Verwendung von Unschärfe in semantischen Modellen

Bachelorarbeit
Technische Universität Dresden
September 2012

Jonas Schulz

Betreuer: Dipl.-Medieninf. Martin Voigt, Dipl.-Medieninf. Sandro Schmidt
Hochschullehrer: Prof. Dr.-Ing. Klaus Meißner

Fakultät Informatik
Institut für Software- und Multimediatechnik
Seniorprofessur für Multimediatechnik

Aufgabenstellung

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

Erklärung

Hiermit erkläre ich, Jonas Schulz, die vorliegende Bachelorarbeit zum Thema

Verwendung von Unschärfe in semantischen Modellen

selbstständig und ausschließlich unter Verwendung sowie korrekter Zitierung der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel verfasst zu haben.

Dresden, 24. September 2012

Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In diesem einleitenden Kapitel werden die Motivation sowie die grundlegende Problemstellung für diese Bachelorarbeit aufgezeigt und die Arbeit im Kontext des *Topic/S* Projektes[1] der fink & partner Media Services GmbH in Zusammenarbeit mit der Professur Multimediatechnik der Technischen Universität Dresden eingeordnet. Anschließend werden die Ziele verdeutlicht, die mit dieser Arbeit erreicht werden sollen, und der Aufbau dieser Bachelorarbeit erläutert.

1.1 Motivation und Problemstellung

Beschreibt man Daten semantisch, so trifft man meist eindeutige Aussagen über Ressourcen, welche man modellieren möchte. Das *World Wide Web Consortium* (W3C) hat z. B. mit dem *Ressource Description Framework* (RDF) und der *Web Ontology Language* (OWL) Technologien geschaffen, die diese Modellierung ermöglichen und sogar für Maschinen lesbar machen. Nicht selten modelliert man aber Sachverhalte aus der realen Welt, deren Zusammenhänge nicht eindeutig bestimmt sind, was bei der Abbildung auf eine eindeutige Logik Probleme bereitet [Hol09]. So ist die Aussage »der Ball ist rund« für einen Football, welcher eine ovale Form hat, nicht eindeutig. Speziell in Gebieten, bei denen mit Medienobjekten gearbeitet wird, finden sich viele Anwendungsfälle, in denen diese Unsicherheit (uncertainty) bzw. Unschärfe (fuzziness) auftritt und modelliert werden muss. Das Topic/S-Projekt, in dessen Rahmen sich diese Arbeit eingliedert, beschäftigt sich mit der automatisierten semantischen Beschreibung von journalistischen Medienobjekten und der Möglichkeit, diese abzufragen. Im Mittelpunkt steht hierbei der »NewsRoom«, eine Komponente, in der von journalistischen Quellen, wie Agenturen oder Blogs, Medienobjekte eingehen. Der NewsRoom gibt dabei unter anderem einen Überblick über alle derzeitig verfügbaren Artikel. Diese Artikel werden analysiert und deren Metadaten in einer Ontologie eingeordnet. Neben zeitlichen Aspekten und Metadaten, wie Autor und Quelle, wird ein Artikel je nach Inhalt einem Thema und Schlagwörtern zugeordnet. Schlagwörter helfen dabei den Artikel einer oder mehrerer Kategorien zuzuordnen. Bei all diesen Vorgängen treten Probleme der Unschärfe auf, wobei die Daten mit Gewichtungen beschrieben werden müssen, um genauere Aussagen zu erhalten. Ein Beispiel wäre ein Artikel über die Geschwister Michael und Ralf Schumacher, bei dem unter anderem die Schlagwörter »Michael Schumacher«,»Ralf Schumacher« und »Scuderia Ferrari« herausgezogen wurden. Man kann erkennen, dass der Begriff »Ferrari« eine stärkere Gewichtung zur Kategorie »Formel 1« hat als »Michael Schumacher«, welcher auch dem »Motorradsport« zugeordnet werden kann. Neben der Modellierung zeigen sich aber auch Probleme beim Stellen von Anfragen an die Ontologie. Beschreibt man beispielsweise Kategorien von Zeitungsartikeln, so würde

[1] weitere Informationen und Aktuelles unter `http://topic-s.de/`

ein Artikel über ein Sportevent, an dem ein hoher Politiker zuschaute, die Kategorien »Sport« und »Politik« abdecken. Durch die eindeutigen Aussagen geschieht das nun aber zu gleichen Teilen. Eine Maschine kann nicht, wie ein Mensch, aus dem Kontext erkennen, ob der Sport oder die Politik hier einen höheren Stellenwert hat. Probleme entstehen bei einer Anfrage eines politikbegeisterten, sportdesinteressierten Benutzers, dem nun auch alle Sportartikel präsentiert werden, in denen die Politik nur zu einem Bruchteil eine Rolle spielt. Der Benutzer müsste nun selbst nochmal das Ergebnis nach wirklich relevanten Artikeln durchsuchen. Einen großen Nutzen hat unscharfe semantische Modellierung daher bei Anwendungsfällen, wo entschieden werden muss, ab welchem Grad von relevanten Informationen Daten gefiltert werden sollen.
In vielen semantischen Modellen wurden die eben aufgezeigten Probleme der unscharfen Modellierung nicht von vornherein betrachtet, weswegen dies auch schon seit Jahren ein Forschungsschwerpunkt ist. Dabei sind Erweiterungen von Technologien entstanden, die z. B. auf Basis von *Fuzzy-Logik* arbeiten und diese Modellierung ermöglichen. In Bezug auf das Topic/S-Projekt besteht das Problem, wie diese Daten überhaupt semantisch unscharf beschrieben werden können, einschließlich der Frage, welche Voraussetzungen dafür nötig sind und wie man diese geeignet anfragen kann.

1.2 Zielstellung der Arbeit

Diese Arbeit setzt die in der gleichnamigen Seminararbeit angestrebten Ziele fort. Auf Basis der Erkenntnisse und Anforderungsanalyse soll ein Konzept zur automatisierten Gewichtung von Medienobjekten im Topic/S-Projekt und einer geeigneten Suche der erstellten Daten entwickelt werden. Dieses Konzept muss im Detail aus einer Erweiterung einer vorgegebenen Medienontologie, der Verwendung einer geeigneten Technologie zur Modellierung solcher Daten sowie einem Algorithmus bestehen, mit dessen Hilfe die automatisierte Gewichtung und Kategorisierung der Medienobjekte durchgeführt werden kann. Dabei ist auf die Umsetzbarkeit zu achten und das Konzept prototypisch zu implementieren. Schlussendlich muss das Konzept anhand eines geeigneten Beispielszenarios evaluiert werden.

1.3 Aufbau der Arbeit

In Kapitel 2 wird nochmals die Anforderungsanalyse der Seminararbeit zusammengefasst und an die gesteckten Ziele dieser Bachelorarbeit angepasst. Anschließend wird in Kapitel 3 das Konzept vorgestellt und Designentscheidungen diskutiert. Kapitel 4 behandelt die Implementierungsphase, diskutiert dabei erkannte Probleme und deren Lösungen und gibt einen Überblick über den entwickelten Prototypen aus statistischer und funktionaler Sicht. In Kapitel 5 wird das Konzept schließlich mit Hilfe des entwickelten Prototypen evaluiert. Die Arbeit wird mit einer Zusammenfassung und einem Ausblick für weitere Arbeiten im Zusammenhang mit dem Topic/S-Projekt und der Verwendung von Unschärfe abgeschlossen.

2 Anforderungsanalyse

Dieses Kapitel greift die, anhand eines Referenzszenarios erkannten, Anforderungen an das System auf und stellt die Grundlage für die Konzeption dar. Einführend wird das Szenario kurz beschrieben, wobei nur für diese Arbeit relevante Aktivitäten aufgezeigt werden. Anschließend werden funktionale und nichtfunktionale Anforderungen aufgestellt.

2.1 Referenzszenario

Das Topic/S-System dient zur automatisierten Beschreibung von Medienobjekten mit Metadaten und dem Einpflegen in eine Ontologie. Hierbei stehen die Automatisierung und eine, für einen unerfahrenen Nutzer im Rahmen des Semantic Web, einfache Suche nach Daten im Vordergrund. Das nachfolgende Szenario dient als Grundlage zur Bestimmung von Anforderungen für ein Teilsystem, welches sich mit der automatisierten Beschreibung unscharfer Aussagen beschäftigt.
Es wird angenommen, dass ein Medienobjekt, beispielsweise ein Artikel eines Journalisten oder einer Agentur in das System eintrifft. Durch eine *Named Entity Recognition* bzw. *Named Entity Disambiguation* (NER/NED) werden so genannte Entitäten, also Begriffe wie Personen, Orte etc. im Text erkannt und extrahiert. Diese Entitäten, also existierende Dinge, haben einen Eintrag in einer Wissensdatenbasis, wie DBpedia[1]. Im Rahmen dieser Arbeit wird die automatisierte, gewichtete Kategorisierung der Medienobjekte behandelt. Dies bedeutet, dass diesen Entitäten voreingestellte Kategorien mit einer bestimmten Sicherheit, dem so genannten Confidence-Wert, zugeordnet werden. Anhand dieser Gewichtung der Entitäten wird nun ebenso eine Kategorisierung des eingetroffenen Artikels durchgeführt. Der Ablauf ist in Abbildung 2.1 skizziert.
Als Beispiel wird angenommen, dass ein Verlag das Topic/S-System nutzt und ein Artikel eintrifft, der die Karriere der beiden »Formel 1«-Fahrer und Brüder »Michael Schumacher« und »Ralf Schumacher« betrachtet. Diese Entitäten wurden mit einer gewissen Sicherheit, repräsentiert durch einen Confidence-Wert, durch einen Service erkannt, der eine Textanalyse auf den Artikel angewandt und diese Entitäten durch eine NER/NED erkannt hat. Ein Administrator hat bereits die Kategorien »Sport« und »Motorradsport« in seiner Datenbank. Aufgrund dieser Daten errechnen sich Zugehörigkeiten zwischen den Entitäten und den beiden Kategorien. Ausgehend von diesen Daten wird der Artikel ebenfalls kategorisiert. Da der Verlag des Administrators künftig speziell Artikel zur Formel 1 veröffentlichen möchte, fügt er eine weitere Kategorie namens »Formel 1« hinzu. Nun werden die Kategorien, deren Zugehörigkeiten untereinander und die daran geknüpften Aussagen über die Zugehörigkeit zwischen Entitäten bzw. Medienobjekte und den Kategorien neu bestimmt. Diese

[1] http://dbpedia.org

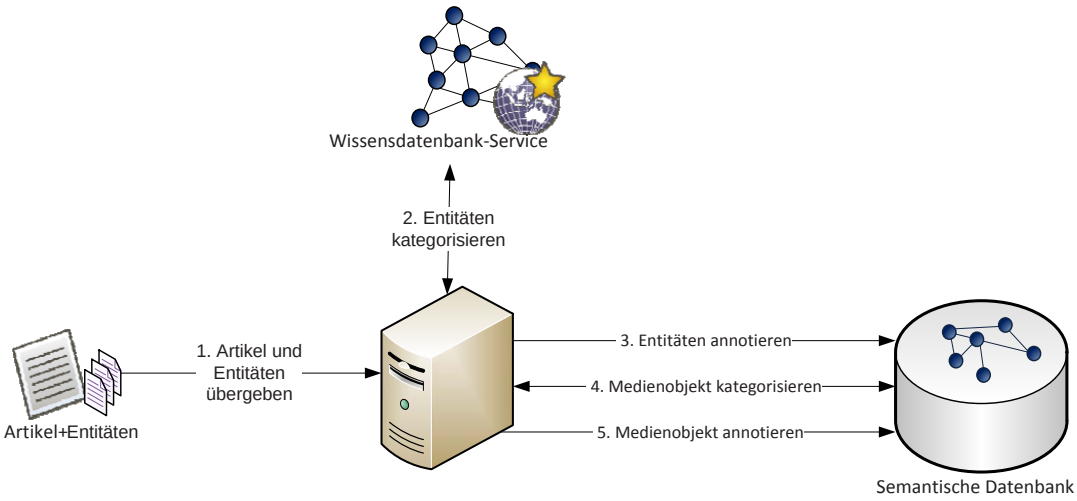


Abbildung 2.1: Darstellung der automatisierten Kategorisierung eines Medienobjekts und seiner Annotation

Änderungen werden in die Ontologie übernommen und sind somit für einen Benutzer durchsuchbar. Während all diesen Vorgängen bleibt das System performant. Die Suche seitens des Benutzers wird durch das direkte Eingeben einer SPARQL-Anfrage ermöglicht.

2.2 Funktionale Anforderungen

Aus diesem Szenario ergeben sich folgende funktionale Anforderungen für diese Arbeit:

1. **Automatisierte Verarbeitung**
   Die Verarbeitung der Metadaten und die Gewichtung der Artikel muss vollkommen ohne menschliches Einwirken geschehen und dabei zuverlässig sein. Es muss eigenständig entschieden werden, ob Daten neu berechnet werden müssen oder bereits bestehen und daher aus der Datenbank übernommen werden können.

2. **Gewichtung**
   Mit Hilfe von geeigneten Services muss bestimmt werden können, mit welcher Sicherheit Kategorien, Entitäten und Medienobjekte - im speziellen journalistische Artikel - von Kategorien abgedeckt werden. Hierzu muss ein geeigneter Algorithmus entwickelt und bestehende Gewichtungen von Aussagen über Zugehörigkeiten zwischen Objekten in der Ontologie berücksichtigt werden.

3. **Services und Datenbasen nutzen**
   Um aus Texten extrahierte Entitäten zu kategorisieren, müssen bestehende Wissensdatenbanken bzw. -ontologien genutzt werden, welche beispielsweise

über einen Service erreichbar sind. Beispiele für nutzbare Services sind DBpedia und Wikipedia[2].

4. **Erweiterung einer Medienontologie**
   Es muss eine bestehende Medienontologie so erweitert werden, dass Gewichtungen modelliert werden können, die bisherige Funktionalität jedoch nicht eingeschränkt wird. Daher muss besonders auf die Verwendung von semantischen Technologien geachtet werden, die vom W3C empfohlen werden und kompatibel zu der bisherigen Ontologie sind.

5. **Suche von unscharf gewichteten Daten**
   Es muss möglich sein, die unscharf modellierten Daten in der Ontologie anfragen zu können. Dazu müssen verschiedene Fälle betrachtet werden und Beispielanfragen evaluiert werden.

6. **Reasoning**
   Aufgrund fehlender Standards ist ein Reasoning von unscharfen Ausdrücken nicht möglich, jedoch darf dadurch die Funktionalität für nicht annotierte Teile der Ontologie nicht eingeschränkt werden. Das heißt, dass trotz der Erweiterung der Ontologie das Nutzen eines Reasoners auf Basis von Standards möglich sein muss. Alternativ müssen für unscharfe Aussagen Wege gefunden werden Zugehörigkeiten zu schlussfolgern.

7. **Kategorisierung**
   Da das Hauptaugenmerk auf der Kategorisierung liegt, müssen entsprechend Kategorien nutzbar gemacht und deren Zugehörigkeiten untereinander modelliert werden. Dazu müssen diese in Form einer Hierarchie strukturiert bzw. einander zugeordnet werden. Des Weiteren werden Kategorisierungen von Medienobjekten und Entitäten durchgeführt. Alle Zugehörigkeiten werden unter Berücksichtigung von Unschärfe modelliert.

2.3 Nichtfunktionale Anforderungen

Außerdem ergeben sich folgende nichtfunktionale Anforderungen für diese Arbeit:

1. **Einfachheit**
   Die genutzte Medienontologie und verwendete Technologien sollen so einfach wie möglich gehalten werden, um sowohl eine bessere Wartbarkeit des Systems, als auch eine gute Performanz sicherzustellen. Hier bietet sich besonders das *RDF Schema* (RDFS) an, da hier ausreichend Sprachmittel bei vergleichsweise wenig Komplexität zur Web Ontology Language zur Verfügung stehen.

2. **Standardkonformität**
   Es muss vor allem darauf geachtet werden, dass bestehende Standards ausgeschöpft und auch genutzt werden. Für das Topic/S-Projekt bedeutet dies die Nutzung der Standards und Empfehlungen des W3C, da diese weit verbreitet und anerkannt sind.

[2] http://www.wikipedia.org/

3. **Performanz**
   Die entwickelten Funktionalitäten sollen die Performanz nicht negativ beeinflussen. So ist vor allem zu beachten, dass Flaschenhälse im System erkannt und möglichst umgangen werden. Bei der Annotation muss eine Datenbasis geschaffen werden, die schnell nutzbar ist und nicht durch ständige Rechenoperationen die Performanz des Systems verschlechtert.

4. **Wiederverwendbarkeit**
   Das Konzept soll für verschiedene Systeme wiederverwendbar sein. Dies bedeutet, dass Beschreibungen für die Benutzung einer semantischen Wissensdatenbasis zum Bestimmen und Berechnen von Zugehörigkeiten und einer semantischen Datenbank zur Speicherung der erzeugten Daten generisch, also allgemein formuliert, sein müssen.

