Institut für Informatik, Freie Universität Berlin

Arbeitsgruppe Software Engineering

Masterarbeit

Eingereicht am 25.09.2014

Entwicklung einer Architektur zur datenflussorientierten Verarbeitung von Daten/Metadaten-Verbänden auf Basis von KNIME

Alexander Niemeier

Matrikelnummer: 4519198

aniemeier@gmx.de

Erstgutachterin: Prof. Dr.-Ing. Ina Schieferdecker (Freie Universität Berlin)

Zweitgutachter: Prof. Dr.-Ing. Edzard Höfig (Beuth Hochschule für Technik Berlin)

I. Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 25. September 2014

II. Abstract

Diese Thesis beschäftigt sich mit der datenflussorientierten Verarbeitung von Daten/Metadaten-Verbänden auf Basis von KNIME. Um eine solche Verarbeitung in KNIME zu ermöglichen, wurden Knoten entwickelt, die eine Integration von Metadaten ermöglichen. Dazu wurde der Umgang mit RDF-Dateien implementiert. Da das Internet sich als hervorragende Datenquelle für Metadaten anbietet und vor allem das Semantic Web sehr bestrebt ist die Daten strukturiert und vernetzt zur Verfügung zu stellen, wurde auch ein SPARQL-Knoten implementiert. Dieser Knoten kann Metadaten von der DBpedia abfragen und in das KNIME-Tabellenformat umwandeln. In dieser Arbeit werden auch mehrere Varianten der Verknüpfung, Verarbeitung und Trennung der Daten/Metadaten-Verbände vorgestellt und diskutiert. Umgesetzt wurde die synchrone Verarbeitung mit nur einem Datenstrom, was durch Datenfusion ermöglicht wird. Alle entwickelten Knoten werden vorgestellt und anhand von realistischen Benutzerszenarien validiert.

III. Danksagung

Danken möchte ich in erster Linie meinem Betreuer, Herrn Prof. Dr. Höfig, für seine ausgiebige Unterstützung. Durch stetig kritisches Hinterfragen und konstruktive Kritik verhalf er mir zu einer durchdachten These und Fragestellung. Dank seiner herausragenden Expertise konnte er mich immer wieder in meiner Recherche und bei meinen Fragen unterstützen. Die entsendende Datenbasis wäre ohne sein fachliches Wissen nicht so solide geworden. Vielen Dank für Zeit und Mühen, die Sie, Herrn Prof. Dr. Höfig, in meine Arbeit investiert haben.

Auch will ich mich bei Sebastian Riedel, Robert Delbrück und Horst Niemeier bedanken, die Zeit in die Korrektur meiner Arbeit investiert haben. Einige Kommata, Satzstellungen und Rechtschreibfehler flogen dank ihrer Hilfe hinaus oder auch hinein.

Einen besonderen Dank gilt meinen Eltern, die mich nicht nur während der Schulzeit sondern auch während des Studiums so herzlich unterstützt haben.

Zu guter Letzt möchte ich meiner Freundin, Amrei Kuhn, dafür danken, dass Sie mich vor allem während des Endspurts seelisch unterstützt hat.

IV. Inhaltsverzeichnis

I. Eidesstattliche Erklärung I

II. Abstract II

III. Danksagung III

IV. Inhaltsverzeichnis IV

1. Einführung 1

1.1 Motivation 2

1.2 Zielsetzung und Problemstellung 2

1.3 Methodik 3

1.4 Struktur der Arbeit 4

2. Grundlagen 5

2.1 KNIME 5

2.2 Metadaten 10

2.3 Semantic Web 12

2.4 RDF & Turtle 13

2.5 SPARQL 17

2.6 DBpedia 19

2.7 Stand der Forschung 20

3. Architektur 22

3.1 Datenfluss-Architektur von KNIME 22

3.2 Erweiterung der Architektur für die gemeinsame Verarbeitung von Daten und Metadaten 24

4. Konzeption der KNIME-Knoten 30

4.1 Integration von RDF in KNIME 30

4.2 Search-Engine zum Lokalisieren der IRIs 32

4.3 Generierung von SPARQL-Anfragen 36

4.4 Zusammenfassung der zu implementierenden Knoten 40

5. Implementierung & Validierung 41

5.1 Entwicklungsumgebung 41

5.2 Validierung durch Benutzerszenarien 42

5.3 Sonstige Knoten 57

6. Ergebnisse 59

6.1 Zusammenfassung 59

6.2 Kritische Betrachtung 60

6.3 Fazit 61

6.4 Ausblick 62

7. Anhänge 63

7.1 Literaturverzeichnis 63

7.2 Quellcode & Workflows 66

7.3 Bilder 67

1. Einführung

Seit der steigenden Bedeutung von kommunikativen Netzwerken in den Achtzigern und dem daraus folgenden Beginn des sogenannten Informationszeitalters, stieg die Anzahl der zugänglichen Daten stetig an. Über das Internet senden und empfangen Menschen auf der ganzen Welt Informationen. Viele Informationssammlungen sind entstanden, wie etwa YouTube („Pro Minute werden 100 Stunden Videomaterial auf YouTube hochgeladen“ [1]) oder Wikipedia (über 30 Millionen Artikel in 287 Sprachen [2]).

Aufgrund der Menge an Informationen rückt ein immer wichtiger werdendes Szenario in den Vordergrund: die Datenanalyse. Um von den reinen Daten bis zu dem gewünschten Ergebnis zu kommen, bedarf es häufig mehrerer Arbeitsschritte, wie etwa Bereinigungen und Filterungen der Daten, sowie die Visualisierung der Ergebnisse. Den kompletten Vorgang von den Daten bis zum Ergebnis nennt man Workflow.

Eine Open-Source-Software mit der solche Workflows erstellt werden können, wurde von der Uni Konstanz entwickelt und hat den Namen Konstanz Information Miner (KNIME) [3]. Als Informationsquelle werden lokale Dateien oder Datenbanken eingelesen. Diese Informationen werden in Tabellenform dargestellt und können nun weiterverarbeitet werden. Die einzelnen Verarbeitungsschritte (z.B. filtern, löschen, berechnen) werden in Knoten definiert. Diese Knoten haben Ports für die eingehenden und ausgehenden Datenströme. Die Ergebnisse sind meist Visualisierungen (Graph), sowie die Speicherung in Dateien oder Datenbanken.

Neben den eigentlichen Daten gibt es noch eine übergeordnete Informationsschicht, die Meta-Daten genannt wird. Häufig sind dies Informationen, welche die Daten noch genauer beschreiben. Beispiel: Bei einer Musikdatei ist das Lied selbst die Information. Die Metadaten davon wären unter anderen der Titel, der Komponist und das Label. Wiederum der Komponist kann auch der Ursprung sein und die Metadaten wären dann zum Beispiel das Geburtsdatum, sein Heimatort und seine Anschrift.

In der Datenanalyse werden in einem Workflow lediglich die Daten analysiert oder im Vorfeld die Daten mit den Metadaten vermischt. Eine Verarbeitung von Daten/Metadaten-Verbänden ist bisher nicht vorgesehen. Um dies zu ermöglichen, soll eine Architektur zur datenflussorientierten Verarbeitung von Daten/Metadaten-Verbänden, auf Basis von KNIME, entwickelt werden.

1.1 Motivation

Mit Workflow-Programmen lassen sich viele aufeinanderfolgende Arbeitsschritte miteinander verbinden und ausführen. Die Quelle lässt sich austauschen, einzelne Arbeitsschritte können geändert, gelöscht oder neue hinzugefügt werden und die Ergebnisse lassen sich ebenfalls variabel gestalten. Sobald ein Workflow angelegt ist, lässt sich dieser wiederholt ausführen. Je nach Datenquelle, können die Ergebnisse rekonstruiert werden (bei statischen Daten) oder situationsbedingt neu erzeugt werden (bei dynamische Daten oder Livedaten).

Der Einsatz von KNIME ist aufgrund seines offenen Quellcodes, seiner Erweiterbarkeit und seiner Kompatibilität, für die Umsetzung dieser Abschlussarbeit geradezu prädestiniert. Für die Entwicklung und das Erstellen der Workflows wird lediglich Eclipse mit dem KNIME-Plugin und Java benötigt.

Durch die Entwicklung von Knoten, welche Daten/Metadaten-Verbände interpretieren und verarbeiten können, werden neue Möglichkeiten der synchronen Verarbeitung geschaffen. Außerdem wird durch die Integration vom Resource Descriptopn Framework (RDF) - Format und der SPARQL Protocol And RDF Query Language (SPARQL) - Schnittstelle der Umfang der Software sinnvoll erweitert.

1.2 Zielsetzung und Problemstellung

Als Grundlage dieser Arbeit dient die Open-Source Software KNIME, welche von der Uni Koblenz entwickelt wurde, um den Nutzer bei der Daten-Analyse zu unterstützen. Voraussetzung um überhaupt eine Daten-Analyse ausführen zu können, ist das Vorhandensein von Daten und das Einlesen dieser in die Software. Die Daten sind meist lokal in Dateien gespeichert und können mit diversen Readern (unter anderen comma-separated values- (csv), File- und Table Reader) eingelesen werden. Auch über ein Database Reader können die Daten eingelesen werden, sollten sich diese in einer Datenbank befinden.

Ziel dieser Arbeit ist es, dem Benutzer zu ermöglichen seine Daten um Meta-Daten zu erweitern, diese Daten/Metadaten-Verbände synchron zu verarbeiten und gegebenenfalls wieder zu trennen.

Eine umfangreiche Quelle an Meta-Daten bietet einem das Internet. Dank des Konzeptes des semantischen Web (englisch Semantic Web) werden viele Informationen über Personen, Orte und Dinge mit einer einheitlichen Semantik gespeichert und miteinander in Beziehungen gesetzt. Diese Informationen werden von mehreren Anbietern zur Verfügung gestellt, wie zum Beispiel DBpedia [4] (Wikipedia) oder GeoNames [5]. Vorzufinden sind die Daten im RDF-Format und sind über eine SPARQL-Schnittstelle abrufbar.

Die Integration der Meta-Daten in KNIME ist problematisch, da keine Knoten für RDF-Dateien und SPARQL-Abfragen vorhanden sind. Auch das Ver- und Entknüpfen der Daten und Meta-Daten ist bisher nicht vorgesehen.

1.3 Methodik

Die Vorgehensweise des Projekts ähnelt dem Wasserfallmodel und durchläuft folgende Phasen:

1. Identifikation

Als erstes muss ein Grundverständnis für KNIME und die privilegierten Dateiformate (RDF), Schnittstellen (SPARQL) und der Datenquelle (DBpedia) geschaffen werden. Das Ziel dieses Schrittes ist es, zu identifizieren, wie die KNIME-Architektur aufgebaut ist und welche vorhandenen Technologien verwendet und integriert werden können.

2. Konzeption

In der Konzeptionsphase soll ermittelt werden, wie die KNIME-Architektur verwendet und erweitert werden kann, damit die Integration der neuen Knoten und das Erzeugen, Verarbeiten und Trennen der Daten/Metadaten-Verbände, erfolgreich umgesetzt werden kann. Das Ziel dieser Phase ist es, die gewünschten Knoten zu konzeptionieren und die Implementierung derer in die KNIME-Architektur aufzuzeigen und sich für eine architektonische Anpassung zu entscheiden.

3. Implementation

Nachdem die Planungen abgeschlossen sind, gilt es, die erarbeiteten Knoten in KNIME zu integrieren. Das Ziel des dritten Schrittes ist es, die Technologien mit den notwendigen Algorithmen in die neu zu entwickelten KNIME-Knoten zu implementieren und in die präferierte Architektur zu integrieren.

4. Überprüfung

Die implementierten Knoten werden auf ihre Richtigkeit und auch Einsatztauglichkeit überprüft. Das Ziel des letzten Schrittes ist es, anhand von realistischen Benutzerszenarien die Richtigkeit der einzelnen Knoten festzustellen und zu Demonstrieren.

1.4 Struktur der Arbeit

Die Abschlussarbeit ist unterteilt in sechs Kapitel:

Kapitel 1 (Einführung) gibt eine Einführung in das Thema, geht auf die Motivation, Zielsetzung und Problematik ein, beschreibt die Vorgehensweise und wie diese Arbeit strukturell Aufgebaut ist.

Kapitel 2 (Grundlagen) vermittelt das Basiswissen, das notwendig ist, um das Grundverständnis zu steigern. Dazu wird beschrieben was KNIME, Metadaten, Semantik Web, RDF, SPARQL und DBpdia sind und warum diese verwendet werden. Im letzten Teil des Kapitels wird auf den Stand der Forschung eingegangen und wie mit der Verarbeitungen von Daten und Metadaten umgegangen wird.

Kapitel 3 (Architektur) fokussiert die aktuelle KNIME-Architektur und geht genauer auf die einzelnen Bestandteile eines Knotens, auf die KNIME-Tabellenstruktur und die Ausführungs-Engine ein. Im zweiten Teil des Kapitels werden drei mögliche Architekturanpassungen für das Erzeugen und Trennen von Daten/Metadaten-Verbänden vorgestellt und diskutiert.

Kapitel 4 (Konzeption der KNIME-Knoten) befasst sich mit den zu entwickelnden Knoten, welche notwendig sind, um RDF in KNIME zu integrieren, Daten/Metadaten-Verbände zu erzeugen und zu trennen (basierend auf die Erkenntnisse aus Kapitel 3), IRIs mit Hilfe einer Search-Engine zu ermitteln und Online-Abfragen via SPARQL zu ermöglichen.

Kapitel 5 (Implementierung & Validierung) thematisiert die eigentliche Implementierung dieser Thesis. Im ersten Teil wird die Entwicklungsumgebung vorgestellt. Anschließend werden anhand mehrerer realitätsnahen Benutzerszenarien, Workflows erstellt, welche die entwickelten Knoten enthalten und die korrekte Funktionalität aufzeigen sollen.

Kapitel 6 (Ergebnisse) fasst das Wesentliche dieser Abschlussarbeit noch einmal zusammen, betrachtet das gesamte Projekt noch einmal aus einer kritischen Perspektive und resümiert über die gewonnen Erkenntnisse. Abschließend wird ein Ausblick gegeben, was mit dem Projekt nach dieser Thesis passiert und welche Weiterentwicklungsmöglichkeiten bestehen.

2. Grundlagen

In diesem Kapitel wird das Basiswissen, was notwendig ist um das Grundverständnis zu steigern, vermittelt. Dazu wird beschrieben was KNIME, RDF, SPARQL, Metadaten und das Semantik Web sind, warum diese verwendet werden und wie deren Popularität ist.

2.1 KNIME

„Data Mining Workbench for Interactive Data Exploration“ [7]

Der Konstanz Information Miner (KNIME) ist eine Open-Source-Software, welche insbesondere zur Datenanalyse entwickelt wurde. Die Grundidee entstand 2004 im Silicon Valley und wurde an der Universität Konstanz unter der Leitung von Herrn Prof. Dr. Michael Berthold ausgearbeitet und entwickelt. Die erste Version von KNIME erschien 2006 und fand schnell anklang, vor allem im Pharmabereich stieg die Beliebtheit der Software. Im Juni 2008 wurde die KNIME.com GmbH [3] in Zürich gegründet, welche sich um den Support kümmert. Entwickelt wird KNIME in Java, wobei Robustheit, Modularität und hohe Skalierbarkeit in den Vordergrund gestellt wird. Durch die Kombination aus Open-Source (unterliegt der GPL-Lizenz 2.1) und der Modularität, ist die Software hervorragend für die Weiterentwicklung geeignet. Als Entwicklungsumgebung wird Eclipse empfohlen, da KNIME dafür ein Plugin zur Verfügung stellt. [vgl. 6]

Benutzeroberfläche & Workflows

Wer KNIME startet findet einen vollwertigen Editor vor, welcher hauptsächlich mit der Maus via Drag-and-Drog gesteuert wird. Vom Aufbau der Benutzeroberfläche ähnelt es der Eclipse-Oberfläche. In der Mitte befindet sich das Bearbeitungsfenster, in dem der sogenannte Workflow kreiert wird.

Workflow heißt auf Deutsch Arbeitsablauf und definiert eine Abfolge von Aktivitäten. In KNIME bestehen die Aktivitäten aus Algorithmen und werden durch einen Knoten repräsentiert. Die Reihenfolge wird durch das Aneinanderreihen dieser Knoten bestimmt. Die Verbindungen zwischen den Knoten werden als Assoziationen oder Kanten bezeichnet. Der komplette Verlauf vom ersten bis zum letzten Knoten wird Workflow genannt, dabei spielt die Anzahl der Knoten keine Rolle. Um die Funktionalität von KNIME zu erweitern, können neue Knoten entwickelt, eingebunden und in den Workflow eingebaut werden.

Auf der linken Seite des Editors befinden sich der KNIME Explorer und das Node Repository. Im Explorer werden alle vorhandenen Workflows angezeigt. Dies können neben lokalen Workflows auch welche sein, die sich auf einem KNIME-Server befinden. Beispielworkflows können somit auch über den publicserver.knime.org:47037 bezogen werden. Des Weiteren können auch KNIME Server für private und kommerzielle Zwecke entgeltlich gemietet werden,

auf dem die Workflows dann gespeichert und aufgeführt werden könne. Das Node Repository enthält alle Knoten die KNIME zur Verfügung stellt. Diese sind in mehrere Bereiche untergliedert, wie etwa IO, Data Manipulation oder Statistics, so dass die gesuchten Knoten schnell gefunden werden können. Noch schneller kann ein Knoten über das Filterfeld gefunden werden. Für die Verwendung eines Knotens muss dieser lediglich in das Bearbeitungsfenster gezogen werden.

Auf der rechten Seite befindet sich die Node Description. Sobald ein Knoten angeklickt wird, wird dieser hier ausführlich beschrieben.

Im unteren Bereich befindet sich das Outline Fenster, in dem der komplette Workflow minimiert angezeigt wird. Daneben ist die KNIME Console, welche für Hinweise, Fehlermeldungen und Statusausgaben verwendet wird, um den Benutzer über aktuelle Geschehnisse oder auch über Fehler in der Benutzung zu informieren.

Knoten, Ports und Assoziationen

Gegliedert ist das Programm in Knoten, Ports und Assoziationen, wobei jeder Knoten einen festen Algorithmus implementiert bekommen hat. Diese können in fünf Kategorien unterteilt werden: Einlese-, Umwandlungs-, Analyse-, Visualisierungs- und Ausgabeknoten. Die Einleseknoten (Reader) sind für das Einlesen und Integrieren von Daten zuständig. Diese Daten werden in KNIME als Tabellen dargestellt, mit Kopfzeile und einer ID-Spalte. Mit den Umwandlungsknoten können diese Daten in die gewünschten Formate umgewandelt oder auch gruppiert oder gefiltert werden. Die Analyse der Daten wird mit Hilfe der Analyse- und Data Mining-Knoten umgesetzt. Das Ende der Workflows bilden die Visualisierungsknoten und die Ausgabeknoten. Damit lassen sich Bilder oder Diagramme erstellen, aber auch Dateien wie etwa CSV.

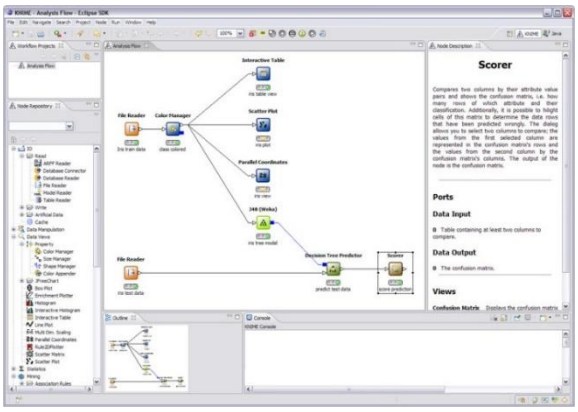

Abbildung 1: Aussehen des KNIME-Editor [8]

Jeder Knoten hat Ports, welche entweder Ein- oder Ausgangsports sind. An einem Eingangsport wird eine Assoziation eines anderen Knotens erwartet (benötigt eine Datenquelle). Einleseknoten besitzen meistens keine Eingangsports, aber auf jeden Fall einen Ausgangsport. An einem Ausgangsport werden die Ergebnisse vom Knoten-Algorithmus zur Verfügung gestellt. An diesem können Knoten mit Eingangsport angeschlossen werden. Visualisierung- und Ausgabeknoten besitzen meistens keine Ausgangsports, aber auf jeden Fall Eingangsports. Alle Knoten, welche die Daten manipulieren oder zur Analyse verwendet werden, haben beide Arten von Ports. Außerdem ist es möglich, dass ein Knoten mehrere Ein- und Ausgangsports hat, wobei die Ausgangsports verschiedene Ergebnisse liefern. An einem Ausgangsport können

mehrere Assoziationen sein, somit können mehrere Verarbeitungen, welche auf den gleichen Daten basieren, in einem Workflow bearbeitet werden.

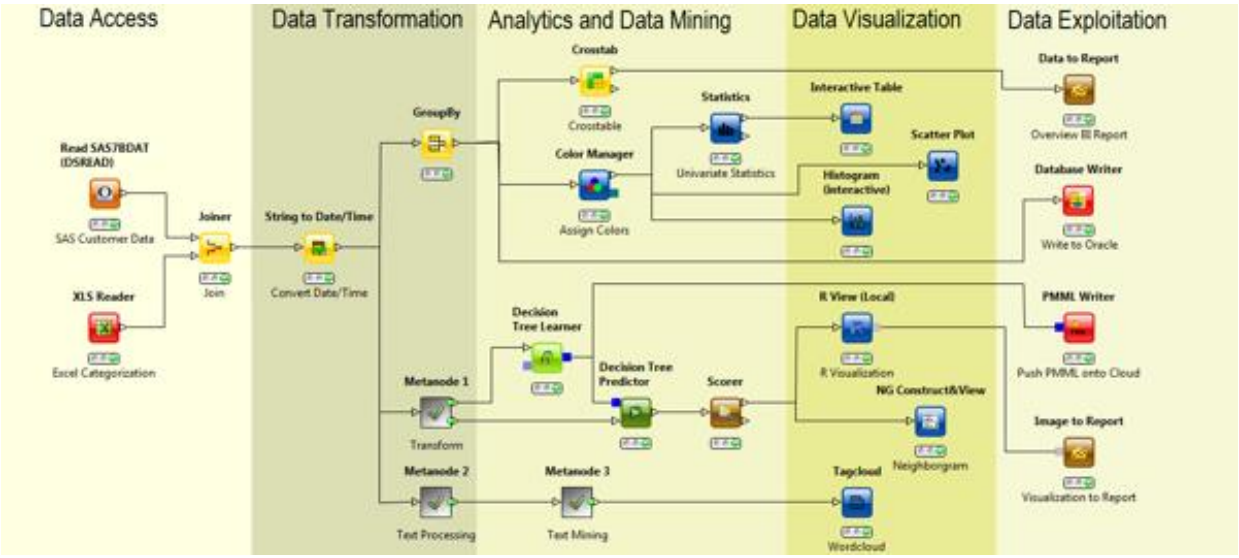

Abbildung 2: Beispielworkflow, mit Einteilung der Knoten in die verschiedenen Knotentypen [9]

Im dritten Kapitel wird vertieft auf die Architektur und die Ausführungssemantik von KNIME eingegangen.

Data-Mining Studie

Professor Dr. Markus Lusti, emeritierter Extraordinarius für Informatik, von der Universität Basel hat Data Mining wie folgt beschrieben: „Ziel des Data Mining ist die automatische und nicht-triviale Suche nach Wissen in Massendaten.“ [12] Demnach bezieht sich das Data-Mining im eigentlichen Sinne nur auf den Analyseschritt des Prozesses. In der folgenden Umfrage wird aber der komplette Workflow mit berücksichtigt, also auch die Erfassung, Speicherung und Verarbeitung der Daten.

Das Analyse- und CRM-Beratungshaus Rexer Analytics [10] führt seit 2007, jährlich (mit Ausnahme von 2012) eine Data-Mining Studie durch. Die aktuellste Studie aus dem Jahr 2013 [11] ist mit 1259 professionellen Data-Mining-Anwendern aus 75 Ländern die größte Umfrage seiner Art. Dabei werden alle Analysetools betrachtet, welche im Bereich des Data-Mining zum Einsatz kommen.

