benutzer durch eine Overview/Detail Ansicht unterstützt (Anforderung O11). Dort wird Pan & Zoom zur Verfügung gestellt (Anforderung Z1).

Eine **Buttonleiste** stellt weitere Funktionen für den Main View zur Verfügung.

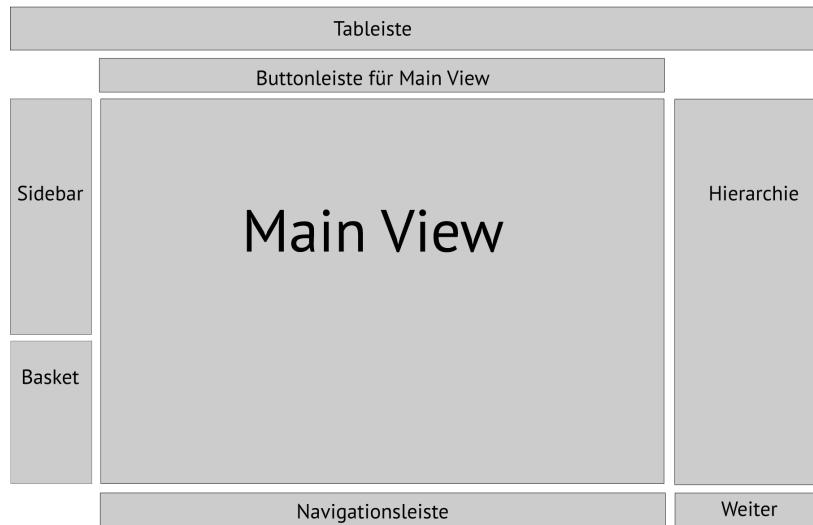


Abbildung 4.1: Layout des Frontends

Hier findet der Benutzer Controls um nähere Informationen aufzurufen (Anforderung D1), Lesezeichen auf Elemente zu setzen (Anforderung N2), Filter basierend auf dem selektierten Element zu erstellen (Anforderung NF3), das Layout zu verändern (Anforderung O8), die Elemente zu clustern (Anforderung O9), nach Ressourcen zu suchen (Anforderung N6) - auch mit Regular Expressions (Anforderung F3) - und Verbindungen zwischen Ressourcen zu finden (Anforderung R1).

Wenn der Benutzer im Main View navigiert, wird er von der **Hierarchieansicht** unterstützt. Diese zeigt die Klassenhierarchie für ein gewähltes Element bis eine Ebene vor *rdfs:Resource*, so wird Anforderung D1 und O1 umgesetzt. Da diese Hierarchie sehr groß werden kann, wird auch hier eine Overview/Detail Ansicht verwendet (Anforderung O11) und Pan & Zoom unterstützt (Anforderung Z1).

Bis jetzt kann der Benutzer den anzuzeigenden Bestandteil der Ontologie nicht umschalten. Das geht mit einer **Tableiste**, die damit Anforderung O1 vervollständigt. Außerdem wird dort ein Hilfe-Button zur Verfügung gestellt, der eine Anleitung zur Nutzung sowie ein kurzes Tutorial anzeigt (Anforderung O13, O12).

Um die angezeigten Elemente einzuschränken, werden dem Benutzer zwei Werkzeuge in einer **Sidebar** in die Hand gegeben: Filter und Facets. Filter stellen globale (tab-übergreifende) Einschränkungen dar, die auf verschiedene Weise anwendbar sind. Mehrere Filter werden automatisch mit einem logischen UND verknüpft (Anforderung F1). Facets werden vom System bereitgestellt und lassen den Benutzer die angezeigten Daten iterativ verfeinern (Anforderung F2).

Jede Aktion des Benutzers - egal ob Navigation, Auswahl, Layout umschalten o. ä. - wird aufgezeichnet und chronologisch dargestellt, sodass er die Abfolge später nachvollziehen kann (Anforderung H1). Dort kann er auch Aktionen rückgängig machen bzw. wiederholen oder alles zurücksetzen (Anforderungen H2, N5, N4). Außerdem werden an dieser Stelle Ressourcen vorgeschlagen, die für ihn vielleicht interessant

sind (Anforderung N1). Das alles wird einer **Navigationsleiste** dargestellt.

Als weitere Unterstützung wird noch analog zu TopBraid Composer ein **Basket** eingeführt. Dieser vereint dabei die Funktion von einer Lesezeichensammlung (Anforderung N2) und der Auswahl des Nutzers (Anforderung C5).

Zuletzt wird noch ein Button benötigt, der Anforderung C6 umsetzt und die nächste CRUISe Komponente aufruft. Im Folgenden werden die hier vorgestellten Bestandteile im Detail behandelt, Entscheidungen erläutert sowie Alternativen aufgezeigt.

#### 4.2.2 Main View

Der Aufbau des Main View ist in Abbildung 4.2 dargestellt. Im Folgenden wird auf die einzelnen Bestandteile eingegangen.

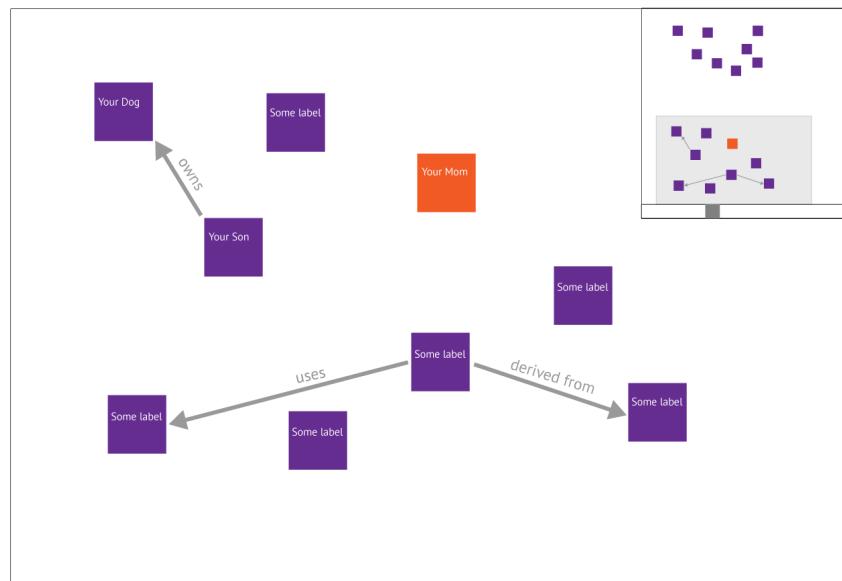


Abbildung 4.2: Mockup des MainViews

Die **Darstellung** erfolgt mit einem Graphen. In den Grundlagen aus Kapitel 2 wurde gezeigt, dass sich diese Art der Darstellung für Hierarchien und Netzwerke eignet.

Andere Möglichkeiten wären:

- **Baum:** Dann müssten Klassen mit mehreren Superklassen mehrfach dargestellt werden, was für den Benutzer nicht sofort verständlich ist.
- **Nested View:** Auch hier müssten Klassen mit mehreren Superklassen mehrfach dargestellt werden. Außerdem ist das Konzept - wenn auch einfach verständlich - nicht sehr geläufig und stellt eine Art Einstiegshürde für Anfänger dar.

Die Bestandteile von Ontologien (Klassen, Instanzen, Propertys) werden als **Knoten** dargestellt. Diese sind je nach Typ eingefärbte Quadrate (Anforderung O3), da

so der Platz bestens ausgenutzt werden kann (Abb. 4.3, Variante A). So wird Anforderung O5 umgesetzt.

**Andere Möglichkeiten** wären:

- *Verschiedene Formen abhängig von der Art des Knotens* (Variante B): So könnten Klassen z. B. Kreise sein, Propertys Dreiecke etc. Ein Vorteil wäre, dass der Benutzer unabhängig von der Farbe des Knotens beurteilen könnte, worum es sich handelt<sup>1</sup>. Bei der Anordnung in einer Matrix verschwenden Dreiecke, Kreise und dergleichen allerdings wertvollen Platz am Bildschirm. Personen mit einer Sehschwäche kann auch durch eine abgestufte Helligkeit oder Sättigung geholfen werden.
- *Verschiedene Formen, eingefärbt* (Variante C): Hier würden sowohl die Form als auch die Farbe auf die Art des Knotens hinweisen. So würden sich Menschen mit und ohne Sehschwäche gleich gut zurechtfinden. Diese Redundanz könnte vom Benutzer aber so interpretiert werden, dass es noch andere Kombinationen gibt (z. B. ein gelbes Quadrat oder einen roten Kreis). Außerdem müsste er sich mehr Information merken. Die freie visuelle Variable kann eventuell für andere Zwecke eingesetzt werden.

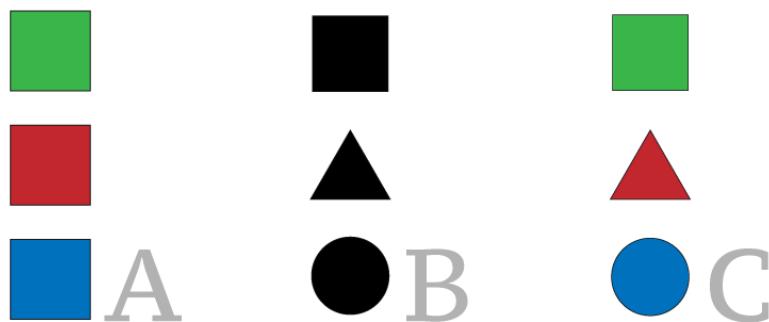


Abbildung 4.3: Mögliche Varianten in der Darstellung von Knoten

Die **Anordnung der Knoten** kann geändert werden (Anforderung Z2). So kann er das Layout beeinflussen und sich besser einen Überblick verschaffen oder schlecht sichtbare Objekte freistellen.

**Andere Möglichkeiten** wären:

- *Feste Anordnung* der Knoten: Die Bedienung wäre dann einfacher, da die Software weniger Funktionen bieten würde. Wahrscheinlich würden sich Benutzer dann über das Layout ärgern, entweder weil es fehlerhaft ist oder weil es einfach nicht ihren Wünschen entspricht. Außerdem macht interaktive Software mehr Spaß zu benutzen.

Die **Beschriftung der Knoten** erfolgt innerhalb derselben, sodass sich außerhalb der Quadrate kein Text überlappen kann. Der Text wird genau so groß gerendert,

<sup>1</sup>Rot- bzw. Grünblindheit betrifft ca. 10 % aller Männer und etwa 1 % der Frauen. [Maz09]

dass er die Breite des Quadrats ausfüllt. Zusätzlich wird über einen Tooltip die vollständige Beschriftung angezeigt. So ist er jederzeit lesbar, solange Knoten sich nicht überlappen (Anforderung O4).

**Andere Möglichkeiten** wären:

- *Beschriftung außerhalb*: Dann könnte sie größer gezeichnet werden und wäre besser lesbar. Allerdings steigt damit die Wahrscheinlichkeit, dass sich Text überlappt und unlesbar wird.
- *Keine Beschriftung*: Sie würde dann mit einem Tooltip o.ä. angezeigt. Hat aber keinen Vorteil gegenüber Beschriftung innerhalb, denn der Tooltip wird dort zusätzlich genutzt.

Die **Beschriftung der Kanten** ist maximal so lang wie die Kante und wird ansonsten abgekürzt. Der Winkel der Schrift entspricht dem der Kante (Abbildung 4.4, Variante B). So überschneidet sich die Beschriftung der Kanten nur, wenn es auch die Kanten überschneiden (Anforderung O4). Da bei hinreichend kleinem Zoomlevel die Beschriftung aber sowieso nicht lesbar ist, kann sie dort weggelassen werden (Variante A) und so ein Semantic Zoom umgesetzt werden (Anforderung Z3).

**Andere Möglichkeiten** wären:

- *Beschriftung nicht an Kante* (Variante C): Sie würde automatisch platziert und über eine Linie mit der Kante verbunden. Allerdings konkurriert der Platz dann mit dem der Knoten und es ist wahrscheinlicher, dass die Beschriftung wegen Überlappungen nicht lesbar ist.

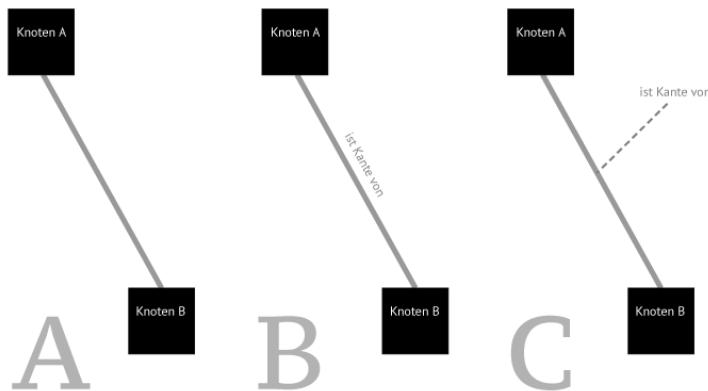


Abbildung 4.4: Platzierung der Beschriftung einer Kante

Mit einer **Overview-Detail** Ansicht weiß der Benutzer zu jeder Zeit, wo er sich im Datensatz befindet (Abb. 4.5). Außerdem kann er dort den Zoom bestimmen. So werden die Anforderungen O11 und Z1 angesprochen. Diese Ansicht kann weitestgehend abstrahiert werden, sie benötigt zum Beispiel keine Beschriftung, weil diese sowieso zu klein wäre. Die Farbe wird allerdings beibehalten um die visuelle Konsistenz zu wahren.

**Zusätzliche Informationen**, wie Datatype Property einer Klasse oder Instanz, aber auch die Anzahl an Subklassen etc. wird über Dialoge angezeigt, die verschoben,

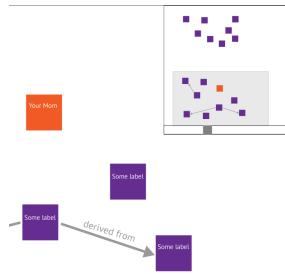


Abbildung 4.5: Overview/Detail Ansicht des Main View

minimiert und in ihrer Größe geändert werden können (Anforderungen D1, D3). Der Inhalt wird dort tabellarisch dargestellt und ist mit einer Textsuche durchsuchbar, außerdem wird eine kurze Zusammenfassung von topologischen Metriken angezeigt (Anforderungen O6, D2, D3).

**Andere Möglichkeiten** wären:

- *Eigener Bestandteil*: Würde zusätzlich Platz benötigen, auch wenn der Benutzer gar keine Details zu Ressourcen sehen will. Ein Vorteil wäre aber, dass er nicht nach dieser Funktion suchen müsste.

Der Benutzer kann die angezeigten Daten mit Hilfe eines Buttons in der Buttonleiste **clustern**. Hier kann entweder ohne weitere Einstellungen geclustert werden, oder eine Property angegeben werden, auf Basis derer die Ähnlichkeitsberechnung. Außerdem kann ein Referenzobjekt angegeben werden, an Hand dessen die Ähnlichkeit berechnet wird (Anforderung O9). Durch diesen Vorgang führt ein Assistent (Anforderung NF3, Abb. 4.6). Wurde das Clustering durchgeführt, sind die Knoten in unterschiedlichen Nuancen ihrer ursprünglichen Farbe eingefärbt und neu angeordnet. Dabei wird unterschieden, ob ein Referenzobjekt angegeben wurde oder nicht. Wenn nicht, bilden die Instanzen die entsprechenden Cluster (Abb. 4.7, Variante A). Wenn doch, steht die Referenz im Zentrum und die Knoten werden darum angeordnet, wobei der Abstand von der Referenz die Ähnlichkeit widerspiegelt. Das Clustering wird in Abschnitt 4.3 genauer behandelt.

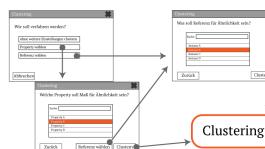


Abbildung 4.6: Assistent für den Vorgang des Clustering

Verbindungen zwischen Ressourcen können durch die **Relate** Funktion gefunden werden. Dabei drückt der Benutzer einen Button in der Buttonleiste und wählt im aufgerufenen Dialog über eine Autocomplete-Suchbox Ressourcen aus und bestätigt. Dann werden sämtliche Verbindungen zwischen diesen Ressourcen angezeigt (Abb. 4.8). Damit wird die Anforderung R1 umgesetzt. Wurden mehrere Knoten ausgewählt, kann der Button auch direkt betätigt werden.

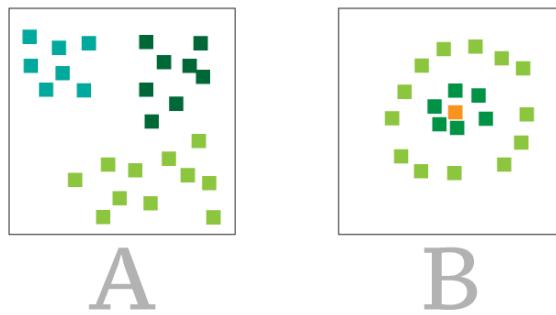


Abbildung 4.7: Clustering

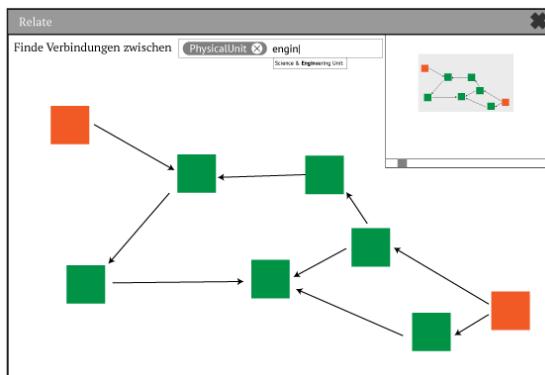


Abbildung 4.8: Verbindungen finden

### 4.2.3 Buttonleiste

Die Buttonleiste stellt zusätzliche Funktionen für den Main View zur Verfügung. Hier kann der Benutzer:

- Details zu Knoten aufrufen
- Lesezeichen auf Knoten setzen
- Filter basierend auf selektierten Knoten erstellen
- das Layout anpassen
- die Knoten clustern
- nach Knoten suchen
- Verbindungen zwischen Knoten finden

Abbildung 4.9 zeigt einen Mockup der Buttonleiste, deren Designentscheidungen im Folgenden erklärt werden. Die Buchstaben stehen für Spring Embedder, Alphabetisch, Radial, Tree und Horizontal Tree, werden aber im Prototypen der Komponente durch Icons ersetzt.

Die **Anordnung** der Buttons ist hierarchisch, was den Funktionen derselben geschuldet ist: Das Layout benötigt verschiedene Buttons, die verschiedene Algorithmen (Vertikal, Horizontal, Force, Radial, Matrix) zur Verfügung stellen. Such- und



Abbildung 4.9: Mockup der Buttonleiste

Relate-Button brauchen Textfelder zur Eingabe. Tangierte Anforderungen: D3, NF2, NF3.

**Andere Möglichkeiten** wären:

- *Flache Struktur*: Alle Elemente wären nach der Reihe angeordnet. Durch fehlende Gruppierung würde hier Übersicht eingebüßt.

Die **Ausrichtung** der Buttons erfolgt aus Platzgründen horizontal, mit einem Trenner zwischen den beiden Hierarchieebenen, damit keine Verwirrung entsteht. Damit werden die Anforderungen O5, NF3 und D3 angesprochen.

**Andere Möglichkeiten** wären:

- *Vertikale Ausrichtung*: Dafür ist im festgelegten Layout kein Platz vorhanden.
- *Kein Trenner, Hierarchien übereinander*: Diese Variante ist wegen Fitt's Law ungünstig, weil die Maus nicht nur horizontal, sondern auch vertikal korrekt positioniert sein müsste, um keine unerwünschte Aktion zu auszuführen. Außerdem verbraucht es mehr Platz.

#### 4.2.4 Hierarchieansicht

Die Hierarchieansicht zeigt sämtliche Superklassen (bzw. Superpropertys) bis eine Ebene unter *rdfs:Resource*, um die Orientierung des Benutzers zu verbessern. *rdfs:Resource* selbst wird nicht angezeigt, da es redundante Information ist (denn ausnahmslos jedes Objekt erbt per Definition von *rdfs:Resource*) und so die Ansicht für Benutzer vereinfacht wird, die mit Ontologien wenig zu tun haben (Anforderung NF3). Abbildung 4.10 zeigt den Aufbau der Hierarchieansicht, der im Folgenden näher erläutert wird.

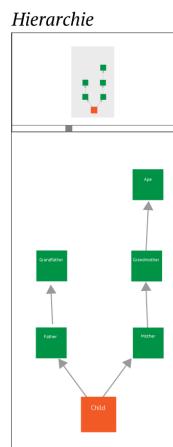


Abbildung 4.10: Mockup der Hierarchieansicht

Die **Knoten und Kanten** in der Hierarchie-Ansicht werden genau so dargestellt wie im Main View. Da dieselben Daten nicht visuell getrennt werden sollten, gibt es hier keine Alternativen (Anforderungen O3, NF3).