2.4 Zusammenfassung

In diesem Kapitel wurde eine Anforderungsanalyse anhand eines Referenzszenarios vorgenommen. Diese dient als Grundlage für die Entwicklung eines Konzepts zur gestellten Problematik der automatisierten, gewichteten Annotation von Medienobjekten. Die Datenbasis ist hierbei semantisch und bedarf besonderer Technologien, wobei sich auf Standardisierungen des W3C beschränkt werden muss. Insbesondere ist auf eine einfache Modellierung und eine möglichst performante Verarbeitung der Medienobjekte durch das System zu achten. Es müssen Beziehungen zwischen Kategorien beachtet und in die Berechnungen der Zugehörigkeiten von Medienobjekten, Entitäten und Kategorien zu relevanten Kategorien eingebracht werden.

3 Konzeption

In diesem Kapitel wird das Thema dieser Arbeit genauer in das Topic/S-Projekt eingeordnet und aufgezeigt, welche Einschränkungen und Voraussetzungen im Rahmen der Arbeit getroffen werden. Anschließend folgt das Konzept zur Lösung des in Kapitel 2.1 beschriebenen Szenarios. Dabei werden auf die Erweiterung einer bestehenden Medienontologie und die Architektur des Systems eingegangen, die Komponenten und deren Beziehungen zueinander beschrieben sowie Algorithmen zur automatisierten Bestimmung von Zugehörigkeiten zwischen Kategorien mit Medienobjekten, Entitäten und Kategorien selbst vorgestellt.

3.1 Einordnung und Abgrenzung im Rahmen des Topic/S-Projekt

Betrachtet man das in der Anforderungsanalyse vorgestellte Szenario, so erkennt man, dass das Hauptaugenmerk dieser Arbeit auf der Berechnung von so genannten Confidence-Werten für Medienobjekte, Entitäten und Kategorien zu anderen Kategorien liegt. Diese Vorgänge werden durch unterschiedliche Aktionen ausgelöst. So wird die Kategorisierung eines Medienobjektes durch das Eintreffen eines solchen im System ausgelöst. Zugehörigkeiten zwischen Kategorien hingegen werden berechnet, wenn durch einen Administrator des Systems Kategorien hinzugefügt oder entfernt werden.
Die manuelle Übergabe von Kategorien an das Systems ist die erste Einschränkung, die betrachtet werden muss. Sie ist mit dem Erhalt der Performanz des Systems begründet. In einer Wissensontologie, wie DBpedia, kann sich eine große Menge von Kategorien befinden[1], die für den Nutzer des Systems nicht relevant sind. Eine Synchronisation aller Kategorien zwischen Service und dem Topic/S-System würde im Fall der DBpedia Millionen von Berechnungen zufolge haben. Dieser Rechenaufwand steht in keinem Verhältnis zu dem Nutzen, den der Betreiber des Systems hat, wenn er nur eine handvoll relevanter Kategorien betrachten möchte. Im Pressewesen, in dem das Topic/S-System genutzt werden soll, sind solche großen Zahlen an Kategorien ebenfalls nicht zu erwarten. Daher wird im Rahmen dieses Konzepts vorausgesetzt, dass der Benutzer entscheidet, welche Kategorien er im System aufnehmen möchte und welche nicht. Dadurch wird das System entlastet. Auf Basis dieser festgelegten Kategorien werden dann die Zugehörigkeiten zu Entitäten und schlussendlich zu Medienobjekten bestimmt.
Eine zweite Einschränkung bzw. klare Abgrenzung ist, dass im Rahmen dieser Arbeit und des Konzepts keine Named Entity Recognition betrachtet, aber vorausgesetzt

[1] so hat die englische DBpedia 24.424.883 verschiedene Kategorien, die deutsche DBpedia hingegen 2.772.410 . Stand: 11.09.2012

wird. So werden dem Teilsystem das behandelte Medienobjekt, seine erkannten Entitäten, deren Typ und ein zugehöriger Confidence-Wert übergeben. Der Confidence-Wert gibt hierbei an, wie sicher die NER über die Richtigkeit der Entität im Kontext des Medienobjektes, z. B. nach einer Textanalyse, ist. Diese Confidence-Werte gehen ebenfalls in die Berechnungen ein und werden auch in die Ontologie übernommen. Die übergebenen Entitäten bzw. deren Internationalized Resource Identifier (IRI) müssen im selben Graphen liegen, der vom Service einer Wissensdatenbank genutzt wird. Das heißt, dass bei der Nutzung verschiedener Services zur Erkennung der Entitäten und Kategorisierung derselbigen die richtigen IRI ermittelt und übergeben werden müssen. Diese klare Abgrenzung des Systems ist wichtig, damit der Nutzer nicht an einen NER- oder NED-Service gebunden ist.
Eine dritte Einschränkung ist, dass die Modellierung der Ontologie dieselbe sein muss, wie sie in Kapitel 3.3 beschrieben wird. Diese Einschränkung wird getroffen, um die für das Konzept in dieser Bachelorarbeit gewählten Modellierungstechnologie umsetzen und geeignete Anfragen für spezielle Problemfelder vorstellen zu können.
In Bezug auf die vorgegebene Ontologie gibt es eine vierte Einschränkung. So werden nicht alle Bestandteile der originalen Ontologie genutzt bzw. betrachtet. Der Grund dafür ist, dass die, auch später in Kapitel 3.2 vorgestellten, Komponenten der Ontologie für die Umsetzung des Konzepts ausreichen. Alle Daten, die hier erfasst werden, können für andere Komponenten übernommen oder referenziert werden. In Kapitel 3.2 wird darauf detaillierter eingegangen.
Die fünfte und letzte Einschränkung dieses Konzepts ist, dass kein User Interface für die administrative und explorative Nutzung der Daten vorgestellt wird, da dies im Rahmen dieser Bachelorarbeit ein zu komplexes Thema wäre. Daher wird hier nur die Modellierung der Unschärfe behandelt und gängige Beispielanfragen an das System betrachtet.
Mit Hilfe dieser Einschränkungen soll vor allem die Umsetzung der folgenden Anforderungen aus der vorangegangenen Anforderungsanalyse unterstützt werden:

- Eine **Medienontologie** wird insoweit vorgegeben, dass eine **Modellierung der unscharfen Aussagen mit einfachen Sprachmitteln** und die **Suche von unscharf gewichteten Daten** möglich sind.

- Die **Performanz** der **automatisierten Kategorisierung** soll durch die Verwaltung ausgewählter Kategorien seitens des Nutzers gesteigert werden. Eine Synchronisierung der gesamten Wissensdatenbank eines Service mit dem System wäre viel zu langsam, um sie in Betracht zu ziehen.

- Durch das Bereitstellen von Schnittstellen zur Übergabe von Medienobjekten und Entitäten erhöht sich die **Wiederverwendbarkeit des Konzeptes**, da Named Entity Recognition Services unabhängig vom System genutzt werden können und die zu übergebenen Objekte einheitlich formatiert werden müssen.

3.2 Erweiterte Medienontologie

Um die unscharfen Aussagen über Zugehörigkeiten zwischen Objekten mit Confidence-Werten in die Ontologie einbringen zu können, muss das Schema der Topic/S-Ontologie erweitert werden. Ein Ausschnitt des Schemas der ursprünglichen Topic/S-Ontologie ist in Abbildung 3.1 dargestellt. Die dort dargestellten Ressourcen sind solche, deren Beziehungen zueinander mit Unschärfe beschrieben werden können.

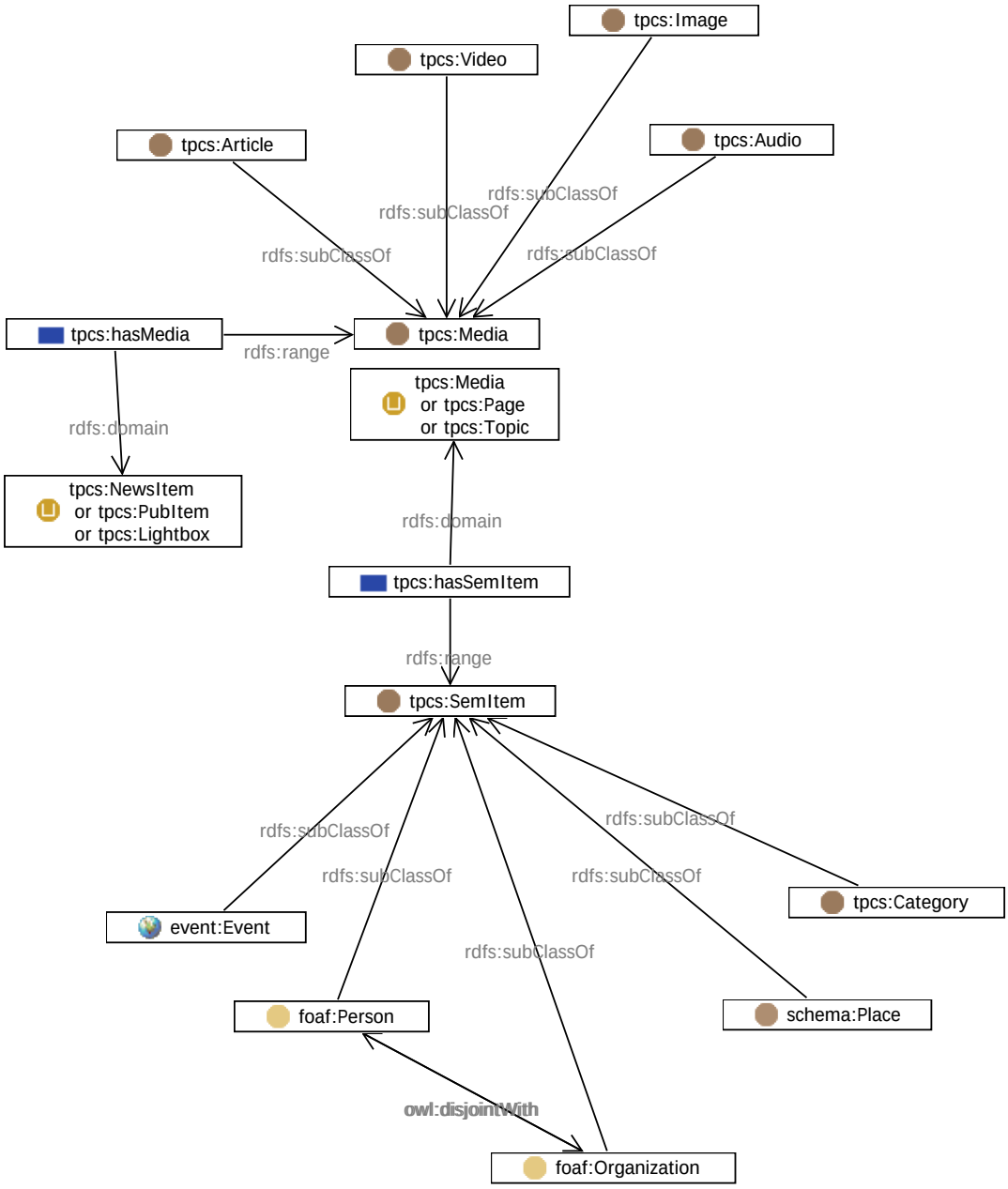


Abbildung 3.1: Schema der ursprünglichen Topic/S-Ontologie als Graph

Die Ontologie ist so aufgebaut, dass Medienobjekte sogenannte semantische Items (`tpcs:SemItems`) besitzen. Dies sind im Speziellen Entitäten wie Events, Orte, Personen oder Organisationen und Kategorien. Diese Medienobjekte können schlussendlich als zu veröffentlichende Items (`tpcs:PubItems`) zusammengefasst und für Zeitungsseiten (`tpcs:Page`) verwendet werden. Einige dieser Beziehungen können hierbei nicht immer eindeutig ausgedrückt werden. So kann man beschreiben, dass ein Medienobjekt einer Kategorie einzuordnen ist, bzw. mit ihr in Beziehung steht, allerdings nicht zu welchem Grad. Genauso kann bei einer Textanalyse eines Artikels eine Menge von Entitäten bestimmt werden. Jedoch würde man in der Ontologie keine Aussage darüber finden, mit welcher Sicherheit man diese Entität in der Analyse bestimmt hat. Im Detail gibt es folgende unscharfe Zugehörigkeiten auf Basis der gesamten Topic/S-Ontologie:

- Eine Kategorie (`tpcs:Category`) ist einer anderen Kategorie unterzuordnen, ihr also zugehörig.
- Eine Entität (`tpcs:SemItem, foaf:Organization, foaf:Person, event:Event, schema:Place`) ist einer Kategorie zugehörig.
- Ein Medienobjekt (`tpcs:Media`) ist einer Kategorie zugehörig.
- Ein Medienobjekt ist einer Entität zugehörig.
- Eine Thema (`tpcs:Topic`) ist einer Kategorie zugehörig.
- Eine Zeitungsseite (`tpcs:Page`) ist einer Kategorie zugehörig.
- Ein veröffentlichtes Objekt (`tpcs:PubItem`) ist einer Kategorie zugehörig.

Auf das Referenzszenario rückblickend sollen die Zugehörigkeiten für Kategorien, Entitäten und Medienobjekten, nicht aber für Zeitungsseiten oder zu veröffentlichende Items zu relevanten Kategorien betrachtet werden. Der Grund hierfür ist, dass die zu betrachtenden Objekte unterschiedliche Arten der Bestimmung und Berechnung dieser Zugehörigkeiten verlangen. Zugehörigkeiten von Kategorien und Entitäten werden durch das Nutzen eines Services mit einer Wissensdatenbank, hingegen die von Medienobjekten nur auf Basis ihrer erkannten Entitäten bestimmt. Analog zur Berechnung für Medienobjekten verhält sich schließlich die Berechnung für Themen, Zeitungsseiten etc. Daher können diese Objekte zunächst außen vor gelassen werden. Da es bisher keinen Standard im Umgang mit unscharfen semantischen Daten gibt, muss eine alternative geeignete Art der Modellierung gewählt werden. Auf Basis bisheriger Recherchen in der vorangegangen Seminararbeit erhielt man das Ergebnis in Tabelle 3.1. Ausgehend von diesem Vergleich wurde für das Konzept die Modellierung mit n-ären Relationen[2] ausgewählt. Die Gründe hierfür sind vor allem die Standardkonformität sowie die einfache Umsetzung der unscharfen Modellierung und Abfrage dieser Daten.

[2] http://www.w3.org/TR/swbp-n-aryRelations/

	Unschärfe-modellierung mit Standard	Einfachheit (Schreiben und Lesen)	Speicherbedarf	Suche mit SPARQL-Standard	Reasoning (von Fuzzy-Logik)
N-äre Relation	✔	❍	❍	✔	✘
Reifikation OWL	✔	❍	✘	✔	✘
Annotations OWL 2	✔	✘	❍	✔	✘
Ontology	❍	✘	✘	✘	✔
N-Quads	✘	✔	✔	✘	✘
aRDFS+AnQL	✘	✔	✔	✘	❍

Tabelle 3.1: Vergleich der Technologien anhand verschiedener Anforderungen.
✔ = gut erfüllt, ❍ = bedingt erfüllt, ✘ = nicht erfüllt

Durch Anwenden der n-ären Relationen zur Modellierung von unscharfe Aussagen wird eine neue Klasse namens `tpcs:FuzzyNode` eingeführt, deren Instanzen als Hilfsknoten funktionieren. Mit diesen Hilfsknoten werden Zugehörigkeiten zwischen Objekten, wie Medien, Zeitungsseiten, Themen, semantischen Items und relevanten Kategorien ausgedrückt. Dies geschieht über die `tpcs:hasFuzzyNode`- und `tpcs:hasSemItem`-Property. Das Gewicht, welches die unscharfe Aussage erhält wird im weiteren Confidence-Wert genannt und gibt an, wie sicher sich das System auf Basis von Berechnungen über diese Zugehörigkeit ist. Der Wert wird über eine `tpcs:hasConfidence`-Property in Bezug auf diese Zugehörigkeit ausgedrückt und mit dem Hilfsknoten modelliert. Zusätzlich wird jedem `tpcs:SemItem` eine `tpcs:externalResource`-Property hinzugefügt, mit der auf die Ressource im genutzten Service verwiesen wird. Außerdem ist es nun für Entitäten und Kategorien möglich Zugehörigkeiten zu anderen Kategorien zu besitzen. Dies war zuvor nur den Medienobjekten vorbehalten. Für die in diesem Konzept relevanten Objekte ergibt sich das erweiterte Schema in Abbildung 3.2. Eine weitere Möglichkeit die Medienontologie zu erweitern ist das Erstellen einer Ontologie, mit der Unschärfe modelliert wird, und anschließende Importieren der Medienontologie. Der in dieser Arbeit gewählte Lösungsansatz erweitert jedoch nicht nur die Ontologie, sondern ändert auch minimal deren Struktur. So sind Medienobjekte mit Kategorien und Entitäten nicht mehr direkt in Relation, da der Hilfsknoten als Mittelpunkt der Relationen definiert wird. Würde man die Medienontologie importieren, wäre es

immer noch möglich Aussagen zu modellieren, welche ohne Gewichtung getroffen werden. Um nicht gleichzeitig unscharfe und scharfe Aussagen zu betrachten und jeweils differenziert behandeln zu müssen, wurde diese Form der Ontologieerweiterung genutzt.