Von den 22 Tools, die in dieser Studie untersucht wurden, haben 8% aller Teilnehmer KNIME als primäres Analysetool gewählt. Nur R [19], STATISTICA [13], Rapid Miner [16] und SAS [45] haben mehr Hauptbenutzer. Während R mit 24% einen großen Vorsprung vor dem Zweitbeliebtesten Tool hat (STATISTICA mit 12%), sind die nächsten fünf Tools dicht beieinander. [11, Folie 29] Bei der Zufriedenheit der Anwender, kann KNIME mit 96% zufriedenen bis sehr zufriedenen Kunden den zweiten Rang hinter STATISTICA einnehmen. Unzufriedene Nutzer gab es keine. [11, Folie 32].

	Overall	IBM SPSS Statistics	IBM SPSS Modeler	KNIME	R	Rapid Miner	SAS	SAS Enterprise Miner	STATISTICA	Weka
Quality and accuracy of model performance	4.28	3.96	4.15	4.30	4.39	4.25	4.20	4.48	4.62	4.16
Variety of available algorithms	4.27	3.66	4.05	4.36	4.74	4.55	3.91	4.23	4.59	4.46
Data manipulation capabilities	4.19	3.91	4.36	4.54	4.24	4.07	4.50	3.74	4.52	3.48
Dependability/Stability of software	4.19	4.02	3.96	4.27	4.24	4.07	4.28	4.16	4.51	4.03
Ability to automate repetitive tasks	4.18	3.79	3.76	4.42	4.35	4.18	4.26	4.10	4.44	3.76
Quality of output / Ease of interpretation	4.11	3.87	3.89	4.17	4.10	4.18	3.84	4.10	4.59	3.82
Ease of use	4.11	4.10	4.67	4.58	3.59	4.39	3.77	4.27	4.58	4.03
Good metrics of model quality	4.08	3.72	3.89	3.91	4.19	4.17	4.01	4.17	4.50	4.06
Data quality assessment & data preparation capabilities	4.05	3.72	4.27	4.37	4.02	4.00	4.26	3.77	4.41	3.47
Ability to easily incorporate data at different levels of granularity (e.g. transaction data and customer data)	4.03	3.87	4.25	4.21	3.94	4.04	4.14	4.10	4.30	3.59
Cost of software	4.03	3.02	2.89	4.85	4.93	4.86	2.33	2.70	3.91	4.89
Ability to modify algorithm options to fine-tune analyses	4.01	3.26	3.63	3.80	4.35	4.10	3.91	3.94	4.28	4.18
Good variable discovery, profiling and selection	4.00	3.64	4.16	4.07	4.03	4.06	3.78	4.23	4.42	3.77
Quality of user interface	3.97	4.02	4.47	4.54	3.49	4.37	3.66	4.10	4.53	3.54
Ease of model deployment (scoring to other data sets)	3.97	3.42	4.01	4.21	3.87	4.19	3.92	4.00	4.43	3.75
Speed	3.95	3.62	4.01	4.00	3.69	3.95	3.93	3.97	4.54	3.70
Enables mining within one's database	3.92	3.59	4.18	4.08	3.92	3.83	3.78	3.93	4.26	3.69
Ability to handle very large data sets	3.84	3.65	4.19	3.90	3.27	3.59	4.35	4.30	4.56	3.18
Strong graphical visualization of models	3.83	2.94	3.60	3.90	4.14	4.01	3.09	3.77	4.58	3.38
Useful help menu, demos and tutorials	3.82	3.87	3.82	4.05	3.86	3.54	3.67	3.90	4.23	3.50

Mean satisfaction rating on 1-5 scale Higher Satisfaction Lower Satisfaction

Abbildung 3: Umfrageergebnis zur Ermittlung der Benutzerzufriedenheit, untergliedert in zwanzig verschiedene Kriterien [11, Folie 33]

Bei den Möglichkeiten der Datenmanipulation, der einfachen Bedienbarkeit, den Kosten und der Qualität der Benutzeroberfläche, gehört KNIME zu den Spitzenreitern. In allen anderen Bereichen liegt die Software über dem Durchschnitt. Lediglich die Kommerzielle Software STATISTICA erreicht häufiger bessere Werte als KNIME.

Alternative: RapidMiner

Die Firma RapidMiner [16] wurde 2007 unter dem Namen Rapid-I in Dortmund gegründet. Heute besteht das Unternehmen aus 40 Entwicklern, 35 Partnern, 500 Enterprise Kunden und 200.000 aktiven Nutzern. Die Umbenennung wurde im November 2013 durchgeführt und aus der gleichnamigen RapidMiner - Open Source Software wurde eine kommerzielle Software [vgl. 17].

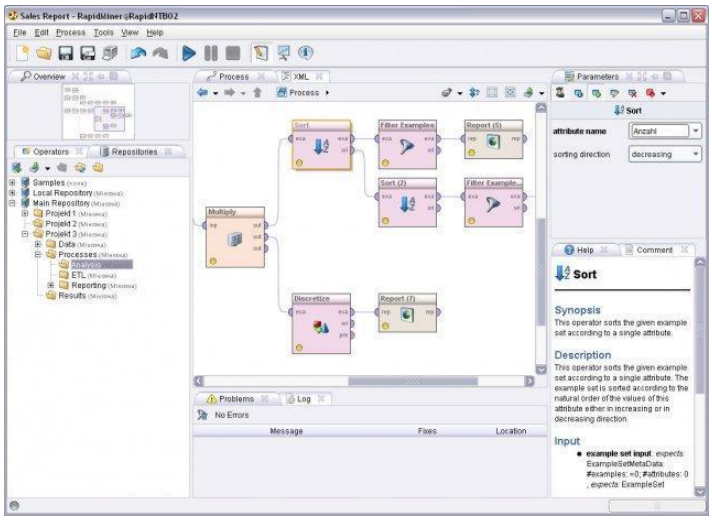

Abbildung 4: Aussehen des RapidMiner-Editor [21]

Durch die Kommerzierung der Software ist es schwieriger geworden sich an der Entwicklung von neuen Knoten zu beteiligen. Die Starter-Version der Software ist noch kostenfrei, aber der Nutzer wird stark eingeschränkt, so kann nur noch mit CSV und Excel-Dateien gearbeitet werden.

Wer die 999$ bezahlt, bekommt eine gute Software mit Zugriff auf allen Knoten. Der Editor ähnelt dem vom KNIME: In der Mitte ist der Bearbeitungsbereich, auf der linken Seite sind das Repository und die Knoten zu finden und auf der rechten Seite befinden sich die Beschreibungen. Auf der rechten Seite befindet sich ein Bereich, wo die Parameter der Knoten angegeben werden können, was vergleichbar mit der Konfiguration von KNIME ist. Darunter ist sind die Knoteninformationen zu finden. Auch hier lassen sich unten die Logs, Probleme und auch eine Miniaturansicht anzeigen.

Anders als in KNIME werden die Workflows beim RapidMiner als Process bezeichnet. Ein zusätzliches Feature gegenüber KNIME, bietet die Möglichkeit, Metadaten für Tabellen und Strings anzulegen. Diese werden bei der Tabellen-View dann über der eigentlichen Tabelle angezeigt und bei den Strings werden dann die Metadaten als Tooltip bei der jeweiligen Zelle mit angezeigt. Diese Metadaten sind aber nur für den Benutzer und können nicht in der Tabelle integriert werden.

Fazit

Seit der Kommerzierung des RapidMiner ab der Version 6.0, ist KNIME vom Leistungsumfang, der Popularität und der Performance unter den kostenfreien Open-Source-Programmen fast konkurrenzlos geworden. Das Data-Mining Tool R (Programmiersprache für statistisches Rechnen und statistische Grafiken) [19] hat sich auf einen festen Bereich festgelegt und bietet nicht die Möglichkeit mit Workflows zu arbeiten. Dieses Tool wird aber von KNIME unterstützt, indem auch für diese Technologie KNIME-Knoten zur Verfügung stehen. Seit der Version 2.7 haben KNIME-Nutzer zudem die Möglichkeit Rapid-Knoten (aus der Version 5.3) in KNIME zu integrieren [18]. Auch für Weka [20], was von der University of Waikato entwickelt wurde und ebenfalls wie KNIME mit Workflows arbeitet und als Open Source Software frei zugänglich ist, aber in der Umfrage viel schlechter Abschnitt, werden Knoten in KNIME zur Verfügung gestellt.

Wer auch kommerzielle Software in Betracht zieht, der kann auch folgende Anwendungen zur Data-Analyse einsetzen:

- STATISTICA Data Miner von StatSoft – Teil von Dell [13]
- SPSS Modeler von IBM [14]
- SAS Enterprise Miner von SAS Institute [15]

Vor allem STATISTICA, ist laut der Studie in vielen Punkten KNIME gegenüber überlegen, schlägt aber mit einem Preis von ca. 950€ für eine Basislizenz zu Buche.

2.2 Metadaten

> Metadaten sind strukturierte Daten zur einheitlichen Beschreibung von Ressourcen jeglicher Art (z. B. Daten, Dokumente, Personen, Gemälde, Orte, Gebäude, Konzepte). Sie sorgen dafür, dass den Beschreibungen von Instanzen ein- und derselben Ressourcenklasse eine einheitliche Struktur zugrunde liegt und erleichtern so das Suchen, Finden und Selektieren relevanter Ressourcen aus einer Vielzahl möglicher Ressourcen. [30]

Viele, die im Internet unterwegs sind, kennen Sie, die Metadaten. Doch kaum jemand weiß genau, was sich hinter dem Begriff verbirgt. Zum Beispiel beim Online-Shopping, werden bei fast jedem Kaufobjekt, wie etwa bei einem Pullover, auch Metadaten der Artikel mit angezeigt, also alle Informationen, welche den Pullover beschreiben. Diese wären etwa die Marke, die Farbe, die Größe, die Materialien oder sogar Pflegehinweise. Aber nicht nur beim Online-Shopping begegnen wir den Metadaten, sondern auch bei Bibliotheken, wo die Bücher ebenfalls durch den Autor, dem Erscheinungsjahr oder IBAN genauer beschrieben werden. Selbst Foto-Dateien, die mit einer Digitalkamera angefertigt werden, werden automatisch Metadaten mit abgespeichert. So kann im Nachhinein herausgefunden werden, wann das Foto geschossen wurde, wie die Auflösung ist und in welchem Dateiformat das Bild vorliegt.

Die Objekte die beschrieben werden, werden Ressourcen genannt.

Arten von Metadaten

Es gibt fünf verschiedene Arten von Metadaten [vgl. 30]:

- Identifizierende/beschreibende Metadaten: beinhalten Informationen über die Ressource, um diese leichter identifizieren zu können. Zum Beispiel der Buchtitel, der Personenname oder das Entstehungsdatum.
- Administrative Metadaten: beinhalten Informationen über die Verwaltung der Ressource. Zum Beispiel von welcher Kamera das Bild geschossen wurde oder wer im Laufe der Zeit Änderungen an der Ressource vorgenommen hat.
- Content Ratings Metadaten: beinhalten Informationen, für wem die Informationen bestimmt sind. Zum Beispiel was für eine Zielgruppe ein Buch hat, wie etwa Kinderbücher für Kinder geeignet sind.
- Linkage/Relationship Metadaten: geben die Beziehung zwischen mehreren Ressourcen an. Zum Beispiel ist Mario Götze ein Teil der deutschen Nationalmannschaft oder Jürgen Klinsmann ist der Vorgänger von Joachim Löw.
- Meta-Metadaten: die Metadaten können wieder von anderen Metadaten beschrieben werden. Diese können zum Beispiel Aussagen über die Modelle, Syntax und Formate der zugrunde liegenden Metadaten sein.

Metadatenelemente

Die Metadatenelemente beschreiben eine bestimmte Art von Ressourcen. Die verschiedenen Arten werden als Klassen (englisch: Classes) bezeichnet. Jede Klasse hat mehrere Eigenschaften (englisch: Properties), denen individuelle Werte zugewiesen werden können. Diese Werte müssen vom korrekten Datentyp sein. Welcher dies ist, wird vom Encoding Schema (englisch: Encoding Scheme) festgelegt. [vgl. 30]

Metadatenelemente für eine Klasse Person könnte wie folgt aussehen:

Klasse: Person Eigenschaften: Geburtsdatum, Name Encoding Schema: Geburtsdatum = xsd:date, Name = xsd:string

Abbildung 5: Beispielhafte Metadatenelemente für die Klasse Person

Anhand dieser Beschreibung, kann eine Ressource vom Typ Person, im folgenden Beispiel wird der Nationaltorhüter der deutschen Fußballnationalmannschaft verwendet, wie folgt aussehen:

Name = „Manuel Neuer“ Geburtsdatum: 1986-03-27+02:00

Abbildung 6: Beispielhafte Ressource für das Metadatenelement Person

Eine Klasse kann auch beliebig viele Unterklassen (englisch: Subclasses) besitzen. So könnte die Klasse Person, auch noch die Unterklasse Fußballer besitzen. Sollte eine Ressource nun von der Klasse Fußballer sein, werden alle Eigenschaften von der Personen- und von der Fußballer-Klasse übernommen.

Interoperabilität

„Interoperabilität ist die Fähigkeit, Daten zwischen verschiedenen Systemen ohne allzu großen Informationsverlust auszutauschen.“ [30]

Dies wird erwünscht um Informationen aus mehreren verschiedenen Datenquellen zu erhalten und um diese untereinander verknüpfen zu können. Damit dies bewerkstelligt werden kann, müssen folgende drei Dinge Interoperabilitäten berücksichtigt werden:

- Strukturelle Interoperabilität: gemeinsames Datenmodell (zum Beispiel das RDF-Modell [22], siehe Kapitel 2.4 RDF & Turtle)
- Syntaktische Interoperabilität: gemeinsame Syntax (zum Beispiel Turtle [23], siehe Kapitel 2.4 RDF & Turtle)
- Semantische Interoperabilität: gemeinsame Metadatenstandards

Leider unterscheiden sich die meisten Informationsquellen in mindestens einem Punkt voneinander, wodurch eine Verknüpfung der Daten meist gar nicht oder nur eingeschränkt möglich ist. Deswegen beschränkt sich diese Abschlussarbeit auf das Semantische Web, was versucht einen einheitlichen Standard zu schaffen.

2.3 Semantic Web

> The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF). [26]

Das World Wide Web (wird zukünftig nur noch als Web bezeichnet) bietet den Nutzern eine große Vielzahl an Informationen und ist aufgrund der weltweiten Erreichbarkeit und der zugrundeliegenden Infrastruktur eine gute und schnelle Variante des Informationsaustausches. Der größte Vorteil des Webs, gegenüber vielen anderen Medien, ist die Möglichkeit die Informationen schnell aktualisieren zu können und dass sich jeder an der Verbreitung dieser Daten beteiligen kann.

Aufgrund der immer größer werdenden Menge an Informationen, wird es für den Nutzer immer schwieriger den Überblick über die Daten zu behalten. Suchmaschinen helfen bei der Findung, doch präsentieren meist viele verschiedene Ergebnisse, welche häufig nur Teilwissen präsentieren. Zudem sind viele Redundanzen zu finden und durch die Vielzahl an verschiedenen Kodierungstechniken und Sprachen, ist es kaum möglich die Informationen miteinander in Verbindung zu setzen.

Das World Wide Web Consortium (W3C) [27] hat deswegen das Semantic Web geschaffen und sich das Ziel gesetzt, mit Hilfe von Standards, die Informationen so zur Verfügung zu stellen, dass diese einer einheitlichen Struktur unterliegen und dass Beziehungen zwischen den einzelnen Daten hergestellt werden können (Interoperabilität). Dies hätte den Vorteil, dass auch eine Verarbeitung durch Maschinen ermöglicht werden würde, was wiederum dem Benutzer das Auffinden der Daten erleichtert. [vgl. 28].

Für diese Abschlussarbeit sind vor allem folgende zwei Standards von der W3C relevant:

- Resource Description Framework (RDF) [22] - Beschreibungssprache für Informationen einer Webressource (siehe Kapitel 2.4 RDF & Turtle)
- SPARQL Protocol and RDF Query Language (SPARQL) [31] – Abfragesprache für das Resource Description Framework (siehe Kapitel 2.5 SPARQL)

Ontologien

„An ontology is an explicit specification of a conceptualization“ [32]

Ontologien sollen die Kommunikation zwischen verschiedenen Kommunikationspartnern (Computersysteme, Menschen, Bereich einer bestimmten Domäne) ermöglichen und verbessern. Dabei besteht eine Ontologie aus einem Basisvokabular, Definitionen und Entitäten und einer Beschreibung der bestehenden Assoziationen.

Eine leichtgewichtige Ontologie für RDF-Daten ist die RDF Vocabulary Description Language (RDF-Schema). Mit RDF-Schema (RDFS) ist es möglich, Begriffe und die damit verbundenen Elemente semantisch zueinander in Beziehung zu setzen. Zum Beispiel kann in RDFS eine Eigenschaft *name* deklariert werden, deren Wert dann der Namen von Personen sein soll. Zudem kann genau definiert werden, was für Werte erlaubt sind. In diesem Fall sollte der *name* nicht nur aus nummerischen Werten bestehen, sondern aus einer Zeichenkette. Des Weiteren kann festgelegt werden, zu welchen anderen Eigenschaften eine Beziehung hergestellt werden kann, was für eine Bedeutung diese Eigenschaft hat und welche Arten von Ressourcen verwendet werden dürfen.

Die W3C hat auch für Ontologien einen Standard entwickelt, dieser heißt Web Ontology Language (OWL) [29] und ist eine Beschreibungssprache für Klassen und Relationen. OWL ist im Gegensatz zu RDFS eine deutlich ausdrucksstärkere Sprache, womit sich die Konzepte besser spezifizieren lassen. In Kapitel 2.6 dbpedia, wird genauer auf die verwendete dbpedia-Ontologie eingegangen.

2.4 RDF & Turtle

Das Ressource Description Framework (RDF) wird im semantischen Web als strukturelle Interoperabilität verwendet und ist demnach die Beschreibungssprache für Informationen einer Webressource. Als syntaktische Interoperabilität wird die Terse RDF Triple Language (Turtle) vorgestellt.

Ressource Description Framework

Das RDF ist ein Standard der W3C [22], für die Beschreibung von Metadaten. Erstmalig wurde der Standard 1999 als Empfehlung veröffentlicht und wird hauptsächlich im Semantik Web verwendet, um Aussagen über die Assoziationen zwischen den Objekten zu treffen. Dabei besteht eine Aussage aus einem Triple mit den folgenden drei Teilen:

1. Das Subjekt, was die zu beschreibende Ressource darstellt
2. Das Prädikat, was ein Property ist (abhängig von die Art der Ressource)
3. Das Objekt, was der Wert für die Property, für genau diese eine Ressource ist

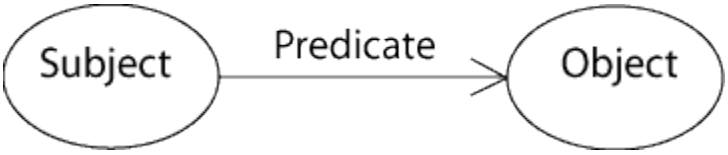

Abbildung 6: Darstellung eines RDF-Tripels als RDF-Graph [33]

In Abbildung 6 wird gezeigt, wie ein RDF-Tripel aus Subjekt, Prädikat und Objekt, visuell als RDF-Graph aussehen würde. Sowohl Subjekt als auch Objekt werden in einem Kreis dargestellt, während das Prädikat als Beschriftung der Kante eingezeichnet wird.

Subjekt	Prädikat	Objekt
Alfred Hitchcock	birthDate	1899-08-13
Alfred Hitchcock	birthPlace	London
Alfred Hitchcock	Surname	Hitchcock

Tabelle 1: Beispiel für drei mögliche RDF-Tripel in vereinfachter Form

Um Verwechslungen vorzubeugen und die Eindeutigkeit zu gewährleisten, wird anstatt eines Literals (oder auch String), eine Internationalized Resource Identifier (IRI) als Ressource und als Property verwendet. Das Objekt kann neben einem Literal, ebenfalls eine IRI einer Ressource enthalten.

Subjekt	Prädikat	Objekt
<http://de.dbpedia.org /resource /Alfred_Hitchcock>	<http://dbpedia.org/ontology /birthDate>	"1899-08-13"^^ <http://www.w3.org/2001 /XMLSchema#date>
<http://de.dbpedia.org /resource /Alfred_Hitchcock>	<http://dbpedia.org/ontology /birthPlace>	<http://de.dbpedia.org /resource/London>
<http://de.dbpedia.org /resource /Alfred_Hitchcock>	<http://xmlns.com/foaf/0.1 /surname>	"Hitchcock"

Tabelle 2: Beispiel für drei RDF-Tripel von der dbpedia

In allen drei Zeilen ist die Ressource, die IRI von Alfred Hitchcock. Als Datenquelle wurde die dbpedia [4] (mehr dazu in Kapitel 2.6) gewählt. In der ersten Zeile wird die Property *birthDate* verwendet und der Wert (das Objekt) beinhaltet das Datum gefolgt vom entsprechenden XMLSchema. In der zweiten Zeile erhalten wir beim Prädikat *birthPlace*, als Wert eine IRI von der Ressource *London*. Auf diese Weise werden Vernetzungen zwischen mehreren Ressourcen geschaffen. In der letzten Zeile erhalten wir als *surname* ein Literal.

Neben Literalen und IRIs gibt es noch blank nodes. Immer wenn ein Knoten weder ein Literal, noch eine IRI ist, wird daraus ein „leerer Knoten“. Dieser kann lediglich als Subjekt oder Objekt auftreten. Häufig entsteht solch ein Knoten, wenn lokale URIs als Ressource verwendet werden, welche global dann nicht für alle verfügbar sind.

Terse RDF Triple Language

Für RDF gibt es zudem mehrere Repräsentationsarten. Die am weitesten verbreiteten Varianten im Web sind das Turtle Format [23] und die RDF/XML Schreibweise [25]. Im semantischen Web wird Turtle verwendet, da die Syntax leichter und einfacher zu verstehen ist und im Gegensatz zur RDF/XML-Schreibweise, nicht durch die XML Regeln eingeschränkt werden. Da Turtle eine Teilmenge von der Notation 3 (N3) [24] ist, kann überall wo N3 unterstützt wird, auch Turtle verwendet werden. Zudem kann Turtle auch über die SPARQL-Schnittstelle (siehe Kapitel 2.5 SPARQL) abgerufen werden. Für Turtle wurde von der W3C ebenfalls als Standard veröffentlich unter dem Namen RDF 1.1 Turtle [23].

Der folgende Textauszug zeigt die drei Tripel aus Tabelle 2 im Turtle Format:

```
<http://de.dbpedia.org/resource/Alfred_Hitchcock> <http://dbpedia.org/ontology/birthDate>
"1899-08-13"^^<http://www.w3.org/2001/XMLSchema#date> .

<http://de.dbpedia.org/resource/Alfred_Hitchcock>
<http://dbpedia.org/ontology/birthPlace> <http://de.dbpedia.org/resource/London> .

<http://de.dbpedia.org/resource/Alfred_Hitchcock> <http://xmlns.com/foaf/0.1/surname>
"Hitchcock"@de .
```

Abbildung 7: Inhalt einer Turtle-Datei, welche drei Tripel von Alfred Hitchcock enthält

Die Unterteilung zwischen den Tripeln erfolgt bei Turtle mit Hilfe von Punkten. Die einzelnen Bestandteile des Statements werden durch ein Leerzeichen getrennt, wobei die Reihenfolge immer eingehalten werden muss (Subjekt Prädikat Objekt).

Ein Literal kann ein String sein, welcher innerhalb von Anführungszeichen geschrieben steht, mit einem Optionalen @-Zeichen, gefolgt von einem Language-Kürzel ("Hitchcock"@de). Ansonsten kann es auch ein integer, double, decimal oder boolean als Wert enthalten. Für alle anderen Datentypen wird der Wert erneut als String interpretiert, gefolgt von `^^` und der IRI zum Datentypschema ("1899-08-13"^^<http://www.w3.org/2001/XMLSchema#date>).