Das **Layout** dieser Ansicht ist abhängig von der Anzahl an selektierten Knoten. Wurde eine Einfachselektion durchgeführt, wird ein verkehrter Baum mit der Klasse des Knoten als Wurzel angezeigt (Abb. 4.11, Variante A). Dies entspricht dem mentalen Modell des Benutzers und der Definition von *subClassOf*: Allgemeine Begriffe befinden sich über Spezifischen. Bei einer Mehrfachselektion sind mehrere solche Bäume zu sehen, wenn die selektierten Knoten keine Superklassen (bis auf *rdfs:Resource*) gemeinsam haben (Abb. 4.11, Variante B). Sollte es doch der Fall sein, wird ein Graph dargestellt (Abb. 4.11, Variante C).

**Andere Möglichkeiten** wären:

- *Normaler Baum* (Top-Down): Bei dieser Variante wäre die Ordnung vom Allgemeinen zu Spezifischen von unten nach oben und entspräche deswegen nicht dem mentalen Modell des Benutzers.

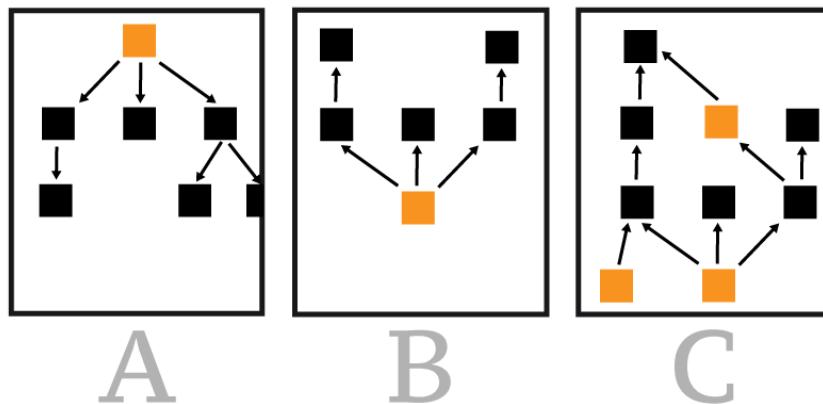


Abbildung 4.11: Mögliche Layouts für die Hierarchie-Ansicht

Das gleiche **Overview-Detail Control** wie im Main View (Abb. 4.5) kommt auch in dieser Ansicht zum Einsatz, weil in großen semantischen Datensätzen auch Hierarchien entsprechend groß werden können (Anforderungen O11, Z1.).

Abschließend ein Mockup, der die Hierarchieansicht anschaulich machen soll. (Abb. 4.10).

## 4.2.5 Sidebar

In der Sidebar befinden sich die Controls, mit denen der Benutzer Regeln für die angezeigten Elemente erstellen, also filtern, kann. Die Controls gliedern sich in Filter (vom Benutzer erstellt, global) und Facets (vom System erstellt, lokal). Global bedeutet, dass angelegte Filter so lange aktiv sind, bis der Benutzer sie deaktiviert oder löscht. Im Gegensatz dazu werden lokale Filter automatisch zurückgesetzt bzw. angepasst, wenn sich die Menge der angezeigten Elemente verändert. Abbildung 4.12 veranschaulicht die Sidebar, die im Folgenden näher behandelt wird.



Abbildung 4.12: Mockup der Sidebar

## Filter

Filter können auf zwei verschiedene Weisen **hinzugefügt** werden. Der Benutzer kann einen oder mehrere Knoten selektieren und den entsprechenden Knopf in der Buttonleiste drücken, dann werden automatisch auf der Selektion basierende Filterregeln erstellt. Sind zum Beispiel die Instanzen A und B selektiert, werden Filter hinzugefügt, sodass

- im Tab »Instanzen« nur diese beiden Instanzen sichtbar sind
- im Tab »Propertys« nur Propertys angezeigt werden, die von diesen Instanzen implementiert werden (Datatype Propertys) oder mindestens eine der Instanzen Teil einer Relation ist (Object Propertys)
- im Tab »Klassen« nur die Klassen der beiden Instanzen angezeigt werden.

Die andere Möglichkeit besteht darin, mittels eines Wizards (Assistenten) eine Filterregel hinzuzufügen.

Der **Assistent** zum Erstellen und Editieren von Filterregeln führt den Benutzer schrittweise durch den Vorgang (Anforderung [NF3](#), Abb. 4.13). Die Schritte sind

1. **Name** der Regel: Der Benutzer muss der Filterregel einen Namen geben, damit er sie später identifizieren kann.
2. **Bestandteil** der Ontologie, auf den sich die Regel bezieht: Entweder Klasse, Property oder Instanz.
3. **Property**, die eingeschränkt werden soll, oder **Datentyp**, den eine Property haben soll. Ersteres dient zum Beispiel dazu, Elemente mit Property »Länge« zwischen 0 und 10 zu finden. Der Anwendungszweck von letzterem ist zum Beispiel, Elemente mit einer Property vom Typ »GeoLocation« zu finden, da später eine Karte angezeigt werden soll (die hier geplante Komponente ist Teil eines größeren Auswahlprozesses).

4. **Wert**, den das Ergebnis aus dem letzten Schritt haben soll: Zum Beispiel Regular Expressions für Strings, Wertebereiche für Zahlen (zuvor Property gewählt) oder der entsprechende Datentyp (zuvor Datentyp gewählt). Hier kann mit einer Checkbox angegeben werden, dass die Filterregel invertiert werden soll (logisches NOT).

**Andere Möglichkeiten** wären:

- *Einzelnes Formular*: Dieses bietet wenig Struktur für den doch relativ komplexen Vorgang und würde es unnötig kompliziert machen. Auch der Entwurf eines solchen Formulars, das übersichtlich und bedienbar ist, gestaltet sich schwieriger als einfach einen Assistenten zu benutzen.

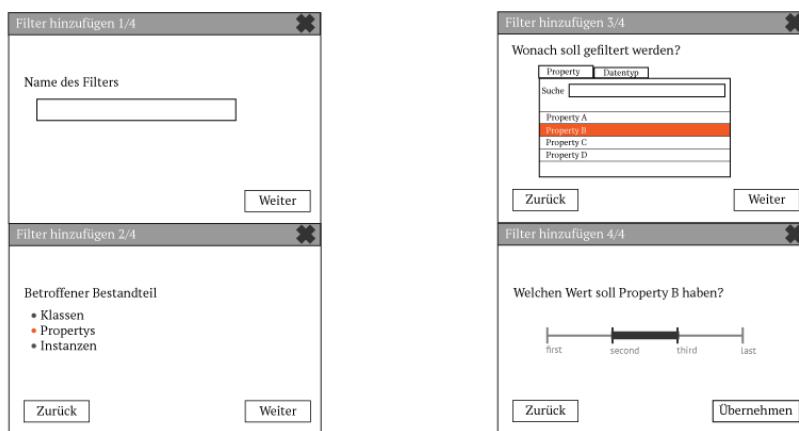


Abbildung 4.13: Assistent für das Hinzufügen/Bearbeiten von Filtern

**Hinzugefügte Filter** werden in einer Liste dargestellt. Das ist platzsparend und übersichtlich (Abb. 4.14, Variante A). Standardmäßig werden fünf Filter angezeigt. Sollten es mehr sein, kann der Benutzer mit dem Pfeil ganz unten die Liste auf volle Größe mit Scrollbar vergrößern und auch Filter innerhalb der Liste verschieben. Die Controls, um Filter zu löschen oder zu editieren, sind rechts neben jeder Zeile zu finden. Der Dialog, um Filter hinzuzufügen, wird sichtbar, nachdem das Plus geklickt wurde.

**Andere Möglichkeiten** wären:

- *Tabelle* (Variante B): Diese Möglichkeit böte den Vorteil, dass mehr Information auf einmal sichtbar ist (mehrere Spalten). Der große Nachteil ist aber, dass die Sidebar möglichst klein sein soll um die Größe des Main Views zu maximieren. Eine Tabelle mit mehreren Spalten auf kleinem Raum ist sehr unübersichtlich.

## Facets

Die Software unterscheidet zwischen nominalen, ordinalen und kontinuierlichen Facets. Die Adjektive beziehen sich dabei auf die zur Darstellung verwendete Skala. Nominalen Daten haben keine inhärente Ordnung, zum Beispiel die Länder Europas. Trotzdem können sie geordnet werden, zum Beispiel alphabetisch oder geografisch.

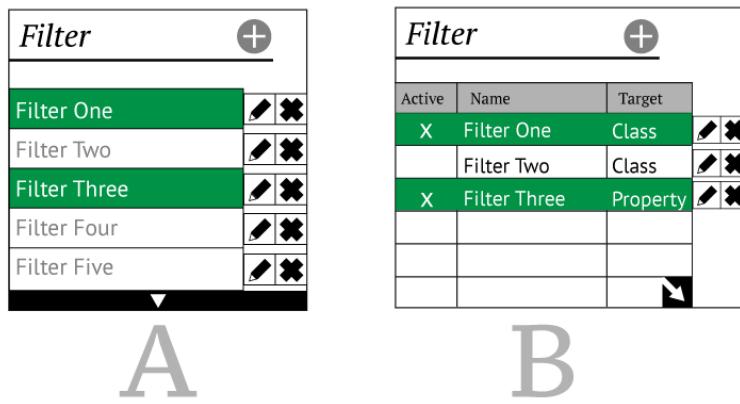


Abbildung 4.14: Mögliche Darstellungen der Filter

Sie werden mit Checkboxes angezeigt. Ordinale Facets sind in sich geordnet, zum Beispiel Gehaltsstufen oder die natürlichen Zahlen. Für sie wird ein einrastender Slider verwendet. Kontinuierliche Facets haben wie Ordinale eine Ordnung, die Differenz zwischen zwei Elementen kann aber beliebig klein sein. Beispiele wären alles, was mit reellen Zahlen ausgedrückt wird, zum Beispiel Gewicht, Größe oder Gehalt. Bei diesen Facets ist der Slider frei verschiebbar. In Abbildung 4.15 wird das Prinzip verdeutlicht.

Facets werden je nach Ontologiebestandteil unterschiedlich gehandhabt. Bei den Klassen werden topologische Kriterien (zum Beispiel Anzahl Subklassen, Anzahl Instanzen, Zentralität) verwendet. Diese können einfach berechnet werden. Bei Instanzen werden Datatype Propertys als Facets aufgefasst. Weil es in einem großen Datensatz sehr viele werden können, werden sie nach Häufigkeit geordnet. Ein Facet für eine Property, welche bei 80 % der angezeigten Instanzen vorkommt, wird höher gewichtet, das heißt weiter oben angezeigt, als ein Facet mit 50 % Vorkommen. Das größte Problem ist jedoch die semantische Trennung. Ein sauberer Datensatz wird für die Länge eines Flusses und für die Länge eines mathematischen Terms zwei verschiedene Propertys bereit stellen. Davon kann jedoch nicht ausgegangen werden, weswegen diese Trennung automatisch durchgeführt werden muss. Böhm et al. [Boh+10] (Abschnitt 2.3.3) clusterten zuerst mit einem schemabasierten Ähnlichkeitsmaß, um danach mit Association Rule Mining innerhalb der Cluster äquivalente Propertys und solche, welche nicht zur Beschreibung des Clusters beitragen, zu identifizieren. Mit dieser Methode können auch Facets für Instanzen bestimmt werden:

1. Schemabasiertes Clustering, um semantisch korrelierende Cluster zu finden.
2. Association Rule Mining innerhalb der Cluster um beschreibende Propertys zu finden, diese werden in einer Tabelle gespeichert
3. Die Zugehörigkeit zu Clustern in einem Vektor an jeder Instanz speichern

Diese Schritte werden vor dem Start der Komponente ausgeführt. Dann kann zur Laufzeit einfach und schnell entschieden werden, welche Facets angezeigt werden müssen: Zuerst wird ermittelt, wie gut die Cluster momentan repräsentiert sind,

indem über angezeigten Instanzen iteriert und die Zugehörigkeitsvektoren addiert werden. Danach werden beschreibende Propertys für die Cluster geladen und als Facets angezeigt.

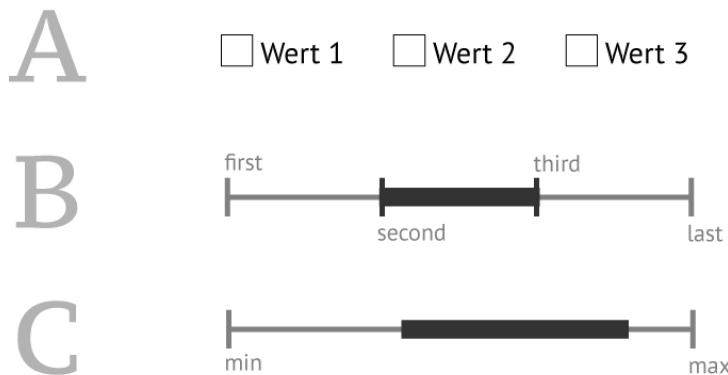


Abbildung 4.15: Die verschiedenen Facets

Die aktiven **Facets** werden samt ihren Controls in Form einer Liste dargestellt (Abb. 4.16, Variante A), welche nach einem Klick auf den Pfeil unten in voller Größe mit Scrollbars dargestellt wird. So sieht der Benutzer wenigstens sofort die Einstellungen von drei Facets. Diese Darstellung benötigt leider relativ viel Platz. Deswegen kann die Position von Facets verändert werden.

**Andere Möglichkeiten** wären:

- *Ausklappbare Liste* (Variante B): So würde sehr viel Platz in der Höhe gespart, denn die Einstellungen werden ausgeklappt, wenn es nötig ist. Der Nachteil ist allerdings, dass die Controls versteckt werden und erstens vom Benutzer gefunden werden müssen, zweitens sind keine Einstellungen sofort sichtbar, was unpraktisch ist.

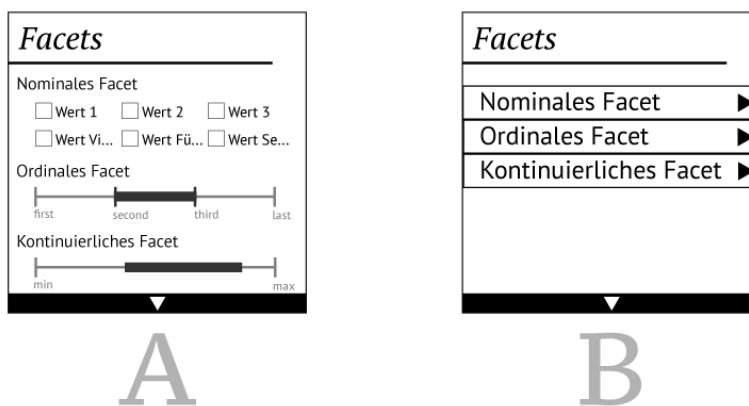


Abbildung 4.16: Mögliche Darstellungen der Facets

## 4.2.6 Navigationsleiste

Es kann passieren, dass der Benutzer eine falsche Aktion ausführt, sich im Datensatz nicht mehr zurechtfindet oder nicht mehr weiter weiß. Mit Hilfe der Navigationsleiste kann er in diesem Fall Undo bzw. Redo oder sogar Reset ausführen oder vorgeschlagene Ressourcen ansehen. Abbildung 4.17 zeigt die Navigationsleiste, die im Folgenden erklärt wird.

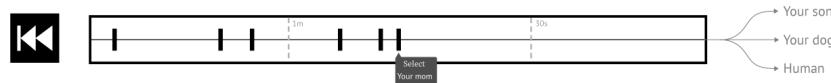


Abbildung 4.17: Mockup der Navigationsleiste

Die **Zeitleiste** hilft beim Nachvollziehen der eigenen Aktionen (Anforderung H1, Abb. 4.18, Variante B). Sie bietet gleichzeitig die Funktionen einer History und eines Undo/Redo-Buttons (Anforderung H2). Standardmäßig wird ein Zeitfenster von drei Minuten angezeigt, Pan & Zoom wird aber unterstützt. Durch die zeitabhängige Darstellung ist die Funktion des Controls leicht zu verstehen.

**Andere Möglichkeiten** wären:

- *Zeitunabhängige Darstellung* (Variante A): Hier würde einfach eine Kette an Aktionen dargestellt. Diese Art der Visualisierung ist abstrakter und schwieriger zu erfassen als Variante B.

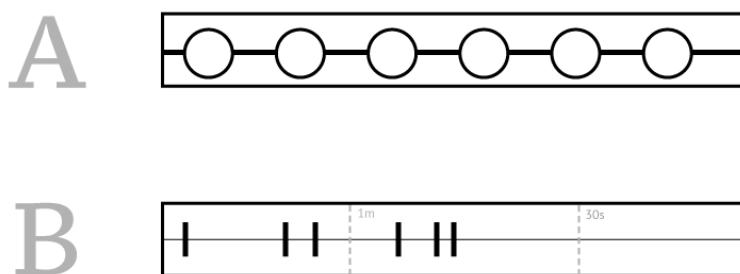


Abbildung 4.18: Mögliche Darstellungen der Zeitleiste

Die **Undo/Redo/Reset** Funktion wird mit dem COMMAND Pattern nach Gamma et al. [Gam+95] umgesetzt. Für jede Interaktion des Benutzers mit der Komponente (z. B. Filter hinzufügen) wird ein Command erzeugt. Für jede wichtige Aktion des Systems (z. B. Deckkraft setzen) wird ein SubCommand erstellt, welches nicht in der Navigationsleiste aufscheint. Makros aus Commands und SubCommands sind möglich und können dynamisch erzeugt werden. Zum Beispiel variiert der Vorgang einer Navigation: Eventuell, aber nicht immer, muss ein Tab umgeschaltet oder die Deckkraft des Ziels angepasst werden. Jedes Command und SubCommand implementiert eine Methode `undo()`, die eine oder mehrere inverse Operationen angibt.

Für die Funktion des **Pivoting**, bei dem Benutzer Ressourcen, die ihn interessieren könnten, vorgeschlagen werden, wird ein verlängerter, sich aufteilender Zeistrahl angedeutet (Abb. 4.19, Variante A). Die Navigation zu einem vorgeschlagenen Element liegt in der Zukunft, weshalb diese Entscheidung naheliegt.

**Andere Möglichkeiten** wären:

- *Liste* (Variante B): Eine Liste funktioniert immer. Diese Liste hat allerdings keinen sichtbaren Bezug zur Zeitleiste, neben der sie direkt angezeigt wird.

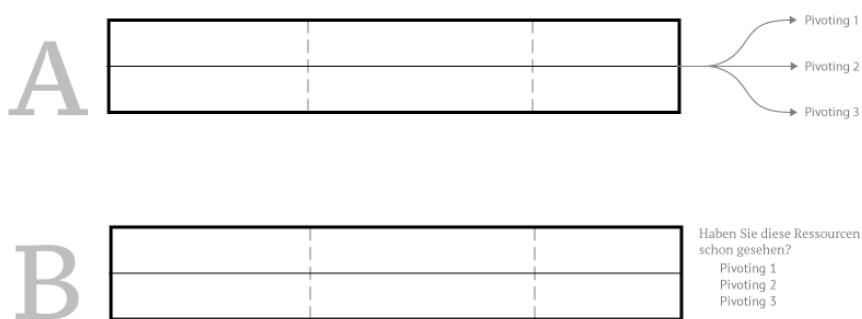


Abbildung 4.19: Mögliche Darstellung der Pivoting-Funktion

Die **Vorschläge für das Pivoting** können auf vielfältige Art und Weise berechnet werden. Das Ziel ist eine optimale Mischung aus Zufälligkeit, inhaltlichem Zusammenhang, topologischer Ähnlichkeit und wie repräsentativ eine Klasse für den Datensatz ist. »Optimal« ist hier irreführend, denn optimal ist, was der Benutzer gerade benötigt und das kann nicht perfekt voraus berechnet, sondern nur angenähert werden. Der inhaltliche Zusammenhang ließe sich durch Aufzeichnung von Nutzerverhalten und Association Rule Mining herstellen. Eine Alternative dazu ist möglicherweise die Berechnung der Ähnlichkeit zwischen Klassen durch den Pfad der spezifischsten gemeinsamen Superklasse zu *rdfs:Resource* [PPM04]. Topologische Ähnlichkeit lässt sich durch verschiedene Zentralitätsmaße in Graphen abbilden, zum Beispiel Betweenness oder Closeness. Um die Wichtigkeit einer Klasse für den Datensatz zu bestimmen, wurde in Abschnitt 2.3.2 die Key Concept Extraction vorgestellt, welche die  $n$  wichtigsten Klassen berechnet. Die drei beschriebenen Metriken (Inhaltlicher Zusammenhang, topologische Ähnlichkeit, Key Concept Extraction) müssten kombiniert und um einen Zufallsfaktor erweitert werden, um das Ziel zu erreichen.