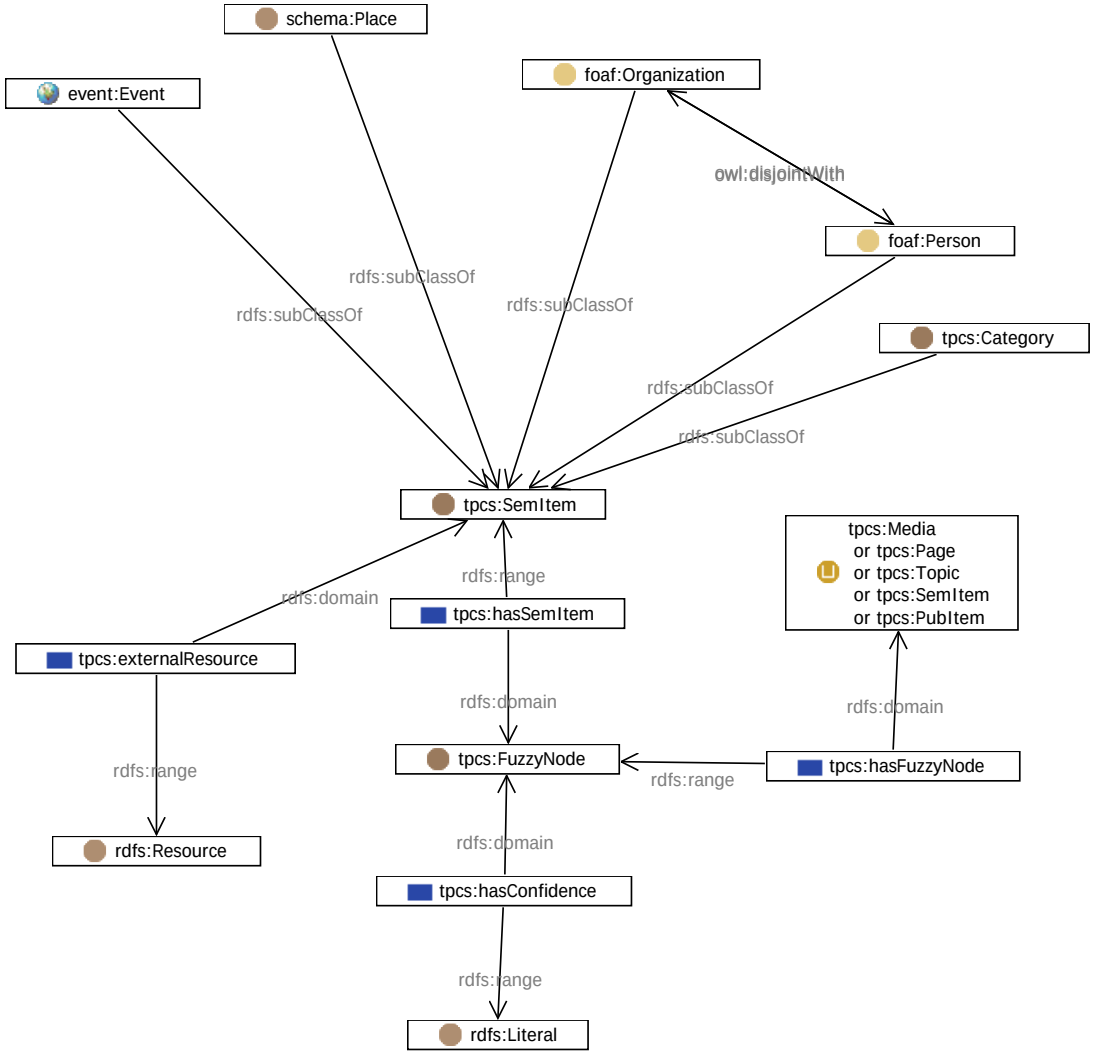


Abbildung 3.2: Erweiterung der Topic/S-Ontologie mit Hilfsknoten und entsprechenden Properties

Ein Beispiel für die Modellierung mit Hilfsknoten in n-ären Relationen ist in Listing 3.1 zu sehen. Hier wird die Kategorie `tpcs:Motorsport` in den beiden relevanten Kategorien `tpcs:Sport` und `tpcs:Motortechnik` mit entsprechenden Confidence-Werten zugeordnet, wobei zwei Hilfsknoten namens `tpcs:MotorsportSport` und `tpcs:MotorsportMotortechnik` instanziiert wurden. Außerdem wird zu der Kategorie `Motorsport` seine externe Ressource, hier am Beispiel der deutschen DBpedia, gespeichert.

```
tpcs:Motorsport a tpcs:Category;
                tpcs:externalResource <http://de.dbpedia.org/resource/Kategorie:
                    Motorsport>.

tpcs:MotorsportSport a tpcs:FuzzyNode;
                tpcs:hasSemItem tpcs:Sport;
                tpcs:hasConfidence "0.5"^^xsd:float.

tpcs:MotorsportMotortechnik a tpcs:FuzzyNode;
                tpcs:hasSemItem tpcs:Motortechnik;
                tpcs:hasConfidence "0.5"^^xsd:float.

tpcs:Motorsport tpcs:hasFuzzyNode tpcs:MotorsportSport;
                tpcs:hasFuzzyNode tpcs:MotorsportMotortechnik.
```

Listing 3.1: Beispielhafte Modellierung durch n-äre Relation

Analog zu diesem Beispiel können alle unscharfen Aussagen modelliert werden. Eine SPARQL-Anfrage, welche alle Kategorien, denen die Kategorie Motorsport zugeordnet ist mit zugehörigen Confidence-Werten zurückliefert, ist in Listing 3.2 dargestellt. Weitere Beispiele für übliche SPARQL-Anfragen an die Ontologie werden im Kapitel 4 vorgestellt.

```
SELECT ?category ?confidence
WHERE{
       tpcs:Motorsport tpcs:hasFuzzyNode ?fuzzynode.
       ?fuzzynode tpcs:hasSemItem ?category.
       ?fuzzynode tpcs:hasConfidence ?confidence.
}
```

Listing 3.2: Beispielhafte SPARQL-Anfrage

Nachdem die Erweiterung der Ontologie nun das Modellieren von unscharfen Aussagen zulässt, muss eine geeignete Architektur für das Teilsystem gefunden werden. Diese Architektur mit ihren Komponenten wird im folgenden Abschnitt vorgestellt.

3.3 Architektur

Die grobe Architektur des Teilsystem zur automatisierten, unscharfen Kategorisierung ist in Abbildung 3.3 dargestellt. Das System kann in vier Hauptkomponenten eingeteilt werden: Frontend, Backend, Triplestore und einem Service mit einer Wissensdatenbank. Nachfolgend werden zu jedem dieser Teile seine Funktion und wichtige Komponenten beschrieben.

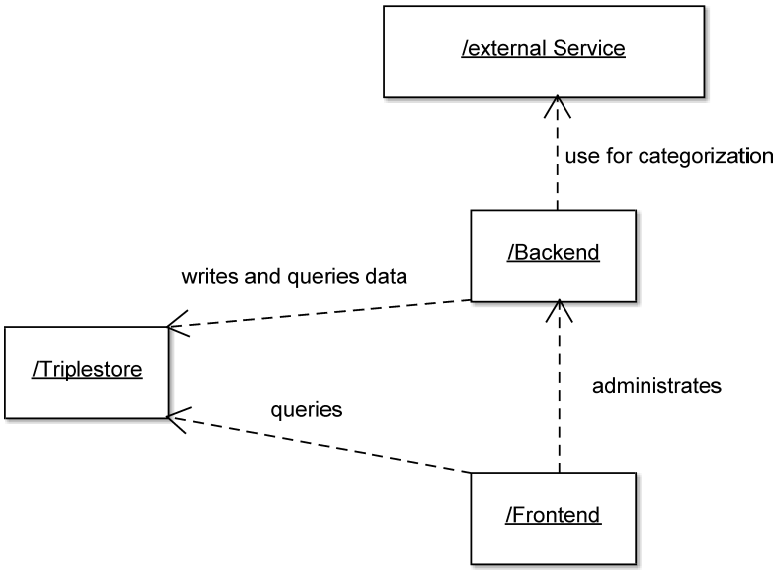


Abbildung 3.3: Grobe Architektur des Systems

Frontend

Mithilfe des Frontends wird einem Nutzer ermöglicht, Suchanfragen an das System zu stellen sowie administrative Aufgaben auszuführen. Administrative Aufgaben können hierbei das Verwalten von Kategorien oder das Hinzufügen von Medienobjekten und Entitäten sein. Es kann sowohl als eine Desktop-Anwendung als auch in Form einer webbasierten Oberfläche realisiert werden. Im Rahmen dieser Bachelorarbeit werden keine weiteren Ausführungen zu einer möglichen Oberfläche und Funktionsumfang behandelt, da der Umfang zur Entwicklung für diese Arbeit zu groß ist.

Backend

Dieser Bestandteil bildet die Kernkomponente des Systems, wie auch für das Konzept. Hier werden alle Daten berechnet und verarbeitet. Aufgrund seiner zentralen Rolle, muss dieser Teil auch so implementiert werden. Dieser kann einerseits in ein bestehendes System integriert oder andererseits als Webservice umgesetzt werden. Die Beziehungen zwischen den einzelnen Komponenten sind dabei in Abbildung 3.4 dargestellt.

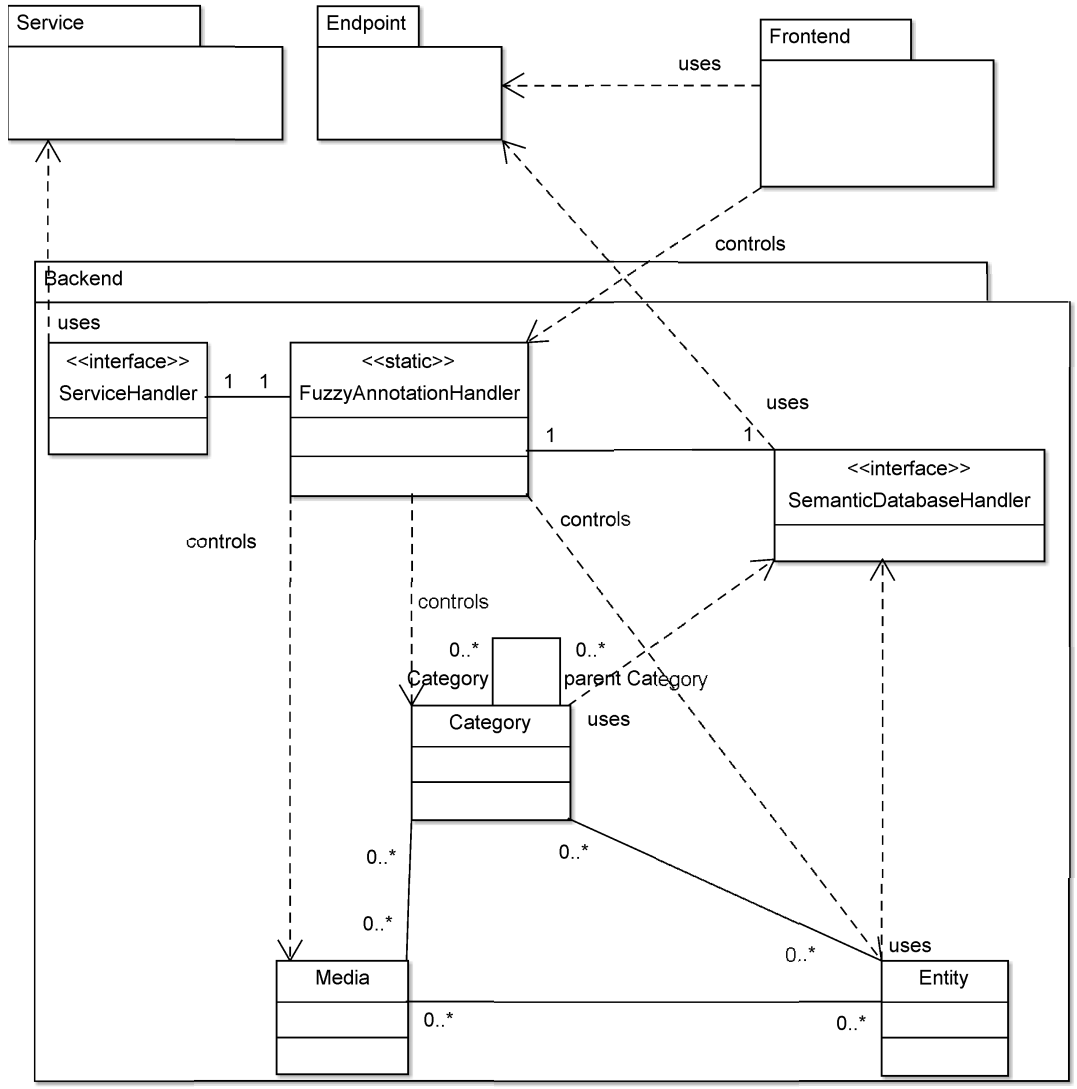


Abbildung 3.4: Beziehungen zwischen den Komponenten im Backend

Das Backend wird als zentrale Komponente vom Frontend angesprochen, verwaltet aber auch gleichzeitig den Triplestore und den Service der Wissensdatenbank. Medienobjekte, Entitäten und Kategorien sind die Komponenten im Backend, die während der Laufzeit verarbeitet werden. Wie schon in Kapitel 3.2 gezeigt wurde, bestehen unter diesen Objekten Zugehörigkeiten, die auch hier in der Abbildung dargestellt sind. Des Weiteren wird eine zentrale Komponente im Backend benötigt, welche Schnittstellen zum Service und zur Datenbank bereitstellt und verwaltet. In Abbildung 3.4 wird diese Komponente als `FuzzyAnnotationHandler` bezeichnet. Die bereitgestellten Schnittstellen dienen zur Wiederverwendung des Konzepts und sind vor der Nutzung des Systems zu implementieren. Beim Start des Systems müssen die implementierten Schnittstellen außerdem beim `FuzzyAnnotationHandler` registriert werden.

Triplestore

Der Triplestore dient als semantische Datenbank und ist daher der wichtigste Bestandteil zur Datenverwaltung und beinhaltet die in Kapitel 3.2 erweiterte Medienontologie. Alle in das Backend eingehenden Medienobjekte, Entitäten und Kategorien werden letztendlich hier mit unscharfer Modellierung in die Ontologie eingefügt. Der Triplestore kann dabei entweder ein bestehendes System, wie Virtuoso[3] oder OWLIM[4], oder ein selbst programmierter Triplestore sein, der über Framework API's, wie von Jena[5] und Sesame[6], angesprochen werden kann. Als Verbindung zum Backend dient dabei der `SemanticDatabaseHandler`. Diese Schnittstelle muss alle wichtigen Funktionalitäten bereitstellen, um SPARQL-Anfragen stellen zu können. Dies umfasst sowohl SELECT- und ASK-Anfragen als auch die Möglichkeit SPARQL-Updates zu nutzen und die Aufbereitung erhaltener Anfrageergebnisse zur geeigneten Weiterverarbeitung.

Service

Unter einem Service versteht man im Zusammenhang mit dieser Arbeit eine semantische Wissensdatenbank, die als Ausgangsbasis zur Bestimmung von Zugehörigkeiten zu Kategorien und entsprechenden Berechnungen der Confidence-Werte dient. Der Service kann sowohl extern als auch intern implementiert sein. Ein Beispiel für solch einen externen Service ist DBpedia. Entitäten und Kategorien, deren Zugehörigkeiten untereinander bestimmt werden sollen, müssen dabei in der selben Datenbank liegen. Wie beim Triplestore gibt es hier im Backend eine Schnittstelle, die implementiert werden muss. Diese Schnittstelle heißt `ServiceHandler`. Funktionalitäten, die dabei im `ServiceHandler` umgesetzt werden müssen, umfassen das Prüfen einzelner Ressourcen auf ihre Existenz und Verarbeiten möglicher Oberkategorien.

Mithilfe dieser Architektur und ihrer einzelnen Komponenten ist es möglich Zugehörigkeiten zwischen Medienobjekten, Entitäten und Kategorien automatisiert zu bestimmen und zu gewichten. In Bezug auf die Anforderungsanalyse ist dieses Konzept durch die Nutzung generischer Schnittstellen für Triplestore und Service wiederverwendbar. Gerade dadurch ist es möglich unterschiedliche Services und Datenbasen zu nutzen, wobei die Funktionalitäten allerdings selbst implementiert werden müssen.
Im nächsten Abschnitt werden die einzelnen Teilalgorithmen, die zur Vervollständigung des Konzepts umgesetzt werden müssen, vorgestellt.