Ressourcen werden bei Turtle in spitzen Klammern geschrieben (<http://de.dbpedia.org/resource/Alfred_Hitchcock>).

Alternativ können auch Präfixe verwendet werden, dann wird eine Ressource wie folgt geschrieben:

```
@prefix dpbedia : <http://de.dbpedia.org/resource/> .

dbpedia:Alfred_Hitchcock <http://dbpedia.org/ontology/birthPlace> dbpedia:London .
```

Abbildung 7: RDF-Tripel in Turtle-Schreibweise unter Verwendung eines Prefix

Die ausführliche Grammatik von Turtle wird im entsprechenden W3C-Standard ausführlicher beschrieben [34].

Zum Vergleich zeigt der folgende Textauszug die gleichen drei Tripel in der RDF/XML-Schreibweise:

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

 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex ="http://xmlns.com/foaf/0.1/"
  xmlns:dbp ="http://de.dbpedia.org/resource/">
   <rdf:Description rdf:about="http://de.dbpedia.org/resource/Alfred_Hitchcock">
    <dbp:birthDate rdf:datatype=“ http://www.w3.org/2001/XMLSchema#date“>
     1899-08-13
    </dbp:birthDate>
    <dbp:birthPlace>
     <rdf:Description rdf:about="http://de.dbpedia.org/resource/London">
     </rdf:Description>
    </dbp:birthPlace>
    <ex:surname>
     Hitchcock
    </ex:surname>
   </rdf:Description>
 </rdf:RDF>
```

Abbildung 9: Inhalt einer RDF/XML-Datei, welche drei Tripel von Alfred Hitchcock enthält

2.5 SPARQL

Die SPARQL Protocol And RDF Query Language (SPARQL) [31] ist wie auch RDF ein Standard von der W3C. Wie der Name schon sagt handelt es sich um eine Anfragesprache zur Abfrage von Instanzen aus RDF-Dokumenten. Anbieter von Metadaten, wie etwa DBpedia bieten ihre Daten im RDF-Format an und der Zugriff kann über eine SPARQL-Schnittstelle erfolgen. Die Abfragen haben dann folgende Syntax:

```
<PREFIX declarations...>

SELECT <variables...>
WHERE {
  <graph patterns with variables...>
}
ORDER BY <conditions...>
OFFSET <offset>
LIMIT <limit>
```

Abbildung 8: Allgemeine Syntax von SPARQL

PREFIX und WHERE sind fester Bestandteil der Grundstruktur. Als Ausgabeformat kann anstatt der SELECT-Klausel auch CONSTRUCT, ASK oder DESCRIBE verwendet werden. Optional hingegen sind die Graph-Muster (unter anderen OPTIONAL oder UNION), Filter (BOUND; isURI, isLITERAL, STR, REGEX, etc.) und Modifikatoren (ORDER BY, LIMIT, OFFSET, DISTINCT).

Innerhalb der WHERE-Klausel wird die Turtle-Syntax angewandt, außerhalb wird die Anfrage durch zusätzliche SPARQL-Anfragevariablen erweitert, um bestimmte Elemente als Ergebnis zu ermitteln. Das folgende Beispiel zeigt eine mögliche SPARQL-Abfrage:

```
PREFIX ex: <http://example.org/>
SELECT ?titel ?autor
WHERE
 { ?buch ex:VerlegtBei   <http://springer.com/Verlag> .
   ?buch ex:Titel        ?titel .
   ?buch ex:Autor        ?autor . }
```

Abbildung 9: Beispiel für eine SPARQL-Anfrage [35]

Die Schlüsselwörter sind in SPARQL durch ihre Großschreibweise hervorgehoben. PREFIX deklariert genauso wie in Turtle das @prefix, den Namensraum. Hinter dem SELECT werden die Variablen angegeben, deren Werte wir als Antwort erhalten möchten. In diesem Beispiel sind das die Variablen ?titel und ?autor. Die eigentliche Anfrage wird innerhalb der WHERE-Klausel durchgeführt, was in dem Fall drei Turtle-Tripel sind, nur das neben IRIs auch die Variablen verwendet werden können. Diese dienen als Platzhalter und werden erst durch die Anfrage mit konkreten Werten gefüllt. [vgl. 35]

Anhand des ersten Tripels erhalten wir alle Bücher, die die Eigenschaft http://example.org/VerlegtBei haben und dort den Wert <http://springer.com/Verlag> zugewiesen haben. Durch die letzten zwei Statements fragen wir alle Bücher ab, die einen Titel besitzen und die einen Autor besitzen. Diese Titel und Autoren sollen auch die Ergebnistabelle liefern. Wichtig ist, dass alle Variablen die wir im SELECT erwarten, müssen auch in der WHERE-Klausel vorkommen. [vgl. 35]

Diese Anfrage wird nun auf folgende RDF-Datei ausgeführt:

```
@prefix ex: <http://example.org/> .
ex:SemanticWeb ex:VerlegtBei <http://springer.com/Verlag> .
ex:SemanticWeb ex:Titel "Semantic Web - Grundlagen" .
ex:SemanticWeb ex:Autor ex:Hitzler .
ex:SemanticWeb ex:Autor ex:Krötzsch .
ex:SemanticWeb ex:Autor ex:Rudolph .
ex:SemanticWeb ex:Autor ex:Sure .
```

Abbildung 10: Turtle-Datei mit sechs Tripeln [vgl. 35]

Als Ergebnis wird folgende Tabelle geliefert:

titel	autor
"Semantic Web - Grundlagen"	http://example.org/Hitzler
"Semantic Web - Grundlagen"	http://example.org/Krötzsch
"Semantic Web - Grundlagen"	http://example.org/Rudolph
"Semantic Web - Grundlagen"	http://example.org/Sure

Tabelle 3: Ergebnistabelle für die SPARQL-Abfrage (Abb. 9), angewandt auf die RDF-Datei (Abb. 10) [35]

Weitere SPARQL-Grundlagen können dem W3C Standard [31] oder dem Semantic Web - Buch [35] entnommen werden. Im vierten Kapitel werden die für dieses Projekt verwendeten SPARQL-Abfragen, sowie auf deren enthaltenen Schlüsselwörter genauer eingegangen.

2.6 DBpedia

Unter den Begriff Linked Open Data (LOD) hat Tim Berners-Lee im Jahr 2006, die Grundidee für das Vernetzen von Daten geschaffen. Die Idee bestand darin, URIs als Namen von Dingen zu nehmen, diese über http zugänglich zu machen, verwenden von Standards wie RDF und SPARQL und Verlinkungen zu anderen URIs verwenden, wenn diese eine Verbindung zueinander haben [36]. Dieses LOD ist fester Bestandteil vom Semantic Web und hat bereits viele Datenbestände involvieren können [68]. Die meisten Daten und Verlinkungen entstammen der DBpedia.

„The importance of DBPedia is not only that it includes Wikipedia data, but also that it incorporates links to *other* datasets on the Web, e.g., to Geonames.“ [38]

Das DBpedia-Projekt wurde 2007, in Zusammenarbeit zwischen der Freien Universität Berlin, der Universität Leipzig und OpenLink Software, gestartet. Ziel war es die Metadaten aus Wikipedia zu extrahieren, diese zu strukturieren und letztendlich zur Verfügung zu stellen. Inzwischen bietet DBpedia Informationen für 12,6 Millionen Dinge in 119 verschiedenen Sprachen und 2,46 Milliarden RDF-Triple (davon 470 Millionen aus der englischen Wikipedia) [4]. Hinzu kommen über 50 Million externe Datensätze von Datenquellen wie FOAF (Friend Of A Friend)[40] oder GeoNames [5].

Diese Strukturierung der Daten hat auch für Wikipedia seine Vorteile. Seit mehreren Jahren befindet sich bei vielen Artikeln in der oberen, rechten Ecke eine Box mit den Metadaten. Diese sind bei gleichen Artikeltypen identisch gegliedert. Wenn es sich zum Beispiel um eine Person handelt, welche Fußball spielt, befinden sich neben Geburtstag, Geburtsort, Größe und Position, auch Informationen über deren aktuellen und ehemaligen Vereine.

Der Datenzugriff kann über zwei verschiedene Wege erlangt werden. Wer die Metadaten offline verwenden möchte, dem werden viele Datasets zum Download als RDF-Dateien zur Verfügung gestellt. Neben der N-Triples Darstellungsart (nt, nq), welche mit URIs (http://dbpedia.org/resource/Bras%C3%ADlia) arbeiten, werden auch Turtle-Dateien zur Verfügung gestellt. Diese verwenden IRIs (http://dbpedia.org/resource/Brasília). Zur Auswahl stehen unter anderen Datensätze mit Geografischen Koordinaten, Personendaten oder auch Homepages [39]. Für den Online-Zugriff steht eine SPARQ-Schnittstelle zur Verfügung, was den Zugriff auf die Live-Daten ermöglicht.

DBpedia Ontologie

Die DBpedia Ontologie umfasst 685 Klassen mit insgesamt 2795 Properties. Die Basisklasse von der sich alle anderen Klassen abzweigen, ist die owl:Thing. In der zweiten Ebene befinden sich Klassen wie Place, Agent oder Work. Unter Agent werden wiederum Klassen wie Person oder Organisation geführt. Insgesamt enthält die DBpedia Ontologie über 4,2 Millionen Instanzen, wobei die meisten davon der Klasse Person (1,45 Millionen) angehören, gefolgt von Place (735.000) und Work (411.000). [42]

Jede Klasse enthält mehrere Eigenschaften (Properties), wobei jede Eigenschaft von einem festen Datentyp ist. Dies können entweder Standardtypen sein wie xsd:string, xsd:date oder xsd:integer, aber auch Objekte von Klassen sind möglich. So hat die Klasse Person das Property employer, was vom Typ Organisation ist. [43]

2.7 Stand der Forschung

Der niederländische Professor Wil van der Aalst von der TU Eindhoven, hat eine Sammlung von verschiedenen Entwurfsmustern im Bereich des Workflow-Managements entworfen und diese in seinem Buch „Workflow Patterns“ [46] [47] veröffentlicht. Dabei werden bei jedem Muster die Abhängigkeiten zwischen den Aktivitäten (Knoten) im Kontrollfluss betrachtet. Viele dieser Muster sind auch in KNIME wieder zu finden, vor allem die Basis Kontrollmuster (englisch Basic Control Patterns), wie etwa Sequence, Parallel Split, Synchronize und Simple Merge, bilden die Grundlagen bei der Verbindung und Trennung der Daten und Metadaten.

Für die Verarbeitung von Daten/Metadaten-Verbünden haben sich drei verschiedene Vorgehensweisen etabliert. Die wohl am weitesten verbreitete Methode um eine Kopplung zwischen den Daten zu erzeugen, ist das Einfügen von Assoziationen. Im Datenbankbereich werden die Metadaten meist in einer anderen Tabelle gespeichert, als die Daten. Um diese miteinander verknüpfen zu können, enthalten die Datensätze eindeutige Schlüssel (Keys). Je nach Assoziationsart, wird der Key der Metadaten in die Tabelle der Daten als Foreignkey hinzugefügt (bei 1:1- oder 1:n-Beziehungen) oder es wird eine neue Tabelle erzeugt, in der die Datensätze aus den Keys beider Tabellen bestehen (n:m-Beziehung). Mit Hilfe der Structured Query Language (SQL) können die verknüpften Daten tabellenübergreifend abgefragt und bearbeitet werden. Anstelle von Datenbanken können auch Dateien verwendet werden. Zum Beispiel, in einer RDF-Datei können die Metadaten stehen und in einer XML die Daten. Die Verknüpfung kann entweder durch das Hinzufügen der entsprechenden Metadaten-Id in der XML geschehen oder durch eine explizite Konfigurationsdatei, in welcher die entsprechenden Verknüpfungen vermerkt sind.

Alternativ kann eine Datenfusion durchgeführt werden. Hier wird aus mehreren zusammengehörigen Datensätzen, ein Datensatz geschaffen. Dabei gibt es viele verschiedene Möglichkeiten, wie mit Zusammenführungskonflikten umgegangen werden kann, wie etwa, wenn in beiden Tabellen eine Spalte identisch ist oder was geschehen soll, wenn zwar die Spalten gleich heißen, aber der Inhalt unterschiedlich ist. Je nach Datenbestand und System sind die Konzepte und Lösungsmöglichkeiten der Datenfusion stark unterschiedlich [48] und wird stets diskutiert und weiter entwickelt. Einige relevante Konzepte der Datenfusion werden im 3. Kapitel vorgestellt. Da zur Verarbeitung der Daten/Metadaten-Verbünde auch das Trennen der Daten gehört, muss die Datenfusion noch durch eine Konfiguration erweitert werden, um Rückschlüsse auf die Datenherkunft zu ermöglichen und somit die Rekonstruktion zu ermöglichen.

Bei nicht verteilten Systemen, welche mit objektorientierten Sprachen entwickelt werden, werden Daten auch mittels Objekten miteinander Verknüpft. Dabei werden zwei Klassen erzeugt, eine für die Daten und eine für die Metadaten. In der Klasse von den Daten, gibt es ein Objekt von der Klasse der Metadaten. Innerhalb des Systems könnten somit die Daten und Metadaten gemeinsam verarbeitet werden und auch wieder problemlos getrennt werden. Um diese Variante auch bei verteilten Systemen oder auch bei der Pipe und Filter – Architektur verwenden zu können, wird auf die Serialisierung zurückgegriffen, bei der die Objekte persistiert und übergeben werden können. [49] Dabei wird der komplette Zustand des Objektes, inklusive aller referenzierten Objekte, in einen Datenstrom umgewandelt, welcher anschließend auf ein Speichermedium geschrieben wird. Auf der Empfängerseite muss anschließend eine Deserialisierung durchgeführt werden, wo der Datenstrom wieder in Objekte umgewandelt wird.

3. Architektur

In diesem Kapitel wird erst die Datenfluss-Architektur von KNIME vorgestellt und anschließend werden die Erweiterungsmöglichkeiten der Architektur, für die gemeinsame Verarbeitung von Daten und Metadaten, erläutert. Abschließend wird noch eine begründete Auswahl getroffen, welche Anpassungen vorgenommen werden.

3.1 Datenfluss-Architektur von KNIME

Dieses Kapitel bezieht sich auf den Technical Report für KNIME [50]. Beim Designen der Architektur für KNIME wurde das Hauptaugenmerk auf drei Kriterien gelegt:

- Es soll ein visuelles und interaktives Framework sein, was mittels einfachen Drag&Drop bedient werden kann.
- Es soll modular aufgebaut sein, so dass alle notwendigen Berechnungen eines Algorithmus nur innerhalb eines Moduls durchgeführt werden und nicht abhängig sind von anderen Modulen. Zudem sollen die Datentypen gekapselt sein und neue Typen einfach integriert werden können.
- Es soll leicht erweitert werden können, indem einfach neue Knoten hinzugefügt oder entfernt werden können. Selbiges gilt für die Assoziationen.

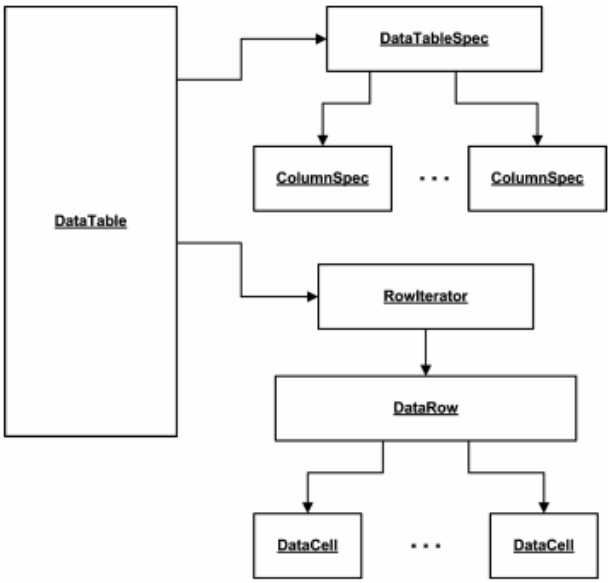

Abbildung 11: Diagramm von der Datenstruktur und den Hauptklassen von den KNIME-Tabellen [50]

In Anbetracht der gewünschten Modularität und der einfachen Erweiterbarkeit, haben sich die Entwickler von KNIME für eine Pipeline-Architektur (Pipes und Filter) entschieden. Bei diesem Datenfluss-Architekturmuster „fließen“ Daten oder Modelle über eine Pipe („Rohrleitung“) bis zur Datenausgabe. Um nun Algorithmen auf diese Daten anwenden zu können, wird die Pipe durch Filter unterbrochen. In KNIME wird die Pipe als Kante und die Filter als Knoten bezeichnet. Die Komplexität kann dabei von einfachen Filterungen bis hin zu komplexen, statistischen Funktionen reichen.

Die Informationen, welche sich zwischen den Knoten bewegen, sind von der Klasse DataTable und beinhaltet die Daten, Metadaten (DataTableSpec) und Spaltentypen (ColumnSpec). Eine solche Tabelle ist vom Aufbau her vergleichbar mit denen in Tabellenkalkulationsprogrammen (z.B. wie bei MS Excel). Die Tabelle ist unterteilt in mehrere Reihen (DataRow), welche

durchiteriert werden können. Jede Reihe besitzt eine eindeutige ID (RowId) und eine bestimmte Anzahl an Zellen (DataCell), welche die Informationen enthalten.

Die Knoten in KNIME sind vom Typ Node und implementieren mehrere abstrakte Klassen:

- NodeDialog: der Dialog wird angezeigt, wenn ein Knoten konfiguriert werden soll. Zum Beispiel um bei einem Reader den Dateipfad auszuwählen
- NodeModel: enthält die Logik vom Knoten.
- NodeView: zeigen den Inhalt der DataTable an.

Während es genau eine NodeModel-Klasse geben muss, ist das schreiben einer eigenen NodeDialog-Klasse optional. NodeViews kann es sogar mehrere geben, wobei die Anzahl der Views meist äquivalent zur Anzahl der Ausgangsports ist. Das zugrunde liegende Designpattern ist das Model-View-Controller Muster [37].

Zudem hat jeder Knoten verschieden viele Eingangs- und Ausgangsports. Eine genaue Übersicht über die Struktur zeigt die folgende Abbildung.

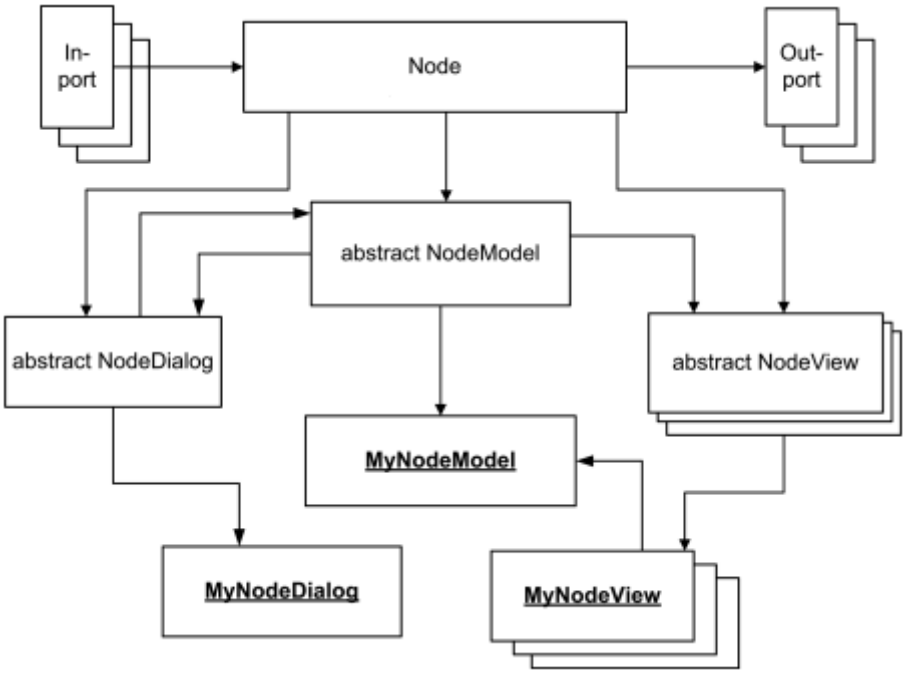

Abbildung 12: Struktur der Hauptklassen eines Knotens [50]

Die Workflows von KNIME erweitern das klassische Pipes und Filter Architekturmuster um weitere Pipes. So ist es möglich, mehrere Datenquellen zu haben, welche in einen Knoten zusammengeführt werden (T-Filter). Zudem gibt es auch Knoten, welche den Datenstrom in mehrere Pipes aufteilen (T-Pipes).

Das eine Verbindung zwischen zwei Knoten entstehen kann, verdankt KNIME dem WorkflowManager. Dieser verleiht den Knoten auch drei verschiedene Status, was er mit den Ampelfarben kenntlich macht. So kann ein Knoten als unkonfiguriert (rot), konfiguriert (gelb) und ausgeführt (grün) gekennzeichnet werden. Wenn ein Knoten keiner Konfiguration bedarf, dann ist der Knoten gleich von Beginn an gelb. Ein Knoten wird nur grün, wenn die Ausführung erfolgreich war. Außerdem kann ein Knoten nur ausgeführt werden, wenn alle vorherigen Knoten erfolgreich ausgeführt worden sind. Bevor ein Knoten neu konfiguriert werden kann, muss vorher ein Reset des Knotens durchgeführt werden, dies hat zur Folge, dass alle folgenden Knoten ebenfalls zurückgesetzt werden.

Durch die einfache Integration neuer Knoten, haben bereits mehrere Analysetools wie Weka, statistische Toolkits wie R oder auch Visualisierungsmöglichkeiten wie JFreeChart einen Knoten erhalten. Um selbst einen Knoten zu entwickeln, bietet Knime eine Möglichkeit in Eclipse an. Dort wird nach der Namensgebung ein Grundgerüst mit allen notwendigen Dateien erstellt.

3.2 Erweiterung der Architektur für die gemeinsame Verarbeitung von Daten und Metadaten

KNIME bieten einem die Möglichkeit mit mehreren Daten zu arbeiten, die Zusammenführung dieser unterliegt aber strengen Voraussetzungen. So muss beim „Column Appender“ die Anzahl der Zeilen, die Reihenfolge und die RowID übereinstimmen, ansonsten kann der Knoten nicht ausgeführt werden. Etwas komfortabler ist der „Joiner“-Knoten. Hier spielt die Reihenfolge und die Anzahl der Zeilen keine Rolle. Anstatt der RowID kann auch eine andere beliebige Spalte ausgewählt werden, welche dem Vergleich unterzogen werden sollen. Aber auch hier müssen beide Zellen dann komplett identisch sein, da sonst die Zeilen nicht miteinander verbunden werden. Dies ist aber im Bereich der Daten und Metadaten selten der Fall. Bei den Daten wird aufgrund des einfacheren Verständnisses ein Literal verwendet, zum Beispiel bei einer Person wird der Name oder bei einem Buch wird der Buchtitel verwendet. Die Metadaten, sofern diese aus dem Semantik Web stammen, werden über URIs identifiziert. Eine Verbindung über die bisherigen Knoten ist somit nicht möglich.

Für das korrekte Assoziieren zwischen zwei zusammengehörigen Daten und Metadaten, muss ein Algorithmus entwickelt und als Knoten zur Verfügung gestellt werden. Die höchste Trefferquote lässt sich mittels Search-Engines erzielen (ziehe Kapitel 4.3). Für die einfache lokale Zusammenführung werden hingegen die URIs in Literale umgewandelt und dann mit den Namen der Daten abgeglichen.

Das Erzeugen der Daten/Metadaten-Verbände ist aber nur der erste Schritt in der Verarbeitungskette. Diese Verbände sollen im zweiten Schritt noch verarbeitet werden. Für die Verarbeitung stellt KNIME sehr viele Knoten zur Verfügung, weswegen es erstrebenswert ist, die Zusammenführung so zu realisieren, dass diese Knoten auch weiterhin verwendet werden können.

Auch die Art der Verarbeitung sollte überdacht werden. Sollte der Nutzer immer mit den kompletten Verbänden arbeiten wollen, empfiehlt sich eine permanente synchrone Verarbeitung. Möchte der Nutzer hingegen nur mit einem Strang der Daten arbeiten wollen, ist eine asynchrone Verarbeitung der bessere Weg. Am komfortabelsten wäre es, wenn dem Benutzer selbst die Entscheidung überlassen wird, wann die Daten synchron und wann asynchron verarbeitet werden sollen. Um dies zu gewährleisten muss auch der dritte Schritt umgesetzt werden, die Trennung.

Bei der Trennung der Verbände werden die Daten und die Metadaten wieder voneinander getrennt. Da bei der Umsetzung des Projekts weder der Kern von KNIME bearbeitet werden, noch die bisherigen Knoten umgeschrieben werden sollen, muss anderweitig Abhilfe geschaffen werden. Dazu muss in der Konfiguration der Spalten (ColumnSpec), beim Zusammenführen der Daten/Metadaten gespeichert werden, welche Spalte zu welcher Quelle gehört. Diese Information muss bis zum Trennungsknoten durchgereicht und dementsprechend verarbeitet werden.

Folgende Umsetzungsvarianten haben sich anhand der Vorgaben und der Architektur ergeben:

- Synchrone Verarbeitung mit einer Kante
- Asynchrone Verarbeitung mit zwei Kanten
- Synchrone Verarbeitung mit zwei Kanten

Synchrone Verarbeitung mit einer Kante

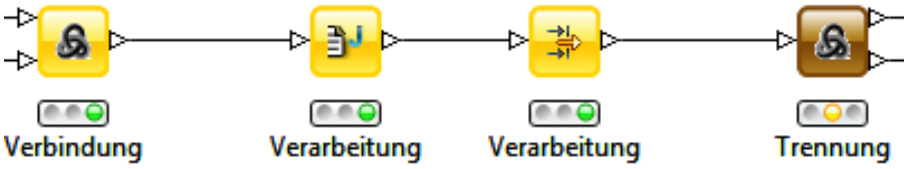

Abbildung 13: KNIME-Workflow-Ausschnitt für die synchrone Verarbeitung mit einer Kante

Bei der synchronen Verarbeitung mit einer Kante werden die Daten und Metadaten in eine Tabelle transferiert. Um dies zu verwirklichen wird die Daten-Quelle als Grundlage genommen. Anhand deren RowIDs werden in der Metadaten-Tabelle die passenden Meta-Informationen gesucht. In der zusammengeführten Tabelle wird auch diese RowID übernommen, die von der Metadaten-Tabelle wird zusätzlich hinten als neue separate Spalte angehangen.

Der Vorteil an der Zusammenführung ist, dass alle bisherigen KNIME-Knoten verwendet werden können und alle Informationen zu einem Datensatz gut in einer Zeile ablesbar sind. Die Manipulation aller Daten ist somit möglich und schnell erkennbar.

Als Workflow Patter wird für die Verbindung der Daten das „Simple Merge“-Pattern verwendet. Ziel des Knotens ist die Fusion der Daten.

Damit die Trennung der Daten/Metadaten-Verbände korrekt verläuft, müssen die Spalten getaggt werden. Dies geschieht indem im Verbindungsknoten alle Spalten, welche Informationen von der Metadaten-Tabelle enthalten, die Property „meta“ erhalten. Ein Sonderfall ist dabei die ehemalige RowID, welche im Verbindungsknoten eine neue Spalte erhalten hatte. Bei dieser wird dem „meta“-Property der Wert „uri“ zugewiesen. Somit wird diese Spalte wieder entfernt und als RowID verwendet.

Beispiel:

Row ID	Lieblingsfilm
Alfred Hitchcock	Topas

Tabelle 4: Beispielhafte Darstellung einer KNIME-Tabelle mit einem Datensatz aus den Daten

Row ID	<http://dbpedia.org/ontology/birthDate>
<http://de.dbpedia.org/resource/ Alfred_Hitchcock>	"1899-08-13"^^ <http://www.w3.org/2001/XMLSchema#date>

Tabelle 5: Beispielhafte Darstellung einer KNIME-Tabelle mit einem Datensatz aus den Metadaten

Row ID	Lieblingsfilm	<http://dbpedia.org/ontology/birthDate>	URI
Alfred Hitchcock	Topas	"1899-08-13"^^ <http://www.w3.org/2001/XMLSchema#date>	<http://de.dbpedia.org/resource/Alfred_Hitchcock>

Tabelle 6: KNIME-Tabelle nach der Zusammenführung der Tabellen 4 & 5 mit Hilfe des Verbindungsknotens

Für die Spalte <http://dbpedia.org/ontology/birthDate> wird in der ColumnSpec das Property „meta“ erzeugt. In die Spalte URI wird dieses Property zusätzlich mit „uri“ belegt, damit erneut eine Aufspaltung von der Tabelle 3 in die Tabelle 1 & 2 geschehen kann.

Asynchron Verarbeitung mit zwei Kanten

Bei der zweiten Variante werden die einzelnen Datenstränge getrennt voneinander verarbeitet. Der Verbindungs- und der Trennungsknoten dienen als Synchronisationsknoten, in dem die Metadatentabelle mit der Datentabelle abgeglichen und angepasst wird. Die Datentabelle bleibt dabei unverändert, während von der Metadatentabelle die Reihenfolge der Zeilen so angepasst werden, so dass die Metadaten in Zeile 25 die Daten aus Zeile 25 beschreiben. Sollte es keine Metadaten zu einem Datensatz geben, so wird eine Leerzeile eingefügt.

In diesem Fall wird also nicht das „Simple Merge“-Pattern verwendet, sondern eine Kombination aus dem „Synchronize“- und dem „Parallel Split“ - Pattern (AND-split). Das bedeutet, dass innerhalb des Verbindungsknotens die Datensätze zwar synchronisiert werden, anschließend aber wieder auf zwei verschiedene Tabellen getrennt werden.

Auch bei dieser Variante können für die Verarbeitung die existierenden KNIME-Knoten verwendet werden. Diese Knoten können in beiden Pipes eingebaut werden. Wenn Zeilen rausgefiltert werden sollen, macht dies nur in dem Daten-Strang Sinn, denn eine Filterung im Metadaten-Strang führt bei der nächsten Synchronisation nur zu einer Leerzeile.

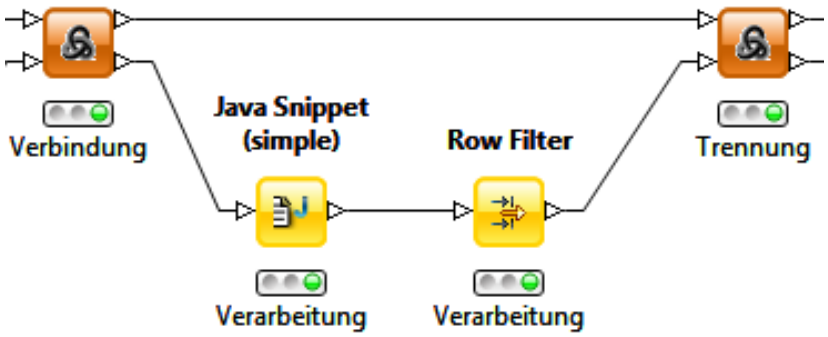

Abbildung 15: KNIME-Workflow-Ausschnitt für die asynchrone Verarbeitung mit zwei Kanten

Für mehr Flexibilität könnten die Synchronisationsknoten erweitert werden, so dass der Benutzer in der Knotenkonfiguration auswählen kann, welche Spalte als Grundlage für die Synchronisation dienen soll. Damit könnte das Problem mit den Leerzeilen umgangen werden, sollte nur eine Zeilenfilterung im Metadaten-Strang durchgeführt werden.

Unvorteilhaft ist das Anzeigen beider Tabellen nebeneinander. Während es bei wenigen Datensätzen noch übersichtlich erscheint, kann schnell in der Zeile verrutscht werden, wenn es sich um mehrere hundert Datensätze handelt. Dem könnte durch farbliche Abhebung oder automatischem Scrollen bei beiden Tabellen entgegen gewirkt werden, aber das würde bei ersteren im Kontrast zu möglichen Einfärbungsknoten stehen und das gemeinsame Scrollen würde ein Eingriff in den KNIME-Kern bedeuten, was beides nicht wünschenswert ist.

Um die beiden Tabellen vereinen zu können, müsste die Metadaten-Tabelle durch einen neuen Knoten geleitet werden, welcher die IRI zu einem Literal umwandelt (siehe CleanNames-Knoten im Kapitel 5.3 „Sonstige Knoten“). Anschließend könnte der „Column Appender“-Knoten verwendet werden, da die Reihenfolge und auch Zeilenlänge bereits übereinstimmen.

Synchrone Verarbeitung mit zwei Kanten

Bei der synchronen Verarbeitung mit zwei Kanten besitzen alle Knoten Ein- und Ausgänge für beide Pipes. Der Verbindungsknoten hat die gleiche Funktion wie im vorherigen Beispiel, wo dieser als Synchronisationsknoten verwendet wird. Auch hier würde mit zwei separaten Tabellen gearbeitet werden. Der Vorteil dieses Vorgehens ist, dass alle Verarbeitungsschritte gleichzeitig auf beiden Pipes ausgeführt werden. Wenn also in der Metadaten-Tabelle Zeilen rausgefiltert werden, dann werden auch die entsprechenden Daten-Zeilen entfernt.

Abbildung 16: KNIME-Workflow-Ausschnitt für die synchrone Verarbeitung mit zwei Kanten

Da die beiden Stränge nie zu einer Pipe zusammengefügt werden, ist auch eine Trennung nicht notwendig und auch keine expliziten Synchronisationen. Ein großer Nachteil an dieser Umsetzung ist, dass die bisherigen Knoten von KNIME entweder angepasst werden müssten oder es müssten komplett neue Knoten entwickelt werden.

Fazit – Auswahl der Architektur für die Realisierung

Aufgrund der Vorgabe, keine Manipulationen am KNIME-Kern vorzunehmen und es weiterhin möglich sein soll die bisherigen Knoten zu verwenden, wird die synchrone Verarbeitung mit zwei Kanten nicht umgesetzt.

Wegen der aktuellen Tabellenstruktur bietet sich das Verwenden einer gemeinsamen Tabelle für Daten und Metadaten an, weswegen die synchrone Verarbeitung mit nur einer Kante komplett umgesetzt wird. Auch bezüglich der SPARQL-Schnittstelle empfiehlt es sich diese Variante zu wählen. Somit können die abgerufenen Metadaten direkt in die Tabelle übernommen werden.

Die asynchrone Verarbeitung mit zwei Kanten besitzt ebenfalls das Potenzial von der Community anerkannt zu werden, vorausgesetzt es wird eine Möglichkeit entwickelt die Verknüpfung zwischen den beiden Tabellen besser Erkennbar dazustellen. Diese Variante wird teilweise umgesetzt, wobei die Synchronisation nur einen einfachen Findungsalgorithmus und auch nur die bisherige Tabellen-Darstellungsart erhält.

4. Konzeption der KNIME-Knoten

Um Daten/Metadaten-Verbände verarbeiten zu können, müssen folgende Schritte in KNIME implementiert werden:

1. Integration von RDF in KNIME
2. Erzeugen und Trennen von Daten/Metadaten-Verbänden (siehe Kapitel 3.2)
3. Search-Engine zum Lokalisieren der URIs
4. Generierung von SPARQL-Anfragen

Eine genauere Analyse folgt in den nächsten Unterkapiteln.

4.1 Integration von RDF in KNIME

Bisher bietet KNIME keine Möglichkeit, RDF-Daten in KNIME zu integrieren. Damit dies in Zukunft möglich ist, muss ein Knoten erstellt werden, welcher RDF Dateien korrekt parsen und in ein für KNIME passendes Format umwandeln kann. Da es bereits hervorragende Parser für RDF-Daten gibt, wird auf eine Neuentwicklung verzichtet. Prädestiniert ist Apache Jena [51], da es ebenfalls in Java entwickelt wurde und somit eine leichte Implementierung ermöglicht, es ein Open-Source Framework ist und weil es auch eine SPARQL Integration enthält. Dies und noch weitere Features (wie Kompatibilität mit mehreren Präsentationsarten wie RDF/XML, Turtle, N-Triples oder RDFa) sind wahrscheinlich auch der Grund, warum Apache Jena eine so hohe Popularität hat.

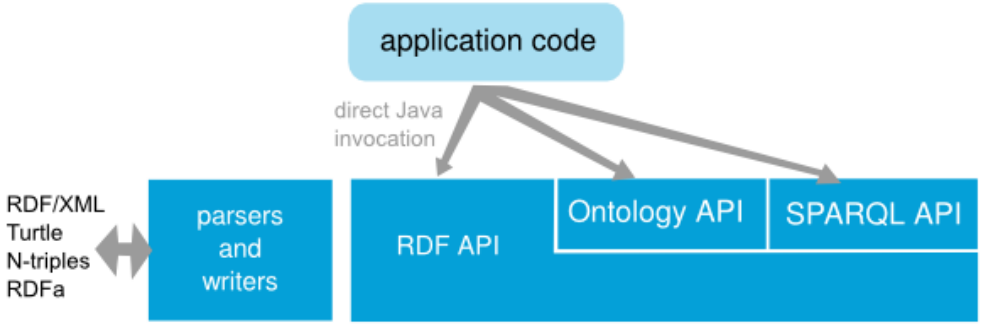

Abbildung 14: Ausschnitt aus der Apache Jena – Architektur [52]

Abbildung 15 zeigt den relevanten Ausschnitt aus der Jena-Architektur. So ist es möglich direkt im Java Quellcode auf die RDF-, Ontology- und SPARQL-API zuzugreifen. Zudem bietet die Software Parser und Writer an, um RDF/XML, Turtle, N-triples oder RDFs Dateien einzulesen oder raus zu schreiben.

Wird eine RDF-Datei mit Apache Jena eingelesen, wird dies in ein *Model* transferiert. Dieses Model enthält den kompletten Graphen der Datei. Dieser Graph besteht aus mehreren *Statement*-Objekten, welche die RDF-Triple (*Subject*, *Predicate* und *Object*) repräsentieren.

Die einzelnen Triple-Bestandteile können wiederum eine *Resource* (IRI) oder ein *Literal* (number, string, dates, etc.) sein.

Der folgende Auszug aus einer Turtle-Datei zeigt zwei RDF-Triple. In Jena wären dies zwei verschiedene *Statement*-Objekte:

```
<http://de.dbpedia.org/resource/Alfred_Hitchcock> <http://dbpedia.org/ontology/birthDate>
"1899-08-13"^^<http://www.w3.org/2001/XMLSchema#date> .

<http://de.dbpedia.org/resource/Alfred_Hitchcock>
<http://dbpedia.org/ontology/birthPlace> <http://de.dbpedia.org/resource/London> .
```

Abbildung 15: Auszug aus einer Turtle-Datei von der DBpedia [39]

Die Unterteilung des Statements in Subjekt, Prädikat und Objekt zeigt der nächste Auszug:

```
<http://de.dbpedia.org/resource/Alfred_Hitchcock> <http://xmlns.com/foaf/0.1/surname>
"Hitchcock"@de .
```

Abbildung 16: Unterteilung eines Turtle-Statements in Subjekt, Prädikat und Objekt

Während das Subjekt und das Prädikat IRIs (Resource) sind, ist das Objekt ein Literal vom Typ String.

Die Darstellung von Informationen ist in KNIME tabellarisch, weswegen auch die RDF-Daten in die Tabellenstruktur überführt werden müssen. Dafür werden die Subjekte als RowID verwendet, das Prädikat bildet den Spaltenheader und das Objekt befindet sich in der Zelle, wo Reihe und Spalte sich überschneiden. Für die drei oben genannten Statements würde die Tabelle wie folgt aussehen:

Row ID	<http://dbpedia.org/ontology/birthDate>	<http://dbpedia.org/ontology/birthPlace>	<http://xmlns.com/foaf/0.1/surname>
<http://de.dbpedia.org/resource/Alfred_Hitchcock>	"1899-08-13"^^<http://www.w3.org/2001/XMLSchema#date>	<http://de.dbpedia.org/resource/London>	"Hitchcock"@de

Tabelle 7: Überführte KNIME-Tabelle, basierend auf die drei Statements aus Abbildung 16 & 17

Immer wenn es ein Subjekt bereits gibt, wird diese Reihe verwendet. Sollte ein neues Subjekt vorkommen, was noch nicht als RowId deklariert ist, so bekommt dies eine neue Zeile. Selbiges gilt für das Prädikat, nur, dass hier der Spaltenheader betrachtet wird.

Für den RDF-Writer wird ebenfalls Jena verwendet. Dazu wird aus der KNIME-Tabelle ein Jena-Graph erstellt (nach dem gleichen Schema, nur in umgekehrter Reihenfolge) und daraus kann dann wieder eine RDF-Datei erstellt werden.

4.2 Search-Engine zum Lokalisieren der IRIs

Es wird selten der Fall eintreffen, dass in dem Datendokument die IRIs enthalten sind. Viel wahrscheinlicher ist es, dass dort lediglich die Personennamen, Orte oder Gegenstände vermerkt sind, für die die Metadaten gewünscht werden. Zwar steckt hinter der Namensgebung der IRIs ein bestimmtes Schema, aber spätestens wenn zwei Personen den gleichen Namen haben oder Sonderzeichen enthalten sind, kann es zu Findungsproblemen kommen. Zwar kann keine Suchmaschine für das richtige Ergebnis garantieren, aber die Wahrscheinlichkeit, die richtigen Daten zu erhalten, kann damit erhöht werden.

Beispiel: Sonderzeichen

Für Jerome Boateng würden wir ohne eine Suchmaschine die IRI wie folgt schreiben:
http://dbpedia.org/resource/Jerome_Boateng

DBpedia lookup liefert uns hingegen folgende IRI:

http://dbpedia.org/resource/Jérôme_Boateng

Über die Webseite würden wir bei beiden URIs die gleichen Daten erhalten. Starten wir hingegen eine Abfrage über die SPARQL-Schnittstelle erhält man nur über den Link mit den Sonderzeichen alle Metainformationen. Als Ergebnis für die erste IRI erhalten wir lediglich das <http://dbpedia.org/ontology/wikiPageRedirects> mit den Verweis auf die zweite IRI.

Um dem zu entgehen, muss eine Such-Engine verwendet werden.

DBpedia Lookup

DBpedia Lookup [54] ist, wie der Name vermuten lässt, eine Such-Engine innerhalb der DBpedia. Anhand eines Strings sucht die Engine nach passenden Ergebnissen in der Datenbank und liefert das Ergebnis im XML-Format. Dabei werden zwei Varianten zur Verfügung gestellt, entweder wird direkt nach einem Schlüsselwort (Keyword) gesucht oder nach einem Prefix. Eine Anfrage kann über eine zur Verfügung stehenden API via URL durchgeführt werden. Diese sieht wie folgt aus:

```
http://lookup.dbpedia.org/api/search.asmx/<API>?<parameters>
```

Abbildung 17: Grundstruktur der DBpedia Lookup - URL

Für <API> wird die Art der Suche angegeben, mit KeywordSearch wird der genaue String, welcher übergeben wird gesucht und mit PrefixSearch werden alle Ergebnisse geliefert, welche mit den entsprechenden String beginnen. Als Parameter sind folgende drei Keys gültig:

- QueryString: dies ist der String nach dem gesucht werden soll. Je nach Art der Suche ist dies also das Schlüsselwort oder der Präfix.
- QueryClass: damit können die Suchergebnisse nach einer bestimmten Ontologie (Place, Person, Work, Species, Organisation) gefiltert werden.
- MaxHits: damit kann die Anzahl der Ergebnisse eingeschränkt werden.

Beispiel: Wir wollen die Person Alfred Hitchcock finden und auch nur ein Ergebnis bekommen. Die Anfrage würde dementsprechend so aussehen:

```
http://lookup.dbpedia.org/api/search.asmx/KeywordSearch?QueryClass=person&QueryString=alfred%20hitchcock&MaxHits=1
```

Abbildung 18: Beispiel-URL für die Verwendung der DBpedia Lookup - API

Um Ergebnisse zu erhalten, sollte der QueryString immer angegeben werden. QueryClass und MaxHits sind hingegen optional. Ohne QueryClass werden Suchergebnisse aus allen Kategorien angezeigt, so erscheint bei der Suche nach Berlin auch die „Free University of Berlin“. Wenn MaxHits weggelassen wird, dann werden die ersten fünf Ergebnisse angezeigt.

Die Reihenfolge der Ergebnisse wird anhand der Häufigkeit der Referenzen festgelegt. Dies hat den Effekt, dass bei der Schlüsselsuche „Berlin“ als erstes Suchergebnis die deutsche Hauptstadt angezeigt wird, da dieser Artikel fast 19000mal referenziert wird. Die weiteren Ergebnisse sind die olympischen Spiele von 1936 (ca. 2400 Referenzen) und der Axis powers (ca. 2200 Referenzen).

So ist in den meisten Fällen das korrekte Ergebnis zu erwarten, aber eine Garantie ist dafür nicht gegeben, vor allem wenn nach Orten gesucht wird die namentlich öfters Vorkommen. Als gutes Beispiel ist „Neukirchen“, welches einer der häufigsten Ortsnamen in Deutschland ist.

Die Vorteile dieser Such-Engine sind, dass der Umgang mit Sonderzeichen beherrscht wird (egal ob nach Jérôme Boateng oder Jerome Boateng gesucht wird, es wird die korrekte URI gefunden), dass auch Kurzformen und Abkürzungen erkannt werden (USA wird weitergeleitet zu United States) und dass die API mit Hilfe von Post- und Get –Anfragen in Java integriert werden kann.

Okkam

Die Suchmaschine Okkam [55] verfolgt einen ähnlichen Ansatz wie DBpedia lookup, nur dass dort nicht nur die Daten von Wikipedia als Grundlage dienen, sondern noch andere Quellen, deren Daten dem Standards des Semantik Webs unterliegen. Unterstützt wird das Projekt von der Okkam SRL und der Universität in Trento (Italien).

Bei der Suche wird nur nach Schlüsselwörtern gesucht, eine Präfix-Suche wird nicht unterstützt. Um die Suchergebnisse etwas einzugrenzen, stehen auch hier bestimmte Ontologin zur Verfügung (Any, Person, Artifact, Organization, Artifact type, Location, Event und Other). Die Suchergebnisse enthalten neben den Namen und dem Typ auch typabhängige Informationen, die URI zur Datenquelle und auch alternative URIs.

Typabhängige Informationen (sind optional) können folgende sein:

- Location: latitude, longitude, timezone, population, country, location type
- Person: first name, last name, full name, birthdate, domain, occupation, tag
- Organization: city, activity sector, country, street adress, foundation date

Das Ergebnis für Alfred Hitchcock würde wie folgt aussehen:

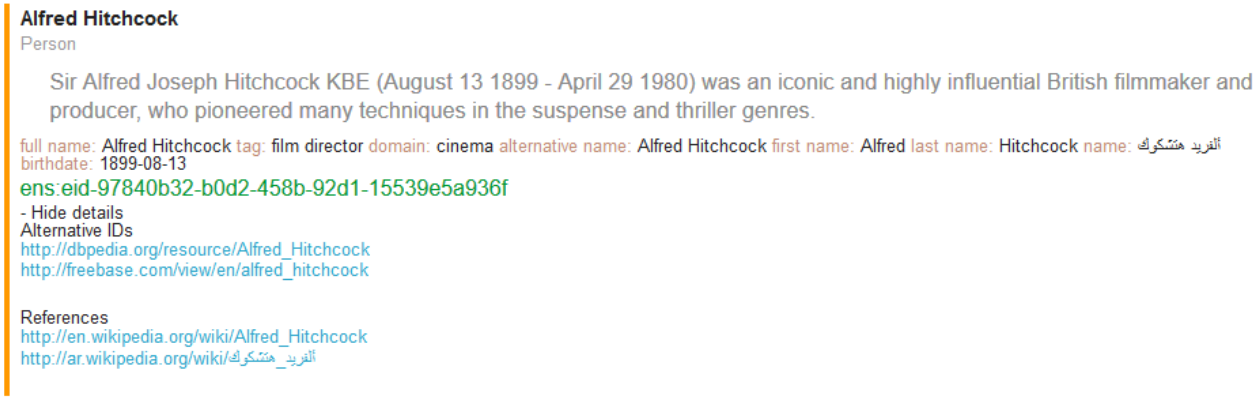

Abbildung 19: Suchergebnis für "Alfred Hitchcock" von der Okkam-Suchmaschine [57]

Die Suche kann über mehrere Wege durchgeführt werden, entweder über die Webseite [57], über eine SOAP- oder REST-Schnittstelle oder mit Hilfe des Java Okkam Client [56]. Für die Integration in KNIME würde sich der Java Client am besten eignen. Leider wird für den Verbindungsaufbau ein Zertifikat benötigt was ein Benutzeraccount voraussetzt, was für ein Knoten unvorteilhaft ist.

Über die REST-Schnittstelle ist kein Zertifikat notwendig und das Ergebnis ist im JSON-Format. Die Anfrage [58] besteht aus zwei Teilen, der QUERY und den METADATA. In der QUERY wird angegeben wonach gesucht wird (dies kann ein Schlüsselwort oder spezielle Attribute) und in der METADATA wird angegeben wo gesucht werden soll (entityType). Die Suche nach Alfred Hitchcock würde dann wie folgt aussehen:

```
http://api.okkam.org/okkam-core/jsearch?q=QUERY{name="Alfred Hitchcock"}
METADATA{entityType=person}
```

Abbildung 20: Beispiel-URL für die Verwendung der Okkam - API

Sindice

Sindice [59] ist ebenfalls eine Suchmaschine und ein gemeinsames Forschungsprojekt der DERI [60], der Fondazione Bruno Kessler [61] und der OpenLink software [62]. Früher wurde das Projekt auch von Okkam unterstützt, weswegen so einige Gemeinsamkeiten erkennbar sind. Der Funktionsumfang von Sindice ist gegenüber den anderen Suchmaschinen sehr viel größer, aber dadurch auch komplexer.

Neben der normalen Keyword-Suche kann auch nach bestimmten Pattern gesucht werden. Auch Filter (url, domain, class, predicate, ontology, format, date) können verwendet werden um die Suchergebnisse zu präzisieren. Als Ausgabeformate gibt es neben JSON auch ATOM und RDFXML. [63]

Das Verwenden der vielen Filter ist bei der Suche auch notwendig, denn ohne Filter ist die Reihenfolge der Suchergebnisse doch überraschend. Auf der Suche nach „Berlin“ ist das erste Ergebnis eine RDF Datei mit den Titel „RDF Description of Berlin“ von der FU Berlin, gefolgt vom „song by Irving Berlin“ und dem Bahnhof „Anhalter Bahnhof“, beide von der DBpedia.

Fazit

In punkto Vielseitigkeit und Datenbasis ist Sindice die umfangreichste Suchmaschine. Damit der KNIME-Nutzer das gewünschte Ergebnis bekommt müsste dieser vieles Konfigurieren. Um Fehleingaben zu vermeiden, müssten dem Benutzer dann Auswahlfelder zur Verfügung gestellt werden, welche aber gewartet werden müssten, da neue Funktionen ansonsten nicht genutzt werden könnten (zum Beispiel wenn eine neue Ontologie hinzugefügt wird oder eine neue Datenquelle). Alternativ könnten Textfelder verwendet werden, was aber voraussetzt, dass der Nutzer die genaue Syntax kennt.

Dieses Problem gibt es bei Okkam nicht, da die Filtermöglichkeiten überschaubar sind, aber diese richtig angewandt, zu guten Ergebnissen führen. Hier stellt sich aber die Frage wie qualitativ hochwertig die anderen Quellen im Vergleich zur DBpedia sind. Zudem ist verwunderlich, warum das Projekt immer noch den Beta-Status besitzt, obwohl es die Software schon seit 2009 gibt. Auch darf die Software nur für den privaten Gebrauch genutzt werden, was nicht bei jedem KNIME-Nutzer gewährleistet werden kann.

Den geringsten Umfang hat DBpedia Lookup. Hier kann neben zweier Suchvarianten, sowie der Filterung nach der Ontologie, nur noch die Anzahl der Ergebnisse angegeben werden. Zudem beschränkt sich die Suche nur auf die DBpedia.

Da sich diese Abschlussarbeit nur mit der DBpedia befasst und der Benutzer nur so viel konfigurieren soll wie nötig, fällt die Entscheidung auf DBpedia lookup.

4.3 Generierung von SPARQL-Anfragen

Daten und deren Informationen sind selten final, sondern neigen dazu, ergänzt und verbessert zu werden. Zwar umfasst das Semantik Web bereits mehrere Millionen Metadaten, aber nicht für alle Objekte sind alle bereits vorgesehenen Informationen enthalten. Einige können auch erst in der Zukunft ergänzt werden, wie etwa der Todestag. Zudem können sich auch bereits integrierte Werte ändern, zum Beispiel wenn jemand heiratet kann sich der Nachname ändern. Aufgrund dieses Datenwandels empfiehlt es sich, mit den Live-Daten zu arbeiten, anstatt mit statischen Dateien, welche zuvor heruntergeladen werden müssen.

Eine komfortable Schnittstelle, um an die dynamischen Metadaten zu gelangen, ist die SPARQL-Schnittstelle. Viele Datenquellen, wie etwa DBpedia, bieten eine solche Schnittstelle an. Für die Integration der Metadaten in KNIME, wird eine Technologie benötigt, welche SPARQL-Anfragen erzeugen, ausführen und die Ergebnisse interpretieren kann. Eine solche Technologie ist Apache Jena, welche bereits für das integrieren von RDF in KNIME verwendet wurde.

SPARQL ist vom groben Aufbau her SQL sehr ähnlich. Für das Abfragen von Informationen benutzen wir das Schlüsselwort SELECT, gefolgt von den Spalten, welche wir angezeigt haben wollen. Im WHERE-Bereich wird die Suche spezifiziert.

Ziel der SPARQL-Anfragen soll es sein, für jedes Objekt, welches durch eine IRI dargestellt wird, sollen alle vorhandenen Informationen (Property) abgerufen werden.

Die Suche wird erneut mithilfe eines Triple durchgeführt. Dieses besteht wie bei RDF aus einem Subjekt, Prädikat und Objekt. Das Subjekt ist erneut IRI des Objektes. Als Prädikat suchen wir alle Properties die dieses Subjekt besitzt. Das Objekt ist wieder das zugewiesene Literal oder eine weitere IRI auf die verwiesen wird.

Eine vollständige Anfrage sieht wie folgt aus:

```
SELECT DISTINCT ?property ?o WHERE {
      ?s ?property ?o .
      FILTER (?s = <http://dbpedia.org/resource/Alfred_Hitchcock>) .
}
```

Abbildung 21: SPARQL-Abfrage an die DBpedia, um alle Properties mit den Objekten, für Alfred Hitchcock zu erhalten

Mit DISTINCT werden alle Tripel mit identischen Prädikaten & Objekt-Kombinationen für das gleiche Subjekt entfernt (zur Vermeidung von Redundanzen). Unser Ergebnis soll die Property und das dazugehörige Objekt anzeigen (?property ?o). In der WHERE-Klausel geben wir das gesuchte Tripel an (?s ?property ?o). Um nur die Ergebnisse von unserem gewünschten Subjekt zu erhalten, weisen wir dem ?s mit Hilfe eines FILTERS das Subjekt zu (in diesem Beispiel die IRI von Alfred Hitchcock).

Die Ergebnisse sehen wie folgt aus (Auszug):

Property	O
http://xmlns.com/foaf/0.1/name	"Alfred Joseph Hitchcock"@en
http://dbpedia.org/property/birthDate	1899-08-13+02:00
http://dbpedia.org/property/birthName	"Alfred Joseph Hitchcock"@en
http://dbpedia.org/property/birthPlace	"Leytonstone, London, England"@en

Tabelle 8: Ergebnisstabelle von der SPARQL-Abfrage aus der vorigen Abbildung

Filterung nach IRIs

Neben den oben gezeigten Properties mit dem Namespace xmlns.com/foaf/ und dbpedia.org, werden auch Ergebnisse aus dem www.w3.org und purl.org Namespace angezeigt. Von der xmlns.com/foaf/ werden folgende Properties angeboten (nicht jedes Objekt hat alle Properties):

account | accountName | accountServiceHomepage | age | aimChatID | based_near | birthday | currentProject | depiction |depicts | dnaChecksum | familyName | family_name | firstName | focus | fundedBy | geekcode | gender | givenName | givenname |holdsAccount | homepage | icqChatID | img | interest | isPrimaryTopicOf | jabberID | knows | lastName | logo | made | maker | mbox |mbox_sha1sum | member | membershipClass | msnChatID | myersBriggs | name | nick | openid | page | pastProject | phone | plan |primaryTopic | publications | schoolHomepage | sha1 | skypeID | status | surname | theme | thumbnail | tipjar | title | topic | topic_interest |weblog | workInfoHomepage | workplaceHomepage | yahooChatID |

Abbildung 22: Alle Properties von der FOAF [40][64]

Für Alfred Hitchcock existieren nur die Informationen für depiction, givenName, isPrimaryTopicOf, name und surname.

Von dbpedia.org werden in der aktuellen Version (3.9) insgesamt 2333 verschiedene Properties verwendet, aber nicht jedes Objekt wird durch alle Properties beschrieben. Welche Properties verwendet werden liegt nämlich daran, welchen Klassen das Objekt zugeordnet ist. Insgesamt gibt es 529 Klassen und jede Klasse wird durch einige Properties beschrieben [42][43]. So hat Alfred Hitchcock die Klassen Agent und Person zugewiesen bekommen und kann demnach auch nur deren Eigenschaften erhalten. Auch hier gilt, dass nicht jedes Property einen Wert zugewiesen haben muss.

Purl.org liefert eine kurze Beschreibung über das Objekt und liefert Informationen zu Kategorien (http://purl.org/dc/terms/subject), denen das Objekt angehört. So befindet sich Herr Hitchcock zum Beispiel in den Gruppen American_film_directors (http://dbpedia.org/resource/Category:American_film_directors) und Edgar_Award_winners (http://dbpedia.org/resource/Category:Edgar_Award_winners). Da sich die Beschreibung des Objektes bereits in der DBpedia befindet und die Kategorisierung lediglich der Gruppierung von Objekten dient und weniger der Beschreibung des Objektes, werden die Properties von der purl.org herausgefiltert.

Als vierte Namespace-Quelle dient die w3.org. Die Property http://www.w3.org/1999/02/22-rdf-syntax-ns#type dient der Klassifizierung des Objektes. Hier werden alle Klassen von allen Namespaces angegeben, dem dieses Objekt angehört, nicht nur von den bisher angegebenen Quellen, sondern auch von umbel.org, schema.org, etc. Aufgrund der vielen verschiedenen Sprachen mit verschiedenen Sonderzeichen und möglichen Abkürzungen, gibt es auch noch das Property http://www.w3.org/2002/07/owl#sameAs. Dieses liefert alle möglichen Resource-URIs für dieses Objekt. Auch bietet die w3.org die Beschreibung (http://www.w3.org/2000/01/rdf-schema#comment) und den Namen (http://www.w3.org/2000/01/rdf-schema#label) des Objektes in mehreren Sprachen. Da wir diese Informationen aber bereits von der DBpedia erhalten, wird die w3.org als Property-Quelle ebenfalls nicht berücksichtig.

Um die oben beschriebenen Filterungen umzusetzen, wird die SPARQL-Abfrage um folgende Zeile ergänzt:

```
SELECT DISTINCT ?property ?o WHERE {
      ?s ?property ?o .
      FILTER (?s = <http://dbpedia.org/resource/Alfred_Hitchcock>) .
      FILTER regex(?property, 'http://xmlns.com/foaf|http://dbpedia.org*') .
}
```

Abbildung 23: SPARQL-Abfrage an die DBpedia, um alle Properties mit den Objekten, für Alfred Hitchcock zu erhalten, gefiltert nach deren Datenquelle, so dass nur noch die Ressourcen von der DBpedia oder der FOAF vorhanden sind

Um die Abfrage mit Jena auszuführen, wird folgender Java-Code verwendet:

```
String queryString=
      "SELECT DISTINCT ?property ?o WHERE {"+
      "?person ?property ?o ."+
      "FILTER (?person = <"+uri+">) ."+
      "FILTER regex(?property, 'http://xmlns.com/foaf|http://dbpedia.org*') ."+
      "}";

Query query = QueryFactory.create(queryString);
QueryExecution qexec =
      QueryExecutionFactory.sparqlService("http://dbpedia.org/sparql", query);

ResultSet results = qexec.execSelect();
```

Abbildung 24: Integration der SPARQL-Anfrage aus Abbildung 24, in Apache Jena (Java)

Die SPARQL-Anfrage wird direkt in einen String gespeichert, wobei die IRI variabel ist. Dieser String wird mit Hilfe der QueryFactory zu einer Query umgewandelt. Mit der QueryExecutionFactory wird die Query mit der SPARQL-Schnittstelle verknüpft, so dass diese ausführbereit ist. Die von der dbpedia ist unter `http://dbpedia.org/sparql` erreichbar. Der SELECT wird durch den Aufruf von qexec .execSelect() ausgeführt und liefert das Ergebnis (ResultSet). Dies kann durchiteriert werden und liefert somit die einzelnen Zeilen (Solution).

Verhalten bei doppelten Properties

Property	O
http://xmlns.com/foaf/0.1/name	"Alfred Joseph Hitchcock"@en
http://xmlns.com/foaf/0.1/name	"Sir Alfred Hitchcock"@en
http://dbpedia.org/ontology/birthPlace	http://dbpedia.org/resource/Essex
http://dbpedia.org/ontology/birthPlace	http://dbpedia.org/resource/Leytonstone
http://dbpedia.org/ontology/birthPlace	http://dbpedia.org/resource/London

Tabelle 9: Auszug aus den Anfrageergebnissen von Abbildung 24, zum Aufzeigen von doppelten Properties

Leider kommt es regelmäßig vor, dass einige Properties Mehrfachzuweisungen haben. Da jeder Benutzer dies gerne anders handhaben möchte, werden diesem mehrere Möglichkeiten angeboten:

1. Das Objekt was in der Abarbeitungsreihenfolge zuerst verarbeitet wird, wird in KNIME übernommen, alle weiteren Properties mit dem gleichen Namen werden ignoriert.
2. Egal ob es bereits einen Eintrag in der entsprechenden Zelle gibt, das Objekt wird eingetragen. Somit werden mögliche vorherige Werte überschrieben und das zuletzt auftretende Objekt wird verwendet.
3. Sollte bereits ein Wert existieren, so wird an diesem am Ende ein Komma gesetzt und der neue Eintrag wird angehängt.
4. Das zuerst aufkommende Objekt eines Property wird wie gewohnt in die dafür vorgesehene Zelle geschrieben. Für jedes gleiche Property wird eine neue Spalte erzeugt mit dem Property-Namen, einem Leerzeichen und einer laufenden Nummer (zum Beispiel: Name, Name 1, Name 2, …)

Filterung nach der Sprache

Einige Properties kommen doppelt vor, da sich die Objekte in mehreren Sprachen verfasst sind. Dies kommt zum Beispiel bei den Namen oder der Objektbeschreibung vor. Erkennbar sind diese Fälle an dem „@“-Zeichen, gefolgt vom Kürzel der Sprache, am Ende eines Objektes (nur bei Literalen). Für diesen Fall wird nur das Objekt mit der ausgewählten Sprache in die KNIME-Tabelle übernommen.

Property	O
http://dbpedia.org/ontology/abstract	"Sir Alfred Joseph Hitchcock fue un director de cine británico."**@es**
http://dbpedia.org/ontology/abstract	"Sir Alfred Joseph Hitchcock war ein britischer Filmregisseur und Filmproduzent. […]"**@de**
http://dbpedia.org/ontology/abstract	"Sir Alfred Joseph Hitchcock, KBE (13 August 1899 – 29 April 1980) was an English film director and producer. […]“**@en**
http://dbpedia.org/ontology/abstract	"サー・アルフレッド・ジョゼフ・ヒッチコック[…]“**@ja**
http://dbpedia.org/ontology/abstract	"阿尔弗雷德·希区柯克爵士，[…]“**@zh**

Tabelle 10: Auszug aus den Anfrageergebnissen von Abbildung 24, zum Aufzeigen von doppelten Properties in verschiedenen Sprachen

4.4 Zusammenfassung der zu implementierenden Knoten

Für die Integration von RDF in KNIME werden zwei Knoten entwickelt, der RDFReader und der RDFWriter. Die zugrunde liegende Technologie zum Einlesen und Erzeugen der RDF-Dateien ist Apache Jena.

Beim Erzeugen und Trennen von Daten/Metadaten-Verbänden werden drei Knoten benötigt. Dieses wird der Matching-Knoten, welcher für die Datenfusion von den beiden Datenquellen zuständig ist, der Separating-Knoten zum Trennen der Daten und für die alternative Umsetzung erneut ein Matching-Knoten, welcher für die Synchronisation der Daten zuständig ist.

Zum Lokalisieren der URIs wird ein URIConverter-Knoten entwickelt, welcher aus einem Literal, eine passende URI erzeugt. Dafür wird die Suchmaschine DBpedia Lookup verwendet.

Für das Abfragen von Online-Daten wird ein SPARQL-Knoten entwickelt, welcher ebenfalls Apache Jena verwendet.

5. Implementierung & Validierung

In diesem Kapitel geht es um die Implementierung und Validierung der geplanten Knoten. Zuerst wird die Entwicklungsumgebung vorgestellt, auf der die Implementierung stattfand. Anschließend werden anhand von realistischen Benutzerszenarien die implementierten Knoten vorgestellt und validiert.

5.1 Entwicklungsumgebung

Die Entwicklung fand auf einem PC mit dem Betriebssystem Windows 8.1, in der 64-Bit Version statt. Als Entwicklungsumgebung wurde Eclipse Indigo (Version 3.7.2) verwendet. Diese kann inklusive KNIME auf der KNIME Homepage heruntergeladen werden [65]. Zum Start der Entwicklung war die offizielle Version von KNIME die 2.8. Zudem wurde Java 7 verwendet.

Zudem wurde das Projekt auch auf einem Laptop getestet. Dieser hatte als Betriebssystem Windows 7 in der 64-Bit Version. Auch hier wurde die Eclipse Indigo Version (3.7.2) verwendet. Diese Version ist zum Zeitpunkt der Abgabe weiterhin auf der KNIME Homepage erhältlich. Da die Tests später angefangen haben als die Entwicklung, wurde hier bereits Version 2.8.1 von KNIME verwendet. Die Java-Version war ebenfalls JRE7.

Für das Ausführen von KNIME wurden folgende Dinge in der Running-Konfiguration verwendet:

```
Program arguments:
-os ${target.os} -ws ${target.ws} -arch ${target.arch} -nl ${target.nl} –consoleLog