#### 4.2.7 Farbgebung & Schrift

Als Schrift kommen PT Serif und PT Sans zum Einsatz, weil diese frei über Google Webfonts<sup>2</sup> verfügbar sind und verschiedene Schnitte beinhalten. PT Serif wird für statische Beschriftungen in der Software verwendet, also für Text, der immer an derselben Stelle vorhanden ist und sich nicht ändert. Alles, was sich zur Laufzeit

<sup>2</sup><http://www.google.com/webfonts>

verändern kann, wird im serifenlosen Pendant gesetzt.

In Vizboard existiert bereits ein Farbschema, welches die Farbcodes für Blau, Grau, Orange und Rosa definiert. Dieses wird um ein sanftes Grün erweitert, sodass schwarzer Text lesbar bleibt (Abb. 4.20).



Abbildung 4.20: Farbschema mit einem Pangramm in PT Sans

### 4.2.8 Interaktion

Im Folgenden wird beschrieben, wie das Frontend auf Benutzereingaben reagiert.

Im **MainView** kann der Benutzer Knoten mit der linken Maustaste selektieren und verschieben, sowie mit dem Mausrad und dem Slider an der Overview/Detail Ansicht zoomen. Er kann sich mit der linken Maustaste wie bei Google Maps durch den Graphen bewegen oder den Ausschnitt der Overview/Detail Ansicht verschieben. Wenn ein Knoten selektiert wurde, kann mit einem Klick in der Buttonleiste der Detaildialog aufgerufen, ein Filter hinzugefügt und der Knoten zum Basket hinzugefügt werden. Die letzten beiden funktionieren auch mit Mehrfachauswahl und Drag & Drop. Außerdem kann bei Mehrfachauswahl direkt der Relate-Button benutzt werden, sodass Verbindungen zwischen den gewählten Knoten angezeigt werden.

Die **Buttonleiste** enthält folgende Buttons:

1. Detailansicht für Knoten
2. Auswahl zum Basket hinzufügen
3. Filterregel für Auswahl erstellen
4. Layout anpassen
5. Clustering
6. Suche
7. Relationen finden

Die ersten drei Buttons sind ausgegraut, wenn kein Knoten selektiert wurde. Beim Button für Suchen erscheint auf der rechten Seite ein Textfeld. Es stellt eine Live-Suche mit Auto vervollständigung und Regex-Unterstützung zur Verfügung. Bei Relate wird ein Dialog aufgerufen, der dasselbe Textfeld und eine Canvas enthält, aber

mehrere Eingaben annimmt (Abb. 4.21). So wird vor allem Anforderung **NF1** umgesetzt.



Abbildung 4.21: Eingabe in das Relate-Textfeld

Da die **Hierarchie** potenziell sehr groß ist, kann sich der Benutzer wie im Main-View mit Pan & Zoom durch die Anzeige bewegen oder alternativ das Overview-Detail Control benutzen. Selbstverständlich können Knoten mit der linken Maustaste verschoben werden. Wenn ein Knoten im Main View angezeigt werden soll, kann das mit linkem Mausklick bei gleichzeitig gedrückter Steuerung-Taste veranlasst werden.

**Filter** hingegen können über den Plus-Button hinzugefügt werden, er zeigt den entsprechenden Wizard an. Editieren bedeutet die Anzeige des Wizards mit bereits ausgefüllten Feldern. Durch einfachen Linksklick auf den Filter wird dieser aktiviert bzw. deaktiviert. Sowohl Filter als auch **Facets** können in der jeweiligen Liste verschoben werden, um so schnell Zugriff auf die Wichtigsten zu haben.

Die **Zeitleiste** wird bei entsprechend langer Nutzung auch sehr lang, weswegen hier Pan & Zoom funktioniert. Um bis zu einer Aktion ein Undo durchzuführen, klickt der Benutzer auf die entsprechende Repräsentation in der Zeitleiste. Falls er einen Vorschlag aus dem Pivoting wahrnehmen will, passiert dies wie bei den anderen Controls auch mit STRG+Mausklick, da dies eine Navigation zu einer Ressource darstellt.

Wenn der Benutzer ein Element des **Baskets** löschen will, kann er entweder die Entfernen-Taste oder den dazugehörigen Löschen-Button drücken. Mit STRG+Klick kann er es im Main View anzeigen lassen.

## 4.3 Backend

Da die Komponente im Browser laufen wird und deren Funktionen speicher- und rechenintensive Vorgänge beinhalten (z. B. Clustering), ist es sinnvoll, diese auf ein Backend auszulagern. Im Folgenden werden zuerst die Funktionen bestimmt, welche das Backend dem Frontend zur Verfügung stellen muss. Danach wird beschrieben, wo

diese Funktionen in die bestehende Architektur von VizBoard eingegliedert werden. Zuletzt erfolgt eine genauere Beschreibung der Funktionen und deren Schnittstellen.

### 4.3.1 Funktionen

Um die gewünschten Funktionen, wie zum Beispiel Clustering, Datenverwaltung mit Filtern oder Relationen finden, für das Frontend zur Verfügung zu stellen, muss das Backend verschiedene Algorithmen implementieren (Abb. 4.22).

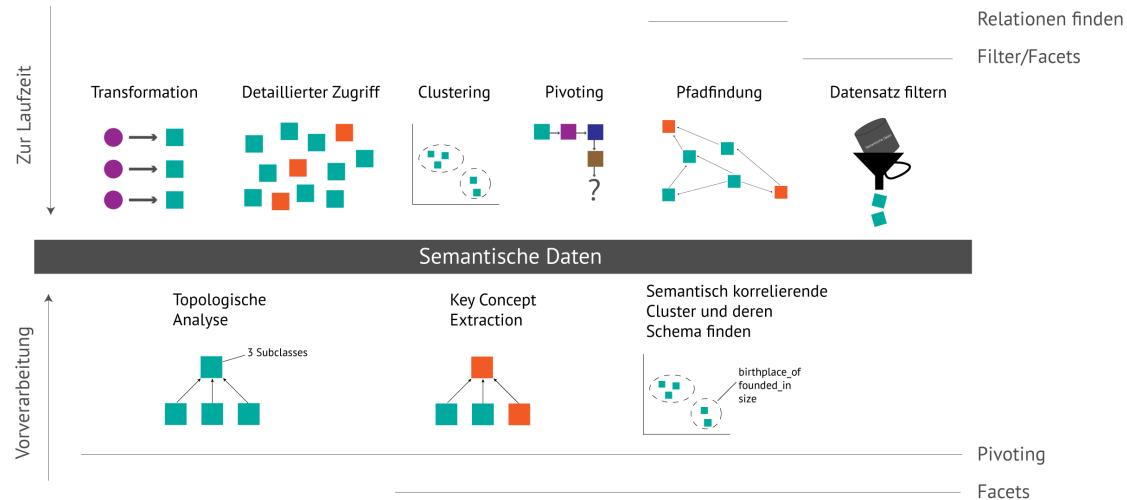


Abbildung 4.22: Die Aufgaben des Backends

### Zur Laufzeit

Das Backend muss die semantischen Daten aus dem Data Repository in ein für das Frontend leicht konsumierbares Format **transformieren**, zum Beispiel geeignet strukturiertes JSON. Außerdem muss es **detaillierten Zugriff** auf Teile des Datensatzes ermöglichen, weil das Frontend sehr selten den gesamten Datensatz benötigt, dafür aber zum Beispiel sehr oft die Superklassen von ausgewählten Ressourcen. Das Backend muss zur Laufzeit **clustern** können, wobei der Benutzer die Bedingungen (Datensatz, Ähnlichkeitsfunktion etc.) vorgibt. Zusätzlich muss das Backend die Vorschläge für interessante Ressourcen, die das Frontend dem Benutzer anbietet, berechnen (**Pivoting**). Das Frontend bietet dem Benutzer die Möglichkeit, Verbindungen zwischen Ressourcen zu finden. Aus diesem Grund muss das Backend beliebige **Pfade** zwischen gegebenen Ressourcen finden können. Der Benutzer kann den angezeigten Datensatz mit selbst definierten Filtern einschränken. Das Backend muss diese **Filter** lesen und auf den Daten ausführen können.

### Vorverarbeitung

Für die schnelle Berechnung des Pivoting, welches verschiedene Faktoren beinhaltet (Key Concepts, semantisch ähnliche Ressourcen, topologisch ähnliche Ressourcen, Zufall), muss das Backend verschiedene Informationen im Voraus sammeln. Es muss

eine **topologische Analyse** durchführen, die beispielsweise die Anzahl der Subklassen berechnet. Ein weiterer Einflussfaktor in das Pivoting sind die **Key Concepts** (Abschnitt 2.3.2), auch diese muss das Backend im Voraus bestimmen. Für einen weiteren Faktor des Pivoting, welcher auch benötigt wird, um die Facets für Instanzen zu finden, muss das Backend **semantisch korrelierende Cluster** und deren beschreibendes Schema finden.

### 4.3.2 Integration in das DaRe

Um entscheiden zu können, wie die im vorigen Abschnitt beschriebenen Funktionen umgesetzt werden, wird hier der Workflow von VizBoard[VPM12] und die Rolle des DaRe beschrieben.

VizBoard ist eine Visualisierungs-Workbench für semantische Daten. Der Workflow gestaltet sich wie folgt (Abb. 4.23): Zuerst lädt der Benutzer die Daten hoch (4.23-1). Das Data Repository annotiert diese, zum Beispiel wird die Scale of Measurement für Datatype Propertys angegeben: Quantitativ, nominal oder ordinal (4.23-2). Darauf folgt die in dieser Arbeit behandelte Komponente zur Vorauswahl, wo der Benutzer den Datensatz auf eine interessante Teilmenge einschränkt, die später visualisiert wird (4.23-3). Hier muss das Data Repository die Daten, wie im vorhergehenden Abschnitt beschrieben, clustern (4.23-4). Später wählt der Benutzer mögliche Visualisierungen aus (4.23-5) und wird dabei von VizBoard unterstützt, indem es Visualisierungen vorschlägt. Zum Beispiel bietet sich für Daten mit Geokoordinaten eine Karte an (4.23-6). Diese Visualisierungen müssen danach vom Benutzer konfiguriert (4.23-7) und von VizBoard in die CRUISe-Runtime integriert werden, sodass die Kommunikation untereinander möglich ist (4.23-8). Zuletzt kann der Benutzer mit Hilfe der gewählten Visualisierungen Informationen aus den Daten beziehen und in Wissen »umwandeln« (4.23-9). VizBoard versucht während des ganzen Prozesses, Wissen über den Benutzer zu sammeln (z. B. Präferenzen bei Visualisierungen) um es später wiederverwenden zu können (4.23-10).

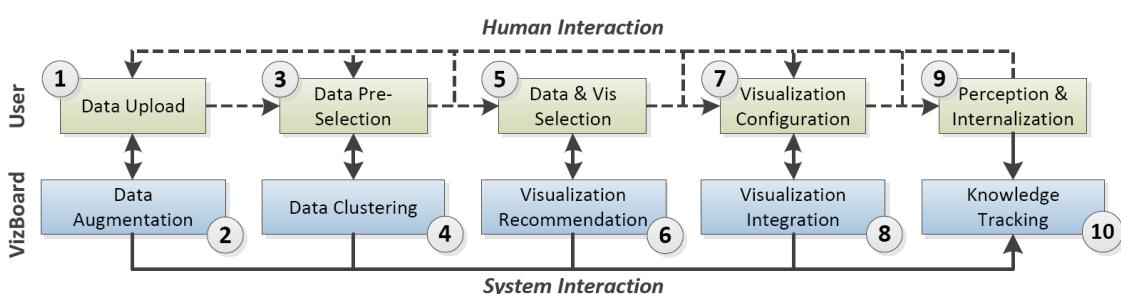


Abbildung 4.23: Der Workflow von VizBoard

Das Data Repository ist in den Schritten 2, 4 und 6 aktiv beteiligt, danach fragen die Visualisierungskomponenten nur noch Daten ab. Hierbei durchläuft es selbst mehrere Phasen (Abb. 4.24): Zunächst werden die gewünschten Daten geladen, entweder lädt der Benutzer sie selbst hoch oder gibt eine URL an, von der sie dann geladen werden (4.24-Upload). Das Data Repository unterstützt nicht nur semantische Daten, sondern auch andere strukturierte Datenformate wie etwa Excel oder

eine relationale Datenbank. Falls die Daten in einem solchen Format vorhanden sind, transformiert sie das Data Repository in RDF ([4.24-Transformation](#)). Danach analysiert es die Daten und annotiert sie, unter anderem wird die Anzahl der Instanzen einer Klasse angegeben ([4.24-Analyse](#)). Der Benutzer kann selbst noch Regeln erstellen, auf Basis derer die Daten annotiert werden sollen. Diesen Schritt setzt das Data Repository nach der Transformation um ([4.24-Annotation](#)). Nach diesem Schritt ist das aktuelle Data Repository mit der Datenaufbereitung fertig und die Visualisierungskomponenten können über eine REST API auf die Daten zugreifen ([4.24-RESTful API](#)).

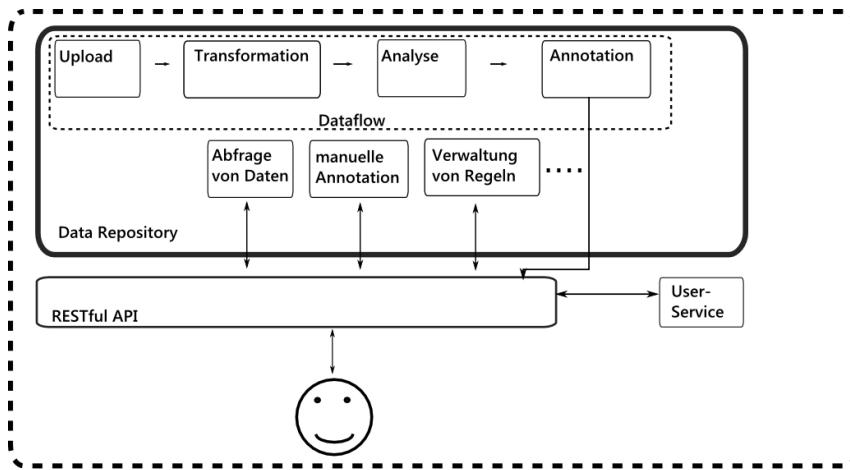


Abbildung 4.24: Die Struktur des Data Repository

Konzeptionell passen das Clustering zur Auffindung von semantisch korrelierenden Daten, die Key Concept Extraction und die topologische Analyse am Besten in die Analysephase des Data Repositorys (Abb. [4.25](#), oben). Es ist ausreichend, dass diese Algorithmen einmal pro Datensatz ausgeführt werden, da sich die gefundenen Metadaten nicht ändern. Wie die anderen Ergebnisse der Analysephase können auch diese Metadaten an die Daten annotiert werden. Beispielsweise ist es möglich, an eine Datatype Property die Cluster zu annotieren, zu dessen Schema sie gehört. Oder an Klassen, ob sie zu den Key Concepts gehören. An Instanzen kann ähnlich den Datatype Propertys annotiert werden, zu welchem Cluster sie gehören.

Das Frontend muss die anderen Funktionen (Pfadfindung, Clustering, Filtern, Pivoting, detaillierter Zugriff) zwecks Parameterübergabe ansprechen können. Deswegen stellt das Backend die Kommunikationsmöglichkeit mit den internen Funktionen am Besten über die RESTful API zur Verfügung (Abb. [4.25](#), links).

### 4.3.3 Schnittstellen

Im Folgenden werden die benötigten Schnittstellen der zuvor beschriebenen Funktionen (Abschnitt [4.3.1](#)) informell erklärt.

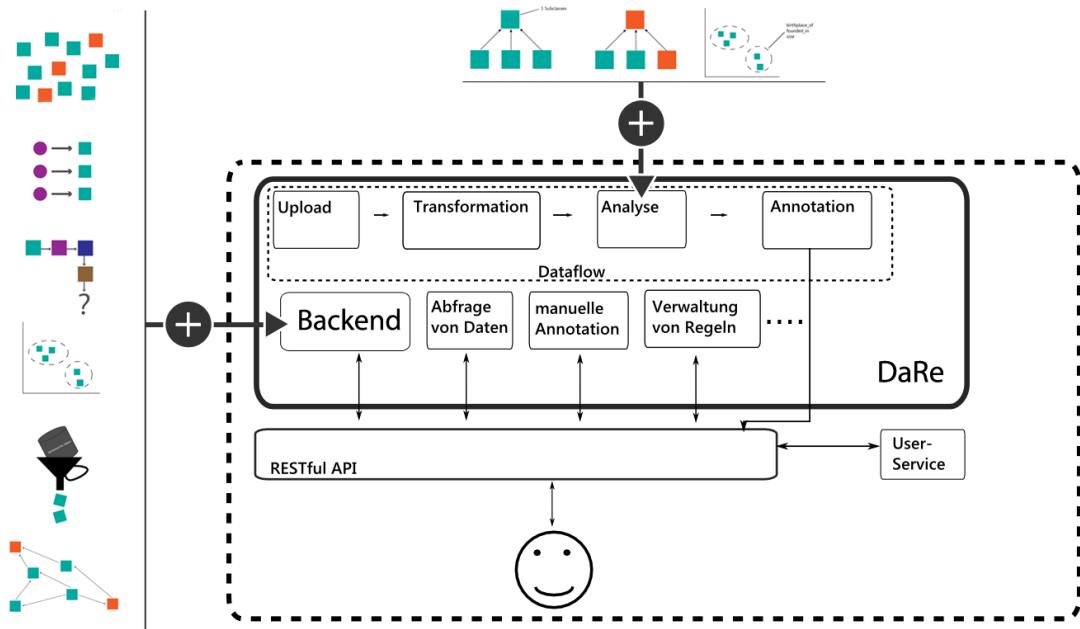


Abbildung 4.25: Integration des Backends in das DaRe

## Zur Laufzeit

Das Backend muss die semantischen Daten in ein für das Frontend leicht zu verarbeitendes Format **transformieren**. Hier ist es nötig, dass die Transformation ein Jena Model in JSON mit zwei Attributen *nodes* und *links* umwandelt. So wird begünstigt, dass nur Daten gesendet werden, die das Frontend auch benötigt. Zum Beispiel ist es für die Anzeige des Graphen völlig ausreichend, die URI und das Label der Ressourcen zu kennen. Die überschüssigen Informationen können während der Transformation entfernt werden. Diese kann der Benutzer über den Detail-Dialog bei Bedarf aufrufen (Abschnitt 4.2.3). Durch diese Vorgehensweise werden Rechenaufwand im Frontend und Netzwerkverkehr reduziert.

Das Frontend benötigt nicht zu jedem Zeitpunkt den vollständigen Datensatz, weshalb das Backend in der Lage sein muss, eine Teilmenge davon zur Verfügung zu stellen (**detaillierter Zugriff**). Beispielsweise benötigt das Frontend für die Hierarchieansicht (Abschnitt 4.2.4) nur die Superklassen der vom Benutzer ausgewählten Ressourcen. Außerdem gehören dazu die detaillierten Informationen zu einer Ressource, zum Beispiel Kommentar, Name, Beschreibung, äquivalente Klassen, ob eine Object Property transitiv ist etc. Da diese Zugriffe aber sehr oft getätigt werden (Hierarchieansicht: Bei jeder Selektion), wird eine Schnittstelle zur Verfügung gestellt. Die Eingabe dieser Funktion besteht aus einer Menge von Ressourcen (bzw. deren URIs) und der Angabe, ob die Hierarchie oder Detailinformationen geladen werden sollen. Die Ausgabe ist dementsprechend ein Jena Model bzw. eine Menge an Key-Value Paaren.

Der Benutzer hat die Möglichkeit, die angezeigten Daten zur Laufzeit zu **clustern**. Dabei hat er verschiedene Optionen (vgl. Abschnitt 4.2.2): Er kann ohne weitere

Einstellungen clustern oder eine Property angeben, auf Basis derer die Ähnlichkeit zwischen Ressourcen berechnet werden soll. Es ist ihm zusätzlich möglich, eine Referenzressource anzugeben, die das Maß der Dinge für die Ähnlichkeitsberechnung ist. Die Eingabe für diese Funktion besteht also aus einem Tupel (*Property, Referenz*), welches vollständig oder teilweise leer sein kann. Das Clustering, welches ausgegeben wird, ist ein Jena Model. Für das Clustering sind Algorithmen nötig, die eine Menge von Objekten und eine Ähnlichkeitsfunktion als Eingabe annehmen und ein Clustering ausgeben. Ein bekannter Vertreter davon ist zum Beispiel K-means (Abschnitt 2.3.1). RapidMiner mit der Erweiterung RMonto stellen solche Algorithmen zur Verfügung.

Das Frontend bietet dem Benutzer Vorschläge für interessante Ressourcen an (**Pivoting**). Diese basieren auf der aktuell ausgewählten Ressource. Die Faktoren, die in diese Vorschläge einfließen, sind topologische Ähnlichkeit, semantische Ähnlichkeit, Key Concepts und Zufall. Dementsprechend ist die Eingabe in diese Funktion eine Ressource bzw. deren URI und die Ausgabe eine Liste von Ressourcen, die dem Benutzer vorgeschlagen werden.