[3] http://virtuoso.openlinksw.com
[4] http://www.ontotext.com/owlim
[5] http://jena.apache.org/documentation/tdb/java_api.html
[6] http://www.openrdf.org

3.4 Algorithmen zur unscharfen Kategorisierung

Dieser Abschnitt widmet sich den Algorithmen zur Erfüllung der Ziele dieser Bachelorarbeit. Dazu werden zunächst Kernprobleme erläutert und deren Lösung anschließend an konkreten Algorithmen mit Hilfe von Aktivitätsdiagrammen aufgezeigt. Diese dienen schließlich der prototypischen Umsetzung in Kapitel 4.

3.4.1 Grundlegende Probleme

Grundlegend ergeben sich aus der Zielstellung der Arbeit folgende Hauptprobleme: *Das Finden eines Algorithmus' zur automatisierten Gewichtung und Beschreibung von Medienobjekten, die Nutzung einer geeignete Modellierung der erhaltenen Daten, sowie die Möglichkeit der Abfrage der modellierten Daten.* Im vorhergehenden Abschnitt wurde bereits die Modellierung anhand der erweiterten Medienontologie erläutert und auch die Möglichkeit solche Daten mit SPARQL abzufragen vorgestellt.
Für die ausstehende Problemstellung, der automatisierten Gewichtung, wurde bereits im Referenzszenario ein grober Ablauf skizziert. Dabei wird im Topic/S-System durch ein ankommendes Medienobjekt und seinen übergebenen Entitäten eine Reihe von Aktivitäten ausgelöst, wobei Zugehörigkeiten von diesen Objekten zu Kategorien berechnet werden. Dieser Ablauf, welcher im Detail in diesem Kapitel vorgestellt wird, ist im Sequenzdiagramm in der Abbildung 3.5 dargestellt. Hier wird ein Medienobjekt mit seinen Entitäten dem Teilsystem übergeben. Eine Überprüfung beim Service soll zeigen, ob diese Entitäten existieren und damit überhaupt möglich ist, Zugehörigkeiten zu Kategorien zu bestimmen. Anschließend werden alle Zugehörigkeiten jeder Entität zu Kategorien berechnet und diese Informationen auch in die Ontologie geschrieben. Aus diesen Informationen lässt sich schließlich eine Kategorisierung für das übergebene Medienobjekt vornehmen, welche ebenfalls in der Ontologie modelliert wird.

Vor der Entwicklung des eigentlichen Algorithmus wurden schon grundlegende Probleme ersichtlich, die bei der Umsetzung beachtet werden müssen. Angefangen bei der eigentlichen Berechnung der Confidence-Werte für die Aussagen der Zugehörigkeiten bis hin zu Problemen, die im Nutzen der Services auftreten können, werden in diesem Kapitel die Probleme und Lösungsansätze vorgestellt, die bei der Entwicklung des Algorithmus umgesetzt wurden. Zur besseren Anschaulichkeit wird nun ein Beispiel eingeführt, an welchem die Probleme und Lösungsansätze gezeigt werden.
Im Referenzszenario wurde angenommen, dass es ein Medienobjekt, in diesem Fall einen Artikel, gibt, welcher sich mit den Karrieren der Brüder »Michael Schumacher« und »Ralf Schumacher« in der »Formel 1« beschäftigt. Das Topic/S-System ist über eine Schnittstelle mit einem Service verbunden, der den in Abbildung 3.6 gezeigten Teilgraphen besitzt. Hierbei sei angenommen, dass die Kanten die Bezeichnung `hasCategory` besitzen.

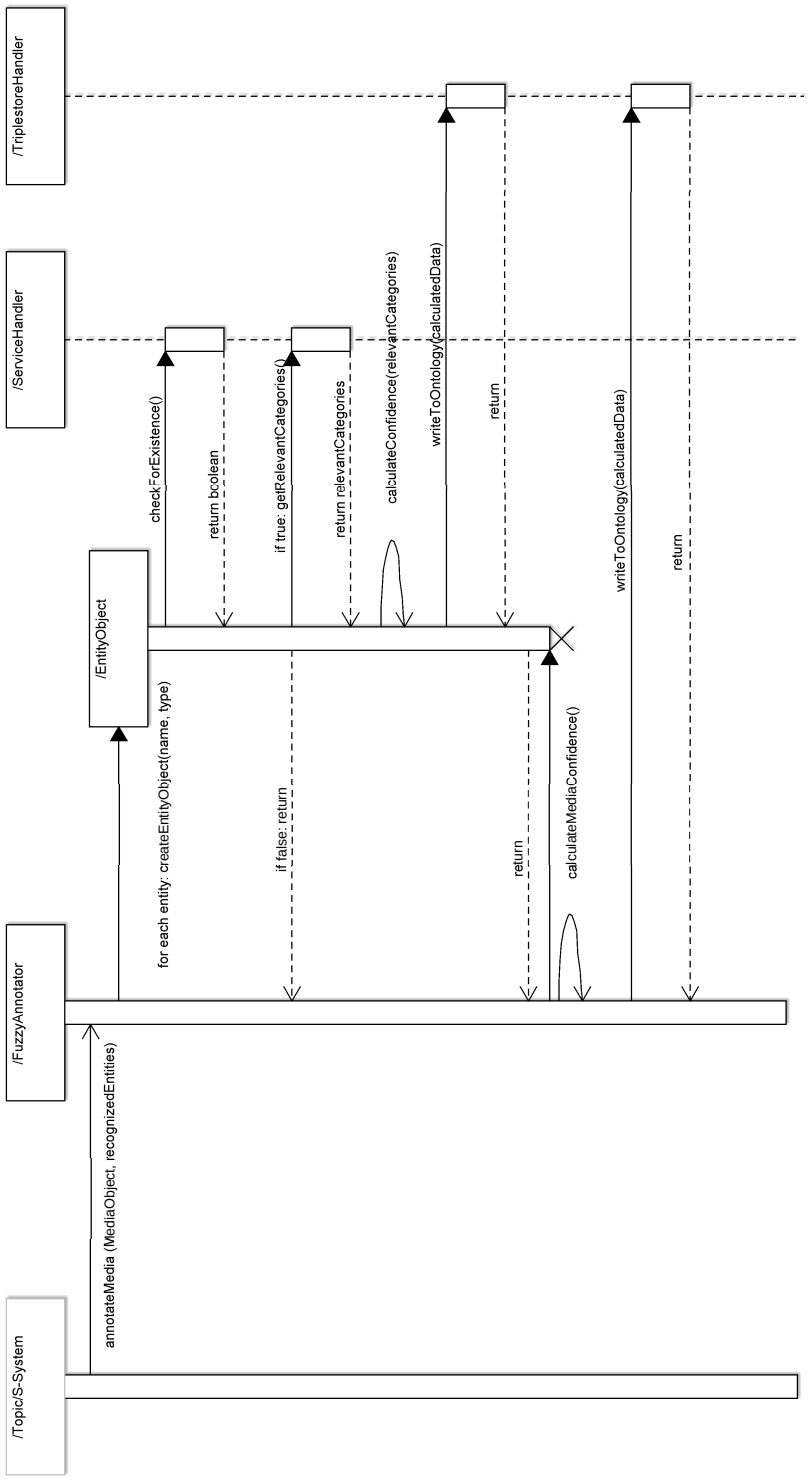


Abbildung 3.5: Allgemeiner Ablauf der Annotation eines eingehenden Medienobjekts

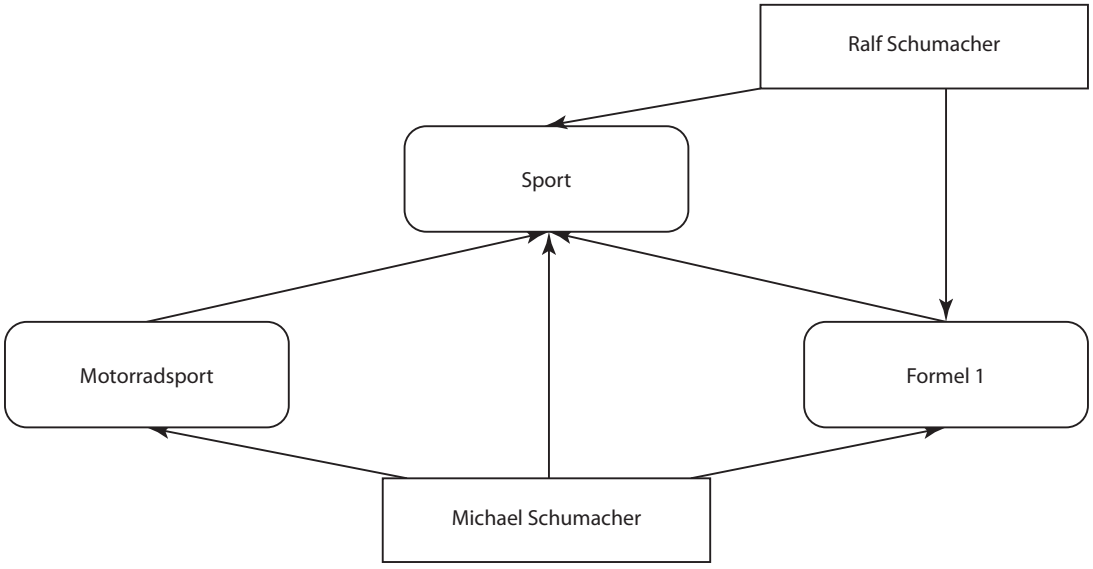


Abbildung 3.6: Beispiel für Kategoriezugehörigkeiten der Entitäten »Michael Schumacher« und »Ralf Schumacher«

Bestimmung der Gewichtungen von unscharfen Aussagen

Allgemein besteht die Frage, welche Art von Aussagen man als unscharf betrachten und damit auch gewichten kann. Zuvor wurde in Kapitel 3.2 festgestellt, dass es in Bezug auf die Medienontologie des Topic/S-Systems verschiedene solcher Aussagen, dort Zugehörigkeiten genannt, gibt. So werden bei der Textanalyse durch eine NER/NED Entitäten mit einer bestimmten Sicherheit, dem Confidence-Wert, erkannt. Dieser beschreibt gleichzeitig die Zugehörigkeit eines Medienobjekt zu seinen Entitäten. Um einem Medienobjekt nun Kategorien zuzuordnen, kann man diese erkannten Entitäten nutzen, indem man einen Services einer Wissensdatenbank nutzt. Hat man, wie in den Einschränkungen in Kapitel 3.1 festgelegt, eine Liste von relevanten Kategorien, so kann man durch Absuchen dieser Kategorien in dem Service eine Kategorisierung der Entitäten treffen. Da durch die Art der Tripel in semantischen Datenbanken gerichtete Graphen entstehen, ist eine Abarbeitung der direkten Kategorien einer Entität bis hin zu Oberkategorien einfach möglich. Zunächst besitzt die Entität direkte Kategorien, denen sie zugeordnet ist. Diese Kategorien wiederum sind weiteren Kategorien, ihren Oberkategorien, zugeordnet. Nun muss für eine Kategorisierung jeder Pfad des Graphen ausgehend von der Entität durchlaufen und das Erreichen einer gesuchten Kategorie protokolliert werden. Wird eine relevante Kategorie gefunden, wird die Suche abgebrochen, der Treffer für diese Kategorie gespeichert und der nächste Pfad durchsucht. Sind alle Pfade abgelaufen, weil jeder Pfad zu einer relevanten Kategorie führt, oder es keine Oberkategorien mehr gibt, wird für die Entität anteilig jede Zugehörigkeit zu jeder relevanten Kategorie mit der nachfolgenden Gleichung berechnet. Hierbei repräsentieren $\mathrm{conf}_{\mathrm{ENT\ CAT}}$ den Confidence-Wert der Aussage über die Zugehörigkeit zwischen einer Entität und

einer Kategorie und $\text{counts}_{\text{ENT CAT}}$ die Zahl der Treffer zu dieser Kategorie:

$$\text{conf}_{\text{ENT CAT}} = \frac{\text{counts}_{\text{ENT CAT}}}{\sum_{\text{CAT}=1}^{n} (\text{counts}_{\text{ENT CAT}})} \tag{3.1}$$

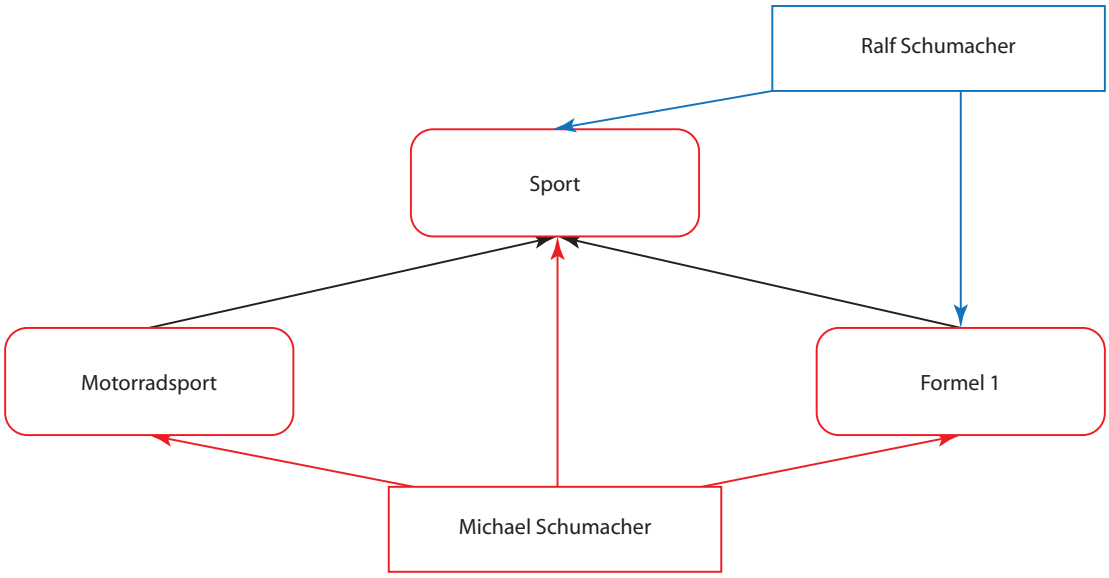


Abbildung 3.7: Suche von direkten relevanten Kategorien für »Michael Schumacher«

In Abbildung 3.7 wird der Vorgang der Kategoriesuche dargestellt. Hierbei wird die Entität `Michael Schumacher` untersucht. In der Liste der relevanten Kategorien, die vom Administrator festgelegt wurde, sind `Motorradsport`, `Sport` und `Formel 1`. Ausgehend von der Entität werden die Pfade nach oben verfolgt. Wird eine der festgelegten Kategorien erreicht, wird das Erreichen gezählt und die Suche für diesen Pfad beendet. Das Ergebnis der Untersuchung ist in Tabelle 3.2 dargestellt.

CAT	$\text{counts}_{\text{Michael CAT}}$	$\text{conf}_{\text{Michael CAT}}$
Motorradsport	1	$\frac{1}{3}$
Sport	1	$\frac{1}{3}$
Formel 1	1	$\frac{1}{3}$

Tabelle 3.2: Berechnung der Zugehörigkeiten $\text{conf}_{\text{Michael CAT}}$ der Entität `Michael Schumacher` (ENT) zu seinen direkten Kategorien (CAT)

Man erkennt, dass die Zugehörigkeit einer Entität zu einer relevanten Kategorie direkt bestimmt werden muss. Dies unterscheidet sich von der Bestimmung der Zugehörigkeiten von Medienobjekten. Deren Einordnung in relevanten Kategorien ergibt sich aus den für sie erkannten Entitäten. Hat ein Medienobjekt beispielsweise die Entitäten `Michael Schumacher` und `Ralf Schumacher`, so errechnen sich

die Confidence-Werte der Zugehörigkeit von dem Medienobjekt und einer relevanten Kategorie aus den für die Entitäten bestimmten Werten mit der nachfolgenden Gleichung:

$$\text{conf}_{\text{MO CAT}} = \frac{\sum_{\text{ENT}=1}^{n} (\text{conf}_{\text{ENT CAT}} * \text{conf}_{\text{MO ENT}})}{n} \tag{3.2}$$

Der Confidence-Wert $\text{conf}_{\text{MO ENT}}$ ist dabei der aus der NER/NED überlieferte Wert.

In Tabelle 3.3 sind die Ergebnisse für die Berechnung der Zugehörigkeiten zwischen dem im Beispiel erwähnten Medienobjekt und seinen relevanten Kategorien dargestellt.

ENT	$\text{conf}_{\text{MO ENT}}$	$\text{conf}_{\text{ENT MS}}$[7]	$\text{conf}_{\text{ENT Sport}}$	$\text{conf}_{\text{ENT Formel1}}$
Michael Schumacher	1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
Ralf Schumacher	1	0	$\frac{1}{2}$	$\frac{1}{2}$
$\text{conf}_{\text{MO CAT}}$	-	$\frac{2}{12}$	$\frac{5}{12}$	$\frac{5}{12}$

Tabelle 3.3: Berechnung der Zugehörigkeiten $\text{conf}_{\text{MO CAT}}$ des Medienobjekts (MO) zu seinen relevanten Kategorien (CAT) mit Hilfe der zugehörigen Entitäten (ENT)

Daraus ergeben sich für die Zugehörigkeit zwischen dem Medienobjekt und den Kategorien `Sport` und `Formel 1` höhere Werte, als für `Motorradsport`. Man kann also sagen, dass das Medienobjekt mit größerer Wahrscheinlichkeit ein Thema behandelt, welches die `Formel 1` betrifft, als den `Motorradsport`.