VM arguments:
-ea -Xmx512M -XX:MaxPermSize=192m
```

Abbildung 25: Notwendige Konfiguration in Eclipse. Bei Bedarf kann auch die Speicherkapazität im Arbeitsspeicher erhöht werden.

5.2 Validierung durch Benutzerszenarien

Die folgenden Szenarien wurden umgesetzt und sollen die Funktionalität unter realistischen Anforderungen aufzeigen.

5.2.1 Verarbeitung von RDF-Daten in KNIME

Mit diesem Szenario soll KNIME den Umgang mit RDF-Dateien erlernen. Das RDF Format wird sehr häufig verwendet, um Metadaten zu speichern. Vor allem die 3-Tripel Schreibweise wird verwendet, da damit eine genaue Zuordnung zwischen mehreren Objekten erzeugt werden kann. Da KNIME das RDF-Format nicht unterstützt, ist das Ziel einen RDF-Reader zu entwickeln, welcher eine lokale Datei einliest und in das KNIME bekannte Tabellenformat umwandelt. In diesem Beispiel wird eine Turtle-Datei verwendet. Nach einer beliebigen Verarbeitung soll wieder eine RDF-Datei erstellt werden.

Gegeben:

Die Metadaten beschreiben Personen und befinden sich in einer ttl-Datei. Diese gibt es auf der Seite von DBpedia zum Herunterladen [39] und beinhaltet Personendaten wie zum Beispiel Name, Geburtsort und –tag. Da die Datei über 6,7 Millionen Tripel umfasst und über 750mb groß ist, werden für das Beispiel nur die ersten 5000 Tripel verwendet. Die Kürzung der Daten wird durchgeführt, da die Verarbeitungszeit sonst zu lang ausfällt[1] und wegen der Begrenzung des Arbeitsspeichers kann es zu einem Fehler kommen (OutOfMemoryError).

Ausgabe:

Nach den Verarbeitungsschritten soll am Ende des Workflows eine ttl-Datei erstellt werden.

Verarbeitungsschritte:

1. Der RDF-Reader liest die ttl-Datei mit den Personendaten ein und wandelt die Triple in die übliche KNIME-Tabellenform um.
2. Die Datensätze werden mit einem Row Filter (bereits in KNIME enthalten) gefiltert, so dass nur noch Personen angezeigt werden, die in Berlin geboren sind.
3. Nur die gefilterten Meta-Daten, von den in Berlin geborenen Personen, werden in eine neue ttl-Datei geschrieben.

[1] eine Datei mit ca. 100.000 Tripeln dauert auf einem aktuellen Standard-PC über fünf Minuten

Realisierung

Abbildung 26: KNIME-Workflow für das Einlesen, Verarbeiten und Ausgeben von RDF-Dateien

RDFReader

Der RDFReader liest die ttl-Datei ein und generiert aus den eingelesenen Tripeln eine KNIME-Tabelle. Dazu wird in der Knoten-Konfiguration ausgewählt, wo sich die Datei befindet. Das Einlesen übernimmt Jena, was aus den Daten einen RDF-Graph generiert. Dieser Graph wird in mehreren Schritten verarbeitet. Im ersten Durchlauf werden die Subjekte und Prädikate gesammelt und in zwei getrennten Listen gespeichert, wobei jeder Eintrag einmalig sein muss. Die Subjekte sind auch gleichzeitig die Row IDs (zum Beispiel „http://de.dbpedia.org/resource/Tom_Hanks“) und die Prädikate sind die Spaltenköpfe (zum Beispiel: „http://xmlns.com/foaf/0.1/givenName“). Nach diesem Durchlauf kann die Größe des zweidimensionalen Arrays bestimmt werden, in dem die Objekte geschrieben werden, was im zweiten Durchlauf geschieht. Anschließend werden die Reihen mit allen enthaltenen Zellen erstellt, basierend auf dem Array. Alle Felder die dabei leer sind, werden in KNIME als „MissingCell“ deklariert, da leere Zellen nicht erstellt werden können. Erkennbar sind die Zellen am „?“ in der KNIME-Tabelle. Diese sieht wie folgt aus (Auszug):

Row ID	S http://xmlns.com/foaf/0.1/...	S http://dbpedia.org/ontology/birthDate
http://de.dbpedia.org/resource/Carl_Barks	Carl@de	1901-03-27^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/James_Cameron	James@de	1954-08-16^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Vannevar_Bush	Vannevar@de	1890-03-11^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/James_L._Brooks	James L.@de	1940-05-09^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Alexander_Michailowitsch_Rodtschenko	Alexander Michailowitsch@de	1891-12-05^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Frédéric_Chopin	Frédéric@de	1810-03-01^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Siddhartha_Gautama	Siddhartha@de	?
http://de.dbpedia.org/resource/Vespasian	?	0009-11-17^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Leonardo_da_Vinci	?	1452-04-15^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Jean-Jacques_Annaud	Jean-Jacques@de	1943-10-01^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Terry_Gilliam	Terry@de	1940-11-22^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Max_Reinhardt	Max@de	1873-09-09^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Karl_Schwarzschild	Karl@de	1873-10-09^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Joan_Slonczewski	Joan@de	?
http://de.dbpedia.org/resource/Louise_Aston	Louise@de	1814-11-26^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Mira_Sorvino	Mira@de	1967-09-28^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Robert_Anton_Wilson	Robert Anton@de	1932-01-18^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Caligula	?	0012-08-31^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Heinrich_Himmler	Heinrich@de	1900-10-07^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Steve_Wozniak	Steve@de	1950-08-11^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Niki_de_Saint_Phalle	Niki de@de	1930-10-29^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Michael_Ellis_DeBakey	Michael Ellis@de	1908-09-07^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Jack_Nicholson	Jack@de	1937-04-22^^http://www.w3.org/2001/XMLSchema#date
http://de.dbpedia.org/resource/Al-Ghazali	al-@de	?

Abbildung 27: Auszug von einer KNIME-Tabelle, nach dem Einlesen einer RDF-Datei mit Personendaten

Folgende Schritte werden bei der Umwandlung von RDF in die KNIME-Tabelle durchgeführt:

1. Das Model wird via Jena und dem RDFDataMgr eingelesen.

2. Die einzelnen Tripel werden durchiteriert, dabei werden die Header und die RowIDs ermittelt und Listen gespeichert.

3. Ein zweidimensionales Array wird erzeugt, wobei die Größe der Listen gleichzeitig auch die Größe der Dimensionen darstellt.

4. Die Tripel werden erneut durchiteriert, wobei dieses Mal das Array mit den Objekten befüllt wird.

5. Die ColumnSpecs werden definiert, als Spaltenkopf dienen die Werte aus der Header-Liste.

6. Im letzten Durchlauf werden alle Zellen, Reihe für Reihe auf Basis des zweidimensionalen Arrays, befüllt. Sollte ein Feld null sein, so wird eine MissingCell stattdessen verwendet. Als RowID werden die Werte aus der RowID-Liste genommen.

Row Filter

Der Row Filter ist ein bereits integrierter Knoten in KNIME. Dieser ermöglicht es, einzelne oder mehrere Reihen herauszufiltern. Die Filterkriterien können dabei den Inhalt einer Spalte, Reihennummern oder ReihenIDs betreffen. Die Kriterien selbst können als Patter definiert werden, bei Ziffern eine Range haben oder die MissingCells betreffen. In dem Beispiel sollen nur noch Personen enthalten sein, die in Berlin geboren sind. Die nötige Konfiguration des Knotens ist in Abbildung 29 zu sehen.

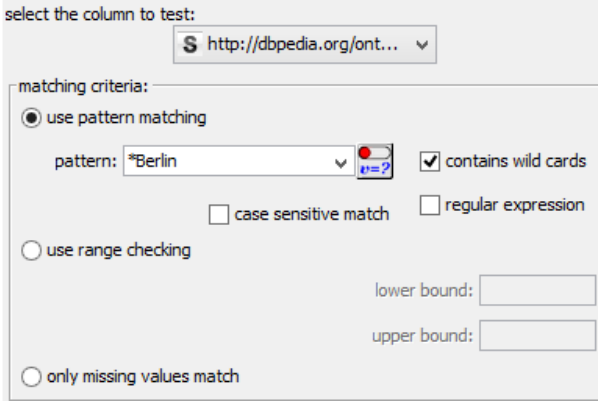

Abbildung 28: Konfigurationsfenster für den Row Filter-Knoten in KNIME

RDFWriter

Der RDFWriter ist der Gegenpart zum RDFReader. Während der RDFReader eine RDF-Datei in das KNIME-Tabellen-Format umwandelt, generiert der RDFWriter aus der KNIME-Tabelle eine RDF-Datei. Was für ein Darstellungsschema extrahiert werden soll, kann in der Konfiguration des Knotens ausgewählt werden. Dabei stehen alle Jena kompatiblen Formate zur Verfügung (Turtle, n-Triples, RDFXML, N3, JSONLD, RDFJSON) [66]. Sollte eine Zelle als MissingCell deklariert sein, so wird dieses Tripel nicht beachtet, da nur Tripel mit vorhandenen Subjekt, Prädikat und Objekt einen benötigten Informationsgehalt besitzen.

5.2.2 Erzeugen und Trennen von Daten/Metadaten-Verbänden

In diesem Szenario sollen Daten und Metadaten zu Verbänden fusioniert, verarbeitet und wieder getrennt werden. Zwischen der Zusammenführung und der Trennung teilen sich die fusionierten Daten eine Tabelle und dementsprechend auch eine Pipe. Alle Verarbeitungsschritte werden dabei auf die komplette Tabelle ausgeführt. Diese Vorgehensweise soll den primär vorgesehenen Ablauf darstellen. Als Alternative wird anschließend noch das Vorgehen mit zwei voneinander getrennten Strängen aufgezeigt, welche teilweise umgesetzt wurden.

In diesem Szenario stehen die Fusionierung und die Trennung der Verbände im Vordergrund. Der Findungsalgorithmus spielt hierbei nur eine untergeordnete Rolle, weswegen auf eine komplexe Such-Engine verzichtet wird. Hier wird lediglich eine einfache String-Suche verwendet. Der Einsatz der Such-Engine findet erst im nächsten Szenario statt.

Gegeben:

Es werden zwei Dateien zur Verfügung gestellt, wobei eine davon die Metadaten und die andere die persönlichen Daten enthalten. Die Daten befinden sich in einer csv-Datei und beinhalten die Namen von Fußballspielern, welche in der Saison 2013/2014 für den Club Hertha BSC aufgelaufen sind. Zudem befinden sich noch drei weitere Spalten in der Tabelle mit der Anzahl der Spiele, der Tore und den Assists (direkte Torvorlagen). Die Metadaten befinden sich erneut in einer ttl-Datei und enthalten alle Tupel zu den Spielern, welche sich auch in der heruntergeladenen Datei von der DBpedia-Seite befunden haben [39].

Ausgabe:

Nach den Verarbeitungsschritten soll am Ende des Workflows eine ttl-Datei mit den gefilterten Metadaten und eine csv-Datei mit den gefilterten Daten erstellt werden.

Verarbeitungsschritte:

1. Der RDF-Reader liest die ttl-Datei mit den Personendaten ein und wandelt die Triple in die übliche KNIME-Tabellenform um.

2. Der CSV Reader (bereits in KNIME enthalten) liest die csv-Datei mit den Spielerdaten ein und wandelt die Triple ebenfalls in die KNIME-Tabellenform um.

3. Mit Hilfe des Matching-Knotens werden die beiden Datensätze synchronisiert und in eine Tabelle fusioniert. Die Zeilen werden von der Datentabelle bestimmt.

4. Mit Hilfe des Java Snippet-Knotens (bereits in KNIME enthalten) wird eine Spalte mit den Scores hinzugefügt (Scores = Tore + Assists).

5. Die Datensätze werden mit einem Row Filter (bereits in KNIME enthalten) gefiltert, so dass nur noch Personen angezeigt werden, welche mindestens einen Score von 1 haben.

6. Der Separating-Knoten trennt den Verbund wieder in Daten und Metadaten.

7. Nur die gefilterten Metadaten, von den Spielern mit einen Score von mindestens eins, werden mittels dem RDFWriter-Knoten in eine neue ttl-Datei geschrieben.

8. Auch die gefilterten Daten werden rausgeschrieben. Dies geschieht mit den CSV Writer (bereits in KNIME enthalten). Bestandteil der Daten ist auch die neu erzeugte Score-Spalte.

Realisierung

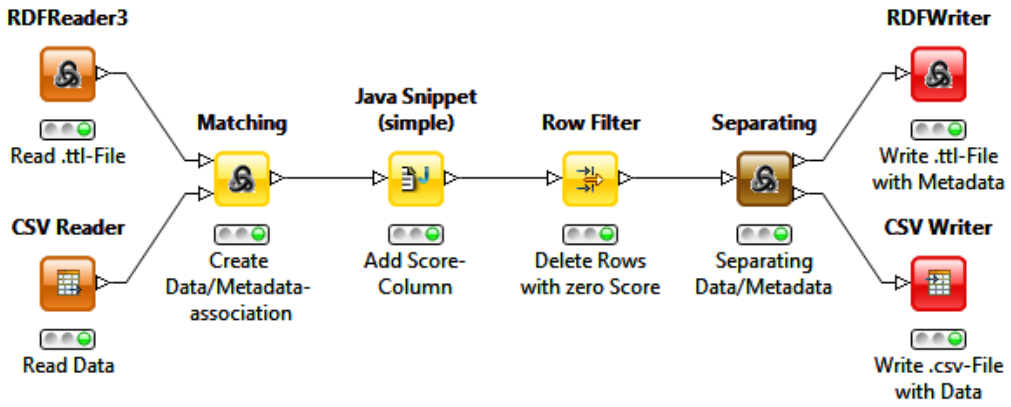

Abbildung 29: KNIME-Workflow für das Fusionieren und Trennen von Daten/Metadaten-Verbänden und einer synchronen Verarbeitung mit nur einem Datenstrom

RDFReader & CSV Reader

Der RDFReader ist bereits aus dem vorherigen Szenario bekannt und ist dafür zuständig die RDF-Daten in die KNIME-Tabellenstruktur zu transformieren. Ähnlich funktioniert der CSV Reader, welcher bereits fester Bestandteil von KNIME ist. Dieser liest die CSV-Datei mit den Spielerdaten ein, dabei kann genau konfiguriert werden wie die CSV-Datei aufgebaut ist. Dies wären: Datenquelle, Aussehen des Spaltentrenners, des Reihentrenners, des Zitatzeichens und des Kommentarzeichens. Zudem kann ausgewählt werden ob die Datei einen Reihen-, bzw. Spaltenkopf besitzen und ob verkürzte Zeilen erlaubt sind. Letztendlich kann auch eine bestimmbare Anzahl an Zeilen übersprungen werden und es kann auch ein Limit an Reihen angegeben werden.

Matching

Der Matching-Knoten übernimmt komplett die Datentabelle. Auf Basis derer werden die RowIDs als Suchgrundlage verwendet. In diesem Szenario sind es die Namen der Spieler. Diese werden nun mit den RowIDs aus der Metadatentabelle verglichen, was in dem Fall die URIs sind. Sollte der Name in einer URI vorkommen, wird dies als Übereinstimmung gewertet. Anschließend werden die dazugehörigen Meta-Informationen an die Datenreihe angehängt. Damit die URI der Metadatentabelle nicht verloren geht, wird zusätzlich eine Spalte angelegt, wo diese vermerkt wird. Alle Spalten von der Metadatentabelle werden zusätzlich in der Columnspec als „meta“ getaggt und die URI bekommt im „meta“-Property zusätzlich den Wert „uri“ zugewiesen. Dies sind wichtige Schritte um ein späteres Trennen zu ermöglichen. Sollte es für einige Spieler keine passenden Metainformationen geben, so erhalten die Zellen von den übernommenen Spalten die Deklaration MissingCell.

Row ID	Spiele	Tore	Assissts	http://dbpedia.org/ontology/birthPlace	http://dbpedia.org/ontology/birthDate
Thomas Kraft	32	0	0	http://de.dbpedia.org/resource/Kirchen_(Si...	1988-07-22^^http://www.w3.org/2001/XMLSchema#c
Rune Almenni...	1	0	0	?	1984-09-29^^http://www.w3.org/2001/XMLSchema#c
Marius Gersbeck	1	0	0	?	?
Fabian Lusten...	19	0	0	http://de.dbpedia.org/resource/Wolhusen	1988-05-02^^http://www.w3.org/2001/XMLSchema#c
John Anthony...	16	2	0	http://de.dbpedia.org/resource/Berlin	1993-01-28^^http://www.w3.org/2001/XMLSchema#c
Sebastian Lan...	29	0	0	http://de.dbpedia.org/resource/Speyer	1988-01-15^^http://www.w3.org/2001/XMLSchema#c
Christoph Jan...	5	0	0	http://de.dbpedia.org/resource/Cham_(Ob...	1985-02-14^^http://www.w3.org/2001/XMLSchema#c
Lewan Kobias...	15	1	0	http://de.dbpedia.org/resource/Tiflis	1977-07-10^^http://www.w3.org/2001/XMLSchema#c
Johannes van...	26	0	1	http://de.dbpedia.org/resource/Viersen	1986-11-21^^http://www.w3.org/2001/XMLSchema#c
Fabian Holland	3	0	0	http://de.dbpedia.org/resource/Berlin	1990-07-11^^http://www.w3.org/2001/XMLSchema#c
Peter Pekarik	31	0	3	?	?
Marcel Ndjeng	17	0	0	http://de.dbpedia.org/resource/Bonn	1982-05-06^^http://www.w3.org/2001/XMLSchema#c
Hajime Hosogai	33	0	1	http://de.dbpedia.org/resource/Maebashi	1986-06-10^^http://www.w3.org/2001/XMLSchema#c
Peter Niemeyer	20	0	0	http://de.dbpedia.org/resource/Mettingen	1983-11-22^^http://www.w3.org/2001/XMLSchema#c
Peer Kluge	2	0	1	http://de.dbpedia.org/resource/Frankenber...	1980-11-22^^http://www.w3.org/2001/XMLSchema#c
Per Ciljan Skel...	28	2	7	?	?
Tolga Cigerci	19	1	3	?	?

Abbildung 30: Auszug von einer KNIME-Tabelle, nach der Daten/Metadaten-Fusion

Java Snippet & Row Filter

Mit den Java Snippet-Knoten können neue Spalten erzeugt oder bisherige Spalten ersetzt werden. Die Konfiguration des Knotens ist in sechs verschiedene Bereiche unterteilt. In der linken oberen Ecke werden alle Spalten der Tabelle aufgelistet, darunter stehen die existierenden Variablen. Auf der rechten Seite befinden sich zwei Textfelder, wobei im oberen globale Variablen deklariert werden können. Im unteren wird der Methodeninhalt hingeschrieben. In der Ecke unten links kann noch gewählt werden, ob eine neue Spalte erzeugt werden und wie die heißen soll oder ob eine bisherige Spalte ersetzt werden soll. Der letzte Bereich unten rechts gibt dem Benutzer die Möglichkeit den Datentyp zu bestimmen (Integer, Long, Double, String) und ob es ein Array ist oder nicht.

In diesem Beispiel soll eine neue Spalte mit den Namen Score hinzugefügt werden. Da lediglich die Tore- mit der Assists-Spalte addiert werden soll, werden keine zusätzlichen Variablen benötigt. Auf den Inhalt einer Zelle wird mit $Spalte$ zugegriffen. Die Methode sieht wie folgt aus:

```
return $Tore$+$Assists$;
```

Abbildung 31: Quellcode für den "Java Snippet"-Knoten, um eine neue Spalte hinzuzufügen, welche als Inhalt die Summe zweier anderer Spalten besitzt

Der Row Filter-Knoten kam bereits im vorherigen Szenario zum Einsatz. Passend für dieses Szenario beziehen wir uns auf die neue Score-Spalte und inkludieren alle Zeilen wo der Score mindestens eins beträgt (use range checking - lower bound: 1). Nach dieser Filterung bleiben nur noch 15 der ehemals 25 Spieler übrig.

Separating

Der Separating-Knoten trennt die erhaltenen Metadaten wieder von den Spielerdaten. Diese Trennung ist nur möglich, wenn die URI-Spalte nicht gelöscht wird, da den Metadaten sonst die RowIDs fehlen. Wenn der Knoten ohne explizite Konfiguration verwendet wird, dann enthält die Datentabelle, neben den bisherigen Daten, auch alle neu hinzugefügten Spalten. Da dies vielleicht nicht immer gewünscht ist, kann dieser Knoten auch konfiguriert werden. Hier kann genau bestimmt werden, welche Spalten zu welcher Tabelle gehören sollen. Nicht austauschbar sind die jeweiligen RowIDs (Spielername/URI), welche den Tabellen fest zugewiesen werden. Alle Zeilen, bei denen es keine URI gibt, werden nicht mit in die Metadatentabelle übernommen.

KNIME stellt bereits einen ähnlichen Knoten zur Verfügung, den Splitter. Mit diesem kann eine Tabelle ebenfalls in zwei Tabellen gesplittet werden und alle darin enthaltenen Spalten können frei auf die neuen Tabellen verteilt werden. Dieser hat aber Nachteile gegenüber dem Separating-Knoten. Für beide ausgehenden Tabellen wird die gleiche RowID verwendet, anstatt für die Metadaten-Tabelle die IRI zu verwenden. Diese wird wiederum als eigenständige Spalte übernommen, was ebenfalls nicht der Anfangsaufteilung, vor dem Zusammenführen der Daten und Metadaten, entsprechen würde.

Table "default" - Rows: 10 | Spec - Columns: 7 | Properties | Flow Variables

Row ID	S http://dbpedia.org/ontology/b...	S http://dbpedia
http://de.dbpedia.org/resource/John_Anthony_Brooks	http://de.dbpedia.org/resource/Be...	1993-01-28^^http
http://de.dbpedia.org/resource/Lewan_Kobiaschwili	http://de.dbpedia.org/resource/Tiflis	1977-07-10^^http
http://de.dbpedia.org/resource/Johannes_van_den_Bergh	http://de.dbpedia.org/resource/Vi...	1986-11-21^^http
http://de.dbpedia.org/resource/Hajime_Hosogai	http://de.dbpedia.org/resource/M...	1986-06-10^^http
http://de.dbpedia.org/resource/Peer_Kluge	http://de.dbpedia.org/resource/Fr...	1980-11-22^^http
http://de.dbpedia.org/resource/Nico_Schulz_(Fußballspieler)	http://de.dbpedia.org/resource/Be...	1993-04-01^^http
http://de.dbpedia.org/resource/Alexander_Baumjohann	http://de.dbpedia.org/resource/W...	1987-01-23^^http
http://de.dbpedia.org/resource/Änis_Ben-Hatira	http://de.dbpedia.org/resource/Be...	1988-07-18^^http
http://de.dbpedia.org/resource/Sami_Allagui	http://de.dbpedia.org/resource/Dü...	1986-05-28^^http
http://de.dbpedia.org/resource/Sandro_Wagner	http://de.dbpedia.org/resource/M...	1987-11-29^^http

Table "default" - Rows: 15 | Spec - Columns: 4 | Properties | Flow Variables

Row ID	I Spiele	I Tore	I Assissts	I Score
John Anthony...	16	2	0	2
Lewan Kobias...	15	1	0	1
Johannes van...	26	0	1	1
Peter Pekarik	31	0	3	3
Hajime Hosogai	33	0	1	1
Peer Kluge	2	0	1	1
Per Ciljan Skel...	28	2	7	9
Tolga Cigerci	19	1	3	4
Nico Schulz	23	0	1	1
Ronny	27	4	4	8
Alexander Ba...	9	0	4	4
Änis Ben-Hatira	21	3	2	5
Sami Allagui	29	9	2	11
Adrian Ramos	32	16	8	24
Sandro Wagner	25	2	0	2

Abbildung 32: KNIME-Tabellen nach der Trennung der Metadaten (Tabelle links) von den Daten (Tabelle rechts)

RDFWriter & CSV Writer

Der RDFWriter schreibt, wie schon im vorherigen Szenario, die Metadaten in eine RDF-Datei. Der CSV Writer bekommt hingegen die Spielerdatentabelle und wandelt diese ins csv-Format um. Wie auch schon beim Einlesen von csv-Dateien erhält der Benutzer jede Menge Konfigurationsmöglichkeiten, um die für ihm passende Formatierung zu erhalten.

Alternative asynchrone Verarbeitung

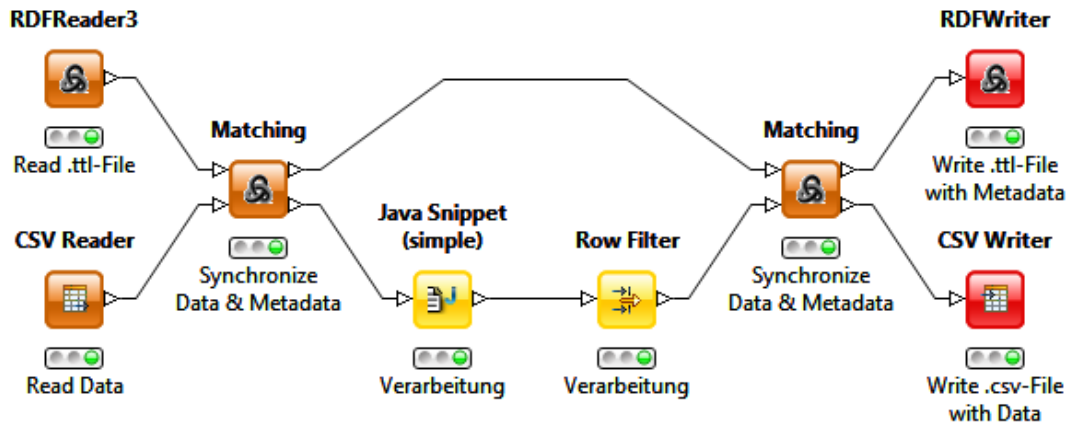

Abbildung 33: KNIME-Workflow für das Synchronisieren von Daten und Metadaten und einer asynchronen Verarbeitung mit getrennten Datenströmen

Matching / Synchronisation

Der Matching-Knoten in dieser alternativen Variante fungiert als Synchronisation zwischen den Tabellen. Es wird der gleiche Suchalgorithmus wie zuvor verwendet, jedoch werden die Tabellen nicht miteinander fusioniert, sondern lediglich die Länge und die Reihenfolge der Metadatentabelle wird an die Datentabelle angeglichen. Um die Zugehörigkeiten erkennen zu können, müssen beide Tabellen nebeneinander geschoben werden (siehe Bild). Bei sehr vielen Reihen kann aber schnell der Überblick verloren gehen. Da die Tabellen nie wirklich miteinander verbunden werden, sondern lediglich synchronisiert werden, kann nach dem Matching jede einzelne Tabelle separat verarbeitet werden. Um nach den Verarbeitungsschritten wieder eine Synchronität zu erlangen wird erneut der Matching-Knoten verwendet. Bisher ist nur eine Anpassung der Metadatentabelle an die Datentabelle möglich, in Zukunft könnte der Knoten durch eine intelligenteren Synchronisationsalgorithmus ersetzt werden.

Table "default" - Rows: 25 | Spec - Columns: 7 | Properties | Flow Variables

Row ID	http://dbpedia.org/ontology/birthPlace	http://dbpedia.org/ontology/bi
http://de.dbpedia.org/resource/Thomas_Kraft	http://de.dbpedia.org/resource/Kirchen_(Si...	1988-07-22^^http://www.w3.org/
http://de.dbpedia.org/resource/Rune_Almenning_Jarstein	?	1984-09-29^^http://www.w3.org/
http://de.dbpedia.org/resource/Marius_Gersbeck	?	?
http://de.dbpedia.org/resource/Fabian_Lustenberger	http://de.dbpedia.org/resource/Wolhusen	1988-05-02^^http://www.w3.org/
http://de.dbpedia.org/resource/John_Anthony_Brooks	http://de.dbpedia.org/resource/Berlin	1993-01-28^^http://www.w3.org/
http://de.dbpedia.org/resource/Sebastian_Langkamp	http://de.dbpedia.org/resource/Speyer	1988-01-15^^http://www.w3.org/
http://de.dbpedia.org/resource/Christoph_Janker	http://de.dbpedia.org/resource/Cham_(Ob...	1985-02-14^^http://www.w3.org/
http://de.dbpedia.org/resource/Lewan_Kobiaschwili	http://de.dbpedia.org/resource/Tiflis	1977-07-10^^http://www.w3.org/
http://de.dbpedia.org/resource/Johannes_van_den_Bergh	http://de.dbpedia.org/resource/Viersen	1986-11-21^^http://www.w3.org/
http://de.dbpedia.org/resource/Fabian_Holland	http://de.dbpedia.org/resource/Berlin	1990-07-11^^http://www.w3.org/
http://de.dbpedia.org/resource/Peter_Pekarik	?	?
http://de.dbpedia.org/resource/Marcel_Ndjeng	http://de.dbpedia.org/resource/Bonn	1982-05-06^^http://www.w3.org/
http://de.dbpedia.org/resource/Hajime_Hosogai	http://de.dbpedia.org/resource/Maebashi	1986-06-10^^http://www.w3.org/
http://de.dbpedia.org/resource/Peter_Niemeyer	http://de.dbpedia.org/resource/Mettingen	1983-11-22^^http://www.w3.org/
http://de.dbpedia.org/resource/Peer_Kluge	http://de.dbpedia.org/resource/Frankenber...	1980-11-22^^http://www.w3.org/
http://de.dbpedia.org/resource/Per_Ciljan_Skelbred	?	?
http://de.dbpedia.org/resource/Tolga_Cigerci	?	?
http://de.dbpedia.org/resource/Nico_Schulz_(Fußballspieler)	http://de.dbpedia.org/resource/Berlin	1993-04-01^^http://www.w3.org/
http://de.dbpedia.org/resource/Ronny	?	?
http://de.dbpedia.org/resource/Alexander_Baumjohann	http://de.dbpedia.org/resource/Waltrop	1987-01-23^^http://www.w3.org/
http://de.dbpedia.org/resource/Hany_Mukhtar	http://de.dbpedia.org/resource/Berlin	1995-03-21^^http://www.w3.org/
http://de.dbpedia.org/resource/Änis_Ben-Hatira	http://de.dbpedia.org/resource/Berlin	1988-07-18^^http://www.w3.org/
http://de.dbpedia.org/resource/Sami_Allagui	http://de.dbpedia.org/resource/Düsseldorf	1986-05-28^^http://www.w3.org/
http://de.dbpedia.org/resource/Adrian_Ramos	?	?
http://de.dbpedia.org/resource/Sandro_Wagner	http://de.dbpedia.org/resource/München	1987-11-29^^http://www.w3.org/

Table "Kader2.csv" - Rows: 25 | Spec - Columns: 3 | Properties | Flow Variables

Row ID	Spiele	Tore	Assissts
Thomas Kraft	32	0	0
Rune Almenni...	1	0	0
Marius Gersbeck	1	0	0
Fabian Lusten...	19	0	0
John Anthony...	16	2	0
Sebastian Lan...	29	0	0
Christoph Jan...	5	0	0
Lewan Kobias...	15	1	0
Johannes van...	26	0	1
Fabian Holland	3	0	0
Peter Pekarik	31	0	3
Marcel Ndjeng	17	0	0
Hajime Hosogai	33	0	1
Peter Niemeyer	20	0	0
Peer Kluge	2	0	1
Per Ciljan Skel...	28	2	7
Tolga Cigerci	19	1	3
Nico Schulz	23	0	1
Ronny	27	4	4
Alexander Ba...	9	0	4
Hany Mukhtar	10	0	0
Änis Ben-Hatira	21	3	2
Sami Allagui	29	9	2
Adrian Ramos	32	16	8
Sandro Wagner	25	2	0

Abbildung 34: KNIME-Tabellen nach der ersten Synchronisation der Metadaten (Tabelle links) mit den Daten (Tabelle rechts)

5.2.3 Online-Daten abfragen via SPARQL

Im Gegensatz zu den vorigen Szenarien, sollen die Metadaten nun über das Internet bezogen werden. Dafür wird die SPARQL-Schnittstelle von der DBpedia verwendet, worüber Daten im RDF-Format abgerufen werden können. Damit auch die richtigen Daten abgerufen werden, werden zuvor mittels einer Suchmaschine (DBpedia lookup) die URIs abgefragt.

Gegeben:

Erneut ist eine csv-Datei gegeben, welche die deutschen Nationalspieler enthält, die Deutschland 2014 zum 4. Weltmeistertitel geschossen haben. Neben den Namen, welcher als RowID fungiert, enthält die Datei noch Spalten für deren Position, der Rückennummer und den Verein bei dem sie unter Vertrag standen. Für die Metadaten stellt DBpedia einen Virtuoso Server mit SPARQL-Schnittstelle zur Verfügung, über dem auf die Wikipedia-Daten zugegriffen werden kann [67].

Verarbeitungsschritte:

1. Der CSV Reader (bereits in KNIME enthalten) liest die csv-Datei mit den Spielerdaten ein und wandelt die Informationen in die KNIME-Tabellenform um.

2. Es wird eine Spalte ausgewählt für die die URIs ermittelt werden sollen. In diesem Beispiel die der Spieler, anhand der Spielernamen.

3. Mittels der gefundenen URIs werden SPARQL-Abfragen erzeugt und an DBpedia geschickt. Die somit erhaltenen Metadaten werden dann in die KNIME-Tabelle eingearbeitet.

4. Anschließend kann der Daten-/Metadaten-Verbund beliebig bearbeitet werden, wie etwa im vorherigen Szenario ab Verarbeitungsschritt 4.

Realisierung

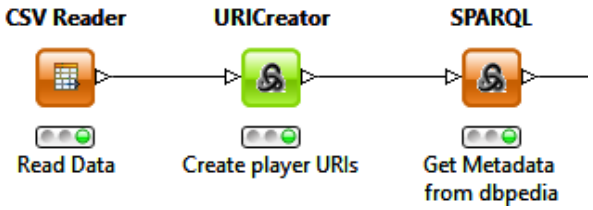

Abbildung 35: KNIME-Workflow für das Erstellen von URIs und Abfragen von Metadaten über die SPARQL-Schnittstelle von der DBpedia

URICreator

Um den URICreator-Knoten verwenden zu können, muss in der Konfiguration des Knotens eine Spalte vom Typ String oder die RowID ausgewählt werden. Auf Basis des Inhalts in den einzelnen Zellen wird daraufhin die URI gesucht. In diesem Szenario wählen wir die RowID, welches die Namen der Fußballspieler sind. Die Erstellung und Ausführung der Anfrage erfolgt im DBpediaLookupClient [53], welcher von Mark Watson [41] entwickelt und für dieses Projekt angepasst wurde. Dies sieht wie folgt aus:

```
1 String url =
  "http://lookup.dbpedia.org/api/search.asmx/KeywordSearch?MaxHits=2
   &QueryString=" + query;

2 HttpClient client = new HttpClient();
3 HttpMethod method = new GetMethod(url);
4 method.getParams().setContentCharset("UTF-8");
5 method.setRequestHeader("Content-Type", "application/x-www-form-url-
  encoded; charset=utf-8");
6  try {
7     client.executeMethod(method);
8     InputStream ins = method.getResponseBodyAsStream();
9     SAXParserFactory factory = SAXParserFactory.newInstance();
10    SAXParser sax = factory.newSAXParser();
11    sax.parse(ins, this);
12 } catch (HttpException he) {
      System.err.println("Http error connecting to lookup.dbpedia.org");
13 } catch (IOException ioe) {
      System.err.println("Unable to connect to lookup.dbpedia.org");
14 }
```

Abbildung 36: Der Auszug aus dem Quellcode vom DBpediaLookupClient zeigt die Erzeugung einer Http-Methode im UTF-8 Format, das Ausführen der Anfrage an DBpedia Lookup und das Parsen des Ergebnisses mit Hilfe des SAXParsers

Die Suchanfrage wird an DBpedia Lookup geschickt, was die URL „http://lookup.dbpedia.org/api/search.asmx/“ hat. Die Syntax der Parameter wird bereits in Kapitel 4.2 erklärt. In der Variable query befindet sich der Name des Fußballspielers. Die Anfrage wird über den Jakarta HttpClient [44] gestellt, die Deklaration wird bereits in Zeile zwei durchgeführt. Als Übertragungsmethode wird GET verwendet (Zeile 3). Damit es keine Probleme mit den Sonderzeichen gibt, muss sowohl die Anfrage, als auch das Ergebnis in UTF-8 sein (Zeile 4&5). Mit der executeMethod wird die Anfrage ausgeführt (Zeile 7). Die Ergebnisse werden als Stream zurückgegeben (Zeile 8) und der SAXParser ist für die Interpretation des Streams zuständig (Zeile 9-11).

Der folgende Quellcodeauszug zeigt wie die Anfrage aufgerufen und das Ergebnis verarbeitet wird:

```
1   key = URLEncoder.encode(r.getCell(colindex).toString(), "UTF-8");
2   DBpediaLookupClient lookupClient = new DBpediaLookupClient(key);

3   if(!lookupClient.variableBindings().isEmpty()){
4    List<Map<String, String>> resultList = lookupClient.variableBindings();
5     if(resultList.get(0).containsKey("URI")){
6       String uri = resultList.get(0).get("URI");
7       cells[columns] = new StringCell(uri);
8     } else if(resultList.get(1).containsKey("URI")) {
9       String uri = resultList.get(1).get("URI");
10      cells[columns] = new StringCell(uri);
11    } else {
12      cells[columns] = org.knime.core.data.DataType.getMissingCell();
13    }
14  } else {
15    cells[columns] = org.knime.core.data.DataType.getMissingCell();
16  }
```

Abbildung 37: Verarbeitung des Requests vom DBpedia Lookup - Suchergebnis

Dieser Quellcode wird für jede Reihe, von der Tabelle, ausgeführt. Dazu wird in der Reihe der Inhalt der Zelle genommen, deren Spalte in der Konfiguration ausgewählt wurde und anschließend in UTF-8 kodiert (Zeile 1). Dieser Key wird dem DBpediaLookupClient übergeben und das Ergebnis wird in der Variable lookupClient gespeichert (Zeile 2). Diese beinhaltet nun alle Variablen, welche in der Request enthalten waren. Sollten keine Variablen enthalten sein, dann wurde kein Ergebnis für die Suchanfrage gefunden, dann wird anstatt der URI eine MissingCell übernommen (Zeile 3 → Zeile 15). Wenn die Suche erfolgreich war und somit Suchergebnisse in der Request enthalten sind, dann wird eine Liste mit den Suchergebnissen erzeugt (Zeile 4). Da in der Anfrage das MaxHits auf 2 gestellt wurde, enthält die Liste nun maximal zwei Maps mit den jeweiligen Variablen. Der Vorteil an einer Map ist, dass einfach nach einem Schlüßelwort gesucht werden kann, was auch gleichzeitig der Name des Feldes/Wertes ist. Da wir nur an der URI interessiert sind, wird überprüft ob das erste Suchergebnis diesen Key beeinhaltet (Zeile 5). Wenn dies der Fall ist, wird die URI abgefragt und es wird eine Zelle vom Typ String mit dieser URI erzeugt (Zeile 6&7). Manchmal kommt es vor, dass ein Suchergebnis keine URI enthält, dann wird auf das zweite Suchergebnis zurückgegriffen (Zeile 8-10). Sollte es kein zweites Suchergebnis geben oder sollte es auch dort keine URI geben, dann wird erneut eine MissingCell erzeugt und als Suchergebnis verwendet (Zeile 12).

Als Beispiel würde für den Fußballspieler Jerome Boateng die Anfrage-URL wie folgt aussehen:

http://lookup.dbpedia.org/api/search.asmx/KeywordSearch?MaxHits=2&QueryString=Jerome+Boateng

Das Ergebnis sieht dann folgendermaßen aus (dargestellt als XML):