Das Frontend stellt ein Feature zur Auffindung von Verbindungen zwischen Objekten, also **Pfaden** im RDF Graphen, zur Verfügung. Das Backend muss dementsprechend eine Menge von Ressourcen (URIs) entgegennehmen und sie zusammen mit ihren Verbindungen als Jena Model ausgeben. Dazu wird ein Algorithmus benötigt, der zu  $n$  Ressourcen alle vorhandenen Pfade im Datensatz ausgibt. Bei der Implementierung muss darauf geachtet werden, dass dieser Vorgang nicht zu teuer (Rechenzeit) wird. Um dies sicherzustellen, können maximale Werte für  $n$  und die Länge der Pfade festgelegt werden. Um diese Funktion umzusetzen können beispielsweise SPARQL Querys konstruiert werden, die einen Pfad der Länge  $k$  ausgeben, welche danach für mehrere  $k$  ausgeführt werden.

Der Benutzer kann den angezeigten Datensatz zur Laufzeit **filtern** (vgl. Abschnitt 4.2.5). Der Assistent zum Erstellen solcher Filter nimmt dabei

1. den Bestandteil der Ontologie (Klasse, Instanz oder Property)
2. eine Property oder einen Datentyp, den eine Property haben soll
3. einen Wert

entgegen. Das entspricht der Eingabe der Filter-Funktion, die diese Informationen in eine SPARQL Query umwandelt und das resultierende Model zurückgibt.

## Vorverarbeitung

Zur Identifikation der Facets von Klassen und der Vorschläge für Pivoting müssen dem Backend **topologische Metriken** wie zum Beispiel Anzahl Subklassen, Closeness oder dergleichen bekannt sein. Dabei wird für die Berechnung ein Jena Model entgegengenommen, in dem die Ergebnisse an jede Ressource annotiert werden.

Für die Berechnung der Vorschläge im Pivoting wird auch **Key Concept Extraction** (Abschnitt 2.3.2) verwendet. Dabei werden die  $n$  wichtigsten Ressourcen einer Ontologie bestimmt. Die Eingabe in diese Funktion ist ein Jena Model, die Ausgabe wird an die Klassen im Model in der Form *isKeyConcept=true/false* annotiert. Das Backend kann diese mit Hilfe einer Java API<sup>3</sup> durchführen.

Für die Identifikation der Facets von Instanzen und der Vorschläge für Pivoting muss das Backend **semantisch korrelierende Cluster** und deren wichtigste Prädikate finden. Das ist nötig, weil eine Datatype Property in semantisch unterschiedlichen Kontexten verwendet werden kann. Dieser Algorithmus nimmt ein Jena Model entgegen und annotiert die Ergebnisse an die Ressourcen (für die Instanzen die Zugehörigkeit zu Cluster, für Propertys Teil welcher Clusterschema sie sind). Die Menge an gefundenen Clustern wird am Datensatz selbst annotiert. Zur Durchführung kann das Open Source Tool RapidMiner<sup>4</sup> mit der Erweiterung für RDF-Daten RMonto [PL11] verwendet werden.

---

<sup>3</sup><http://sourceforge.net/projects/kce/>

<sup>4</sup><http://sourceforge.net/projects/rapidminer/>

# 5 Implementation

Auf dem Wissen aus den Grundlagen (Kapitel 2), dem Feedback aus der Nutzerstudie (Kapitel 3) und dem daraus entwickelten Konzept für eine Datenauswahlkomponente (Kapitel 4) aufbauend wurde ein Prototyp entwickelt. Er besteht aus einem Frontend und einem Backend, deren Umsetzung in diesem Kapitel diskutiert wird. Der Prototyp wurde in einer kleinen Nutzerstudie evaluiert, deren Ergebnisse zum Schluss vorgestellt werden.

## 5.1 Frontend

In den folgenden Abschnitten wird auf die Umsetzung des Frontends (Abb. 5.1) eingegangen. Zuerst wird die Architektur betrachtet und die wichtigsten Design Patterns in einem Diagramm anschaulich gemacht. Danach wird auf die verwendeten Frameworks und Bibliotheken eingegangen, welche für die Umsetzung benutzt wurden.



Abbildung 5.1: Ein Screenshot des Frontends

### 5.1.1 Architektur

Für die Architektur des Frontends wurde im Prinzip auf drei Design Patterns zurückgegriffen (Abb. 5.2).

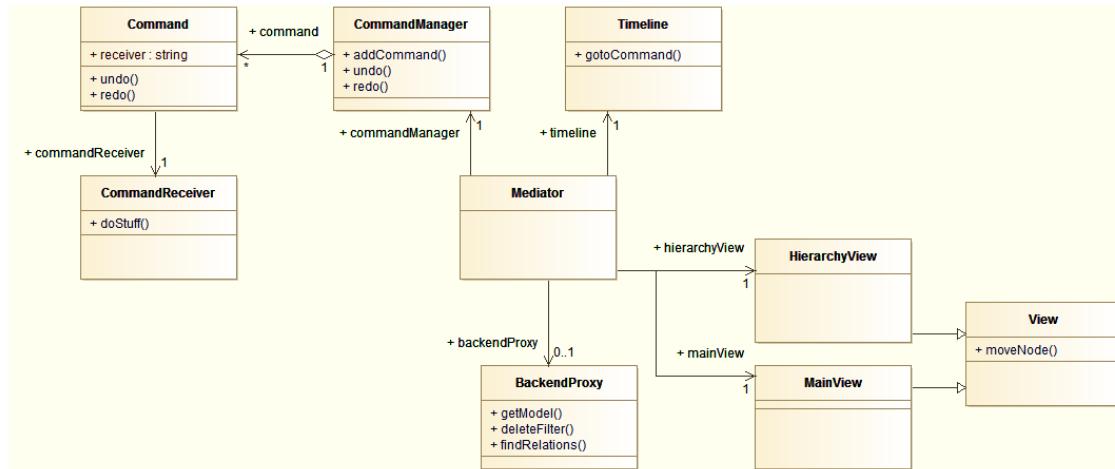


Abbildung 5.2: Die wichtigsten Klassen des Frontends

Ein MEDIATOR [Gam+95] verbindet sämtliche Klassen und steuert die Kommunikation zwischen ihnen. Verschiebt der Benutzer z. B. einen Knoten, feuert der View ein Event mit einem MoveCommand als Parameter. Auf dieses Event reagiert der Mediator, in dem er das MoveCommand an den CommandManager und die Timeline weitergibt (Abb. 5.3).

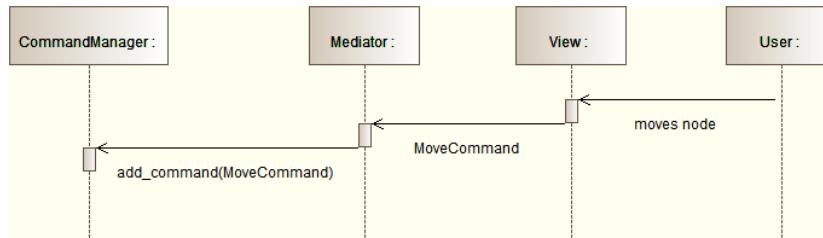


Abbildung 5.3: Der Programmablauf, wenn der Benutzer einen Knoten verschiebt

Um reibungsloses Undo/Redo zu ermöglichen, wurde das COMMAND [Gam+95] Pattern verwendet. Der Receiver des Commands kann entweder eine Methode einer beliebigen Klasse sein (z. B. `moveNode()` im View) oder ein deziidierter CommandReceiver. So wird die Beschreibung eines Commands von der Implementierung getrennt und Flexibilität sowie Codewiederverwendung erhöht. Ein CommandReceiver kann so mit jedem beliebigen Command verwendet werden. Die ausgeführten Commands werden von einem CommandManager verwaltet, der die Schnittstelle für die Undo/Redo Funktion ist.

Ein PROXY [Gam+95] kapselt die Kommunikation mit dem Backend. So wird die Quelle der Daten abstrahiert und könnte einfach ausgetauscht werden.

### 5.1.2 Frameworks & Bibliotheken

Für das Frontend wurden verschiedene JavaScript Frameworks und Bibliotheken eingesetzt. Die visuellen Elemente wie Views oder Timeline sind SVG Elemente, die mit Hilfe von D3<sup>1</sup> [BOH11] verwaltet werden. Mögliche Alternativen dazu wären RaphaelJS<sup>2</sup> und Protopis<sup>3</sup> [BH09]. Protopis wird zugunsten von D3 nicht mehr weiterentwickelt und RaphaelJS ist weniger verbreitet als D3<sup>4</sup>, weswegen die Wahl auf D3 fiel.

Die Fenster, Listen und dergleichen sind ExtJS<sup>5</sup> 3.4 Komponenten. Da die anderen Komponenten in VizBoard ebenfalls ExtJS 3.x verwenden, war dies mehr oder weniger Vorgabe, um die Kompatibilität zu gewährleisten.

Für kurzes Feedback an den Benutzer (wie z. B. »Filter erfolgreich hinzugefügt«) wird humane.js<sup>6</sup> benutzt, da die Web Notification API [W3C11] noch ein Working Draft ist. Um mit Datumsangaben einfacher umgehen zu können, benutzt das Frontend außerdem date.js<sup>7</sup>.

## 5.2 Backend

In den folgenden Abschnitten wird auf die Implementierung des Backends eingegangen. In Abschnitt 4.3.2 wurde festgestellt, dass die benötigten Funktionen des Backends in eine Analyse- bzw. Vorverarbeitungsphase und eine Berechnungsphase zur Laufzeit aufgeteilt werden sollten (Abb. 5.4).

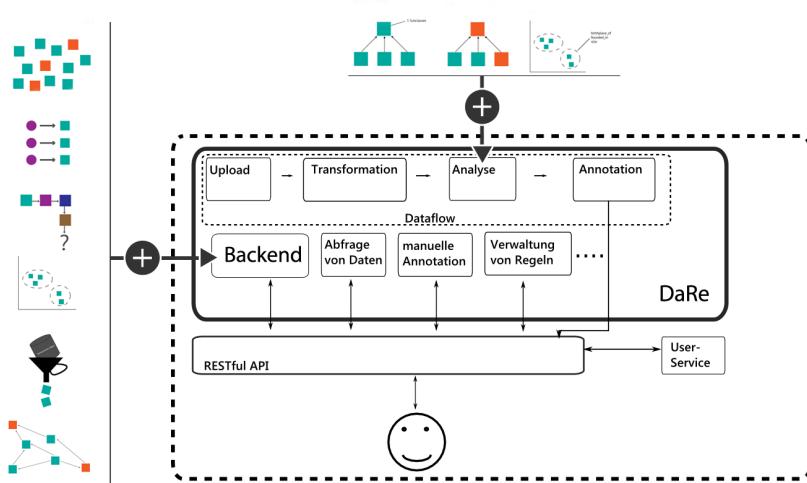


Abbildung 5.4: Integration des Backends in das Data Repository

<sup>1</sup><https://github.com/mbostock/d3>

<sup>2</sup><http://raphaeljs.com/>

<sup>3</sup><http://mbostock.github.com/protovis/>

<sup>4</sup>327 Forks und 3646 Watcher bei github gegenüber 635/5648 von D3 am 10. Mai 2012.

<sup>5</sup><http://www.sencha.com/products/extjs3>

<sup>6</sup><http://wavedevel.github.com/humane-js/>

<sup>7</sup><http://www.datejs.com/>

Im Folgenden wird die Architektur der beiden Phasen diskutiert und danach die verwendeten Frameworks und Bibliotheken erläutert, bevor die Analyse- und Annotationsphase detailliert beschrieben werden.

### 5.2.1 Architektur

Das Backend durchläuft drei Phasen (Abb. 5.5): Zuerst werden die Daten analysiert (Analysephase). Dazu zählen Clusteranalyse, Berechnung einer Ähnlichkeitsmatrix, Finden der Key Concepts, Bestimmen der Clusterschemas etc. Danach annotiert das Backend die gefundenen Metadaten an den Datensatz (Annotationsphase). Wenn der Benutzer auf die Daten zugreifen will, müssen diese zuerst wieder extrahiert werden (Extraktionsphase), um die nötigen SPARQL Querys zur Laufzeit zu verringern.

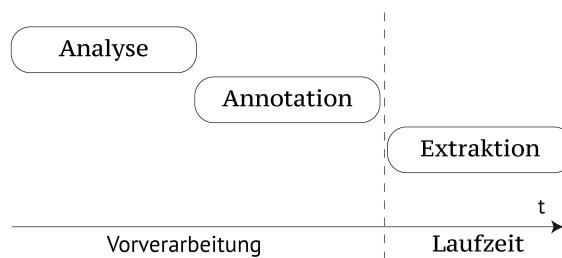


Abbildung 5.5: Die zwei Vorverarbeitungsphasen des Backends

Die REST API des Backends stellt passive Methoden, die nur Daten abrufen, und aktive Methoden für die Filter zur Verfügung (Abb. 5.6) bereit. Methoden wie `/model`, `/hierarchy` und `/relate` rufen die JSON Repräsentation eines Jena Models ab. `/facets` und `/filter` geben die JSON Repräsentationen der aktiven Filter bzw. Facets aus. `/pivot` zeigt die Vorschläge für eine gegebene Ressource als JSON Array von URIs an und `/details` gibt die Detailinformationen zu einer Ressource aus. `/propertys` wird für den Filterdialog verwendet, der die im aktuellen Model verfügbaren Property's kennen muss. `textit/cluster` ordnet alle vorhandenen Ressourcen einem Cluster zu, indem deren URIs in entsprechende JSON Arrays geschrieben werden. `/filter/add` fügt einen übergebenen Filter hinzu, `/filter/addresource` erstellt mehrere Filter auf Basis einer Ressource. Mit `/toggle` kann ein Filter an- oder ausgeschaltet werden, wohingegen er mit `/edit` verändert und mit `/delete` entfernt wird.

<code>/model</code>	<code>/filter</code>
<code>/hierarchy</code>	<code>/add</code>
<code>/pivot</code>	<code>/addresource</code>
<code>/relate</code>	<code>/toggle</code>
<code>/cluster</code>	<code>/edit</code>
<code>/propertys</code>	<code>/delete</code>
<code>/facets</code>	
<code>/details</code>	

aktive Methoden  
passive Methoden

Abbildung 5.6: Die REST API des Data Repository

## 5.2.2 Frameworks & Bibliotheken

Das Backend wurde mit Hilfe von Jena<sup>8</sup> (Analyse, Annotation) und Jersey<sup>9</sup> (REST) umgesetzt. Diese waren vorgegeben, da das bestehende DaRe sie benutzt. Apache Jena ist ein Java Framework für semantische Daten. Es stellt eine API für wichtige Operationen zur Verfügung, wie zum Beispiel Querys erstellen, Tripel schreiben etc. Jersey ist eine Java API, mit der REST Webservices erstellt werden können.

Das Clustering der Ressourcen und die Ähnlichkeitsberechnung wurde mit Hilfe von RapidMiner<sup>10</sup> 5.2 und RMonto<sup>11</sup> 1.0 [PL11] durchgeführt. RapidMiner ist ein verbreitetes Open Source Clustering/Analysis Tool und RMonto erweitert es um die Möglichkeit, semantische Daten clustern zu können. Eine Alternative wäre das Plugin rapidminer-semweb<sup>12</sup> [KGD10], welches ebenfalls eine Ähnlichkeitsmatrix berechnen kann. Die Wahl fiel auf RMonto, weil dieses zusätzlich noch ein Clustering durchführen kann. Für Graphalgorithmen benutzt das Backend die Java Bibliothek JGraphT<sup>13</sup> 0.8.3, da es sich im Gegensatz zu dem verbreiteten Framework JUNG<sup>14</sup> speziell auf die Algorithmen konzentriert und Aspekte wie Visualisierung nicht betrachtet. Um die Key Concepts (Abschnitt 2.3.2) zu bestimmen, wird die KCE API<sup>15</sup> verwendet, welche direkt von einem der Autoren des Papers stammt. Da das Backend über JSON mit dem Frontend kommuniziert, wurde die Bibliothek json-lib<sup>16</sup> 2.4 eingebunden, welche z. B. die Konvertierung von/nach Arrays und String Escaping vereinfacht.

## 5.2.3 Analysephase

Wird ein neuer Datensatz in das Backend hinzugefügt, startet es nach einer eventuellen Transformationsphase als erstes die Analysephase. Als Eingabe dienen der Datensatz als Jena Model und der Pfad zu der dazugehörigen Datei. Das Backend berechnet zuerst die Ähnlichkeiten der Ressourcen zueinander. Diese dienen als Basis für den nächsten Schritt, das Clustering der Ressourcen. Danach werden Graphalgorithmen zur Berechnung der Betweenness und Subklassen je Klasse ausgeführt. Das Backend berechnet daraufhin das Schema jeden Clusters und die Key Concepts des Datensatzes. Zuletzt werden Informationen gesammelt, die für die Anzeige der Facets benötigt werden. Im Folgenden werden die einzelnen Schritte näher diskutiert.

### Berechnung der Ähnlichkeitsmatrix

Für Ähnlichkeitsberechnung und Clustering kommt RapidMiner 5.2 mit der Erweiterung RMonto 1.0 zum Einsatz. In RapidMiner gibt es das Konzept eines Prozesses

<sup>8</sup><http://incubator.apache.org/jena/>

<sup>9</sup><http://jersey.java.net/>

<sup>10</sup><http://rapid-i.com/content/view/181/190/lang,en/>

<sup>11</sup><http://semantic.cs.put.poznan.pl/RMonto>

<sup>12</sup><http://code.google.com/p/rapidminer-semweb/>

<sup>13</sup><http://jgrapht.org/>

<sup>14</sup><http://jung.sourceforge.net/>

<sup>15</sup><http://sourceforge.net/projects/kce/>

<sup>16</sup><http://json-lib.sourceforge.net/>

mit In- und Outputports, der wiederum Operatoren mit In- und Outputports enthält. RMonto stellt verschiedene Operatoren für den Umgang mit semantischen Daten zur Verfügung, zum Beispiel die Berechnung der Ähnlichkeiten (»Calculate gram/distance matrix«) oder »K-medoids Clustering«.

Diese werden zu einem Prozess verkettet (Abb. 5.7): Zuerst lädt der Operator »Load file« den Datensatz aus der Datei, das Ergebnis wird in »Build knowledge base« eingegeben. Dieser Operator ist die Grundlage für »SPARQL selector«, wo die zu clusternden Elemente aus der Knowledge Base geholt werden und »Calculate gram/distance matrix«, wo die eigentliche Ähnlichkeitsberechnung stattfindet. Die Definition von Ähnlichkeit enthält der Common Classes »Bloehdorn kernel« [BS07], konkret wird die Menge an gemeinsamen Superklassen herangezogen. Die berechnete Matrix wird dann als Ergebnis des Prozesses an einen Outputport desselben weitergegeben.

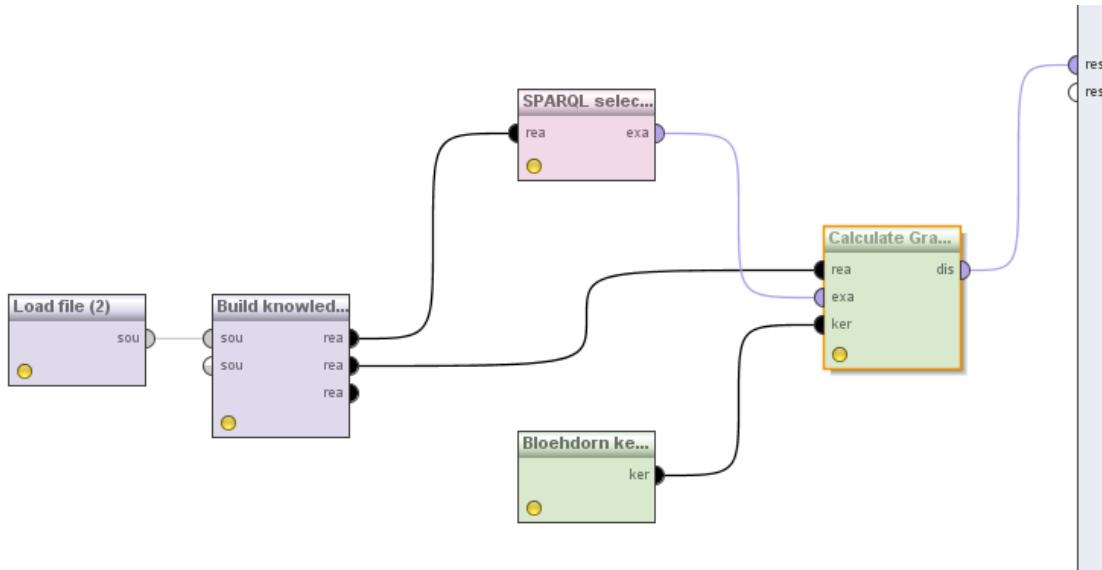


Abbildung 5.7: Verkettung der RMonto-Operatoren in RapidMiner zu einem Prozess

Das Ergebnis der Ähnlichkeitsberechnung ist eine Matrix, welche die Unähnlichkeit zwischen den Ressourcen enthält (Tab. 5.1). Die Werte liegen zwischen 0 (Identität) und 1 (etwas ganz anderes) und können einfach invertiert werden, um die Ähnlichkeit zu erhalten.