Festlegung der Confidence-Werte bei fehlenden Informationen

Wie eben beschrieben ist das Bestimmen von Kategorien mithilfe eines Services möglich, welcher an eine Wissensdatenbank angebunden ist. Durch die Einschränkung der eigens verwalteten Liste von relevanten Kategorien ergibt sich jedoch ein weiteres Problem. Es ist nicht garantiert, dass überhaupt relevante Kategorien gefunden werden können. So würde bei der Liste der sportbasierten Kategorien aus dem angeführten Beispiel für einen Musiker, der keinen Zusammenhang zu Sport hat, keine passende Kategorie gefunden werden. Die Lösung ist, eine Standardkategorie festzulegen. Diese könnte beispielsweise »uncategorized« lauten. Eine Entität ist also »uncategorized«, wenn keine relevante Kategorie, beim Absuchen der Kategorie-Pfade im Graphen des Service, ausgehend von dieser Entität, gefunden wurde. Ein Medienobjekt hingegen würde als »uncategorized« bezeichnet werden, wenn keine ihrer Entitäten einer relevanten Kategorie zuzuordnen ist.

[7] Motorsport

Ein ähnliches Problem ergibt sich, wenn die für ein Medienobjekt mitgelieferten Entitäten keinen Confidence-Wert übergeben bekommen haben. Das heißt, wenn Entitäten für ein Medienobjekt durch eine NER/NED bestimmt wurden, aber keine Aussage getroffen wird, wie sicher diese Zugehörigkeit ist. Für diesen Fall muss man einen Standardwert verwenden. Welcher Wert dies ist, ist dem Programmierer überlassen, jedoch wird empfohlen, einen festgelegten Wert, wie »0« oder »1«, zu verwenden. Bei Wahl eines in diesem Intervall befindlichen Wertes könnte der Nutzer verwirrt werden und diese besondere »Nicht-Kategorie« falsch interpretieren.

Zugehörigkeiten zwischen Kategorien

Betrachtet man das bisherige Beispiel, so erkennt man, dass die Berechnung noch ungenau ist. Bisher wurden zu einer Kategorie die Zugehörigkeiten zu seinen direkten Kategorien bestimmt. Wenn man nun aber zusätzlich die Zugehörigkeit der Kategorie `Formel 1` zur Kategorie `Sport` betrachtet, so erweitert sich die Berechnung und man erhält genauere Confidence-Werte. In Abbildung 3.6 ist einfach zu erkennen, dass jede Entität, die von der `Formel 1` abhängt, auch implizit eine Zugehörigkeit zu `Sport` hat. Um die bisherigen Berechnungsmethoden nicht umzuwerfen, werden für jede Kategorie, der die relevante Kategorie unterzuordnen ist (auch über mehrere Ebenen gesehen) diese Zugehörigkeiten berechnet. Konkret bedeutet dies, dass man zunächst für jede relevante Kategorie der Entität die direkte Zugehörigkeit berechnet. Anschließend betrachtet man alle Oberkategorien dieser relevanten Kategorie und berechnet ebenfalls die Zugehörigkeit der Entität zu diesen Oberkategorien. Dadurch ergibt sich ein Zusatz zu der oben angeführten Gleichung, wobei $\text{conf}_{\text{ENT CAT'}}$ den Confidence-Wert zwischen der Entität und einer Oberkategorie der aktuellen Kategorie beschreibt:

$$\text{conf}_{\text{ENT CAT}} = \frac{\text{counts}_{\text{ENT CAT}}}{\sum_{\text{CAT}=1}^{n} (\text{counts}_{\text{ENT CAT}})} \tag{3.3}$$

$$\text{conf}_{\text{ENT CAT'}} = \text{conf}_{\text{ENT CAT'}} + (\text{conf}_{\text{ENT CAT}} * \text{conf}_{\text{ENT CAT'}}) \tag{3.4}$$

Die oben angegebenen Gleichungen sind analog gültig für die Bestimmung der Zugehörigkeit zwischen zwei Kategorien.
In unserem Beispiel mit `Michael Schumacher` würde die Zugehörigkeit der `Formel 1` und des `Motorradsport` zur höheren Kategorie `Sport` den Confidence-Wert 1 tragen. Fügen wir diese hinzu, sehen wir in Tabelle 3.4, dass wir nun mit einem Confidence-Wert von 1, also mit absoluter Sicherheit, sagen können, dass `Michael Schumacher` der Kategorie Sport, jedoch immer noch nur mit 33%-iger Sicherheit dem `Motorradsport` und der `Formel 1` zuzuordnen ist.

CAT	$\text{conf}_{\text{CAT}_{\text{Motorsport}}}$	$\text{conf}_{\text{CAT}_{\text{Sport}}}$	$\text{conf}_{\text{CAT}_{\text{Formel1}}}$
Motorsport	-	1	0
Sport	0	-	0
Formel 1	0	1	-
$\text{conf}_{\text{Michael}_{\text{CAT}_{\text{ges}}}}$	$\frac{1}{3}$	1	$\frac{1}{3}$

Tabelle 3.4: Berechnung der Zugehörigkeiten $\text{conf}_{\text{Michael}_{\text{CAT}_{\text{ges}}}}$ zwischen der Entität `Michael Schumacher` und den relevanten Kategorien unter Berücksichtigung der Zugehörigkeit zwischen den Kategorien selbst

Dieses Problem ist eine Folgeerscheinung des Lösungsansatzes, um Gewichtungen von unscharfen Aussagen über Zugehörigkeiten zu bestimmen. Hier bricht die Pfadsuche ab, sobald eine relevante Kategorie gefunden wurde. Allerdings wurde diese Lösung aus Gründen der Performanz gewählt. Verfolgt man den Pfad weiter, so stoppt der Algorithmus erst, wenn es keine höheren Kategorien gibt. Dies kann zuweilen eine sehr lange Zeit dauern. Durch das Bestimmen der Zugehörigkeiten zwischen den Kategorien selbst wird diese Suche jedoch nur einmalig, anstatt für jede Entität durchgeführt, wodurch insgesamt eine Steigerung der Performanz erreicht wird. Würde die Pfadsuche weitergeführt, bekäme man schlussendlich die in diesem Abschnitt errechneten Werte bekommen.

Zyklen in Services

Ein Problem, welches die Services bei der Suche nach relevanten Kategorien verursachen können, sind Zyklen. Solche Zyklen entstehen, wenn die Kategorien nicht absolut hierarchisch aufgebaut sind. Ein Beispiel ist in Abbildung 3.8 skizziert.

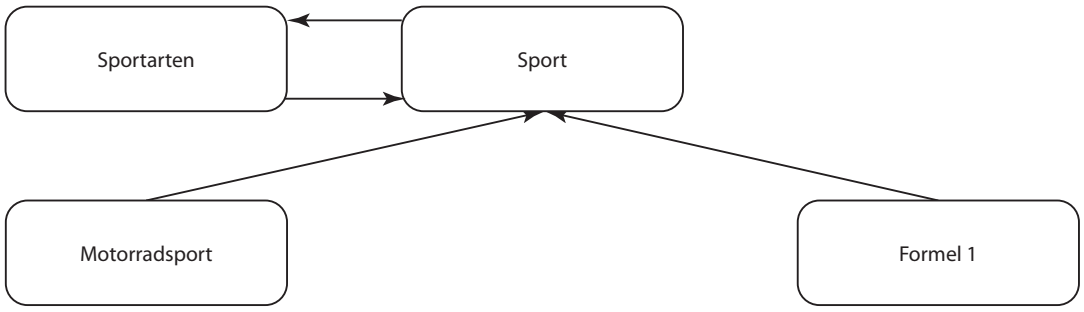


Abbildung 3.8: Zyklus zwischen den Kategorien `Sport` und `Sportarten`

Durchläuft der Algorithmus bei der Pfadverarbeitung solch einen Zyklus, gibt es keine Chance, diesen aufzulösen. Daher muss eine Möglichkeit dafür geschaffen werden, dass das System die Suche im Graphen protokolliert und bei Erkennen eines

Zyklus abbricht. Dies kann durch Speichern der abgelaufenen Pfade realisiert werden. Für jede direkte Kategorie, die zu Beginn des Algorithmus gefunden wird, wird eine Liste gespeichert. Diese Liste repräsentiert den bisher abgelaufenen Pfad. Für jede Liste werden für das letzte (das heißt jüngst hinzugefügte) Element Oberkategorien ermittelt. Für jede Oberkategorie wird nun geprüft, ob sie sich bereits in der Pfadliste befindet. Wenn dies der Fall ist, wurde ein Zyklus erkannt und die Oberkategorie darf nicht mehr betrachtet werden. Ist die Oberkategorie noch nicht im aktuellen Pfad, gibt es keinen Zyklus und es kann weiter gesucht werden. Mitunter können bis zum Erkennen eines Zyklus viele Kategorien durchlaufen und neue Zyklen erreicht worden sein. Dies geht sehr zu Lasten der Performanz. Daher ist es zu empfehlen einen Service mit einem gut strukturierten Graphen zu nutzen.

Mehrdeutigkeit von Entitäten in Services

Es kann vorkommen, dass eine Entität, z. B. wenn sie bei einer NER/NED in einem anderen Service erkannt und für den hier genutzten Service transformiert wurde, mehrdeutig ist oder nur eine Referenz auf eine andere Entität darstellt. Am Beispiel der englischen DBpedia lässt sich das gut mit der Entität »CDU«[8] erklären. Ist die Christlich Demokratische Union in Deutschland weitesgehend bekannt, so führt sie in der englischen DBpedia auf eine so genannte Disambiguation-Page (dt.: Begriffserklärung). Tritt dieser Fall ein, so müsste man annehmen, dass jede dieser möglichen Ressourcen gemeint sein könnte. Diesen Fall kann man auf unterschiedliche Weisen behandeln, die schlussendlich für das betroffene Medienobjekt aber keinen Unterschied bewirken. Es ist möglich, die übergebene Ressource »CDU« zu übernehmen und die Confidence-Werte zu jeder Kategorie unter Berücksichtigung jeder Ressource der Begriffserklärung zu berechnen und mit dieser einen Entität zu annotieren. Andererseits ist es aber auch möglich, dass man die übergebene Entität verwirft, ihren mitgelieferten Confidence-Wert zu gleichem Anteil jeder neuen Entität der Begriffserklärung übergibt und jede dieser Entitäten dem Medienobjekt zuordnet. Es werden hier unterschiedliche Zuordnung von Entitäten zu einem Medienobjekt getroffen. Die Zugehörigkeit vom Medienobjekt zu Kategorien hingegen bleibt hierbei unberührt.
Ein weiterer Fall, den es zu beachten gilt, ist die Behandlung von direkten Weiterleitungen oder Referenzen. So erhält man in der deutschen DBpedia für die »CDU«[9] keine Begriffserklärung, allerdings eine Weiterleitung zur »Christlich Demokratischen Union Deutschlands«. Die Behandlung dieses Falls ist analog zur Behandlung der Begriffserklärung.

[8] http://dbpedia.org/resource/CDU
[9] http://de.dbpedia.org/resource/CDU

Aktualisierung der Datensätze

In semantischen Datenbanken ist es üblich, dass Daten aktualisiert, hinzugefügt oder gelöscht werden. Im Fall des Topic/S-Systems kann eine Aktualisierung verschiedene Inkonsistenzen zufolge haben. Wird zum Beispiel eine Kategorie hinzugefügt, muss überprüft werden, ob neue Zugehörigkeiten zwischen den Kategorien entstehen. Ist dies der Fall, so müssen diese neu berechnet werden. Derselbe Fall tritt ein, wenn eine Kategorie gelöscht wird. Wurden neue Zugehörigkeiten festgestellt und in die Datenbank aufgenommen, so entstehen neue Inkonsistenzen, da nun auch neue Zugehörigkeiten zwischen Entitäten und Kategorien entstanden sein können. Dasselbe gilt für Medienobjekte, deren zugeordnete Entitäten aktualisiert wurden, z. B. durch neu berechnete Zugehörigkeiten zu Kategorien. Im Rahmen dieser Arbeit ist es vorerst nur möglich, eine Aktualisierung für alle Objekte durchzuführen, wenn dieser Fall der Aktualisierung eintritt. Diese kann jedoch optimiert werden, wenn unter Berücksichtigungen von zeitlichen Informationen eine effiziente Reihenfolge bei der Abarbeitung der Objekte vorgenommen wird. Eine Möglichkeit, Zeitinformationen geeignet zu nutzen, wäre, Entitäten und Medienobjekte, welche in naher Vergangenheit hinzugefügt oder benutzt wurden, bei Neuberechnungen zu bevorzugen.
Ein weiterer Sachverhalt, der eine Aktualisierung der Daten verlangt, ist, wenn relevante Daten in einem Service geändert wurden. Wenn beispielsweise Michael Schumacher eine Karriere in der Politik startet, so muss dies auch in der Topic/S-Ontologie berücksichtigt werden. Auch hier werden temporale Informationen benötigt, weswegen dieser Fall im Rahmen dieser Arbeit nicht behandelt werden kann.

Ausgehend von diesen Kernproblemen und deren Lösung lassen sich nun konkrete Algorithmen zur Bestimmung der Zugehörigkeiten von Medienobjekten, Entitäten und Kategorien untereinander mit Hilfe von Confidence-Werten aufstellen, welche in den nächsten Abschnitten genauer erläutert werden.

3.4.2 Kategorisierung von Kategorien

Ein Anwendungsfall für die Nutzung des Systems ist das Einfügen einer Kategorie durch einen Benutzer. Diese administrative Funktion ist wichtig, um eine gute Performanz des Systems zu erhalten. Im letzten Abschnitt wurde außerdem erläutert, dass für eine korrekte Berechnung bei der Kategorisierung Zugehörigkeiten zwischen den relevanten Kategorien selbst automatisiert erkannt und diese auch mit einem Confidence-Wert beschrieben werden müssen. Der Ablauf für die Annotation wird zur besseren Übersicht in drei Teilalgorithmen aufgeteilt: die Annotation der Kategorie selbst, die Bestimmung der Zugehörigkeiten und die Berechnung der Confidence-Werte für diese Zugehörigkeiten.
In Abbildung 3.9 ist ein Aktivitätsdiagramm dargestellt, welches den Ablauf der Annotation skizziert.

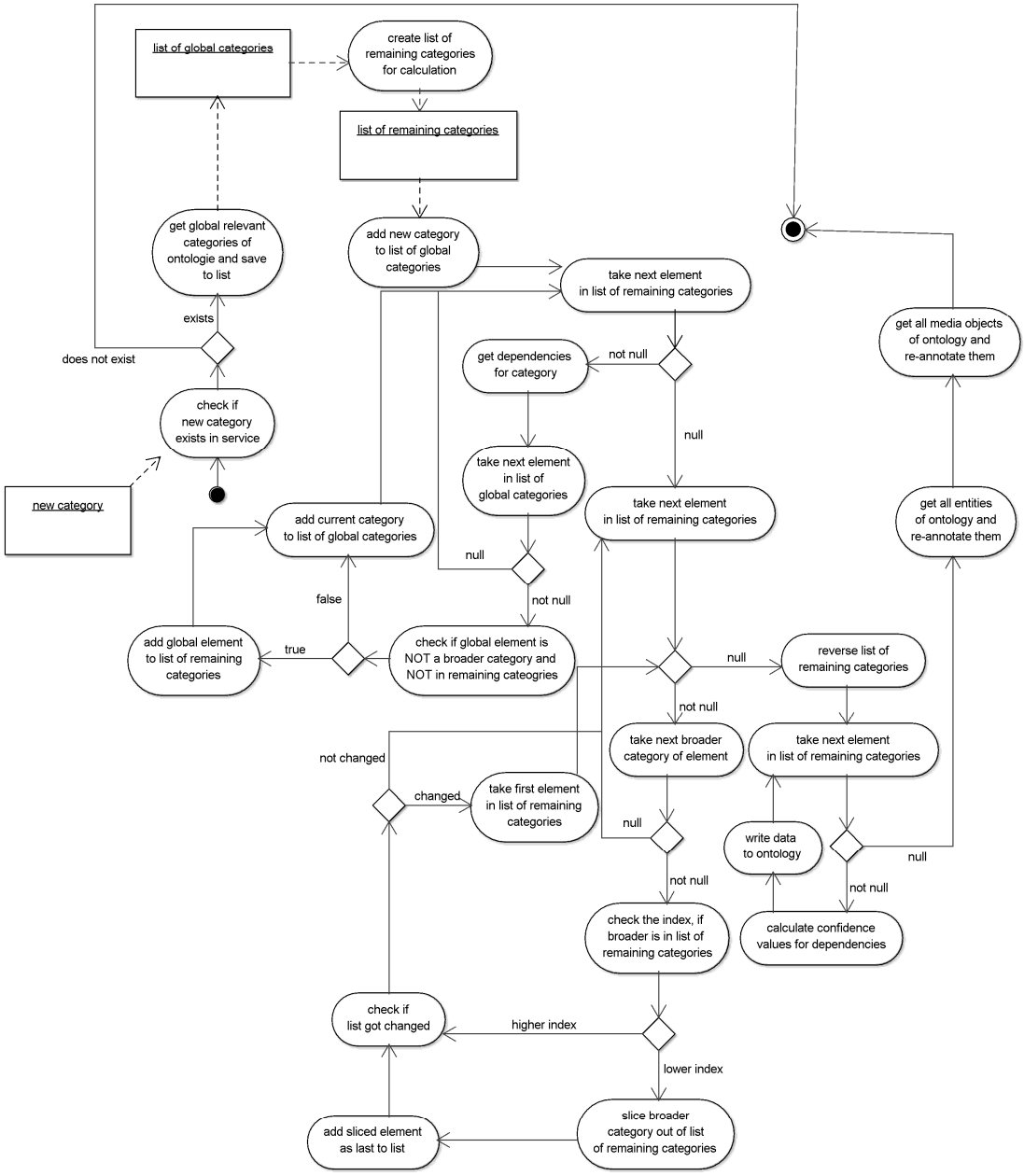


Abbildung 3.9: Aktivitätsdiagramm zur automatisierten Kategorisierung von Kategorien