```
<ArrayOfResult>
    <Result>
<Label>Jérôme Boateng</Label>
<URI>http://dbpedia.org/resource/Jérôme_Boateng</URI>
<Description/>
<Classes/>
<Categories/>
<Templates/>
<Redirects/>
<Refcount>26</Refcount>
    </Result>
    <Result>
…
<URI>
http://dbpedia.org/resource/2007%E2%80%9308_Hertha_BSC_season__Jerome_Boateng__1
</URI>
…
    </Result>
</ArrayOfResult>
```

Abbildung 38: Ergebnis einer DBpedia Lookup - Suchanfrage im XML-Format (Auszug)

Die Map würde dann die Einträge Label, URI und Refcount enthalten. Demnach würden wir als Suchergebnis folgende URI haben: http://dbpedia.org/resource/Jérôme_Boateng

Die Reihenfolge der Results wird durch die Anzahl der Refcount bestimmt, also der Anzahl wie oft eine URL von einem anderen Artikel referenziert wird.

Row ID	S Position	I Rücken...	S Verein	S TestURI
Manuel Neuer	Torhüter	1	Bayern München	http://dbpedia.org/resource/Manuel_Neuer
Roman Weidenfeller	Torhüter	22	Borussia Dortmund	http://dbpedia.org/resource/Roman_Weidenfeller
Ron-Robert Zieler	Torhüter	12	Hannover 96	http://dbpedia.org/resource/Ron-Robert_Zieler
Jerome Boateng	Abwehr	20	Bayern München	http://dbpedia.org/resource/Jérôme_Boateng
Erik Durm	Abwehr	15	Borussia Dortmund	?
Kevin Grosskreutz	Abwehr	2	Borussia Dortmund	http://dbpedia.org/resource/Kevin_Großkreutz
Benedikt Höwedes	Abwehr	4	FC Schalke 04	http://dbpedia.org/resource/Benedikt_Höwedes
Mats Hummels	Abwehr	5	Borussia Dortmund	http://dbpedia.org/resource/Mats_Hummels
Philipp Lahm	Abwehr	16	Bayern München	http://dbpedia.org/resource/Philipp_Lahm
Per Mertesacker	Abwehr	17	Arsenal FC	http://dbpedia.org/resource/Per_Mertesacker
Shkodran Mustafi	Abwehr	21	Sampdoria	http://dbpedia.org/resource/2010%E2%80%9311_Everton_F.C._season__Shkodran_Mustafi__1
Julian Draxler	Mittelfeld	14	FC Schalke 04	http://dbpedia.org/resource/Julian_Draxler
Matthias Ginter	Mittelfeld	3	SC Freiburg	http://dbpedia.org/resource/Matthias_Ginter
Mario Götze	Mittelfeld	19	Bayern München	http://dbpedia.org/resource/Mario_Götze
Sami Khedira	Mittelfeld	6	Real Madrid	http://dbpedia.org/resource/Sami_Khedira
Christoph Kramer	Mittelfeld	23	Bor. Mönchengladbach	http://dbpedia.org/resource/Christoph_Kramer
Toni Kroos	Mittelfeld	18	Bayern München	http://dbpedia.org/resource/Toni_Kroos
Thomas Müller	Mittelfeld	13	Bayern München	http://dbpedia.org/resource/Thomas_Müller
Mesut Özil	Mittelfeld	8	Arsenal FC	http://dbpedia.org/resource/Mesut_Özil
Lukas Podolski	Mittelfeld	10	Arsenal FC	http://dbpedia.org/resource/Lukas_Podolski
Andre Schürrle	Mittelfeld	9	Chelsea FC	http://dbpedia.org/resource/André_Schürrle
Bastian Schweinsteiger	Mittelfeld	7	Bayern München	http://dbpedia.org/resource/Bastian_Schweinsteiger
Miroslav Klose	Sturm	11	Lazio Roma	http://dbpedia.org/resource/Miroslav_Klose

Abbildung 39: Die KNIME-Tabelle zeigt die Testdaten, zuzüglich der neu hinzugefügten URI-Spalte mit den via DBpedia Lookup ermittelten URIs

Die Tabelle zeigt, dass in 21 von 23 Fällen die korrekten URIs zu den Nationalspielern gefunden wurden. Für Erik Durm gibt es aktuell noch keinen Eintrag, weswegen nichts gefunden werden kann. Erkennbar ist die MissingCell an dem Fragezeichen in der Zelle. Bei Shkodran Mustafi wurde eine falsche URI gefunden. Dies ist damit zu begründen, dass sowohl die gefundene URI, als auch die eigentliche von Mustafi keine Refcount besitzen und somit die Reihenfolge der Ergebnisse falsch ist. Der korrekte Datensatz hätte sich an Position drei der Ergebnisse befunden. Um diesen Fehler korrigieren zu können, müsste ein Knoten zum Ändern der URI entwickelt werden (siehe Kapitel 4.3 Sonstige Knoten – URIChanger).

SPARQL

Der SPARQL-Knoten ist der umfangreichste Knoten, da dieser die Abfrage erzeugt und ausführt, den Request interpretiert und in die KNIME-Tabellenstruktur umwandelt und diese Ergebnisse mit den bisherigen Daten zusammenführt. Somit Kombiniert er Teil-Algorithmen vom RDFReader (Umwandlung von RDF in KNIME) und dem URICreator (Abfrageergebnisse und Daten zusammenführen).

Nach der Konfiguration, welche Spalte als Basis der Abfrage genommen werden soll, durchläuft der Knoten folgende Schritte:

1. Für jede Zeile wird eine SPARQL-Abfrage generiert und ausgeführt (wie die SPARQL-Abfrage auszusehen hat, wird bereits in Kapitel 3.3.4 gezeigt).

2. Die Abfrageergebnisse werden mit Hilfe von Jena zu einem RDF-Graphen transferiert. Dabei ist die URI das Subjekt, das Property das Prädikat und das Objekt bleibt das Objekt.

3. Die einzelnen Tripel vom RDF-Graphen werden durchiteriert, dabei werden die Header und die RowIDs ermittelt und in Listen gespeichert.

4. Ein zweidimensionales Array wird erzeugt, wobei die Größe der Listen gleichzeitig auch die Größe der Dimensionen darstellt.

5. Die Tripel werden erneut durchiteriert, wobei dieses Mal das Array mit den Objekten befüllt wird. Dieser Schritt beinhaltet zudem die Filterung nach den richtigen Eintrag, sollte es das Property für ein Subjekt öfters in mehreren Sprachen geben. Für die Bestimmung des Verhaltens bei doppelten Einträgen ist noch keine Auswahl möglich, es wird, egal ob es bereits einen Eintrag in der entsprechenden Zelle gibt, das Objekt eingetragen. Somit werden mögliche vorherige Werte überschreiben und das zuletzt auftretende Objekt wird verwendet.

6. Die ColumnSpecs werden definiert, dazu werden die Specs der bisherigen Spec übernommen und direkt folgend die neuen für die Metadaten generiert.

7. Im letzten Durchlauf werden alle Zellen, Reihe für Reihe befüllt. Als Basis werden die bisherigen Werte der Eingangstabelle und die des zweidimensionalen Arrays verwendet. Sollte ein Feld leer sein, so wird eine MissingCell stattdessen verwendet. Die RowID bleibt erhalten.

Von einer Tabelle mit vormals vier Spalten, wird nach der Verarbeitung des Knotens eine Tabelle mit insgesamt 68 Spalten generiert. Diese Tabelle bildet die Grundlage für weitere Verarbeitungsschritte. Da nicht alle Spalten sehr Aussagekräftig sind oder gar Irrelevant für die Auswertung sind, bietet sich zunächst ein Column Filter an. Nach diversen weiteren Verarbeitungsschritten kann mit Hilfe des Separating-Knotens wieder eine Trennung der Daten erfolgen.

5.3 Sonstige Knoten

CleanNames

Die Metadaten aus dem semantischen Web bestehen aufgrund der Eindeutigkeit und für die bessere Vernetzung größtenteils aus URIs. Es ist eher selten der Fall, dass Literale verwendet werden und dann meistens auch nur als Objekt. Was im semantischen Web ein Vorteil ist, ist aber bei der Datenanalyse eher unübersichtlich. Deswegen wurde auch ein CleanNames-Knoten entwickelt, welcher URIs in leserliche Strings umwandelt. Eine Rückumwandlung ist nicht möglich, weswegen dieser Knoten nur verwendet werden sollte, wenn zum Ende des Workflows keine RDF-Datei erstellt werden soll, es sei denn es werden Triple ohne URIs gewünscht.

Bei Daten und Zahlen wird hinter dem Wert noch das Typ-Schema von der w3c mit angegeben, dies wird entfernt, so dass nur der Wert angezeigt wird. Bei einer URI wird nur der Teil genommen, welcher hinter dem letzten Backslash steht. Zudem werden die Unterstriche durch ein Leerzeichen ersetzt. Bei Literalen die mit einem Sprachkürzel am Ende versehen sind, wird dieser ebenfalls abgeschnitten.

Die folgende Tabelle zeigt einige Umwandlungsbeispiele:

URI	String
1986-03-27+02:00^^http://www.w3.org/2001/XMLSchema#date	1986-03-27
98^^http://www.w3.org/2001/XMLSchema#integer	98
http://dbpedia.org/resource/FC_Schalke_04	FC Schalke 04
Gladbeck, Germany@en	Gladbeck, Germany

Tabelle 11: Beispiele für eine Umwandlung von URIs in leserliche Strings

Problematisch ist, dass nicht unterschieden werden kann, ob es sich um eine URI oder eine Homepage handelt. So werden zum Beispiel externe Links ebenfalls konvertiert von:

URI	String
http://www.arsenal.com/first-team/players/lukas-podolski	lukas-podolski
http://www.jboateng.com	www.jboateng.com
http://commons.wikimedia.org/wiki/Special:FilePath/Bastian_Schweinsteiger_20120609.jpg?width=300	Bastian_Schweinsteiger_20120609.jpg?width=300

Tabelle 12: Beispiele für ungewollte Umwandlungen von URIs in Strings

URIChanger

Der URIChanger-Knoten wurde nicht entwickelt, aber wär eine sinnvolle Ergänzung zum URICreator-Knoten. Wie bereits erklärt, kann es bei der Suche nach der URIs zu nicht erwünschten Ergebnissen kommen. Zum Beispiel wenn jemand im Bundesstaat New York in der Stadt Berlin geboren wurde, dann wird die Suchmaschine unter den Begriff „Berlin“ die URL der deutschen Hauptstadt liefern. Um dies im Nachhinein ändern zu können, bevor die SPARQL-Abfrage durchgeführt wird, wäre der URIChanger genau der richtige Knoten. Dafür müsste der Nutzer in der Konfiguration des Knotens die Spalte, in der die URI enthalten ist, auswählen und anschließend die falsche URI in einem zweiten Dropdown-Menu auswählen. In einem Textfeld daneben müsste dann die richtige URI eingetragen werden. Über einen Button können noch weitere Zeilen mit der Dropbox und dem Textfeld hinzugefügt werden, sollten mehrere URIs auf einmal geändert werden.

6. Ergebnisse

Im letzten Kapitel dieser Arbeit, werden die wesentlichen Punkte noch einmal zusammengefasst, wird das gesamte Projekt noch einmal aus einer kritischen Perspektive betrachtet und es wird resümiert über die gewonnen Erkenntnisse. Abschließend gibt es noch ein Ausblick, wie es mit dem Projekt weiter geht und was noch in Zukunft umgesetzt werden könnte.

6.1 Zusammenfassung

Im Rahmen dieser Abschlussarbeit wurden sieben Knoten für KNIME entwickelt. Als Datenquellen wurden RDF-Dateien und die Daten von der DBpedia ausgewählt. Für das Einlesen der RDF-Daten wurde der RDFReader entwickelt, welcher die enthaltenen Tripel mit Hilfe von Jena in das KNIME-Tabellenformat umwandelt. Für das Exportieren der Tabellen in RDF-Dateien wurde der RDFWriter geschaffen, welcher ebenfalls Jena integriert. Auch hinzugekommen ist der SPARQL-Knoten, welcher die Metadaten mittels der SPARQL-Schnittstelle von dbpedia in KNIME importiert.

Hauptziel war es, eine Architektur zur datenflussorientierten Verarbeitung von Daten/Metadaten-Verbänden zu entwickeln. Darunter zählt neben der Verarbeitung auch die Verknüpfung und Trennung dieser Verbände.

Für die Fusionierung wurden zwei Knoten entwickelt, welche abhängig von der Datenquelle sind. Bei der Verwendung einer RDF-Datei, werden Daten und Metadaten mit Hilfe des Matching-Knotens miteinander verschmolzen. Wenn hingegen die Daten aus dem Internet bezogen werden, dann wird der SPARQL-Knoten verwendet. Um die korrekten Daten von der dbpedia zu erhalten, wurde auch der URICreator entwickelt, welcher mit Hilfe einer Suchmaschine (dbpedia lookup), die URI zum Datensatz ermittelt.

Die fusionierten Daten können mit allen bisher existierenden Verarbeitungsknoten von KNIME weiter verarbeitet werden. Einzig der Erhalt der URI-Spalte ist notwendig, wenn der Verbund wieder getrennt werden soll. Wer von der Abspaltung der Spalten absieht und nicht mit den URIs und den w3c-Datenformaten arbeiten möchte, kann den neu entwickelten URIConverter verwenden, der aus den URIs und w3c-Typen leserliche Strings generiert.

Für Trennung wurde der Separating-Knoten entwickelt, welcher ohne Konfiguration automatisch die Spaltung in zwei Tabellen vornimmt. Sollte der Benutzer den Wunsch haben einzelne Spalten in die andere Tabelle zu überführen, so ist dies in der Konfiguration des Knotens möglich.

Als Alternative zur Fusionierung der Daten, wurde ein Prototyp einer Architektur geschaffen, bei der die Daten und Metadaten lediglich synchronisiert werden (wobei nur die Metadaten an den Daten angepasst werden). Nach dem Durchlaufen des Matching-Knotens gibt es demnach weiterhin zwei verschiedene Ports. Jeder Strang kann so weiterhin separat verarbeitet werden.

6.2 Kritische Betrachtung

In der vorangegangenen Recherche, ob es bereits eine RDF oder SPARQL Umsetzung gibt, wurde ich nicht fündig, konnte aber mehrere Anfragen finden, die den Bedarf an einer Umsetzung von solchen Knoten zu erkennen gaben. Wie hoch der Bedarf letztendlich ist, wird erst nach der Veröffentlichung der Knoten bekannt sein.

Dass von der DBpedia die Datensätze heruntergeladen geladen und dann in KNIME eingelesen werden sollen, ist aufgrund der Größe unrealistisch. Zudem ist es in vielen Fällen aufgrund des begrenzten Arbeitsspeichers nicht möglich oder wegen der langen Verarbeitungszeit nicht zu empfehlen. Realistischer ist hingegen, dass die Metadaten über die SPARQL-Schnittstelle heruntergeladen werden. Dies hat den Vorteil, dass nur die Daten integriert werden, die auch benötigt werden. Anschließend können diese Daten dann in eine RDF-Datei gespeichert werden, damit hat der Benutzer die Möglichkeit im nächsten Workflow auf die relevanten Daten auch lokal zuzugreifen.

Die Informationen, die von der Wikipedia bezogen werden sind immer, bezüglich der Qualität der Daten, mit Bedacht zu sehen. Nicht alles was dort geschrieben steht muss auch der Wahrheit entsprechen. Außerdem unterliegen die Daten einem ständigen Wandel, so dass häufig Informationen hinzukommen oder wegfallen, neue Tags hinzukommen, geändert oder ausgetauscht werden oder veralten und mehr gewartet werden.

Die SPARQL-Schnittstelle von der dbpedia ist über einem Virtuoso Server zu erreichen. Wenn dieser aktualisiert wird, ist die Schnittstelle erst wieder erreichbar, wenn der Administrator diese wieder freigibt. Während dieser Arbeit war deswegen eine Woche lang ein Abfragen der Daten nicht möglich. Selbiges könnte auch in Zukunft erneut passieren, was dazu führt, dass der Knoten vorübergehend unbrauchbar ist.

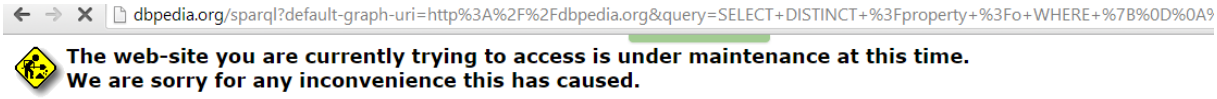

Abbildung 40: Fehlermeldung, wenn der Virtuso Server aktualisiert und noch nicht freigegeben wurde

Da die Online-Abfrage in dieser Arbeit mehr im Fokus stand und somit eine gute Suchmaschine mit DBpedia lookup für das Auffinden der Daten verwendet wurde, ist das Matching etwas in den Hintergrund geraten. Für das Matching wurde lediglich ein String-Vergleichsalgorithmus verwendet, welcher, gegenüber einer Suchmaschine, nicht sonderlich intelligent ist. Dieser dient eher der Veranschaulichung des Knotens und sollte vor der Veröffentlichung mit einer besseren Logik versehen werden.

6.3 Fazit

Ziel dieser Arbeit war es, dem Benutzer zu ermöglichen, seine Daten um Meta-Daten zu erweitern, diese Daten/Metadaten-Verbände zu verarbeiten und gegebenenfalls wieder zu trennen. Für die Umsetzung standen drei verschiedene Varianten zur Verfügung, wobei die Entscheidung zugunsten des synchronen Verarbeitens mit nur einer Kante gefallen ist.

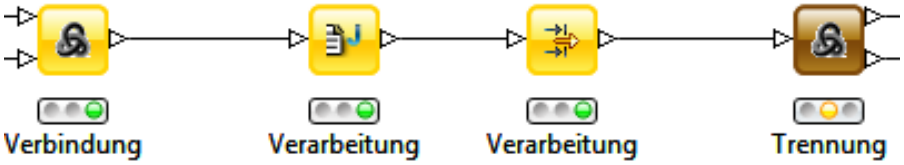

Abbildung 41: KNIME-Workflow-Ausschnitt für die synchrone Verarbeitung mit einer Kante

Diese Auswahl stellt sich im Nachhinein als eine gute Entscheidung heraus. Durch die Datenfusion werden sowohl die Daten, als auch die Metadaten in eine Tabelle verbunden. So ist sofort ersichtlich, welche Informationen zu welcher Reihe gehört. Bei den beiden anderen Varianten, wo die Daten nur synchronisiert werden und dann weiterhin in zwei Tabellen aufgeführt werden, geht schnell die Übersicht verloren, vor allem wenn es sich um besonders viele Reihen handelt. Um dort die Übersicht zu erhöhen, hätte zusätzlich eine neue Darstellungsform geschaffen werden müssen.

Wer dennoch die Daten und Metadaten asynchron bearbeiten möchte, bekommt auch mit der gewählten Umsetzung die Möglichkeit dafür. Dazu müssen lediglich die Daten mit dem Matching- oder SPARQL-Knoten fusioniert werden und anschließend gleich wieder mit den Separating-Knoten getrennt werden. So enthält die Metadaten-Tabelle nur alle relevanten Metadaten und dennoch sind beide Stränge unabhängig voneinander bearbeitbar.

Die synchrone Verarbeitung mit zwei Kanten wurde zurecht nicht in Betracht gezogen, da für diese Verarbeitungsart alle bisherigen Knoten dupliziert und angepasst werden müssten, so dass diese jeweils zwei Ein- und Ausgangsports besitzen und die Funktionalitäten auf beide Datenströme angewandt werden. Eine Anpassung ist bei der gewählten Version nicht notwendig, alle bisherigen Knoten können ohne Probleme verwendet werden.

Aufgrund der bestehenden Struktur war es auch im Bereich der Verknüpfung die geeignetste Variante. Auf den Pipes werden die Tabellen und die Konfigurationen im xml-Format übermittelt. Durch die Datenfusion muss nicht in die Architektur eingegriffen werden. Eine Verknüpfung über IDs, wäre nur mit einem Eingriff in die Architektur möglich gewesen. Entweder indem eine RowSpec geschaffen wird (bisher gibt es nur eine ColumnSpec), in der die Verknüpfung gespeichert wird oder direkt in der Tabelle hätte eine Spalte hinzugefügt werden müssen, welche aber nicht mit angezeigt werden darf. Für das nicht anzeigen hätte wiederum der KNIME-Core bearbeitet werden müssen, damit alle Knoten die Verknüpfung kennen, diese nicht anzeigen und dennoch weitervermitteln an den nächsten Knoten. Bei der dritten Möglichkeit, die Objekte zu serialisieren/deserialisieren wär ebenfalls eine Anpassung des KNIME-Cores notwendig gewesen.

Durch die Umsetzung dieses Projektes können sich die Nutzer von KNIME auf eine sinnvolle Erweiterung freuen. Nun ist es möglich RDF-Daten in KNIME einzulesen und zu exportieren, Daten mit den passenden Metadaten zu fusionieren und weitere Metadaten über den SPARQL-Knoten, von DBpedia abrufen. Diese Daten/Metadaten-Verbände können gemeinsam verarbeitet und auch wieder voneinander getrennt werden.

6.4 Ausblick

Bevor die Knoten als Plugin der Community zur Verfügung gestellt werden, müssen noch Beschreibungstexte erstellt werden, damit der Nutzer gleich sehen kann, was der Knoten bewirkt und was an den Ports angeschlossen werden kann.

Bisher beschränken sich der URICreator- und der SPARQL-Knoten auf die DBpedia. Dies kann zukünftig weiter ausgebaut werden, indem eine andere Suchmaschine verwendet wird, die auch URIs von anderen Datenquellen findet. Als Suchmaschine könnte die bereits vorgestellte Sindice-Engine verwendet werden. Als Datenquellen könnten andere beteiligte Unternehmen des semantischen Webs, welche auch eine SPARQL-Schnittstelle anbieten, in Betracht gezogen werden.

Im bisherigen Algorithmus zur Erstellung der KNIME-Tabellen aus den RDF-Graphen, werden nur Suchen nach den Subjekten durchgeführt. Sollte hingegen das Gesuchte im Triple als Objekt vorkommen, so wird dies nicht mit einbezogen. Da müsste überprüft werden, ob im semantischen Web bidirektionale Verknüpfungen vorhanden sind und wenn nicht, ob die Menge der Daten ansonsten zu groß zum Erfassen wäre. Aktuell ergibt die Suche für Berlin hauptsächlich Informationen direkt über die Stadt. In dem Fall, dass auch nach Objekten gesucht wird, würden sehr viele Personen, welche eine Verbindung mit Berlin haben (zum Beispiel weil sie dort geboren wurden) mit angezeigt.

Durch die Verknüpfung von Daten und Metadaten wird die Grundlage für weitere Projekte gelegt, die eine Realisierung als KNIME-Knoten sinnvoll machen. So auch der Algorithmus von Sebastian Schulz, welcher ebenfalls an der Freien Universität Berlin seine Masterarbeit mit dem Thema „Design and Implementation of a Gateway Process for Runtime Data / Metadata Fusion“ verfasst hat, worin es über eine weitere Fusionierungsart der Metadaten geht, um die Größe der Tabellen und somit auch des Speicherplatzes zu reduzieren und um sinnvolle Gruppierungen der Daten zu ermöglichen.

7. Anhänge

7.1 Literaturverzeichnis

[1] YouTube LLC – Presse. *YouTube Statistiken*. 25.09.2014.
URL: http://www.youtube.com/yt/press/de/statistics.html

[2] Wikimedia. *List of Wikipedias*. 23.09.2014.
URL: https://meta.wikimedia.org/wiki/List_of_Wikipedias

[3] KNIME.COM AG. *KNIME Homepage*. 25.09.2014
URL: https://www.knime.org/

[4] Prof. Dr. Christian Bizer und andere. *DBpedia Homepage*. 09.09.2014
URL: https://www.knime.org/

[5] GeoNames Team. *GeonNames Homepage*. 25.09.2014
URL: http://www.geonames.org/

[6] KNIME.COM AG. *KNIME Open Source Story*. 25.09.2014
URL: http://www.knime.org/knime-open-source-story

[7] KNIME.COM AG. *KNIME Desktop – Marketing Plakat*. 07/2011
URL: http://www.knime.org/files/Marketing/Datasheets/KNIME_Desktop_PDS.pdf

[8] KNIME.COM AG. *Screenshot der Benutzeroberfläche*. 25.09.2014
URL: https://www.knime.org/files/baseflow_800.jpg

[9] KNIME.COM AG. *KNIME Analytics Platform*. 25.09.2014
URL: http://www.knime.org/knime

[10] Rexer Analytics. *Rexer Analytics Homepage*. 25.09.2014
URL: http://www.rexeranalytics.com/

[11] Rexer Analytics. *Data Miner Survey 2013*. 25.09.2014
URL: http://www.rexeranalytics.com/Data-Miner-Survey-Results-2013.html

[12] Markus Lusti. *Data Warehousing und Data Mining: Eine Einführung in entscheidungsunterstützende Systeme*, Springer-Lehrbuch. 2. Auflage (2002)

[13] StatSoft GmbH. *STATISTICA Data Miner*. 25.09.2014
URL: http://www.statsoft.de/produkte/enterprise-produkte/data-miner/

[14] IBM. *SPSS Modeler*. 25.09.2014
URL: http://www-01.ibm.com/software/analytics/spss/products/modeler/

[15] SAS Institute Inc. *SAS Enterprise Miner*. 25.09.2014
URL: http://www.sas.com/en_us/software/analytics/enterprise-miner.html

[16] RapidMiner. *RapidMiner Homepage*. 25.09.2014
URL: http://rapidminer.com/

[17] RapidMiner. *RapidMiner FAQ*. 25.09.2014
URL: http://rapidminer.com/learning/faqs/

[18] KNIME.COM AG. *RapidMiner integration for KNIME*. 27.09.2013
URL: http://tech.knime.org/community/rapidminer-integration

[19] Robert Gentleman und Ross Ihaka. *R Project Homepage*. 25.09.2014
URL: http://www.r-project.org/

[20] Machine Learning Group der University of Waikato. *WEKA Homepage*. 25.09.2014
URL: http://www.cs.waikato.ac.nz/ml/weka/

[21] *RapidMiner Screenshot*. 25.09.2014
URL: http://a.fsdn.com/con/app/proj/rapidminer/screenshots/261388.jpg

[22] W3C – RDF Working Group. *Resource Description Framework (RDF) Standard*. 25.02.2014. URL: http://www.w3.org/RDF/

[23] W3C – RDF Working Group. *RDF 1.1 Turtle Standard*. 25.02.2014
URL: http://www.w3.org/TR/2014/REC-turtle-20140225/

[24] W3C – Tim Berners-Lee. *Notation 3 Logic*. 27.09.2011
URL: http://www.w3.org/DesignIssues/Notation3

[25] W3C – RDF Working Group. *RDF 1.1 XML Syntax*. 25.02.2014
URL: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

[26] W3C – Semantic Web Coordination Group. *Semantic Web*. 25.09.2014
URL: http://www.w3.org/2001/sw/

[27] W3C. *W3C Homepage*. 25.09.2014
URL: http://www.w3.org/

[28] PD Dr. Pascal Hitzler, M.Sc. Markus Krötzsch, Dr. Sebastian Rudolph und Dr. York Sure. *Semantic Web: Grundlagen (eXamen.press)*. 1. Auflage (2007) - Kapitel 1, Seite 9ff

[29] W3C. *OWL 2 Standard*. 11.12.2012
URL: http://www.w3.org/standards/techs/owl

[30] Stefanie Rühle vom Kompetenzzentrum Interoperable Metadaten (KIM). *Kleines Handbuch Metadaten*. 07.09.2012
URL: http://www.kim-forum.org/Subsites/kim/SharedDocs/Downloads/DE/Handbuch/metadaten.html

[31] W3C. *SPARQL 1.1 Query Language Standard*. 21.03.2013
URL: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[32] Thomas R. Gruber. *A Translation Approach to Portable Ontology Specifications*. Seite 2 (Introduction). 1993
URL: http://tomgruber.org/writing/ontolingua-kaj-1993.htm

[33] W3C - Guus Schreiber. *RDF-Graph*.17.06.2014
URL: https://dvcs.w3.org/hg/rdf/raw-file/c8c62e757bed/rdf-concepts/Graph-ex.gif

[34] W3C – RDF Working Group. *Turtle Grammar*. 28.03.2011
URL: http://www.w3.org/TeamSubmission/turtle/#sec-grammar-grammar

[35] PD Dr. Pascal Hitzler, M.Sc. Markus Krötzsch, Dr. Sebastian Rudolph und Dr. York Sure. *Semantic Web: Grundlagen (eXamen.press)*. 1. Auflage (2007) - Kapitel 7, Seite 203ff

[36] W3C - Tim Berners-Lee. *Linked Data*. 27.07.2006
URL: http://www.w3.org/DesignIssues/LinkedData.html

[37] Bernhard Lahres und Gregor Rayman. *Objektorientierte Programmierung*. Galileo Computing, 2. Auflage (2009) – Kapitel 8.2.
URL: http://openbook.galileocomputing.de/oop/oop_kapitel_08_002.htm

[38] W3C. *Linked Data*. 27.07.2006
URL: http://www.w3.org/standards/semanticweb/data

[39] DBpedia. *DBpedia 3.9 Downloads*. 25.09.2014
URL: http://wiki.dbpedia.org/Downloads39

[40] Dan Brickley und Libby Miller. *FOAF Project Homepage*. 25.09.2014
URL: http://www.foaf-project.org/

[41] Mark Watson. *Practical Semantic Web and Linked Data Applications*. 03.11.2010.
URL: http://markwatson.com/opencontent_data/book_lisp.pdf

[42] DBpedia. *The DBpedia Ontology*. 08.09.2014
URL: http://wiki.dbpedia.org/Ontology2014

[43] DBpedia. *The DBpedia Ontology Classes*. 08.09.2014
URL: http://mappings.dbpedia.org/server/ontology/classes/

[44] Apache Software Fundation. *Jakarta Commons httpClient*. 16.06.2011
URL: http://hc.apache.org/httpclient-3.x/

[45] SAS Institute Inc. *SAS Homepage*. 25.09.2014
URL: http://www.sas.com

[46] W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski und A.P. Barros. *Workflow Patterns*. 07/2003.
URL: http://www.workflowpatterns.com/documentation/documents/wfs-pat-2002.pdf

[47] Workflow Patterns Initiative. *Workflow Patterns*. 25.09.2014
URL: http://www.workflowpatterns.com/

[48] H.B. Mitchell. *Data Fusion: Concepts and Ideas*. Springer-Verlag und Heidelberg GmbH & Co. K (2012).

[49] Christian Ullenbom. *Java ist auch eine Insel*. Galileo Computing, 9. Auflage (2010) – Kapitel 17.10.
URL: http://openbook.galileocomputing.de/javainsel9/javainsel_17_010.htm

[50] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Giuseppe Di Fatta, Thomas R. Gabriel, Florian Georg, Thorsten Meinl, Peter Ohl, Christoph Sieb, und Bernd Wiswedel. *Technical Report* von *KNIME: The Konstanz Information Miner*.
URL: http://www.s-in.it/file/prodotto-776.pdf

[51] Apache Software Foundation. *Apache Jena Homepage*. 25.09.2014
URL: https://jena.apache.org/

[52] Apache Software Foundation. *Apache Jena architecture overview*. 25.09.2014
URL: https://jena.apache.org/about_jena/architecture.html

[53] Mark Watson. *DBpediaLookupClient*. 13.11.2013
URL: https://github.com/mark-watson/java_practical_semantic_web/blob/master/src/com/knowledgebooks/info_spiders/DBpediaLookupClient.java

[54] Max Jakob und Chris Bizer. *DBpedia Lookup Service*. 24.04.2013
URL: http://wiki.dbpedia.org/Lookup

[55] OKKAM Srl. *Okkam Homepage*. 25.09.2014
URL: http://api.okkam.org/

[56] OKKAM Srl. *Java Okkam Client*. 17.05.2013
URL: http://community.okkam.org/index.php?option=com_content&view=article&id=55:java-okkam-client&catid=2&Itemid=7

[57] OKKAM Srl. *Okkam Search API*. 25.09.2014
URL: http://api.okkam.org/search/

[58] OKKAM Srl. *Okkam Entity ID request language*. 17.05.2013
URL: http://community.okkam.org/index.php?option=com_content&view=article&id=60:entity-id-request-language&catid=2&Itemid=7

[59] DERI. *Sindice Homepage*. 25.09.2014
URL: http://sindice.com/

[60] DERI. *DERI Homepage*. 25.09.2014
URL: http://deri.ie/
[61] Fondazione Bruno Kessler. *FBK Homepage*. 25.09.2014
URL: http://www.fbk.eu/
[62] OpenLink Software. *OpenLink Software Homepage*. 25.09.2014
URL: http://www.openlinksw.com/
[63] DERI. *Sindice Query Language*. 25.09.2014
URL: http://sindice.com/developers/queryLanguage#QueryLanguage
[64] Dan Brickley und Libby Miller. *FOAF Vocabulary Specification 0.99*. 14.01.2014
URL: http://xmlns.com/foaf/spec/
[65] KNIME.COM AG. *KNIME Analytics Platform - Downloadseite*. 25.09.2014
URL: http://www.knime.org/downloads/overview
[66] Apache Software Foundation. *Writing RDF in Apache Jena*. 25.09.2014
URL: https://jena.apache.org/documentation/io/rdf-output.html
[67] OpenLink Software. *Virtuoso SPARQL Query Editor für die DBpedia*. Virtuoso Version 07.10.3211 vom 25.09.2014
URL: http://dbpedia.org/sparql
[68] Anja Jentzsch. *Grafik von der Cloud der Linked Open Data*. 09/2011
URL: http://de.wikipedia.org/wiki/Linked_Open_Data#mediaviewer/File:LOD_Cloud_Diagram_as_of_September_2011.png

7.2 Quellcode & Workflows

Der Quellcode und die beschriebenen Benutzerszenarien (als KNIME-Workflows) befinden sich auf der angefügten DVD.

7.3 Bilder

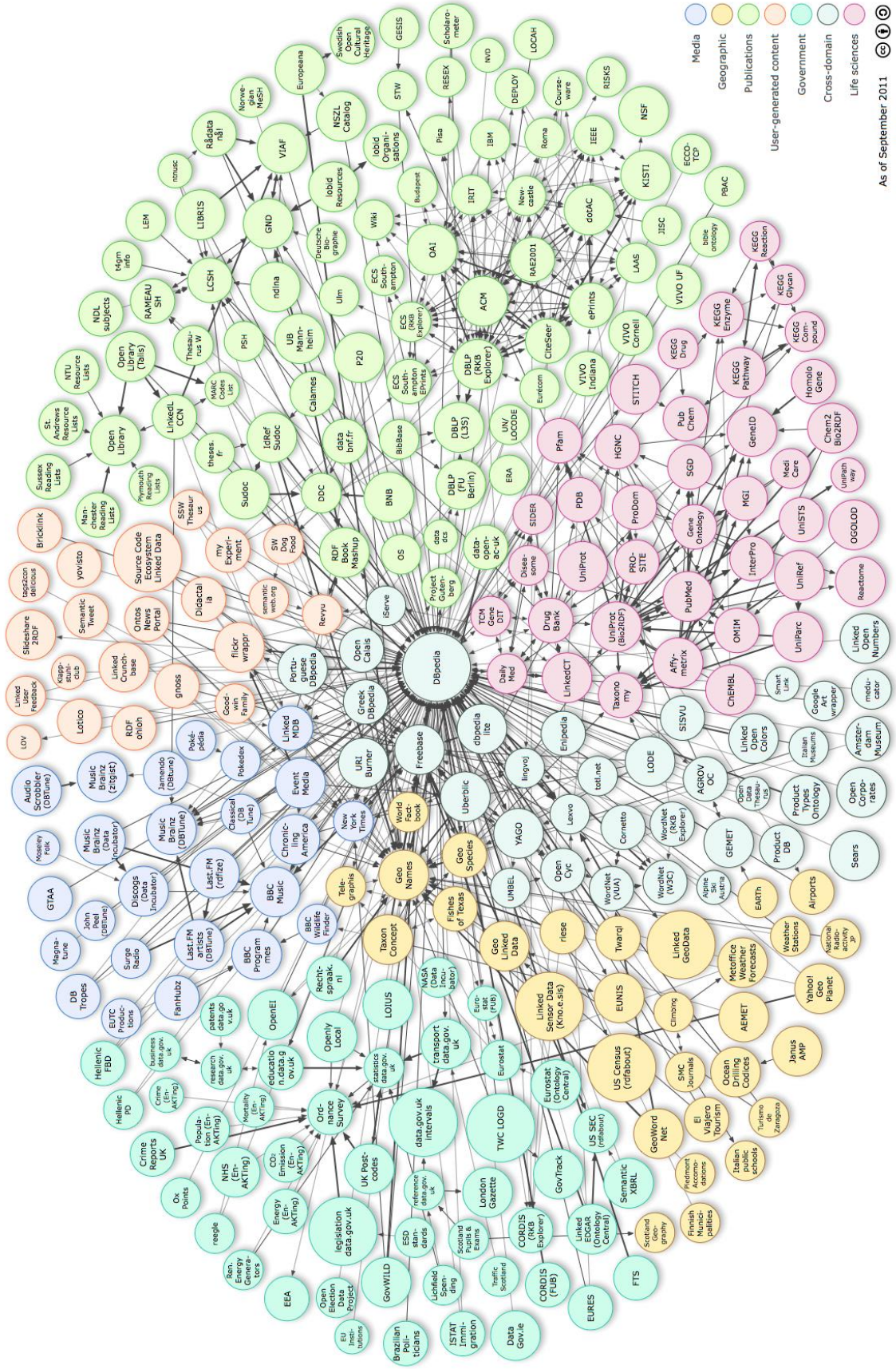

Abbildung 42: Cloud der Linked Open Data [68]