	A	B	C	D
A	0,0	0,3	0,2	0,1
B	0,3	0,0	0,5	0,9
C	0,2	0,5	0,0	0,7
D	0,1	0,9	0,7	0,0

Tabelle 5.1: Beispiel einer Unähnlichkeitsmatrix von vier Ressourcen

Eigentlich sollte für die Bestimmung der semantisch korrelierenden Cluster die Ähnlichkeit zweier Ressourcen die Anzahl der gemeinsamen Prädikate (nicht Superklassen) zur Grundlage haben. Um das umzusetzen, müsste ein entsprechender Kerneloperator in die Bibliothek SeSiL eingefügt werden, welche die Implementation der von RMonto verwendeten Algorithmen enthält. Des Weiteren müsste RMonto erweitert werden, sodass dieser Kerneloperator auch ausgewählt werden kann. Da der Quelltext von SeSiL und RMonto nicht öffentlich verfügbar ist und diese Erweiterung nicht Fokus dieser Arbeit ist, wurde sie nicht umgesetzt.

## Clustering

Im Prinzip verläuft das Clustering (vgl. Abschnitt 2.3) gleich wie die zuvor beschriebene Ähnlichkeitsberechnung. Der einzige Unterschied ist ein Operator »Semantic K-medoids« statt »Calculate gram/distance matrix«, der aber die gleichen Inputports hat. Dieser Operator gibt das Clusteringmodell (Cluster ID und Repräsentant des Clusters, Tabelle 5.2) und das Clustering selbst (Cluster ID und Ressource, Tabelle 5.3) aus.

Cluster ID	Medoid
0	<a href="http://www.ressource.com/A">http://www.ressource.com/A</a>
1	<a href="http://www.ressource.com/B">http://www.ressource.com/B</a>

Tabelle 5.2: Beispiel eines Clusteringmodells

Cluster ID	URI
0	<a href="http://www.ressource.com/A">http://www.ressource.com/A</a>
1	<a href="http://www.ressource.com/B">http://www.ressource.com/B</a>
0	<a href="http://www.ressource.com/F">http://www.ressource.com/F</a>
0	<a href="http://www.ressource.com/E">http://www.ressource.com/E</a>
1	<a href="http://www.ressource.com/C">http://www.ressource.com/C</a>

Tabelle 5.3: Beispiel eines Clusterings

Intern berechnet der Clusteringoperator »Semantic K-medoids« auch die Unähnlichkeitsmatrix, dasselbe wie der Operator »Calculate gram/distance matrix« im vorigen Abschnitt (5.2.3). Da das Backend sowohl das Clustering als auch die Ähnlichkeitsmatrix benötigt, wäre es sinnvoll, zuerst die Matrix zu berechnen und auf deren Basis das Clustering auszuführen. Leider stellt der Clusteringoperator keinen Port zur Ein- oder Ausgabe der Ähnlichkeitsmatrix zur Verfügung. Deswegen wird sie in der Vorverarbeitung unnötigerweise zweimal berechnet. Die Erweiterung des Clusteringoperators in diese Richtung sollte als zukünftige Verbesserung des Prototypen in Betracht gezogen werden, da es im Rahmen dieser Arbeit nicht umgesetzt werden konnte.

## Topologische Metriken

Nachdem die Berechnungen für das Clustering abgeschlossen sind, wird das Jena Model des Datensatzes mit JGraphT in einen gerichteten Graphen überführt, um Graphalgorithmen wie z. B. Tiefensuche benutzen und topologische Metriken berechnen zu können. Für jede Klasse werden die Anzahl an Subklassen (direkt und gesamt) und die Betweenness nach Brandes [Bra01] berechnet. Dies entspricht der Anzahl an kürzesten Pfaden zwischen beliebigen Knoten, in denen die Klasse enthalten ist.

Zuerst wird die enthaltene Klassenhierarchie mit einer SPARQL Query extrahiert. Das sind alle Tripel der Form (Subjekt, rdf:type, rdfs:Class) bzw. (Subjekt, rdf:type, owl:Class), welche die Klassen darstellen, und (Subjekt, rdfs:subClassOf, Objekt), welche die Hierarchie der Klassen enthalten. Für jedes Statement im resultierenden Model wird eine beschriftete Kante (Quelle, Beschreibung, Senke) im gerichteten Graphen hinzugefügt. Danach können die Nachbarn jedes Knotens einfach gefunden werden, diese entsprechen den direkten Subklassen. Mit einer Tiefensuche werden weitere Subklassen gefunden. Dabei werden zuerst alle Kanten eines Knoten betrachtet, bevor der Algorithmus den Nächsten besucht. Im Gegensatz dazu steht die Tiefensuche, wo zuerst der nächste Knoten gesucht wird (Abb. 5.8). Die Tiefensuche wurde gewählt, weil sie im Gegensatz zur Breitensuche eine geringere Speicherkomplexität aufweist. Ist dies abgeschlossen, berechnet das Backend für jede Klasse die Betweenness im Graphen. Dafür wird der Algorithmus nach Brandes benutzt.

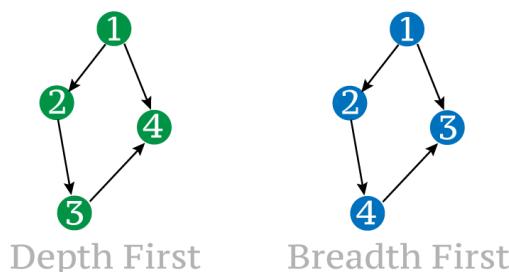


Abbildung 5.8: Tiefensuche vs. Breitensuche

## Clusterschema

Für die Cluster sucht das Backend nun nach Prädikaten, die diese gut beschreiben. Dazu geht es alle Prädikate der Subjekte in den Clustern durch und hält für jedes davon einen Zähler (siehe Pseudocode in Listing 5.1). Danach wird dieser Zähler durch die Größe des Clusters dividiert. Das Ergebnis ist die Relevanz für den Cluster. Ist diese  $\geq 80\%$ , wird das Prädikat in das Schema des Clusters übernommen.

Listing 5.1: Algorithmus zum Auffinden der Clusterschemata

```

// die Zähler für die Prädikate
Map<Resource, int> pred_counters;
// die Cluster
List<Cluster> clusters = getCluster();

for each (Cluster cluster in cluster)
    // Anzahl der Elemente im Cluster
    int cluster_size = cluster.size();

```

```

for each (Resource subject in cluster)
    //Verschiedene Prädikate des Elements holen
    List<Resource> predicates = model.query("SELECT DISTINCT ?p WHERE "+ 
        "?s ?p ?o");
    
    //Überall den Zähler um 1 erhöhen
    for each (Resource predicate in predicates) {
        pred_counters[predicate] = pred_counters[predicate] + 1;
    }

    //Die Prädikate noch einmal durchlaufen
    for each (Resource predicate in pred_counters)
        //Relevanz für Cluster berechnen
        double relevance = pred_counters[predicate] / cluster_size;
        //Prädikat ist relevant, wenn es ein Statement (clusteritem, Prädikat, o)
        //für mindestens 80 % der Elemente im Cluster gibt
        if (relevance >= 0.8)
            //Übernehmen und annotieren

```

Der Threshold von 80 % wurde ohne weitere Untersuchungen gewählt. Natürlich ist es theoretisch möglich, dass kein gemeinsames Schema für einen Cluster gefunden werden kann. In diesem Fall werden keine Facets für die betroffenen Instanzen angezeigt. Möglichkeiten, um den Threshold abschätzen zu können, bleiben für die weitere Verbesserung des Prototypen. Denkbar wäre zum Beispiel, den Parameter abhängig von der Güte des Clustering zu machen. Dafür gibt es verschiedene Verfahren [BA03], zum Beispiel die Cluster Silhouette. So wäre es möglich, die Bedingung für das gemeinsame Schema zu lockern, wenn die Ressourcen im Datensatz sich generell eher nicht ähnlich sind.

## Key Concept Extraction

Zur Bestimmung der Key Concepts, der  $n$  wichtigsten Konzepte im Datensatz (siehe Abschnitt 2.3.2), wird die KCE API benutzt. Sie nimmt einen Datensatz in RDF/XML als String entgegen und gibt ein Java Set von Strings (den URIs der Konzepte) aus.

## Facets

Um zeitintensive Querys zur Laufzeit zu vermeiden, extrahiert das Backend benötigte Informationen zur Darstellung der Facets aus dem Datensatz (vgl. 2.1.3). Das umfasst Minimum/Maximum Werte für ordinale oder quantitative Prädikate (z. B. Datum, Integer, Double) und sämtliche verschiedene Werte für nominale Prädikate (Strings). Diese Werte werden mit SPARQL Querys bestimmt.

### 5.2.4 Annotationsphase

In der Annotationsphase annotiert das Backend die Ergebnisse aus der vorangegangenen Analysephase an den Datensatz, damit sie zur Laufzeit nur noch extrahiert werden müssen. Dies spart Zeit, wenn der Benutzer involviert ist.

## Ähnlichkeitsmatrix

Die Ähnlichkeit zwischen zwei Ressourcen aus Abschnitt 5.2.3 kann als Quadrupel (Ressource A, ist ähnlich zu, Ressource B, Ähnlichkeit) aufgefasst werden. Da RDF aus Tripeln besteht, wird eine Klasse *Similarity* und Property's *hasSimilaritySubject*, *hasSimilarityObject*, *hasSimilarityValue* eingeführt. Im Datensatz sieht das für zwei Ressourcen A und B, deren Ähnlichkeit 0.25 beträgt, wie folgt aus:

### Listing 5.2: Annotation der Ähnlichkeiten

```

1 <SimilarityStatement_AB> <hasSimilaritySubject> <http://www.ressourceA.com>
2 <SimilarityStatement_AB> <hasSimilarityObject> <http://www.ressourceB.com>
3 <SimilarityStatement_AB> <hasSimilarityValue> 0.25
4 <SimilarityStatement_AB> rdf:type <Similarity>
```

Da für jedes unterschiedliche Paar an Ressourcen (Identität nicht ausgeschlossen) vier Statements hinzugefügt werden, erhöht sich die Anzahl der Statements im Datensatz um  $2n^2$ , wo  $n$  die Anzahl an Instanzen im Datensatz ist.

## Cluster

Um zur Laufzeit schnell bestimmen zu können, zu welchen Clustern die Instanzen gehören (Abschnitt 5.2.3), die der Benutzer gerade sieht, werden auch die Elemente der Cluster annotiert. Das passiert durch einen einfachen Tripel (Listing 5.3).

### Listing 5.3: Annotation der Clusterzugehörigkeit

```

1 <I> <isPartOfCluster> 1
2 <I> <isPartOfCluster> 2
```

## Topologische Metriken

Topologische Metriken (Abschnitt 5.2.3) spielen für die Facets von Klassen eine Rolle. Das Backend fügt für eine Klasse C drei Tripel in den Datensatz ein (Listing 5.4):

### Listing 5.4: Annotation der topologischen Metriken

```

1 <C> <hasTotalSubclasses> 24
2 <C> <hasDirectSubclasses> 6
3 <C> <hasBetweenness> 5
```

## Clusterschema

Die Clusterschema (Abschnitt 5.2.3) sind für die Facets von Instanzen von Bedeutung. Die Zugehörigkeit zu einem Clusterschema wird mit einem einzelnen Tripel annotiert (Listing 5.5 Zeile 1-2). Die Werte der Relevanz für jedes Schema eines Clusters werden in einen String umgewandelt und annotiert (Listing 5.1 Zeile 3).

### Listing 5.5: Annotation der Clusterschema

```

1 <P> <isSchemaOfCluster> 1
2 <P> <isSchemaOfCluster> 3
3 <P> <relevanceForClusterSchema> 0.0,0.9,0.0,0.85
```

## Key Concepts

Die Key Concepts, welche in Abschnitt 5.2.3 berechnet wurden, werden mit einem Prädikat *isKeyConcept* und dem Wert *true* annotiert, alle anderen mit *false*. Das erleichtert die Querys zur Laufzeit, da nicht mehr auf das Vorhandensein eines Prädikats geprüft werden muss. Sei für Listing 5.6 KC ein Key Concept und NKC ein normales Konzept.

### Listing 5.6: Annotation der Key Concepts

```

1 <KC> <isKeyConcept> true
2 <NKC> <isKeyConcept> false
```

Die Key Concepts eines Datensatzes sind in der Regel relativ wenige Klassen, da sie nur die Wichtigsten repräsentieren sollen. Deswegen wäre es nur um (Nicht-) Key Concepts zu finden ausreichend, (Concept, isKeyConcept, true) zu annotieren. Normale Klassen könnten zum Beispiel mit einer FILTER NOT EXISTS Query extrahiert werden.

Jedoch wird die Property isKeyConcept im Frontend als nominales Facet für Klassen angezeigt. Die oben beschriebene Vorgehensweise stellt sicher, dass isKeyConcept wie alle anderen Properties behandelt werden kann: Der generische Ablauf für nominale Facets stellt die einzelnen Werte dabei als Checkboxen dar. Wenn der Benutzer eine davon auswählt, teilt das Frontend dies dem Backend über die REST API mit. Das Backend generiert dann eine SPARQL Query, um den Datensatz zu filtern. Bei dieser hat die WHERE Klausel die Form *WHERE {P IN (Q)}*, wobei *Q* die Menge an ausgewählten Werten ist. Wäre *Q = false* und die Key Concepts nur mit *true* annotiert, würde fälschlicherweise eine leere Ergebnismenge zurückgegeben. Zwar könnte als Workaround noch eine *FILTER NOT EXISTS* Klausel mit den ausgeschlossenen Werten hinzugefügt werden, aber wegen der durchgängigen Konsistenz ist es besser, wenn explizit angegeben wird, ob eine Klasse ein Key Concept ist oder nicht.

## Facets

Die Ergebnisse aus der Berechnung für Facets (Abschnitt 5.2.3) werden unterschiedliche Art und Weise annotiert. Für ordinale und quantitative Prädikate reichen zwei Tripel für Minimum- und Maximumwerte (Listing 5.7) aus. Eigentlich müssten diese Werte nicht annotiert werden, da mit SPARQL 1.1 Aggregatfunktionen eingeführt wurden und das Backend sie genausogut zur Laufzeit berechnet könnte (von eventuellen Bedenken wegen Performance abgesehen). Jedoch sind MIN() und MAX() nicht in Jena ARQ implementiert, es stellt nur COUNT() und SUM() (und implizit AVG()) zur Verfügung. Um die wahrscheinlich langsamere manuelle Implementierung zur Laufzeit zu umgehen, wird dieser Schritt in der Vorverarbeitung ausgeführt.

Listing 5.7: Annotation der ordinalen/quantitativen Prädikate

```

1 <P> <hasMinValue> 51.1203
2 <P> <hasMaxValue> 51.2325

```

Nominale Prädikate werden nicht annotiert, da nur der Inhalt dupliziert würde und es keinen Unterschied macht, ob das Backend zur Laufzeit dasselbe Graph Pattern mit dem Prädikat selbst oder einem anderen sucht.

### 5.2.5 Laufzeit

Das Backend stellt auch verschiedene Funktionen zur Laufzeit zur Verfügung (Abschnitt 4.3.3).

#### Laden des Datensatzes

Das Data Repository stellt eine REST API zur Verfügung und REST ist zustandslos, hat also keine Sessions. Für die Erweiterung des DaRe waren Sessions aber nötig,

weil das Backend die aktuellen Models, Filter etc. für jeden User im Speicher hält, um Zeitaufwand und Netzwerktraffic zu minimieren. Gelöst wurde dies mit einem Singleton, der User, FileID, Cache und letzten Zugriff darauf speichert. Wenn eine Anfrage ankommt, sucht das Backend in diesem Singleton nach dem Schlüssel User+FileID und gibt den Cache zurück, falls der letzte Zugriff nicht länger als 20 Minuten zurück liegt. Ist kein Cache vorhanden, muss er erst aufgebaut werden. Dies geschieht mit einigen SPARQL Querys, welche die Ergebnisse aus der Analysephase aus dem Datensatz extrahieren.

## Pfadfindung

Das Backend bietet eine Schnittstelle zum Auffinden von Relationen zwischen Ressourcen an, dies entspricht Pfaden im RDF Graphen. Dazu wird intern ein Model, zwei URIs und zwei Parameter  $n$  und  $k$  angenommen. Der Parameter  $n$  entspricht der Anzahl an Relationen, die gefunden werden sollen,  $k$  gibt die maximale Länge des Pfades an. Voreingestellt sind  $k = 3$  und  $n = 5$  um den Benutzer nicht mit Informationen zu überladen. Das Model wird mit dem Framework JGraphT in einen ungewichteten gerichteten Graphen umgewandelt (siehe Abschnitt 5.2.3) und danach der Algorithmus »K shortest Paths« mit den übergebenen Parametern  $n$  und  $k$  ausgeführt. Die Kanten der Pfade werden danach mit einer SPARQL CONSTRUCT Query in ein neues Jena Model kopiert und zurückgegeben.

Sollen Pfade zwischen mehreren URIs gefunden werden, wird die Funktion für alle möglichen Paare an URIs ausgeführt und die resultierenden Jena Models gemerged.

Falls  $\geq n$  Pfade der Länge  $\leq k$  vorhanden sind, werden diese aktuell nicht beachtet und es gibt keine Möglichkeit zu beeinflussen, welche Pfade angezeigt werden. Dem könnte man beikommen, indem  $n$  mit einem großen Wert statisch oder dynamisch initialisiert wird und der Benutzer im Frontend mit Hilfe eines Sliders o. ä.  $k$  einstellen kann. Dabei besteht immer die Gefahr, dass  $n$  zu klein gewählt wurde. Eine weitere Möglichkeit wäre die Kanten zu gewichten, sodass niedrig gewichtete Pfade zuerst ausgegeben werden.

## Filter

Filter schränken die aktuell angezeigten Elemente ein. Bei einem Filter sind besonders das Ziel (Subjekt oder Property) und die Bedingung, welche erfüllt sein muss (Äquivalenz, enthalten in einer Menge oder innerhalb eines Wertebereichs) von Bedeutung. Der Benutzer kann Filter manuell erstellen oder automatisch basierend auf einer ausgewählten Ressource erstellen lassen.

Bei einem automatischen Filter ist die Semantik abhängig von der Ressource. Ist es eine **Klasse**, wird bei Klassen nur noch diese und deren Subklassen angezeigt. Bei den Propertys sind nur jene sichtbar, die diese Klasse als rdfs:domain oder rdfs:range haben und bei den Instanzen nur jene der Klasse bzw. Subklassen und die damit verbundenen (z. B. Klasse Event, Instanzen Papstwahl und Rom sichtbar, da Papstwahl → hasCity → Rom). Wurde eine **Property** ausgewählt, werden nur Klassen angezeigt, die rdfs:domain oder rdfs:range der Property sind. Bei Propertys werden

nur die ausgewählte Property und deren Subpropertys angezeigt, bei Instanzen nur jene, die diese Property implementieren. Wurde eine **Instanz** ausgewählt, werden nur dessen Klassen angezeigt und nur Propertys, die diese Instanz implementiert. Bei den Instanzen wird nur noch die Ausgewählte dargestellt.

Der Benutzer kann die Filter aber auch selbst mit dem in Abschnitt 4.2.5 beschriebenen Filterdialog erstellen. Dabei wurde der Schritt ausgelassen, wo der Benutzer den betroffenen Bestandteil angibt, da dies die Komplexität des Prototypen zu sehr erhöht hätte. Der betroffene Bestandteil ist im Prototypen immer der aktuell ausgewählte.

## Clustering

Das Clustering zur Laufzeit unterscheidet sich kaum vom Clustering in der Analysephase (Abschnitt 5.2.3). Es kann eine Menge an Attributen angegeben werden, die beim Clustering zusätzlich in Betracht gezogen werden sollen. Leider ist es im Prototypen nicht möglich, den Datensatz bezüglich der Ähnlichkeit zu einer gegebenen Ressource (Referenz) zu clustern, da zu diesem Zweck ein entsprechender Kerneloperator geschrieben werden müsste, was im Rahmen dieser Arbeit nicht möglich war und nicht im Fokus stand. Clustering zur Laufzeit funktioniert ansonsten prinzipiell, es müsste aber noch im Frontend ein entsprechendes Layout implementiert werden, welches die Ähnlichkeit zwischen Ressourcen passend visualisiert (vgl. Abschnitt 4.2.2).