Der Ablauf dieses Teilalgorithmus ist außerdem als Pseudocode in Listing 3.3 ersichtlich und wird im Rahmen dieses Abschnitts erläutert.

```
1  //begin algorithm 3.4.2-a
2  function annotateCategory(Category newCategory){
3
4  if(category.existsInService()){
5         List globalCategories = getCategoriesOfOntology();
6         List remainingCategories = new List();
7         remainingCategories.add(newCategory);
8
9         while(remainingCategories.hasNext()){
10                Category currentElement = remainingCategories.next();
11                currentElement.getDependencies(); // go listing 3.4
12
13                for each(globalCategory : globalCategories){
14
15                       if(!currentElement.dependencyList.
16                                       contains(globalCategory) && !
                                            remainingCategories.contains(
                                            globalCategory)){
17                                remainingCategories.add(globalCategory)
18                       }
19                }
20        globalCategories.add(currentElement);
21        }
22        Category currentElement = remainingCategories.getFirst();
23        while(remainingCategories.hasNext()){
24
25                for each (category : remainingCategories){
26
27                       if(category.index() < currentElement.index() &&
                             currentElement.dependencyList.contains(category)){
28                                remainingCategories.addLast(remainingCategories.
                                     slice(category));
29                       }
30                }
31
32                if(remainingCategories.gotChanged()){
33                       currentElement = remainingCategories.getFirst();
34                }else{
35                       currentElement = remainingCategories.next();}
36        }
37        remainingCategories.reverseList();
38        for(category : remainingCategories){
39                category.calculateConfidence(); // go listing 3.5
40                category.writeToOntology();
41        }
42 updateEntitiesInOntology();
43 updateMediaInOntology();
44 }
45 return;}
```

Listing 3.3: Pseudocode: Annotation der Zugehörigkeiten zwischen Kategorien

Der `annotateCategory` Methode, die im Pseudocode den Teilalgorithmus aufruft, wird das zu beschreibende `Category`-Objekt übergeben. Zunächst wird in Zeile 4 geprüft, ob diese Kategorie mit ihrer IRI im genutzten Service existiert. Ist dies nicht der Fall, so wird die gesamte Annotation abgebrochen bzw. übersprungen und der

Algorithmus endet. Existiert die Kategorie, so werden zunächst alle derzeit in der Ontologie vorhandenen Kategorien ausgelesen und in die Liste `globalCategories` gespeichert. Dies ist nötig, falls das System neu gestartet wurde und die Kategorien noch nicht lokal gespeichert worden sind. Anschließend wird eine Liste von noch zu bearbeitenden Kategorien erstellt, in die das neue Element gespeichert wird. In dieser Liste namens `remainingCategories` werden alle Kategorien aufgenommen, deren Zugehörigkeiten neu berechnet werden müssen. In den Zeilen 9 bis 19 wird nun für jede Kategorie dieser `remainingCategories`-Liste ermittelt, welche Zugehörigkeiten sie zu anderen relevanten Kategorien hat. Die Bestimmung der Zugehörigkeit wird in Zeile 11 aufgerufen und später in Listing 3.4 behandelt. Anhand der erkannten Zugehörigkeiten wird nun entschieden, welche Kategorien der `remainingCategories`-Liste hinzugefügt werden muss. Wenn sie sich in dieser Liste befinden, dann müssen wiederum ihre Zugehörigkeiten neu bestimmt werden. Eine Kategorie wird genau dann in die Liste aufgenommen, wenn ihr die aktuelle Kategorie nicht untergeordnet ist und sie nicht schon in der Liste noch zu bearbeitender Kategorien aufgenommen wurde. Dies spart Rechenaufwand. Lediglich bei den Kategorien, denen die neu eingefügte Kategorie nicht zuzuordnen ist, besteht die Möglichkeit, dass diese wiederum eine Zugehörigkeit zu der neuen Kategorie zeigen. Sind alle Zugehörigkeiten einer Kategorie bestimmt, so wird sie der Liste namens `globalCategories` hinzugefügt.
Der nächste Schritt dieses Teilalgorithmus ist die Sortierung der Liste. Bisher wurden für jede Kategorie die Zugehörigkeiten neu bestimmt. Um die Confidence-Werte dieser berechnen zu können, muss sichergestellt werden, dass zuerst die Kategorien berechnet werden müssen, die anderen Kategorien der `remainingCategories`-Liste nicht untergeordnet sind (Zeile 22 bis 37). Die Begründung dafür folgt bei der Erklärung des Teilalgorithmus zur Berechnung der Zugehörigkeiten in Listing 3.5. Es wird für jede Kategorie der Liste geprüft, ob sie einem anderen Element der Liste untergeordnet ist und diese höhere Kategorie einen kleineren Index in der Liste besitzt. Ist dies der Fall, so wird die höhere Kategorie ausgeschnitten und als letztes Element in der Liste angefügt. Wurde die Position einer Kategorie in der Liste geändert, so wird beim ersten Element neu gestartet, andernfalls wird das nächste Element untersucht. Dadurch werden die Kategorien mit den wenigsten Oberkategorien in der Liste nach hinten verdrängt und die Elemente der Liste nach der Stärke der Zugehörigkeiten zu anderen Kategorien sortiert. Ist der Sortieralgorithmus am Ende der Liste angelangt, wird die Reihenfolge der Elemente in der Liste umgekehrt. Dadurch werden zunächst die Confidence-Werte der Zugehörigkeiten für die »unabhängigeren« Kategorien und zuletzt von den »stark abhängigen« Kategorien berechnet (Zeile 38 bis 41). Die Werte der neu berechneten Kategorien werden anschließend in der Ontologie gespeichert.
Nachdem die Zugehörigkeiten zwischen den Kategorien aktualisiert wurden, ist es möglich, dass Entitäten und dadurch auch Medienobjekte von diesen Aktualisierungen betroffen sind. Daher müssen diese Objekte ebenfalls aktualisiert und gegebenenfalls deren Zugehörigkeiten zu Kategorien neu berechnet werden (Zeile 42 und 43).
Während der Annotation wird ein weiterer Teilalgorithmus aufgerufen, der die Bestimmung der Zugehörigkeiten einer Kategorie untersucht und speichert. Dieser ist in Abbildung 3.10 und als Pseudocode in Listing 3.4 dargestellt.

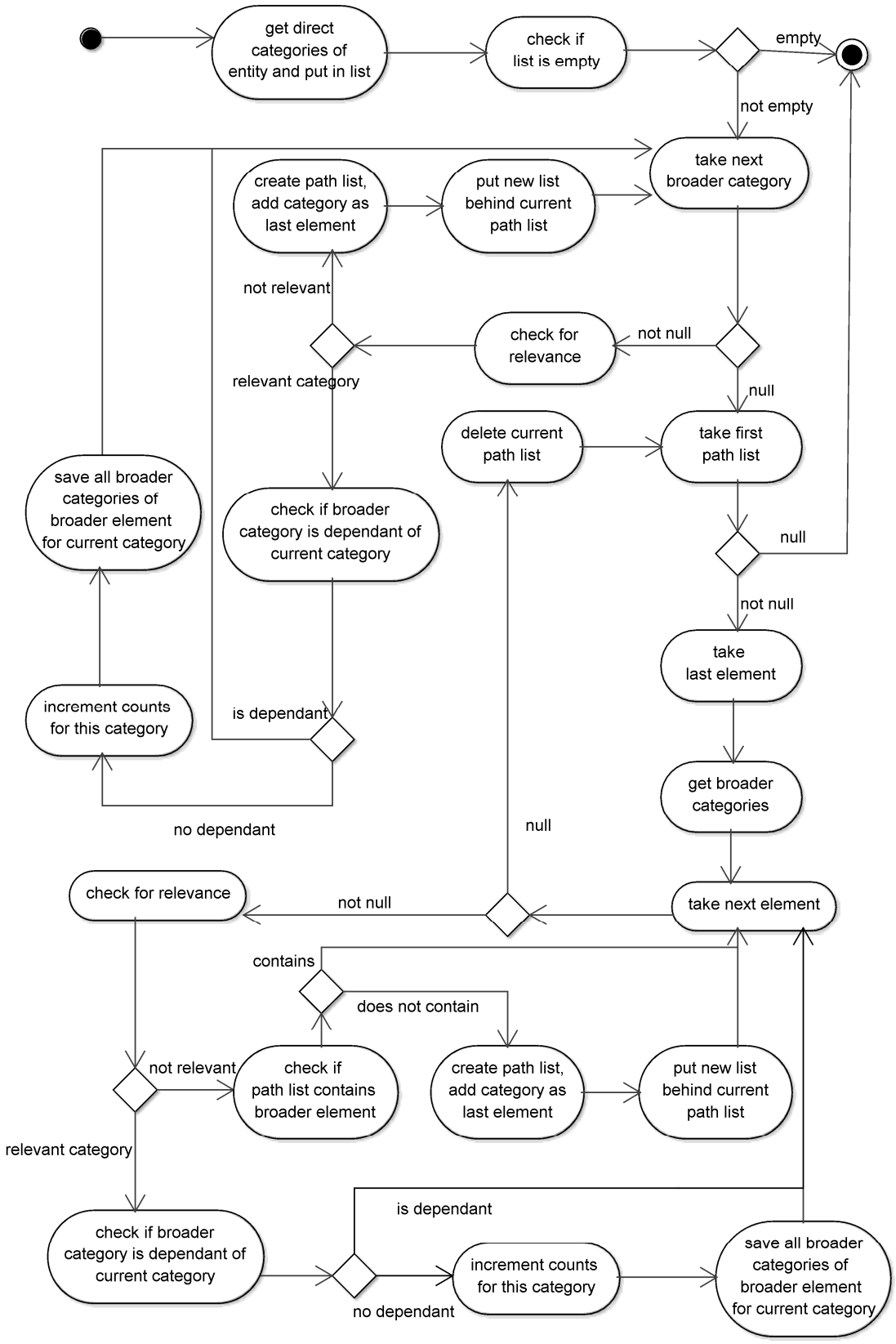


Abbildung 3.10: Aktivitätsdiagramm zur Bestimmung von Zugehörigkeiten zwischen Kategorien

```
1  //begin algorithm 3.4.2-b
2  function getDependencies(){
3
4  if(this.hasBroaderCategories()){
5         List broaderCat = this.getBroaderCat();
6
7         for each (category : broaderCat){
8
9                 if(category.isRelevantCategory() && !category.dependencyList.
                      contains(this)){
10                    setCounts(category, getCounts(category)+1);
11
12                    for each (broader : category.dependencyList){
13                           setCounts(broader, getCounts(broader)+0);
14                    }
15                }else{
16                createPathList(category);
17                }
18        }
19
20        while(this.hasPathList()){
21                List currentPathList = firstPathList();
22                Category currentEl = currentPathList.getLast();
23                List broaderCat = currentEl.getBroaderCat();
24
25                for each(category : broaderCat){
26
27                        if(category.isRelevantCategory() && !category.
                              dependencyList.contains(this)){
28                            setCounts(category, getCounts(category)+1);
29
30                            for each (broader : category.dependencyList){
31                                   setCounts(broader, getCounts(broader)+0);
32                            }
33                        }else{
34                            if(!currentPathList.contains(category)){
35                                   copyPathAndAdd(currentPathList, category);
36                            }
37                        }
38                }
39                deletePathList(currentPathList);
40        }
41
42 }
43 return; } // returns to listing 3.3 line 11
```

Listing 3.4: Pseudocode: Bestimmung von Zugehörigkeiten zwischen Kategorien

Wird die Methode `getDependencies` für ein `Category`-Objekt aufgerufen, so werden zwei in der Ausführung ähnliche, aber im Ergebnis unterschiedliche Codeabschnitte ausgeführt. Im ersten Abschnitt, den Zeilen 4 bis 18, wird zunächst überprüft, ob die aktuell zu betrachtende Kategorie relevante als direkte Oberkategorien besitzt. Hat die Kategorie hierbei keine direkt höhere Kategorie, können keine Zugehörigkeiten bestimmt werden und der Teilalgorithmus terminiert (Zeile 4 und 43). Anderenfalls wird für jede direkt höhere Kategorie untersucht, ob sie relevant ist. Ist dies der Fall,

so wird der Treffer,vorausgesetzt die aktuelle Kategorie selbst keine Oberkategorie der als relevant erkannten, protokolliert (Zeile 9 und 10). Wird die Zahl der Treffer zwischen aktueller und relevanter Kategorie um 1 erhöht, so werden alle Kategorien, denen die relevante Kategorie untergeordnet ist für die aktuelle übernommen. Dies muss geschehen, damit auch indirekte Zugehörigkeiten im zuvor vorgestellten Teilalgorithmus beachtet werden. Ansonsten könnte es sein, dass die indirekten Oberkategorien in der `remainingCategories`-Liste aufgenommen werden, obwohl sie dem aktuellen Element nicht untergeordnet sind. Ist eine direkt höhere Kategorie nicht relevant, so wird eine neue `PathList` angelegt. Diese Listen dienen zur Protokollierung der abgesuchten Pfade, wodurch mögliche Zyklen erkannt werden können.
Nachdem durch diesen Teilschritt die Ausgangspfade erstellt wurden, werden diese in den Zeilen 20 bis 40 abgearbeitet. Dazu zu jedem Pfad untersucht, ob sein jüngstes bzw. letztes Element relevante Oberkategorien besitzt. Wird eine relevante Kategorie gefunden, wird der Treffer protokolliert und weitere Oberkategorien übernommen. Ist es keine relevante Kategorie, wird geprüft, ob sich das Element bereits in der `PathList` befindet. Wenn ja, wurde ein Zyklus erkannt und der Pfad einfach gelöscht. Gibt es keinen Zyklus, wird die `PathList` kopiert und die höhere Kategorie hinzugefügt. Sind alle höheren Kategorien abgearbeitet, wird der aktuelle Pfad gelöscht. Gibt es keine Pfade mehr abzuarbeiten, wird der Teilalgorithmus beendet. Als Ergebnis hat man nun eine Menge von Kategorien erhalten, zu denen eine Zugehörigkeit besteht. Außerdem wurde jeder Treffer protokolliert, wodurch direkte Zugehörigkeit berechnet werden können. Zusätzlich wurden Kategorien gespeichert, von denen die zu betrachtende nur eine indirekte Unterkategorie ist. Dieses Bestimmen der Zugehörigkeiten bildet die Grundlage für die Berechnung eben dieser. Der Teilalgorithmus zur Berechnung der Zugehörigkeiten zwischen Kategorien ist in Abbildung 3.11 und Listing 3.5 dargestellt.