## Pivoting

Beim Pivoting (Abschnitt 2.1.3) werden Vorschläge für weitere, möglicherweise interessante Ressourcen generiert. Die Einflussfaktoren dazu sind topologische Ähnlichkeit, semantische Ähnlichkeit, Key Concepts und Zufall. Der Algorithmus unterscheidet zwischen Klassen, Property und Instanzen, da nicht alle Einflussfaktoren für jeden Ontologiebestandteil anwendbar sind.

Tabelle 5.4 verdeutlicht den Sachverhalt. Topologische Ähnlichkeiten werden aktuell nur für Klassen berechnet, können aber auch für Property (rdfs:subPropertyOf Hierarchie) und Instanzen (RDF Graph) implementiert werden. Es wurde nicht durchgeführt, da diese Ähnlichkeitsberechnung sowieso relativ schnell implementiert ist und die Zeit in die Filterfunktionen besser investiert schien. Die semantische Ähnlichkeit wird aktuell von RMonto durchgeführt, welches aktuell nur Instanzen als Eingabe nimmt. Es ist möglich, RMonto mit eigenen Kerneloperatoren zu erweitern, die die Ähnlichkeiten für Property (z. B. rdf:domain und rdf:range als Basis) oder Klassen (z. B. gemeinsame Superklassen) definieren. Eine zufällige Ressource kann für jeden Ontologiebestandteil gefunden werden, hierzu werden erst die Statements gezählt, eine Zufallszahl  $r$  zwischen Null und Anzahl der Statements gewählt und zuletzt eine SPARQL Query mit OFFSET  $r$  LIMIT 1 ausgeführt. Da Key Concepts nur für Klassen definiert sind, ist es aktuell nicht möglich, etwas Äquivalentes für Property und Instanzen auszuführen.

	Klassen	Property	Instanzen
Topologisch	x	~	~
Semantisch	~	~	x
Zufall	x	x	x
Key Concept	x	-	-

Tabelle 5.4: Verfügbarkeit der Pivoting-Faktoren nach Ontologiebestandteil

## Hierarchie

Wenn der Benutzer Ressourcen auswählt, wird die Hierarchieansicht im Frontend (Abschnitt 4.2.4) entsprechend aktualisiert. Der Algorithmus dazu unterscheidet zwischen Klassen, Property und Instanzen.

Wurden Klassen ausgewählt, so wird mit einer SPARQL Query die Hierarchie aus dem Datensatz extrahiert. Diese entspricht im Prinzip drei Arten von Tripeln (Listing 5.8).

Listing 5.8: Tripel der Klassenhierarchie

```

1 //C und X sind Klassen
2 <C> rdf:type rdfs:Class
3 <C> rdf:type owl:Class
4 <C> rdfs:subClassOf <X>

```

Danach wird der Pfadfindungsalgorithmus aus Abschnitt 5.2.5 auf die Hierarchie angewendet, um Pfade zwischen der vom Benutzer ausgewählten Klasse und *rdf:Resource* zu finden.

Für Property passiert dasselbe, mit dem Unterschied das die Propertyhierarchie (Listing 5.9) verwendet und Pfade zu *rdf:Property* gefunden werden sollen.

Listing 5.9: Tripel der Propertyhierarchie

```

1 //P und Y sind Property
2 <P> rdf:type rdf:Property
3 <P> rdfs:subPropertyOf <Y>

```

Falls der Benutzer eine Instanz ausgewählt hat, wird der Pfadfindungsalgorithmus auf alle Klassen der Instanz ausgeführt und die resultierenden Models gemerged.

## Detaillierter Zugriff

Mit dieser Funktion soll der Benutzer Details zu einer Ressource angezeigt bekommen. Aktuell werden dazu alle verschiedenen Paare aus Prädikat und Objekt der Tripel (Ressource, Prädikat, Objekt) extrahiert und einer Tabelle angezeigt.

## 5.3 Evaluation

Um die Ergebnisse dieses Kapitels bewerten zu können, wurde der Prototyp evaluiert. Dabei wurde das Frontend einer kleinen Nutzerstudie unterzogen und die Performance des Backends untersucht. Darauf wird in den folgenden Abschnitten eingegangen.

### 5.3.1 Nutzerstudie

Um feststellen zu können, ob die Benutzer mit dem Prototypen besser zurecht kommen als mit der alten Komponente, wurde eine kleine Nutzerstudie durchgeführt. Die Methode entspricht dabei derselben wie in der ersten Studie: Die Teilnehmer mussten 10 Aufgaben innerhalb von jeweils 2 Minuten lösen und einen Task Load Index für jede davon ausfüllen.

#### Datenbasis

Eigentlich sollte für die Evaluation des Prototypen derselbe Datensatz verwendet werden wie bei der ersten Nutzerstudie, nämlich die im Szenario (Abschnitt 2.2) beschriebene QUDT. Leider war es nicht möglich, diese zu verwenden, da sie laut dem in RMonto verwendeten Reasoner (Pellet<sup>17</sup>) inkonsistent ist. Das ist beispielsweise der Fall, wenn verwendete Propertys nicht deklariert wurden oder Datenformate ungültig sind (z. B. 1.432 für *xsd:integer*). Aufgrund der nichtssagenden Fehlermeldung (»Reason for inconsistency: null«) konnte dieser Umstand leider nicht behoben werden.

Daraufhin wurde nach einem anderen geeigneten, großen Datensatz gesucht. Dieser sollte mehrere zehntausend Tripel beinhalten, die möglichst gleich in Schemainformationen und Instanzen aufgeteilt sind. Der Grund für die Größenbeschränkung war einerseits die schlechte Performance des Backends (siehe Abschnitt 5.3.2) und dass der Heapspace in der virtuellen Maschine für so große Datensätze nicht ausreichte. Letzteres hätte vermutlich mit einer 64 Bit JVM gelöst werden können, die Installation war aber nicht möglich, weil die Setup-Dateien »keine zulässige Win32-Anwendung« waren.

Die meisten Datensätze in der LOD Cloud übersteigen entweder die geforderte Größe bei weitem (mehrere zehn Millionen Tripel) oder beinhalten keine Schemainformationen. Die wenigen, welche die geforderten Eigenschaften hatten, waren leider ebenfalls inkonsistent<sup>18</sup>. Entweder war der Grund dafür wie bei der QUDT »null« oder die Taxonomie enthielt Zyklen.

Aus diesem Grund wurde der während der Entwicklung verwendete Testdatensatz *eventData.rdf* vom Lehrstuhl Multimediatechnik eingesetzt. Dieser enthält Daten über Musikevents in Dresden und besteht aus 1296 Instanzen, 19 Propertys und 4 Klassen in 9249 Tripeln.

#### Aufgaben

Da ein neuer Datensatz benutzt werden musste, mussten die Fragen, welche sich direkt darauf bezogen, geändert werden. Dabei wurde abgesehen von Frage 7, die auch inhaltlich geändert wurde, nur der Name der Ressource angepasst:

<sup>17</sup><http://clarkparsia.com/pellet/>

<sup>18</sup>Unter anderem wurden die Bibelontologie, die französische Pokédedia, ein aus der DBpedia selbst zusammengestellter Datensatz und eine Ontologie mit menschlichen Krankheiten und Symptomen ausprobiert.

1. Von welchem Konzept gibt es die meisten Instanzen?  
(Task Overview)
2. Welche Klasse hat die meisten direkten Subklassen?  
(Task Overview)
3. Finden Sie das Konzept *Place*. Nennen Sie Beispiele für Eltern, Kinder und Geschwister.  
(Task Zoom)
4. Nennen Sie Beispiele für Details der Ressource *musicbrainz\_guid*.  
(Task Details-on-demand)
5. Welche Ressourcen haben eine Property *hasCity*? Nennen Sie 3 Beispiele.  
(Task Relate)
6. Über welche Konzepte sind *Event* und *Place* verbunden? Geben Sie den vollständigen Pfad an.  
(Task Relate)
7. Nennen Sie Beispiele für Instanzen, die sich in Cluster 1 befinden.  
(Task Filter)
8. Blenden Sie alle Instanzen mit »name« im Namen aus.  
(Task Filter)
9. Legen Sie eine Art Lesezeichen für *Event* für einen späteren Zugriff an.  
(Task History)
10. Welche Ressource hatten Sie 5 Navigationsschritten zuvor ausgewählt?  
(Task History)

## Teilnehmer

Die Studie wurde mit fünf Teilnehmern durchgeführt, wobei drei davon auch an der ersten Studie teilnahmen, also das alte und das neue Konzept vergleichen konnten. Die Teilnehmer waren zwischen 22 und 24 Jahren alt und zu 80 % männlich.

## Diskussion der Ergebnisse

Bevor die konkreten Ergebnisse vorgestellt werden, wird die Vergleichbarkeit der beiden Studien diskutiert. Die erste Studie aus Kapitel 3 wurde mit einem wesentlich größeren Datensatz durchgeführt. Außerdem konnte für die alte Vorauswahlkomponente mehr Entwicklungszeit aufgewendet werden. Sie wurde mit 10 Teilnehmern durchgeführt. Im Gegensatz dazu wurde die zweite Nutzerstudie mit einem relativ kleinen Datensatz durchgeführt, der zum Beispiel gerade einmal vier Klassen beinhaltete. Der Prototyp hat wegen dem engeren Zeitrahmen noch einige Bugs, z. B. funktioniert die Navigation noch nicht ausreichend gut und nachdem die angezeigten Daten in einem View geändert wurden, verschwinden Knoten oder wechseln sprunghaft ihre Position. Die Nutzerstudie wurde mit 5 Teilnehmern durchgeführt. Sie sollte außerdem weniger ein bestehendes Konzept optimieren, sondern ein Neues

auf seine Tauglichkeit überprüfen. Aus diesem Grund werden die Metriken zur Effizienz nicht diskutiert.

Tabelle A.6 zeigt, welche Testperson welche Aufgabe erfolgreich gelöst hat und vergleicht die Ergebnisse mit jenen aus der ersten Studie. In drei Aufgaben (1, 2 und 8) waren die Ergebnisse schlechter als zuvor, in zwei dieser drei aber auch nur knapp (-10 Prozentpunkte). Auf der anderen Seite waren zwei Verbesserungen (bei Aufgaben 5 und 6) ebenfalls nur knapp (+10 Prozentpunkte) und drei blieben gleich (Aufgaben 4, 7 und 9). Als Aufgaben mit signifikantem Unterschied bleiben 1, 3 und 10.

In Aufgabe 1 sollte die Klasse mit den meisten Instanzen gefunden werden. Dies konnte entweder mit einem Facet oder dem Detaildialog gelöst werden. Dass so wenige Testpersonen die Aufgabe lösen konnten, könnte darauf zurückzuführen sein, dass die Facets im Tutorial nicht ausreichend erklärt wurden oder schlecht sichtbar sind.

Aufgabe 3 befasste sich mit der Taxonomie, es sollten Superklassen, Subklassen und »Geschwisterklassen« von *Place* gefunden werden. Dies wurde wesentlich besser gelöst als in der ersten Studie, hängt aber wahrscheinlich sehr damit zusammen, dass nur vier Klassen vorhanden waren und diese keine Superklassen, Subklassen etc. hatten.

In Aufgabe 10 sollten die Teilnehmer bestimmen, welche ihre fünftletzte Aktion war. Im Gegensatz zur alten Studie, wo es niemand schaffte, konnte es hier jede getestete Person.

Das selbst formulierte Feedback der Teilnehmer war verhalten positiv. Gut angekommen sind der Basket (wie in der ersten Studie auch) und die Timeline, mit jeweils 5/5 Nennungen. Auch die Tabs waren eine Verbesserung (2/5) und die automatisch generierten Filter wurden ebenfalls als sinnvoll befunden (2/5). Definitiv noch zu verbessern sind Layout und Navigation (jeweils 5/5). Die Quadrate und die Beschriftung wurden als zu klein empfunden, die Navigation als zu empfindlich und nicht flüssig genug. Am Dialog um Filter hinzuzufügen muss ebenfalls noch gearbeitet werden (1/5). Basierend auf dem Feedback, dem persönlichen Gespräch mit den Teilnehmern und den Beobachtungen kann angenommen werden, dass das Konzept in einer fertigen Software gut funktioniert.

### 5.3.2 Performance

Für die Untersuchung der Performance stand eine Virtuelle Maschine mit Windows XP Professional SP3 und einer Intel Xeon E5405 mit  $2 \times 2$  GHz sowie 3 GB RAM zur Verfügung. Dabei wurde der in der zweiten Nutzerstudie verwendete *eventData* Datensatz verwendet. Dieser enthält 1296 Instanzen, 19 Propertys und 4 Klassen in 9249 Tripeln. Als Datensatz zum Vergleich wurden daraus 100 Tripel mit zufällig gewählten Instanzen extrahiert. Größere Datensätze konnten nicht verwendet werden, da dafür der Heapspace in der VM nicht ausreichte und (als mögliche Lösung) keine

64 Bit JRE installiert werden konnte, weil die Installationsdateien »keine zulässige Win32-Anwendung« waren.

## Ergebnisse

Die Ergebnisse zeigen, dass das Backend in seiner jetzigen Form für einen Einsatz in der echten Welt nicht geeignet ist (Tab. 5.5, VVA steht für Vorverarbeitung), da es überhaupt nicht skaliert. Für den Benutzer ist es nicht akzeptabel beim Upload und der Extraktion jeweils länger als ein paar Minuten zu warten.

	eventData	100 Tripel
Größe vor VVA	1.1 MB / 9249 Tripel	13 KB / 100 Tripel
Größe nach VVA	1.14 GB / 8445247 Tripel	35 KB / 257 Tripel
Zeit für VVA	90 Minuten	< 10 Sekunden
Zeit für Extraktion	> 1 Stunde	< 30 Sekunden

Tabelle 5.5: Vergleich der Performance

Es sind verschiedene Ansätze zur Verbesserung der Performance denkbar, sie teilen sich in Hardware, Software und Programmierung. Die Verbesserung der Hardware ist offensichtlich, ein Rechner mit mehr RAM und CPU-Leistung wird bessere Ergebnisse erzielen. Es könnte zur Verbesserung auf der Softwareseite ein anderer, leistungsfähigerer Triplestore als Apache Jena getestet werden (z. B. Sesame<sup>19</sup> oder OWLIM<sup>20</sup>). Um die Programmierung zu optimieren sollten die SPARQL Querys reduziert werden. Viele davon haben ein offenes Graph Pattern ( $?s ?p ?o$ ) in der WHERE-Klausel, was sehr zur Dauer der Query beiträgt. Beispielsweise werden direkt nach dem Einlesen des annotierten Datensatzes sämtliche Tripel extrahiert, die eine Annotation beinhalten, was im Falle von eventData mehr als ein Gigabyte ist. Zur Laufzeit ist der Datensatz logisch in ein Annotationsmodel und drei Models für Klassen, Propertys und Instanzen aufgeteilt. Würde diese Aufteilung auch physisch auf der Festplatte vorhanden sein, könnten solche Querys zur Trennung der logischen Models eingespart und Zeit gewonnen werden. Eine weitere Möglichkeit wäre die Analysephase so zu implementieren, dass deren Bestandteile möglichst parallel ausgeführt werden können.

<sup>19</sup><http://www.openrdf.org/>

<sup>20</sup><http://www.ontotext.com/owlim>

# 6 Zusammenfassung und Ausblick

Diese Arbeit setzte sich mit den Problemen, auf die Benutzer bei der Navigation und Selektion in großen semantischen Datensätzen stoßen, auseinander und zeigte Lösungsansätze auf. Beispiele für derartige Probleme sind unter anderem relevante Zusammenhänge zu finden, Patterns zu erkennen oder sich einen Überblick zu verschaffen. Im Fokus lag dabei die Entwicklung eines Konzepts einer Datenvorauswahlkomponente für die Visualisierungsworkbench VizBoard. Das Konzept sollte prototypisch umgesetzt werden.

## 6.1 Zusammenfassung

Zu diesem Zweck wurden zuerst die Grundlagen von Ontologien und Visualisierungs-, Navigations- und Interaktionskonzepten erarbeitet. Dabei wurden bei Visualisierungskonzepten die Vor- und Nachteile von z. B. Tabellen und Graphen sowie Nested Views und dreidimensionalen Darstellungen betrachtet und festgestellt, dass Graphen wegen der Semantik der RDF Tripel die verbreitetste Darstellung für Ontologien sind. Verschiedene Navigations- bzw. Interaktionskonzepte wie z. B. Breadcrumbs, Overview/Detail, Pan & Zoom und Faceted Navigation wurden einander gegenübergestellt und auf ihre Einsetzbarkeit in semantischen Datensätzen untersucht. Es stellt sich heraus, dass alle in irgendeiner Form einsetzbar sind. Diejenigen, welche die meisten Menschen schon aus dem Internet kennen, erschienen aber besonders vielversprechend. Drei grundlegende Algorithmen zur Clusteranalyse wurden eingeführt, alle können - eine entsprechende Distanzfunktion vorausgesetzt - mit semantischen Datensätzen arbeiten. Danach wurden zwei speziell für semantische Daten konzipierte Techniken betrachtet: Die Key Concept Extraction bestimmt zur einer gegebenen Ontologie die  $n$  wichtigsten Konzepte, wofür unter anderem auf psycholinguistische Aspekte zurückgegriffen wird. Böhm et al. konnten durch Clusteranalyse semantisch korrelierende Cluster in Datensätzen finden. Damit wurde gezeigt, wie es möglich ist, automatisch relevante Daten aus dem Datensatz zu extrahieren. Weiterhin wurden drei verschiedene Visualisierungstools (Jambalaya, Knoocks, TopBraid Composer) für semantische Datensätze auf ihre Tauglichkeit bezüglich großer Datenmengen untersucht. Dabei wurde festgestellt, dass diese in den Kategorien Navigationshilfen, Filtermöglichkeiten und Layoutflexibilität am schlechtesten abschnitten und dass bei der Konzeption auf diese drei Punkte besonders geachtet werden muss.

Um zusätzliches Feedback bezüglich Visualisierung von und Interaktion mit großen semantischen Daten zu bekommen, wurde mit einem bereits bestehenden Mockup eine Nutzerstudie durchgeführt. Dabei wurden 10 Personen befragt, die jeweils 10 Aufgaben innerhalb eines Zeitlimits von zwei Minuten ohne Hilfe lösen sollten. Danach konnten sie in eigenen Worten angeben, was sie gut fanden und was nicht bzw.

sagen, wo die Probleme lagen. Zusätzlich mussten sie einen Fragebogen in Form eines NASA Task Load Indexes ausfüllen, bei dem verschiedene Ausprägungen von Usabilityproblemen (zu anstrengend, zu frustrierend, zu zeitaufwändig etc.) untersucht werden. In den Ergebnissen der Studie zeigte sich, dass die größten Probleme die schlechte Performance (Ruckeln, keine Reaktion auf Eingaben) und eine schlechte/-überfordernde Darstellung des Datensatzes waren. Gut gelöst waren laut Aussagen der Testpersonen hingegen die Lesezeichenfunktion und die parallele Darstellung von Ontologiebestandteilen. Als weitere Verbesserungen wurden die Automatisierung von einfachen Aufgaben (wie z. B. Instanzen zählen), mehr Rückmeldungen an den Benutzer und die Vereinfachung der Bedienung identifiziert.

Für das Konzept wurde eine Anforderungsanalyse basierend auf Shneidermans »Information-seeking Mantra« und den Untersuchungskriterien der Visualisierungstools durchgeführt, bei der 41 Anforderungen identifiziert werden konnten. Darauf aufbauend wurde ein User Interface (Frontend) entworfen. Es bietet eine visuelle Repräsentation des Datensatzes und verschiedene andere Möglichkeiten, relevante Daten in diesem zu finden (z. B. Pivoting, Clustering, Verbindungen zwischen Ressourcen finden). Zu den schwierigsten Entscheidungen hierbei gehörte zwischen Funktionalität der Software und Komplexität des Interfaces abzuwagen. Zum Beispiel kann man sich sehr viele unterschiedliche Möglichkeiten ausdenken, einen Filter auf den Datensatz anwenden zu wollen. Diese aber alle auch umzusetzen würde das Interface so komplex machen, dass niemand mehr filtern kann. Um rechenintensive Vorgänge (wie z. B. Clustering) auslagern zu können, wurde ein Backend eingeführt, welches diese Funktionen dem Frontend zur Verfügung stellt. Die Entwicklung eines Konzepts für das Backend ging relativ einfach, da durch die Funktionalität des Frontends klar war, welche Schnittstellen angeboten werden müssen. Durch die Rahmenbedingungen, welche das bestehende Data Repositorys setzte, konnte schnell entschieden werden, wo die Funktionen integriert werden sollten.

Die wichtigsten Funktionen des Konzepts wurden danach in einem Prototypen umgesetzt. Das Frontend desselben wurde mit gängigen Webtechnologien realisiert. Bei der Arbeit an diesem zeigte sich, dass D3 viele einfache Schnittstellen zu wichtigen Funktionen bietet (Data Joins, Animationen). Komplexere Funktionen wie Pan & Zoom mussten jedoch selbst implementiert werden. Die Entwicklung der Listen und Fenster ging mit ExtJS einfach von der Hand, dafür waren direkte DOM Manipulationen im Vergleich zu jQuery umständlicher durchzuführen. Das Backend wurde mit Hilfe von Java und diversen Bibliotheken geschrieben. Eine Herausforderung hierbei war die logische Trennung des Datensatzes in Teilmengen, wie z. B. Klassen oder Instanzen. Die Entscheidung, welche Tripel wohin gehören und an welche Models die Anfragen zur Laufzeit gestellt werden, war nicht einfach und änderte sich oft. Auch das Einbinden von RapidMiner als Bibliothek und dem zugehörigen Plugin RMonto dauerte länger als gedacht, weil im Internet praktisch keine Dokumentation dafür zu finden ist.

Der neue Prototyp wurde in einer kleinen Nutzerstudie mit fünf Teilnehmern evaluiert, wobei drei davon schon an der ersten Studie teilnahmen. Dabei wurde ein anderer Datensatz als in der ersten Nutzerstudie verwendet, da der in RMonto ver-

wendete Reasoner viele Ontologien als inkonsistent deklarierte, RMonto deswegen nicht clustern konnte und so die Vorverarbeitung fehlschlug. Es wurde auf den während der Entwicklung verwendeten Testdatensatz *eventData*, der 9249 Tripel enthält, zurückgegriffen. Die Aufgabenstellung wurde an den neuen Datensatz angepasst, indem bei Fragen, die sich direkt auf Ressourcen beziehen, diese geändert wurden. Die Ergebnisse der Studie lassen vermuten, dass das in dieser Arbeit erarbeitete Konzept gut funktionieren kann. Sie zeigen aber auch, dass die Manipulation des Viewports (Pan & Zoom) eine kritische Funktionalität der Software ist und äußerst flüssig und problemlos funktionieren muss.

## 6.2 Ausblick

Für das User Interface des Prototypen sind noch weitere Verbesserungen denkbar. Zum Beispiel hat der Benutzer bei ordinalen bzw. quantitativen **Facets** das Problem, dass er, bevor er den Slider betätigt, nicht weiß, wie groß die Ergebnismenge sein wird. Das kann z. B. mit Histogram Slidern [HS12] verbessert werden. Die nominalen Facets sind bei großen Datensätzen tendenziell problematisch. Attributte wie »Username« treten zu tausenden auf und können nicht durchsucht werden. Außerdem nimmt die Darstellung sehr viel Platz ein. Neben offensichtlichen Aufgaben wie der Umsetzung weiterer Teile des Konzepts sollte die **Berechnung des Layouts** in das Backend verlagert werden. Wenn sehr viele ( $> 1000$ ) Knoten das erste Mal gerendert werden, friert das User Interface für eine unbestimmte Zeit (aber abhängig von der Anzahl der Knoten) ein. Dazu kann z. B. auf das Java Universal Network/Graph Framework<sup>1</sup> (JUNG) zurückgegriffen werden. Dieses würde im Gegensatz zu D3 auch den für die Hierarchieansicht nötigen Layoutalgorithmus nach Sugiyama [STT81] zur Verfügung stellen<sup>2</sup>. Eine Alternative dazu wäre, die D3 Bibliothek auf ihr **Optimierungspotenzial** bezüglich der JavaScript-Befehle und DOM-Manipulationen zu untersuchen.

Für die Weiterentwicklung des Backends des Prototypen bleiben noch einige Themen. Zum Beispiel müssten semantisch korrelierende **Cluster** auch für Klassen und Propertys gefunden werden können, um so die Vorschläge an den Nutzer verbessern und mehr bzw. vielleicht nützlichere Facets anzeigen zu können. Für Instanzen sollten eher die gemeinsamen Propertys anstatt der gemeinsamen Superklassen als Grundlage zur **Ähnlichkeitsberechnung** dienen, um von eventuell nicht vorhandenen Schemainformationen unabhängiger zu werden. Bei der Clusteranalyse wäre es weiters wünschenswert, wenn die Ähnlichkeitsmatrix nicht zweimal berechnet würde. So könnte einiges an Rechenzeit eingespart werden. Außerdem werden in der Analysephase verschiedene **Parameter** statisch verwendet, bei denen noch untersucht werden sollte, ob die Werte brauchbar sind und ob sie dynamisch festgelegt werden können. Dazu gehören der 80 % Threshold bevor ein Prädikat zu einem Clusterschema hinzugefügt wird, die Anzahl an zu extrahierenden Key Concepts (momentan immer 10 %) und die Anzahl an darzustellenden Relationen sowie die maximale Pfadlänge zwischen zwei Ressourcen (5 Relationen zu max. 3 »Hops«). Bei letzteren

<sup>1</sup><http://jung.sourceforge.net/>

<sup>2</sup>Bei D3 steht das Sugiyama Layout auf der Roadmap, wurde aber noch nicht implementiert.

<https://github.com/mbostock/d3/issues/349>

kann auch darüber nachgedacht werden, ob sie nicht besser vom Benutzer selbst angegeben werden sollten (z. B. mit einem Slider in der entsprechenden Anzeige). Zusätzlich dazu müssen Wege gefunden werden, um die **Performance** zu erhöhen. Der Testdatensatz, mit dem entwickelt wurde, ist während der Annotationsphase - die etwa 90 Minuten dauerte - um Faktor 1000 gewachsen und die Extraktionsphase dauerte deswegen länger als eine Stunde.

# A Anhang

## A.1 Erste Nutzerstudie

### A.1.1 Teilnehmer

### A.1.2 Lösungen der Aufgaben

Lösungen zu den Fragen an die Teilnehmer der Nutzerstudie (Abschnitt 3.5):

1. CODATA Value, Physical Constant.
2. Mechanics Unit (15 direkte Subklassen).
3. Eltern: Science And Engineering Unit. Kinder: Atomic Physics Unit, Mechanics Unit, Logarithmic Unit, Chemistry Unit, Electricity And Magnetism Unit. Kinder: Communications Unit, Bio And Medical Unit.
4. Label: »Inverse Second«; Abbreviation:  $s^{-1}$ ; Conversion Multiplier: 1.0; Type: Derived Unit, Frequency Unit, SI Derived Unit; Quantity Kind: Frequency; Conversion: Offset 0.0.
5. Biot  $\leftrightarrow$  Statampere, Gon  $\leftrightarrow$  Grad, Hertz  $\leftrightarrow$  Per Second, Knot  $\leftrightarrow$  Nautical Mile Per Hour, Metric Ton  $\leftrightarrow$  Ton Metric, Minute Angle  $\leftrightarrow$  Arc Minute, Second Angle  $\leftrightarrow$  Arc Second, Siemens  $\leftrightarrow$  Mho, Statcoulomb  $\leftrightarrow$  Franklin.
6. Currency Unit  $\rightarrow$  Financial Unit  $\rightarrow$  Resource Unit  $\rightarrow$  Information Entropy Unit  $\rightarrow$  Logarithmic Unit  $\rightarrow$  Physical Unit  $\leftarrow$  Mechanics Unit.
7. Armenian Dram (0), Bahraini Dinar (3), Belarussian Ruble (0), Burundian Franc (0), CFA Franc BCEAO (0), CFA Franc BAC (0), CFP Franc (0), Chilean Peso (0), Colombian Peso (0), Comoro Franc (0), Djibouti Franc (0), Dobra (0), Forint (0), Guarani (0), Guinea Franc (0), Hong Kong Dollar (1), Iceland Krona (0), Iranian Rial (0), Iraqi Dinar (0), Japanese Yen (0), Jordanian Dinar (3), Kuwaiti Dinar (3), Kwanza (1), Kyat (0), Laos Kip (0), Lebanese Pound (0), Leone (0), Lybian Dinar (3), Malagasy Ariary (0), Million US Dollars ( $1.0^6$ ), New Taiwan Dollar (1), North Korean Won (0), Rial Omani (3), Ouguiya (0), Pataca (1), Riel (0), Rupiah (0), Rwanda Franc (0), South Korean Won (0), Tunisian Dinar (3), Uganda Shilling (0), Unidades de formento (0), Vatu (0), Vietnamese Dong (0), Yemeni Rial (0), Yuan Renminbi (1), Zambian Kwacha (0).

### A.1.3 Ergebnisse der Nutzerstudie

## A.2 Zweite Nutzerstudie

UserID	Age	Sex	Beruf	Level_tools	level_sem	hours_day	Fachgebiet	qudt	tutorial
user1	24	w	Chemikerin	1	1	3	Chemie	FALSE	TRUE
user2	22	m	Student	2	2	10	Medieninformatik	FALSE	TRUE
user3	29	m	Wiss. MA	4	5	8	Informatik	TRUE	FALSE
user4	30	m	Wiss. MA	3	3	9	Informatik	FALSE	FALSE
user5	27	m	Student	3	2	7	Informatik	FALSE	TRUE
user6	22	w	Student	1	1	10	Zahnmedizin	FALSE	TRUE
user7	23	m	Student	4	2	10	Medieninformatik	FALSE	TRUE
user8	27	m	Student	3	4	8	Informatik	FALSE	TRUE
user9	28	m	Softwareentwickler	4	4	10	Medieninformatik	FALSE	TRUE
user10	24	m	Student	2	3	5	Chemieingenieur	FALSE	TRUE

Tabelle A.1: Testpersonen und ihre Facetten

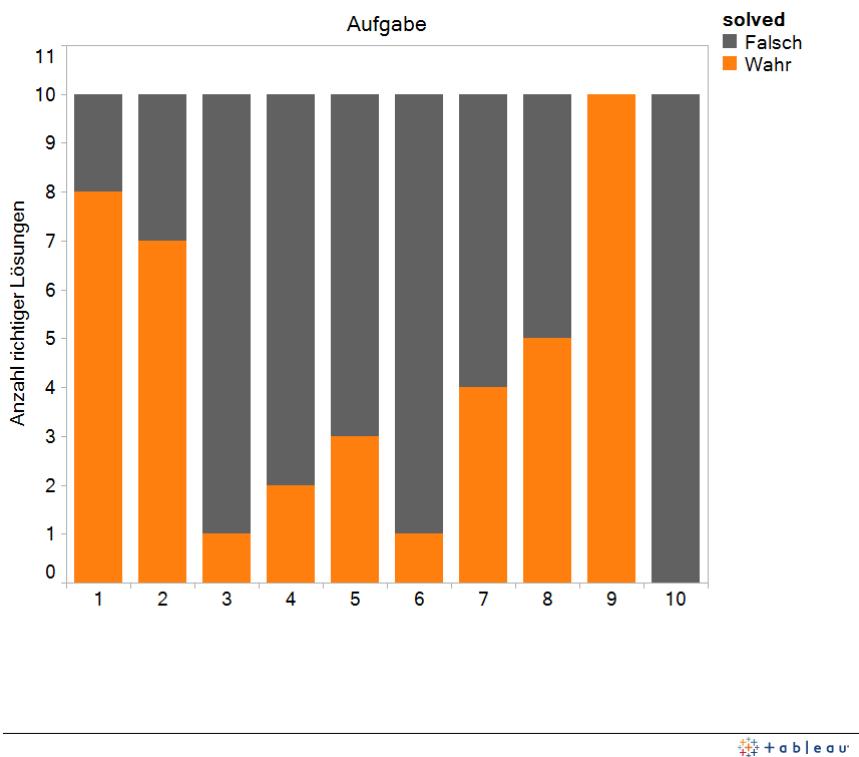


Abbildung A.1: Gelöste Aufgaben

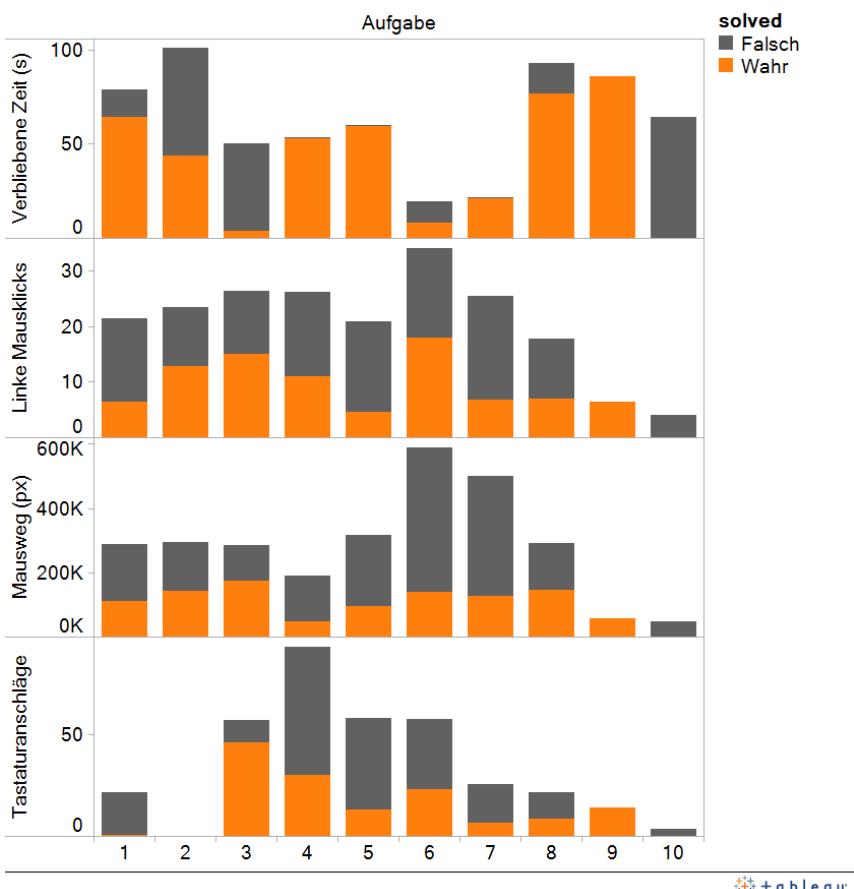


Abbildung A.2: Durchschnitt der Effizienzmetriken pro Aufgabe

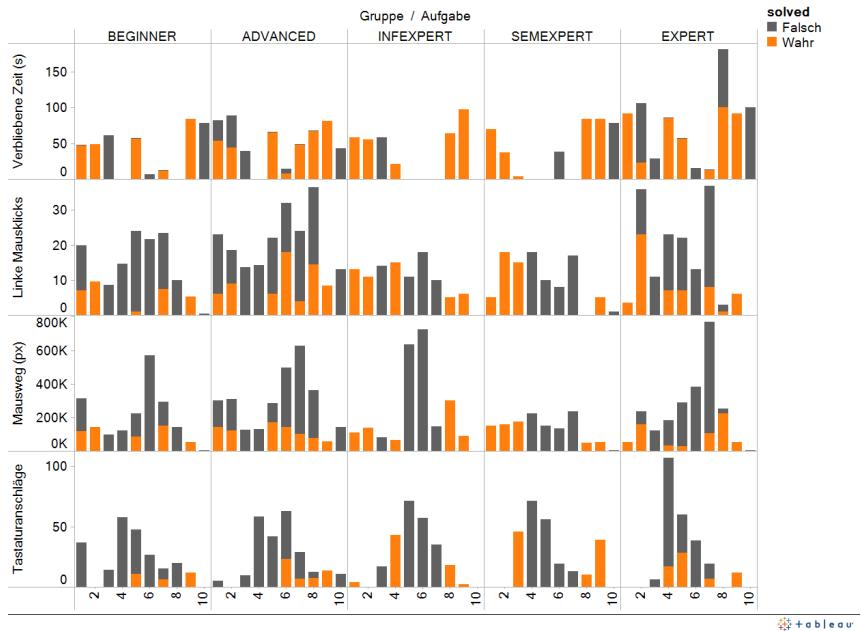


Abbildung A.3: Durchschnitt der Effizienzmetriken pro Aufgabe und Nutzergruppe

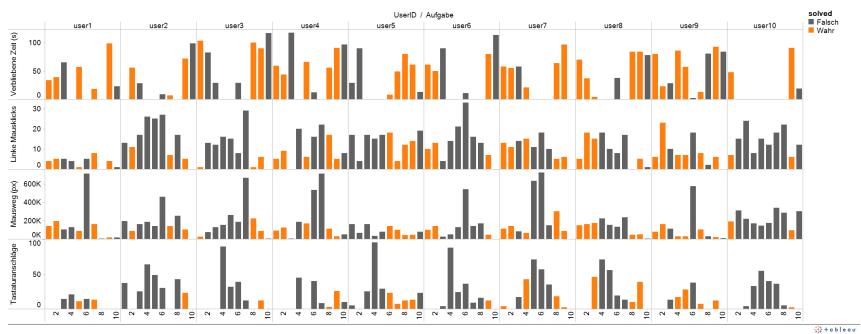


Abbildung A.4: Durchschnitt der Effizienzmetriken pro Aufgabe und Testperson

Was war schwer zu lösen?	Anteil User
Verbindungen finden	40,00%
Suchen/Filtrern	30,00%
Navigation nachvollziehen	30,00%
Propertys lesen	20,00%
Ressourcen ausblenden	10,00%
Namen v. Klassen merken	10,00%
Quantitative Aussagen	10,00%

Tabelle A.2: Schwierige Teilaufgaben

Was war einfach zu lösen?	Anteil User
Lesezeichen	60,00%
Klassen suchen	40,00%
Zählen der Instanzen	30,00%

Tabelle A.3: Einfache Teilaufgaben

Was hat Ihnen nicht gefallen?	Anteil User
Responsiveness	40,00%
Lesbarkeit	20,00%
Layout unübersichtlich	20,00%
Reset-Fkt	20,00%
Begriffe	20,00%
Unerwartete Reaktionen d. SW	10,00%
Vollbildfunktion	10,00%
Regular Expressions	10,00%
Relationen zw. Klassen	10,00%
Arbeiten mit Fenstern	10,00%

Tabelle A.4: Schlecht gelöste Dinge

Was hat Ihnen gut gefallen?	Anteil User
Parallele Darstellung	30,00%
Lesezeichen	10,00%
Klassen suchen	10,00%
Fancy Layout	10,00%
Textsuche	10,00%

Tabelle A.5: Gut gelöste Dinge

User	1	2	3	4	5	6	7	8	9	10
user1	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
user2	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
user3	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
user4	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
user5	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
Summe	20%	60%	60%	20%	20%	40%	20%	40%	100%	100%
Vergleich	80%	70%	10%	20%	30%	10%	40%	50%	100%	0%
Differenz	-60%	-10%	50%	0%	10%	10%	0%	-10%	0%	100%

Tabelle A.6: Ergebnisse der zweiten Nutzerstudie

# Literaturverzeichnis

- [Ala] Harith Alani. »TGVizTab: An Ontology Visualisation Extension for Protege«. In: (Siehe S. 27).
- [Aub04] David Auber. »Tulip - A Huge Graph Visualization Framework«. In: *Graph Drawing Software*. Hrsg. von Michael Juenger u. a. Mathematics and Visualization. 10.1007/978-3-642-18638-7\_5. Springer Berlin Heidelberg, 2004, S. 105–126. ISBN: 978-3-642-18638-7. URL: [http://dx.doi.org/10.1007/978-3-642-18638-7\\_5](http://dx.doi.org/10.1007/978-3-642-18638-7_5) (siehe S. 27).
- [BA03] N. Bolshakova und F. Azuaje. »Cluster validation techniques for genome expression data«. In: *Signal Process.* 83.4 (Apr. 2003), S. 825–833. ISSN: 0165-1684. DOI: [10.1016/S0165-1684\(02\)00475-9](https://doi.org/10.1016/S0165-1684(02)00475-9). URL: [http://dx.doi.org/10.1016/S0165-1684\(02\)00475-9](http://dx.doi.org/10.1016/S0165-1684(02)00475-9) (siehe S. 96).
- [BBP05] Alessio Bosca, Dario Bonino und Paolo Pellegrino. »OntoSphere: more than a 3D ontology visualization tool«. In: *Proceedings of the 2nd Italian Semantic Web Workshop*. 2005 (siehe S. 13).
- [Bed+04] Benjamin B. Bederson u. a. »DateLens: A fisheye calendar interface for PDAs«. In: *ACM Trans. Comput.-Hum. Interact.* 11 (1 März 2004), S. 90–119. ISSN: 1073-0516. DOI: <http://doi.acm.org/10.1145/972648.972652>. URL: <http://doi.acm.org/10.1145/972648.972652> (siehe S. 16).
- [Bed00] Benjamin B. Bederson. »Fisheye menus«. In: *Proceedings of the 13th annual ACM symposium on User interface software and technology*. UIST '00. San Diego, California, United States: ACM, 2000, S. 217–225. ISBN: 1-58113-212-3. DOI: <http://doi.acm.org/10.1145/354401.354782>. URL: <http://doi.acm.org/10.1145/354401.354782> (siehe S. 16).
- [Ber07] Mike Bergman. *An Intrepid Guide To Ontologies*. abgerufen am 12.11.2011. Mai 2007. URL: <http://www.mkbergman.com/374/an-intrepid-guide-to-ontologies/> (siehe S. 5).
- [Bez81] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1981. ISBN: 0306406713. URL: <http://www.amazon.com/exec/obidos/ASIN/0306406713/acmorg-20> (siehe S. 24).
- [BH09] Michael Bostock und Jeffrey Heer. »Protovis: A Graphical Toolkit for Visualization«. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009), S. 1121–1128. ISSN: 1077-2626. DOI: [10.1109/TVCG.2009.174](https://doi.org/10.1109/TVCG.2009.174). URL: <http://dx.doi.org/10.1109/TVCG.2009.174> (siehe S. 90).