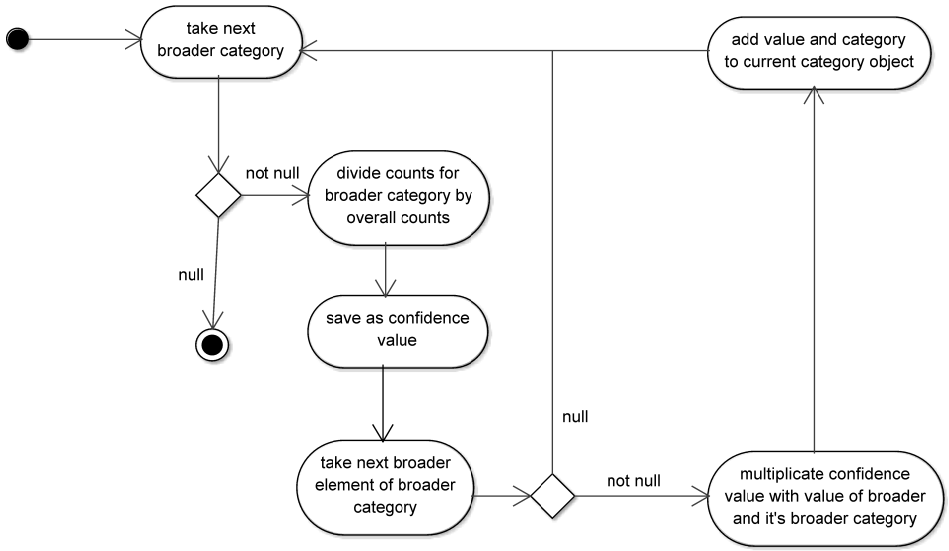


Abbildung 3.11: Aktivitätsdiagramm zur Berechnung von Zugehörigkeiten zwischen Kategorien

```
1  //begin algorithm 3.4.2-c
2  function calculateConfidence(){
3
4  while(dependencyList.hasNext()){
5         broaderCategory = dependencyList.next();
6         float confidenceValue = getCounts(broaderCategory)/getSumOfCounts();
7         addConfidence(broaderCategory, confidenceValue);
8
9         for each (category : broaderCategory.dependencyList){
10                broaderConfVal = broaderCategory.getConfidence(category);
11                addConfidence(broaderCategory, confidenceValue*broaderConfVal);
12        }
13 }
14 return; } // returns to listing 3.3 line 39
```

Listing 3.5: Pseudocode: Berechnung des Confidence-Werts der Zugehörigkeit zwischen Kategorien

Die Bestimmung der Confidence-Werte erfolgt hier nach der in Abschnitt 3.4.1 aufgestellten Gleichung für die Berechnung der Zugehörigkeit zwischen zwei Kategorien. Dazu wird in Zeile 6 der Confidence-Wert für die Zugehörigkeit zwischen aktueller Kategorie und jeder direkt höheren Kategorie bestimmt. Anschließend erhält man mit Hilfe des eben errechneten Wertes die Confidence-Werte der Zugehörigkeiten zu Oberkategorien der direkt höheren Kategorie (Zeile 9 bis 12). Aus diesem Grund wurde im ersten Teilalgorithmus (Abbildung 3.9) die `remainingCategories`-Liste nach dem Grad der Zugehörigkeit sortiert. Nur so können die Werte der Zugehörigkeiten von den »weniger abhängigen« Kategorien zu den »stark abhängigen« Kategorien übermittelt werden.
Mit Hilfe dieses Algorithmus ist es möglich Zugehörigkeiten zwischen Kategorien zu berechnen und damit eine Kategorisierung von Kategorien vorzunehmen. Soll eine Kategorie wieder entfernt werden, so sind nur Neuberechnungen nötig, wenn diese keine Oberkategorien in der Medienontologie besitzt. Hat eine zu löschende Kategorie weitere Nachfolger, so reicht es aus, die Daten mithilfe einer SPARQL-Update Anfrage zu entfernen. Ist eine zu löschende Kategorie nun aber eine oberste Kategorie, so müssen alle Zugehörigkeiten der Unterkategorien und damit auch ihrer zugehörigen Entitäten und Medienobjekte neu berechnet werden. In Listing 3.6 ist der Pseudocode dargestellt. Zunächst wird geprüft, ob das zu löschende Element Oberkategorien besitzt. Wenn nicht, so werden alle Kategorien bestimmt, die ihm untergeordnet sind und in die `remainingCategories`-Liste aufgenommen. Anschließend wird der Sortieralgorithmus, wie beim Hinzufügen einer Kategorie, ausgeführt und die Zugehörigkeiten, sowohl der Kategorien als auch der Entitäten und Medienobjekte, neu berechnet. Anschließend werden alle Tripel bezüglich der zu entfernenden Kategorie aus der Ontologie entfernt.
Mithilfe der errechneten Werte für die Zugehörigkeiten zwischen Kategorien ist es nun möglich, auch Entitäten zu kategorisieren, wofür der Algorithmus im nächsten Teilabschnitt vorgestellt wird.

```
1  function deleteCategory(Category oldCat){
2
3  if(oldCat.inOntology()){
4         if(oldCat.dependencyList.isEmpty()){
5
```

```
6                for each (category : globalCategories){
7                        if (category.dependencyList.contains(oldCat)){
8                                remainingCategories.add(category);
9                        }
10               }
11               globalCategories.delete(oldCat);
12               remainingCategories.sortByDependency();
13               updateEntities();
14               updateMedia();
15       }
16       deleteDataFromOntology(oldCat);
17
18 }
```

Listing 3.6: Pseudocode: Das Löschen einer Kategorie

3.4.3 Kategorisierung von Entitäten

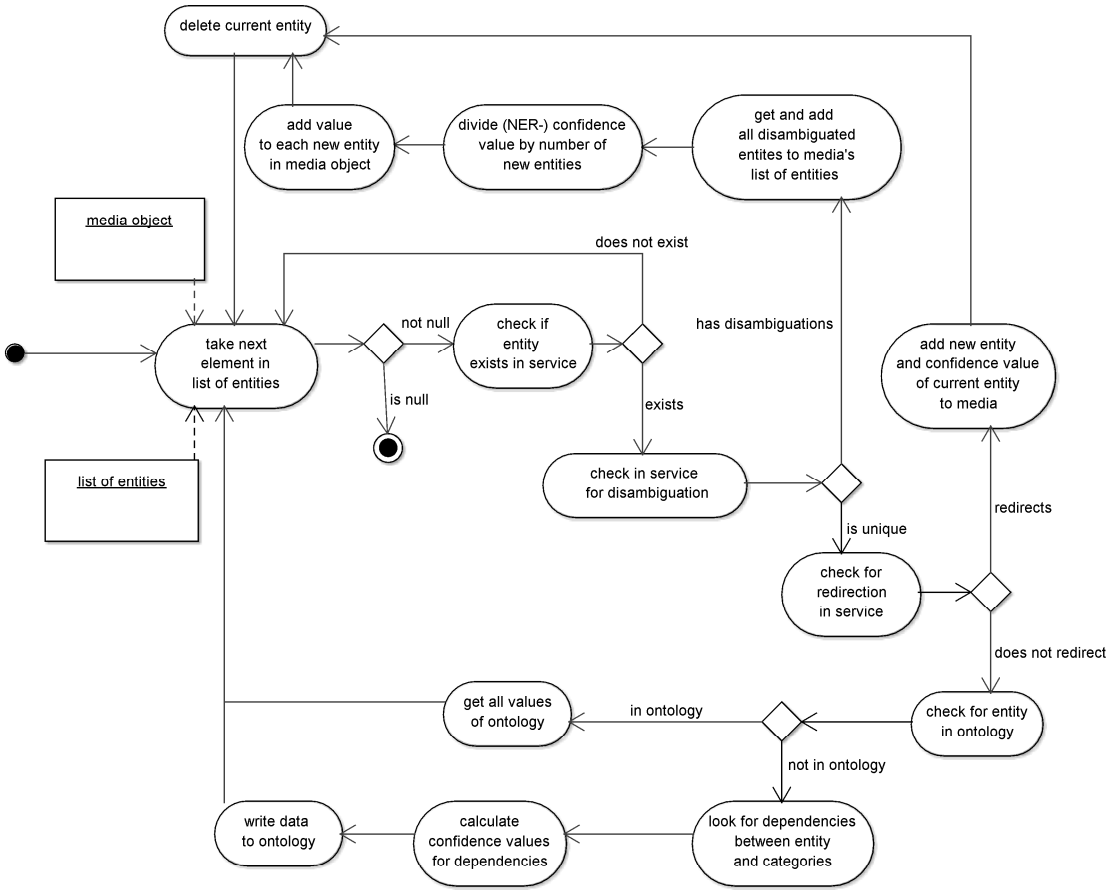


Abbildung 3.12: Aktivitätsdiagramm zur automatisierten Kategorisierung von Entitäten

In diesem Teilabschnitt wird ein Algorithmus zur automatisierten Kategorisierung einer Entität vorgestellt. Dieser Algorithmus wird abgearbeitet, wenn ein Medienob-

jekt in das System eingeht und durch eine NER/NED erkannte Entitäten mitliefert. Wie auch bei dem zuvor präsentierten Algorithmus zur Kategorisierung von Kategorien gibt es hier drei Teilalgorithmen: die Annotation, Bestimmung und Berechnung von Zugehörigkeiten einer Entität zu einer oder mehreren Kategorien.
Kommt von einem Medienobjekt die Anweisung an das System, seine Entitäten zu annotieren, so wird der Teilalgorithmus in Abbildung 3.14 bzw. in Listing 3.7 angestoßen.

```
1  //begin algorithm 3.4.3-a
2  function annotateEntities(Media media, List entities){
3
4  for each (entity : entities){
5
6         if(entity.existsInService()){
7                 if(entity.isDisambiguation()){
8                 for each (resource : entity.getDisambiguation()){
9                         float confidenceValue = media.getNerConfidence(entity)/
                              entity.getDisambiguation().size();
10                        media.addEntity(resource,confidenceValue);
11                }
12                media.dropEntity(entity);
13                }else{
14                        if(entity.redirects()){
15                                media.addEntity(entity.getRedirect(), media.
                                      getNerConfidence(entity));
16                        }else{
17                                if(entity.isInOntology){
18                                        entity.getDataFromOntology();
19                                }else{
20                                        entity.getDependencies(); // go listing 3.8
21                                        entity.calculateConfidence(); // go listing
                                              3.9
22                                        entity.writeToOntology();
23                        }}}
24                }else{
25        media.dropEntity(entity);
26 }}
27 return; } // returns to listing 3.10 line 12
```

Listing 3.7: Pseudocode: Annotation der Zugehörigkeiten zwischen einer Entität und Kategorien

Den Einstieg bietet hier die Methode `annotateEntities`, der das Medienobjekt (`media`) selbst und deren Liste von Entitäten (`entities`) übergeben wurde. Für jedes Element in dieser Liste von Entitäten wird überprüft, ob es in einem Service existiert (Zeile 6). Existiert die Entität dort nicht, so wird sie aus der Liste bzw. dem Medienobjekt gelöscht (Zeile 25). Anderenfalls wird der Teilalgorithmus weiter durchlaufen. Als nächstes wird in Zeile 7 geprüft, ob die Entität zu einer Begriffserklärung führt. Wenn ja, so wird jede Entität, die vorgeschlagen wird, dem Medienobjekt hinzugefügt. Jeder neuen Entität wird der Confidence-Wert, geteilt durch die Zahl der neuen Entitäten, hinzugefügt. Die alte Entität, die auf die Begriffserklärung verwiesen hat, wird für das Medienobjekt gelöscht. Führt die Entität zu keiner Begriffserklärung wird weiterhin geprüft, ob die Entität eine Redirection (dt.: Weiterleitung) ist (Zeile 14). Ein Beispiel für solche eine Weiterleitung wäre die Entität eines Künstlers, welcher sowohl unter seinem Künstlernamen als auch seinem

realen Namen einen Eintrag in der Wissensdatenbank hat. Im Falle einer Weiterleitung wird dem Medienobjekt die neue Entität hinzugefügt, der Confidence-Wert der alten Entität übernommen und die alte Entität, die zur Weiterleitung geführt hat, gelöscht. Bevor nun die Zugehörigkeiten der Entität bestimmt und berechnet werden, wird noch in Zeile 17 geprüft, ob die Entität bereits in der Ontologie vorhanden ist. Wenn ja, so werden alle Daten aus der Ontologie geholt und eine Neuberechnung umgangen. Befindet sich die Entität noch nicht in der Ontologie, so werden ihre Zugehörigkeiten zu Kategorien wie in Listing 3.8 bestimmt und wie in Listing 3.9 berechnet. Sind die Berechnungen für die Entität abgeschlossen, werden alle ermittelten Daten in die Ontologie geschrieben. Ist die gesamte `entities`-Liste abgearbeitet, ist die Annotation aller für das Medienobjekt relevanten Entitäten abgeschlossen. Die Kategorisierung des Medienobjekts baut dabei auf den Zugehörigkeiten der Entitäten zu relevanten Kategorien auf. Die Bestimmung dieser Zugehörigkeiten ist als Aktivitätsdiagramm in Abbildung 3.13 und als Pseudocode in Listing 3.8 zu sehen.

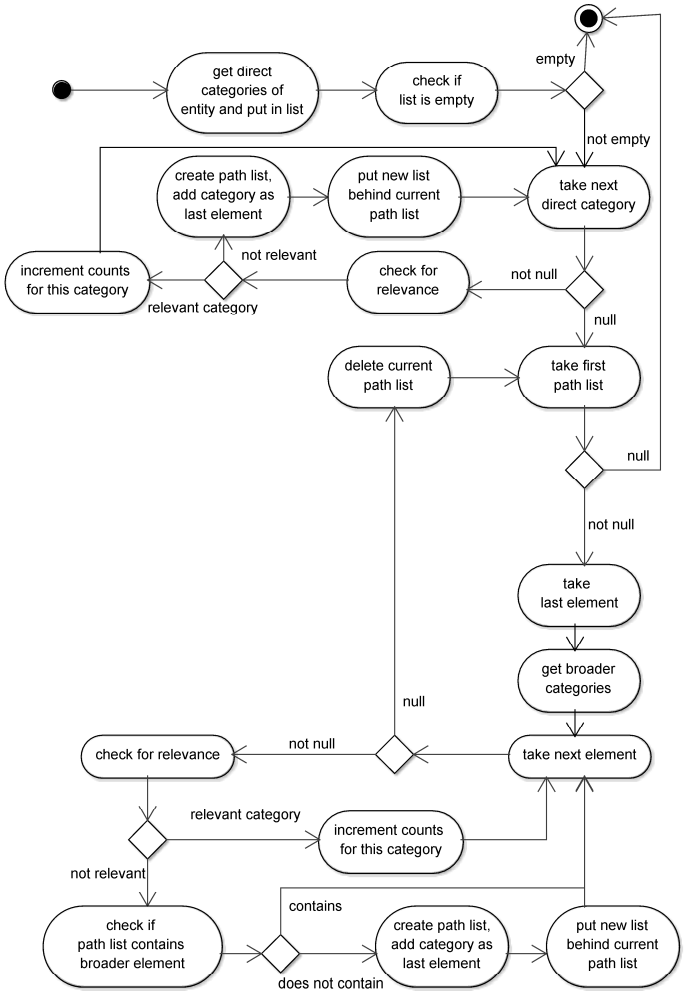


Abbildung 3.13: Aktivitätsdiagramm zur Bestimmung von Kategoriezugehörigkeiten für eine Entität

```
1  //begin algorithm 3.4.3-b
2  function getDependencies(){
3  getDirectCategories();
4  if(!directCategories.isEmpty()){
5         for each(category : directCategories){
6                if(category.isRelevantCategory()){
7                       setCounts(category, getCounts()+1);
8                }else{
9                       createPathList(category);
10               }
11        }
12
13        while(this.hasPathList()){
14               List currentPathList = firstPathList();
15               Category currentEl = currentPathList.getLast();
16               List broaderCat = currentEl.getBroaderCat();
17
18               for each(category : broaderCat){
19
20                      if(category.isRelevantCategory()){
21                             setCounts(category, getCounts(category)+1);
22
23                      }else{
24                             if(!currentPathList.contains(category)){
25                             copyPathAndAdd(currentPathList, category);
26                      }
27                      }
28               }
29               deletePathList(currentPathList);
30        }
31 }
32 return; } // returns to listing 3.7 line 20
```

Listing 3.8: Pseudocode: Bestimmung von Kategoriezugehörigkeiten für eine Entität

Die Bestimmung der Zugehörigkeiten einer Entität zu Kategorien ist dabei der Bestimmung von Zugehörigkeiten zwischen Kategorien sehr ähnlich. Jedoch gibt es auch hier einige Unterschiede. Eine Gemeinsamkeit ist, dass auch für Entitäten zunächst die direkten Kategorien über einen Service ermittelt werden, wie in Zeile 3 zu sehen ist. Gibt es keine direkten Kategorien, wird keine Kategorisierung vorgenommen und der Teilalgorithmus terminiert. Wurden hingegen direkte Kategorien für die Entität ermittelt, so wird zunächst für jede geprüft, ob sie eine relevante Kategorie ist. Trifft dies zu, wird die Zahl der Treffer der relevanten Kategorie von der betrachteten Entität aus erhöht (Zeile 7). Ist die direkte Kategorie keine relevante, wird eine `PathList` angelegt, also eine Liste, die den Fortschritt eines abzuarbeitenden Pfads des Graphen im Service repräsentiert. Gibt es keine direkten Kategorien mehr, so werden die erstellten Pfade abgearbeitet. Dazu wird, wie bei der Kategorisierung von Kategorien, die nächste `PathList` gewählt und deren letztes Element auf höhere Kategorien untersucht. Ist eine höhere Kategorie relevant, so wird ein Treffer für diese protokolliert, die aktuelle `PathList` gelöscht und die nächste verwendet. Ist die höhere Kategorie nicht relevant, so wird sie, vorausgesetzt der Pfad ist kein Zyklus (Zeile 24), einer Kopie der aktuellen `PathList` als letztes Element hinzugefügt.