- [BL+06] Tim Berners-Lee u. a. »Tabulator: Exploring and analyzing linked data on the semantic web«. In: *In Proceedings of the 3rd International Semantic Web User Interaction Workshop*. 2006 (siehe S. 20).
- [Boh+10] Christoph Bohm u. a. »Profiling linked open data with ProLOD«. In: *Data Engineering Workshops, 22nd International Conference on* 0 (2010), S. 175–178. DOI: <http://doi.ieeecomputersociety.org/10.1109/ICDEW.2010.5452762> (siehe S. 26, 76).
- [BOH11] Michael Bostock, Vadim Ogievetsky und Jeffrey Heer. »Data-Driven Documents«. In: *Visualization and Computer Graphics, IEEE Transactions on* 12 (2011), S. 2301 –2309. ISSN: 1077-2626. DOI: <10.1109/TVCG.2011.185> (siehe S. 90).
- [Bol98] Bela Bollobas. *Modern Graph Theory*. Corrected. Springer, Juli 1998. ISBN: 0387984887. URL: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387984887> (siehe S. 8).
- [Bos] Mike Bostock. *Protovis Focus & Context*. abgerufen am 18.11.2011 (siehe S. 18).
- [Bra01] U. Brandes. *A faster algorithm for betweenness centrality*. 2001. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.1.2024> (siehe S. 95).
- [BS07] Stephan Bloehdorn und York Sure. »Kernel methods for mining instance data in ontologies«. In: *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*. ISWC'07/ASWC'07. Busan, Korea: Springer-Verlag, 2007, S. 58–71. ISBN: 3-540-76297-3, 978-3-540-76297-3. URL: <http://dl.acm.org/g/citation.cfm?id=1785162.1785168> (siehe S. 93).
- [CJ11] Richard Cyganiak und Anja Jentzsch. *The Linking Open Data cloud diagram*. abgerufen am 18.11.2011. Sep. 2011. URL: <http://richard.cyganiak.de/2007/10/lod/> (siehe S. 9).
- [CKB06] Andy Cockburn, Amy Karlson und Benjamin B. Bederson. *A review of focus and context interfaces*. Techn. Ber. 2006 (siehe S. 18).
- [DBp11] DBpedia. *DBpedia 3.7 released, including 15 localized Editions*. abgerufen am 25.10.2011. Sep. 2011. URL: <http://blog.dbpedia.org/2011/09/11/dbpedia-37-released-including-15-localized-editions/> (siehe S. 1).
- [Dix11] Alan Dix. *Are five users enough?* abgerufen am 12.12.2011. Juni 2011. URL: <http://www.alandix.com/blog/2011/06/04/are-five-users-enough/> (siehe S. 45, 46).
- [Dun73] J. C. Dunn. »A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters«. In: *Journal of Cybernetics* 3.3 (1973), S. 32–57. DOI: <10.1080/01969727308546046>. eprint: <http://www.tandfonline.com/doi/pdf/10.1080/01969727308546046>. URL: <http://www.tandfonline.com/doi/abs/10.1080/01969727308546046> (siehe S. 24).

- [ES03] Neil Ernst und Margaret-Anne Storey. *A preliminary analysis of visualization requirements in knowledge engineering tools*. Techn. Ber. 2003 (siehe S. 4).
- [FR91] Thomas M. J. Fruchterman und Edward M. Reingold. »Graph Drawing by Force-directed Placement«. In: *Software - Practice and Experience* 21.11 (1991), S. 1129–1164. URL: <http://citeseervx.ist.psu.edu/vi ewdoc/summary?doi=10.1.1.13.8444> (siehe S. 19).
- [FSH05] Christiaan Fluit, Marta Sabou und Frank van Harmelen. »Ontology-based Information Visualisation: Towards Semantic Web Applications«. In: *Visualising the Semantic Web (2nd edition)*. Hrsg. von Vladimir Geroimenko. Springer Verlag, 2005 (siehe S. 9).
- [G.88] HART S. G. »Development of NASA-TLX (Task Load Index) : Results of Empirical and Theoretical Research«. In: *Human Mental Workload* (1988). URL: <http://ci.nii.ac.jp/naid/10015399884/en/> (siehe S. 47).
- [Gam+95] Erich Gamma u. a. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson, 1995 (siehe S. 78, 89).
- [GR69] J. C. Gower und G. J. S. Ross. »Minimum Spanning Trees and Single Linkage Cluster Analysis«. English. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 18.1 (1969), pp. 54–64. ISSN: 00359254. URL: <http://www.jstor.org/stable/2346439> (siehe S. 9).
- [Gua98] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. 1st. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1998. ISBN: 9051993994 (siehe S. 4).
- [GW] Sunil Goyal und Rupert Westenthaler. *RDF Gravity* (siehe S. 27).
- [Har06] Sandra G. Hart. »Nasa-Task Load Index (NASA-TLX) - 20 Years Later«. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. 2006 (siehe S. 47).
- [HK11] Ralph Hodgson und Paul Keller. *QUDT - Quantities, Units, Dimensions and Data Types in OWL and XML*. abgerufen am 12.11.2011. Sep. 2011. URL: <http://qudt.org> (siehe S. 20).
- [HMM00] I. Herman, G. Melancon und M.S. Marshall. »Graph visualization and navigation in information visualization: A survey«. In: *Visualization and Computer Graphics, IEEE Transactions on* 6.1 (Feb. 2000), S. 24 –43. ISSN: 1077-2626. DOI: [10.1109/2945.841119](https://doi.org/10.1109/2945.841119) (siehe S. 14).
- [HS12] Jeffrey Heer und Ben Shneiderman. »Interactive Dynamics for Visual Analysis«. In: (Feb. 2012) (siehe S. 108).
- [JK96] Bonnie E. John und David E. Kieras. »The GOMS family of user interface analysis techniques: comparison and contrast«. In: *ACM Trans. Comput.-Hum. Interact.* 3 (4 1996), S. 320–351. ISSN: 1073-0516. DOI: [http://doi.acm.org/10.1145/235833.236054](https://doi.acm.org/10.1145/235833.236054). URL: <http://doi.acm.org/10.1145/235833.236054> (siehe S. 46).

- [Joh67] Stephen Johnson. »Hierarchical clustering schemes«. In: *Psychometrika* 32 (3 1967). 10.1007/BF02289588, S. 241–254. ISSN: 0033-3123. URL: <http://dx.doi.org/10.1007/BF02289588> (siehe S. 24).
- [Jor90] Anker Helms Jorgensen. »Thinking-aloud in user interface design: a method promoting cognitive ergonomics«. In: *Ergonomics* 33.4 (1990) (siehe S. 47).
- [Kat+07] Akrivi Katifori u. a. »Ontology visualization methods - a survey«. In: *ACM Comput. Surv.* 39 (4 Nov. 2007). ISSN: 0360-0300. DOI: <http://doi.acm.org/10.1145/1287620.1287621>. URL: <http://doi.acm.org/10.1145/1287620.1287621> (siehe S. 6, 13).
- [KGD10] Mansoor Ahmed Khan, Gunnar Aastrand Grimnes und Andreas Dengel. »Two pre-processing operators for improved learning from Semantic Web data«. In: 2010 (siehe S. 92).
- [KJ10] Anil K. und Jain. »Data clustering: 50 years beyond K-means«. In: *Pattern Recognition Letters* 31.8 (2010). Award winning papers from the 19th International Conference on Pattern Recognition (ICPR) 19th International Conference in Pattern Recognition (ICPR), S. 651 –666. ISSN: 0167-8655. DOI: <10.1016/j.patrec.2009.09.011>. URL: <http://www.sciencedirect.com/science/article/pii/S0167865509002323> (siehe S. 22).
- [KMH01] Robert Kosara, Silvia Miksch und Helwig Hauser. »Semantic Depth of Field«. In: *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*. Washington, DC, USA: IEEE Computer Society, 2001, S. 97–. ISBN: 0-7695-1342-5. URL: <http://dl.acm.org/citation.cfm?id=580582.857724> (siehe S. 16).
- [KW10] S. Kriglstein und G. Wallner. »Knoocks - A Visualization Approach for OWL Lite Ontologies«. In: *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*. Feb. 2010, S. 950 –955. DOI: <10.1109/CISIS.2010.55> (siehe S. 38).
- [Loh+10] Steffen Lohmann u. a. »The RelFinder user interface: interactive exploration of relationships between objects of interest«. In: *Proceedings of the 15th international conference on Intelligent user interfaces*. IUI '10. Hong Kong, China: ACM, 2010, S. 421–422. ISBN: 978-1-60558-515-4. DOI: <http://doi.acm.org/10.1145/1719970.1720052>. URL: <http://doi.acm.org/10.1145/1719970.1720052> (siehe S. 20).
- [LOR09] Mia A. Levy, Martin J. O'Connor und Daniel L. Rubin. »Semantic Reasoning with Image Annotations for Tumor Assessment«. In: (2009) (siehe S. 6).
- [Mac67] J. Macqueen. »Some methods for classification and analysis of multivariate observations«. In: *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*. 1967, S. 281–297 (siehe S. 22).
- [Maz09] Ricardo Mazza. *Introduction to Information Visualization*. Springer, 2009 (siehe S. 68).

- [Mot+11] Enrico Motta u. a. »A novel approach to visualizing and navigating ontologies«. In: *The 10th International Semantic Web Conference (ISWC 2011)*. Published in: L. Aroyo et al. (Eds.): ISWC 2011, Part I, LNCS 7031, pp. 470?486, 2011. 2011. URL: <http://oro.open.ac.uk/29913/> (siehe S. 25).
- [NFM01] Natalya Fridman Noy, Ray W. Fergerson und Mark A. Musen. »The knowledge model of Protege-2000: combining interoperability and flexibility«. In: Springer, 2001, S. 17–32 (siehe S. 32).
- [Nie07] Jakob Nielsen. *Breadcrumb Navigation increasingly useful.* abgerufen am 19.11.2011. Apr. 2007 (siehe S. 16).
- [NL93] Jakob Nielsen und Thomas K. Landauer. »A mathematical model of the finding of usability problems«. In: *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems.* CHI '93. Amsterdam, The Netherlands: ACM, 1993, S. 206–213. ISBN: 0-89791-575-5. DOI: <http://doi.acm.org/10.1145/169059.169166>. URL: <http://doi.acm.org/10.1145/169059.169166> (siehe S. 52).
- [NM90] Jakob Nielsen und Rolf Molich. »Heuristic evaluation of user interfaces«. In: *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people.* CHI '90. Seattle, Washington, United States: ACM, 1990, S. 249–256. ISBN: 0-201-50932-6. DOI: <http://doi.acm.org/10.1145/97243.97281>. URL: <http://doi.acm.org/10.1145/97243.97281> (siehe S. 46).
- [Nor86] D. A. Norman. »Cognitive Engineering«. In: *User Centered System Design.* Hrsg. von D. A. Norman und S. W. Draper. Lawrence Erlbaum Association, 1986, S. 31–61 (siehe S. 14).
- [Pie+09] Stefan Pietschmann u. a. »CRUISe: Composition of Rich User Interface Services«. In: *Web Engineering.* Hrsg. von Martin Gaedke, Michael Grossniklaus und Oscar Díaz. Bd. 5648. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, S. 473–476. ISBN: 978-3-642-02817-5 (siehe S. 2).
- [PL11] Jędrzej Potoniec und Agnieszka Lawrynowicz. »RMonto: Ontological Extension to RapidMiner«. In: (2011) (siehe S. 87, 92).
- [PMd08] Silvio Peroni, Enrico Motta und Mathieu d'Aquin. »Identifying Key Concepts in an Ontology, through the Integration of Cognitive Principles with Statistical and Topological Measures«. In: *The Semantic Web.* Hrsg. von John Domingue und Chutiporn Anutariya. Bd. 5367. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, S. 242–256. ISBN: 978-3-540-89703-3. URL: [http://dx.doi.org/10.1007/978-3-540-89704-0\\_17](http://dx.doi.org/10.1007/978-3-540-89704-0_17) (siehe S. 25).
- [Pop+11] Igor Popov u. a. »Connecting the Dots: A Multi-pivot Approach to Data Exploration«. In: 2011 (siehe S. 19, 20).

- [PPM04] Ted Pedersen, Siddharth Patwardhan und Jason Michelizzi. »WordNet: similarity - measuring the relatedness of concepts«. In: *Proceedings of the 19th national conference on Artificial intelligence*. AAAI'04. San Jose, California: AAAI Press, 2004, S. 1024–1025. ISBN: 0-262-51183-5. URL: <http://dl.acm.org/citation.cfm?id=1597148.1597310> (siehe S. 79).
- [Ran64] Shiyali Ramamrita Ranganathan. *Colon Classification: basic classification*. Madras Library Association, 1964 (siehe S. 18).
- [RC94] Ramana Rao und Stuart K. Card. »The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information«. In: *Conference companion on Human factors in computing systems*. CHI '94. Boston, Massachusetts, United States: ACM, 1994, S. 222–. ISBN: 0-89791-651-4. DOI: <http://doi.acm.org/10.1145/259963.260391>. URL: <http://doi.acm.org/10.1145/259963.260391> (siehe S. 16).
- [RG95] Thomas R. und Gruber. »Toward principles for the design of ontologies used for knowledge sharing?« In: *International Journal of Human-Computer Studies* 43.5-6 (1995), S. 907 –928. ISSN: 1071-5819. DOI: [10.1006/ijhc.1995.1081](https://doi.org/10.1006/ijhc.1995.1081). URL: <http://www.sciencedirect.com/science/article/pii/S1071581985710816> (siehe S. 4).
- [RL78] Eleanor Rosch und Barbara B. Lloyd. *Cognition and Categorization*. John Wiley & Sons Inc, 1978. ISBN: 0470263776. URL: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0470263776> (siehe S. 26).
- [RM93] George G. Robertson und Jock D. Mackinlay. »The document lens«. In: *Proceedings of the 6th annual ACM symposium on User interface software and technology*. UIST '93. Atlanta, Georgia, United States: ACM, 1993, S. 101–108. ISBN: 0-89791-628-X. DOI: <http://doi.acm.org/10.1145/168642.168652>. URL: <http://doi.acm.org/10.1145/168642.168652> (siehe S. 16).
- [Roy+08] LA Royer u. a. »Unraveling Protein Networks with Power Graph Analysis«. In: *PLoS Comput Biol* 4.7 (Juli 2008), e1000108. DOI: [10.1371/journal.pcbi.1000108](https://doi.org/10.1371/journal.pcbi.1000108). URL: <http://dx.plos.org/10.1371/2Fjournal.pcbi.1000108> (siehe S. 12).
- [Sch04] Jean Scholtz. »Usability Evaluation«. In: (2004) (siehe S. 45).
- [Sch90] Roger W. Schvaneveldt, Hrsg. *Pathfinder associative networks: studies in knowledge organization*. Norwood, NJ, USA: Ablex Publishing Corp., 1990. ISBN: 0-89391-624-2 (siehe S. 9).
- [SF11] A. Spritzer und C. Freitas. »Design and Evaluation of MagnetViz - A Graph Visualization Tool«. In: *Visualization and Computer Graphics, IEEE Transactions on* PP.99 (2011), S. 1. ISSN: 1077-2626. DOI: [10.1109/TVCG.2011.106](https://doi.org/10.1109/TVCG.2011.106) (siehe S. 19).

- [Shn92] Ben Shneiderman. »Tree visualization with tree-maps: 2-d space-filling approach«. In: *ACM Trans. Graph.* 11 (1 Jan. 1992), S. 92–99. ISSN: 0730-0301. DOI: <http://doi.acm.org/10.1145/102377.115768>. URL: <http://doi.acm.org/10.1145/102377.115768> (siehe S. 11).
- [Shn96] B. Shneiderman. »The eyes have it: a task by data type taxonomy for information visualizations«. In: *Visual Languages, 1996. Proceedings., IEEE Symposium on*. 1996, S. 336–343. DOI: [10.1109/VL.1996.545307](https://doi.org/10.1109/VL.1996.545307) (siehe S. 49, 60).
- [Shn98] Ben Shneiderman. *Treemaps for space-constrained visualization of hierarchies*. abgerufen am 12.5.2012. Dez. 1998. URL: <http://www.cs.umd.edu/hcil/treemap-history/> (siehe S. 11).
- [SIM] SIMCluster. *Simulation*. abgerufen am 25.11.2011 (siehe S. 25).
- [Sto+01] Margaret anne Storey u. a. »Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protege«. In: *in Protege. Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*. 2001 (siehe S. 32).
- [STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiro Toda. »Methods for Visual Understanding of Hierarchical System Structures«. In: *Ieee Transactions On Systems Man And Cybernetics* 11.2 (1981), S. 109–125. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4308636> (siehe S. 108).
- [Tie+11] Vincent Tietz u. a. »Towards task-based development of enterprise mashups«. In: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*. iiWAS '11. Ho Chi Minh City, Vietnam: ACM, 2011, S. 325–328. ISBN: 978-1-4503-0784-0. DOI: [10.1145/2095536.2095594](https://doi.org/10.1145/2095536.2095594). URL: <http://doi.acm.org/10.1145/2095536.2095594> (siehe S. 2).
- [Top] TopQuadrant. *TopBraid Suite*. abgerufen am 20.11.2011 (siehe S. 28).
- [Vir92] Robert A. Virzi. »Refining the test phase of usability evaluation: how many subjects is enough?« In: *Hum. Factors* 34 (4 1992), S. 457–468. ISSN: 0018-7208. URL: <http://dl.acm.org/citation.cfm?id=141691.141700> (siehe S. 52).
- [VPM12] Martin Voigt, Stefan Pietschmann und Klaus Meißner. »Towards a Semantics-Based, End-User-Centered Information Visualization Process«. In: *Proc. of the 3rd International Workshop on Semantic Models for Adaptive Interactive Systems (SEMAIS 2012)*. 2012 (siehe S. 2, 83).
- [W3C04a] W3C. *OWL Web Ontology Language Reference*. abgerufen am 12.11.2011. Feb. 2004. URL: <http://www.w3.org/TR/owl-ref/> (siehe S. 6).
- [W3C04b] W3C. *RDF Vocabulary Description Language 1.0: RDF Schema*. abgerufen am 11.11.2011. Feb. 2004. URL: <http://www.w3.org/TR/rdf-schema/> (siehe S. 6).
- [W3C04c] W3C. *RDF/XML Syntax Specification (Revised)*. abgerufen am 10.11.2011. Feb. 2004. URL: <http://www.w3.org/TR/REC-rdf-syntax/> (siehe S. 5).

- [W3C09] W3C. *OWL 2 Web Ontology Language Quick Reference Guide*. abgerufen am 12.11.2011. Okt. 2009 (siehe S. 6).
- [W3C11] W3C. *Web Notification API*. abgerufen am 10.5.2012. März 2011 (siehe S. 90).
- [Wha+94] Cathleen Wharton u. a. »The cognitive walkthrough method: a practitioner's guide«. In: New York, NY, USA: John Wiley & Sons, Inc., 1994, S. 105–140. ISBN: 0-471-01877-5. URL: <http://dl.acm.org/citation.cfm?id=189200.189214> (siehe S. 46).
- [WP06] Taowei Wang und Bijan Parsia. »CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies«. In: *The Semantic Web - ISWC 2006*. Hrsg. von Isabel Cruz u. a. Bd. 4273. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, S. 695–708. ISBN: 978-3-540-49029-6. URL: [http://dx.doi.org/10.1007/11926078\\_50](http://dx.doi.org/10.1007/11926078_50) (siehe S. 11).
- [XW05] Rui Xu und II Wunsch D. »Survey of clustering algorithms«. In: *Neural Networks, IEEE Transactions on* 16.3 (Mai 2005), S. 645–678. ISSN: 1045-9227. DOI: [10.1109/TNN.2005.845141](https://doi.org/10.1109/TNN.2005.845141) (siehe S. 22).
- [Yee+03] Ka-Ping Yee u. a. »Faceted metadata for image search and browsing«. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. CHI '03. Ft. Lauderdale, Florida, USA: ACM, 2003, S. 401–408. ISBN: 1-58113-630-7. DOI: [10.1145/642611.642681](https://doi.acm.org/10.1145/642611.642681). URL: <http://doi.acm.org/10.1145/642611.642681> (siehe S. 18).