Nachdem alle relevanten Kategorien für die zu betrachtende Entität bestimmt wurden, wird in Zeile 21 des Listing 3.7 die Berechnung der Confidence-Werte für diese Zugehörigkeiten ausgelöst. Der Ablauf des Teilalgorithmus zur Berechnung der Zugehörigkeiten ist in Abbildung 3.14 sowie Listing 3.9 als Pseudocode dargestellt.

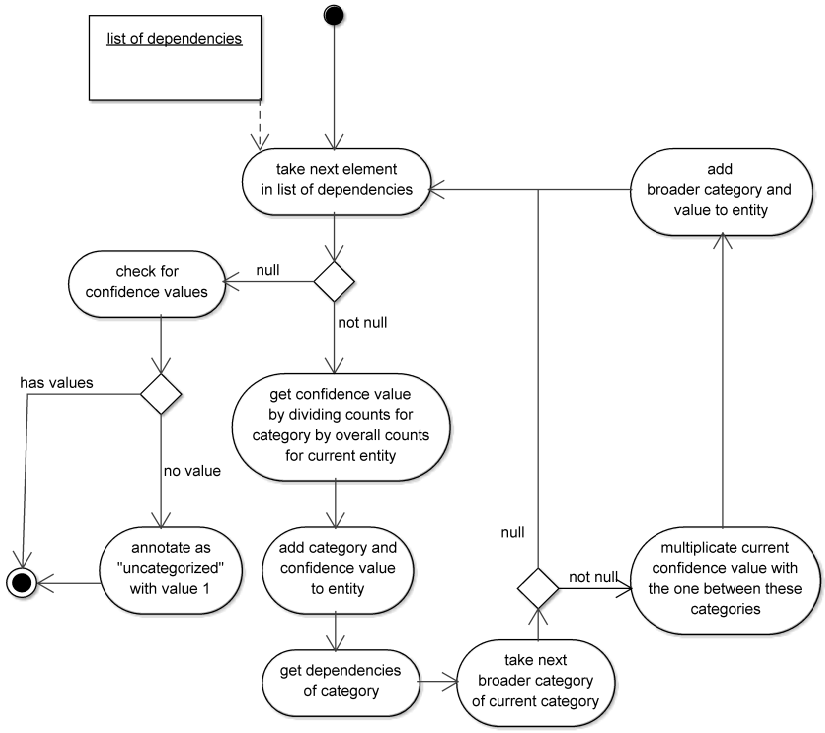


Abbildung 3.14: Aktivitätsdiagramm zur Berechnung von Kategoriezugehörigkeiten einer Entität

```
1  //begin algorithm 3.4.3-c
2  function calculateConfidence(){
3
4  while(dependencyList.hasNext()){
5         broaderCategory = dependencyList.next();
6         float confidenceValue = getCounts(broaderCategory)/getSumOfCounts();
7         addConfidence(broaderCategory, confidenceValue);
8
9         for each (category : broaderCategory.dependencyList){
10                broaderConfVal = broaderCategory.getConfidence(category);
11                addConfidence(broaderCategory, confidenceValue*broaderConfVal);
12        }
13 }
14 if(!this.hasConfidenceValues()) addConfidence("uncategorized", 1.0);
15
16 return; } // returns to listing 3.7 line 21
```

Listing 3.9: Pseudocode: Berechnung des Confidence-Werts zwischen einer Entität und Kategorien

Der Ablauf ist auch hier wieder ähnlich zu der Berechnung bei Kategorien. In den Zeilen 4 bis 7 werden für jede relevante Kategorie die direkten Zugehörigkeiten berechnet. Wurde eine Kategorie direkt von der Entität aus erreicht, errechnet sich hier ein Wert größer 0 und kleiner gleich 1. Anderenfalls, wenn kein Treffer für die relevante Kategorie protokolliert wurde, wird der Confidence-Wert mit 0 beschrieben. Anschließend werden alle indirekten Zugehörigkeiten berechnet (Zeile 9 bis 12). Zusätzlich wird vor der Beendigung des Teilalgorithmus geprüft, ob nun Werte berechnet wurden, d. h. ob Zugehörigkeiten wirklich bestehen. Wenn es keine Zugehörigkeiten gibt, wird die Entität als »uncategorized« markiert (Zeile 14). Dies ist notwendig, damit Entitäten, die nicht kategorisiert werden können, bei SPARQL-Anfragen an das System nicht ausgefiltert werden. Es gibt dem Benutzer bzw. Administrator außerdem die Möglichkeit, speziell nach solchen Objekten zu suchen, welche als »uncategorized« markiert sind, und geeignete Maßnahmen, wie das Einfügen oder manuelle Zuordnen einer Kategorie, durchzuführen.

3.4.4 Kategorisierung von Medienobjekten

Mithilfe der Ergebnisse der bisher vorgestellten Algorithmen ist es nun möglich auch Medienobjekte, wie Artikel, Videos, Bilder etc., zu kategorisieren. Anders als bisher wird dafür kein Service verwendet, stattdessen werden die in der Ontologie vorhandenen Daten genutzt. Im Aktivitätsdiagramm in Abbildung 3.15 wird gezeigt, wie die Annotation generell vorgenommen wird. In Listing 3.10 ist der Ablauf als Pseudocode formuliert.

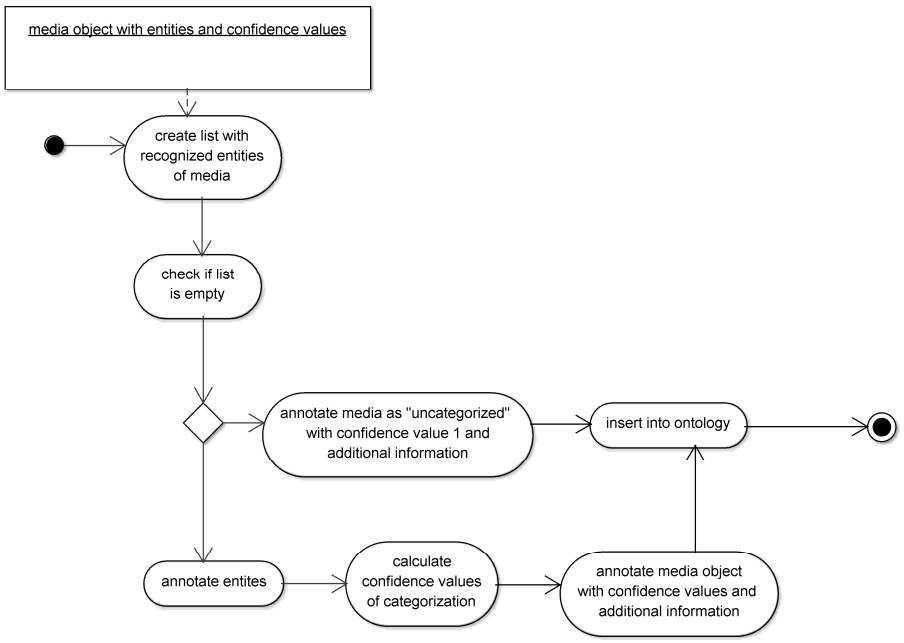


Abbildung 3.15: Aktivitätsdiagramm zur Annotation von Medienobjekten mit gewichteten Zugehörigkeiten

```
1  //begin algorithm 3.4.4-a
2  function annotateNewMedia(Media newMedia){
3
4  List entities = newMedia.entities;
5
6  if(entities.isEmpty()){
7
8         newMedia.addConfidence("uncategorized", 1.0);
9
10 }else{
11
12        annotateEntites(newMedia, entities); // go listing 3.7
13        newMedia.calculateConfidence(); // go listing 3.11
14
15 }
16
17 newMedia.writeDataToOntology();
18
19 return; }
```

Listing 3.10: Pseudocode: Annotation der Zugehörigkeiten zwischen einem Medienobjekt und Kategorien

Wird der Algorithmus ausgelöst, hier repräsentiert als `annotateNewMedia`-Methode, so muss ein Medienobjekt `newMedia` übergeben werden. Informationen, die für das Medienobjekt existieren müssen, sind eine Liste von erkannten Entitäten mit ihren aus einem NER/NED Service mitgelieferten Confidence-Werten und zusätzliche Informationen, wie Autor, Agentur oder Datum der Erstellung. Zunächst muss untersucht werden, ob überhaupt Entitäten übermittelt wurden (Zeile 6). Gibt es keine Entitäten, so wird das Medienobjekt als »uncategorized« beschrieben. Der Grund ist, wie bei der Beschreibung von Entitäten, dass ein Nutzer schneller erkennen kann, dass hier administrativer Handlungsbedarf besteht. Wurden Entitäten bei der NER/NED erkannt und übermittelt, so wird in Zeile 12 zunächst der Algorithmus zur Annotation von Entitäten (siehe Listing 3.7) aufgerufen. Dies ist nötig, da nicht sichergestellt ist, ob die Entität bereits in der Ontologie existiert. Sind alle Daten bezüglich der Entitäten ermittelt, werden auf Basis dieser Daten die Zugehörigkeiten des Medienobjekts zu Kategorien ermittelt bzw. berechnet. Dazu wird in Zeile 13 der Teilalgorithmus in Listing 3.11 aufgerufen. Sind die Berechnungen abgeschlossen oder das Medienobjekt als »uncategorized« beschrieben, so werden alle gewichteten Zugehörigkeiten und zusätzlichen Informationen in die Ontologie geschrieben.

Nachfolgend ist in Abbildung 3.16 sowie Listing 3.11 der Ablauf des Teilalgorithmus zur Berechnung der Zugehörigkeiten eines Medienobjekts zu relevanten Kategorien dargestellt. Dieser wird aufgerufen, wenn alle Entitäten eines Medienobjektes annotiert wurden oder nicht im Service existieren.

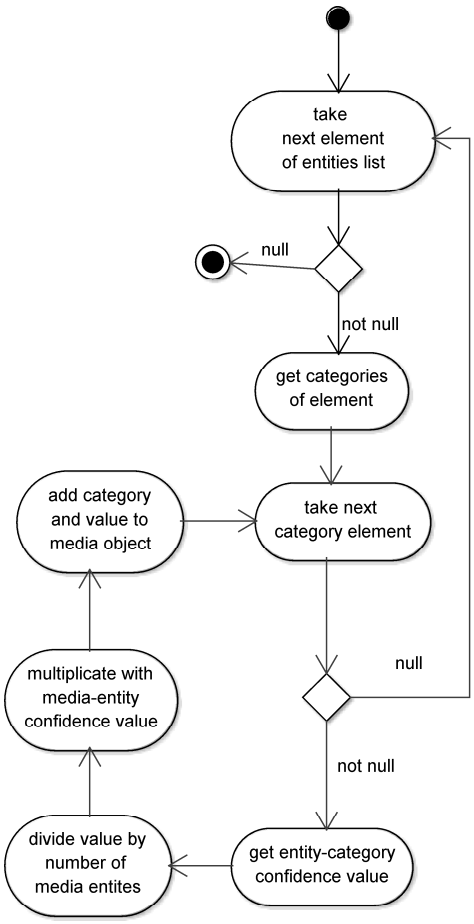


Abbildung 3.16: Aktivitätsdiagramm zur Berechnung von Kategoriezugehörigkeiten von Medienobjekten

```
1  //begin algorithm 3.4.4-b
2  function calculateConfidence(){
3  for each (entity : newMedia.entities){
4
5         for each (category : entity.getDependencyList()){
6
7                 float confidenceValue = (entity.getConfidence(category)/newMedia.
                      entities.size())*newMedia.getNerConfidence(entity);
8                 addConfidence(category, confidenceValue);
9         }
10 }
11 return; } // returns to listing 3.10 line 13
```

Listing 3.11: Pseudocode: Berechnung des Confidence-Werts zwischen einem Medienobjekt und Kategorien

Bei der Berechnung werden die relevanten Kategorien jeder Entität des Medienobjekts untersucht. Die Berechnung in Zeile 7 erfolgt dabei mit der in Kapitel 3.4.1 vorgestellten Gleichung. Der Confidence-Wert des Medienobjekts zu einer Kategorie ergibt sich dabei aus der Summe der Werte, die pro Kategorie errechnet werden (Zeile 8).

Mithilfe dieser drei Algorithmen ist es nun möglich, eine automatisierte Kategorisierung von Medienobjekten, deren Entitäten und Kategorien selbst vorzunehmen. Dazu werden Zugehörigkeiten bestimmt und die Confidence-Werte über die Aussage der Zugehörigkeit berechnet. Alle diese berechneten Daten werden in die Ontologie eingetragen. Im Falle einer Änderung der Menge von relevanten Kategorien werden alle Zugehörigkeiten neu berechnet.

3.5 Zusammenfassung

In diesem Kapitel wurde das Konzept für das Erreichen der Ziele dieser Arbeit erläutert. Es wurde zunächst eine Einordnung des Konzepts in das Topic/S-System vorgenommen und Einschränkungen im Rahmen der aufgestellten Anforderungen aus Kapitel 2.1 getroffen. Eine vorgegebene Medienontologie wurde verwendet und so erweitert, dass sie in ihrer Funktionalität nicht eingeschränkt wird. Dabei ist es nun mit dieser Ontologie möglich, viele im Rahmen des Topic/S-Projekt vorhandene Objekte wie u. a. Entitäten, Medien, zu veröffentlichende Items und Zeitungsseiten mit unscharfen Aussagen zu kategorisieren. Hierbei wurde die Modellierung durch Nutzung von n-ären Relationen gewählt, wodurch Zugehörigkeiten mithilfe von Hilfsknoten und sogenannten Confidence-Werten zwischen 0 und 1 gewichtet werden können. Die erweiterte Medienontologie wird schließlich in einer semantischen Datenbank verwendet. Die Architektur für dieses Konzept trennt diese von einem Backend, Frontend und einem Service, der als Wissensbasis dient, ab. Das Backend spielt hierbei eine zentrale Rolle, da es die Schnittstellen für den Service und die Datenbank bereitstellt. Die Datenbank kann im Frontend durch einen Nutzer angefragt werden, um z. B. mit Hilfe von SPARQL Daten aus der Ontologie zu erhalten. Ein besonderer Handler im Backend verwaltet Service und Datenbank und ruft entsprechende Methoden zum unscharfen, semantischen Annotieren der betrachteten Objekte auf. Die Aufrufe können durch ein eintreffendes Medienobjekt, beispielsweise von einer Agentur aus gesendet, oder durch Hinzufügen bzw. Entfernen einer relevanten Kategorie durch einen Nutzer im Frontend ausgelöst werden. Entsprechend wurden für diese Fälle Algorithmen vorgestellt, die es ermöglichen, Zugehörigkeiten von Medienobjekten, Entitäten und Kategorien zu Kategorien mit einer bestimmten Sicherheit, repräsentiert durch einen Confidence-Wert, zu bestimmen. Zuvor wurden hier grundlegende Probleme vorgestellt, die bei der Erarbeitung des Algorithmus bestanden und gelöst werden mussten.
Diese Teile des Konzeptes werden nun in prototypischer Form für die wichtigsten Funktionalitäten in Kapitel 4 umgesetzt. Anschließend erfolgt in Kapitel 5 eine Evaluation des Konzepts mit Hilfe des Prototypen anhand eines Testszenarios, wobei die wichtigsten Funktionen geprüft und typische Anfragen bei der Verwendung des Systems vorgestellt werden.

4 Implementierung

In diesem Kapitel werden alle Probleme der Implementierung, deren Lösungen sowie wichtige Programmierentscheidung vorgestellt. Einführend wird die prototypische Umsetzung der Architektur und des Konzepts mit Beschreibungen zu den Komponenten präsentiert. Im Anschluss wird ein Überblick über Probleme gegeben, die während der Implementierungsphase auftraten und aufgezeigt wie sie gelöst wurden. Als Abschluss wird die Installation, Konfiguration und Verwendung des Prototypen erklärt.

4.1 Prototypische Umsetzung

In diesem Abschnitt werden die Komponenten und Funktionsweisen des Prototypen erläutert. Der Funktionsumfang des Prototypen beschränkt sich dabei auf die für die Evaluation des Konzepts wichtigsten Funktionalitäten:

- Die automatische Kategorisierung und Annotation eines Medienobjekts
- Die automatische Kategorisierung und Annotation von Entitäten
- Das Einfügen einer Kategorie, deren Kategorisierung und die daraus resultierende Neuberechnung von Daten
- Das Löschen einer Kategorie und die daraus resultierende Neuberechnung von Daten

Für diesen Prototyps wird angenommen, dass übergebene Entitäten eindeutig und korrekt bestimmt sind, sie also zu keiner Begriffserklärung führen und keine Weiterleitung sind. Des Weiteren wird das Medienobjekt über ein browserbasiertes Frontend nur in Form seines Namens und seines Typs, einer Liste von Entitäten und deren Typen sowie weiteren Informationen über Eingabefelder an das System übergeben.

4.1.1 Entwicklungswerkzeuge

Für die Umsetzung des Konzepts als Prototyp wurden folgende Entwicklungswerkzeuge, Services und Software verwendet:

- TopBraid Composer zum Erstellen der erweiterten Medienontologie
- Java als Programmiersprache
- Eclipse als Programmierwerkzeug

- OWLIM Lite 5.2.5331, auf einem Tomcat 7 aufgesetzt und als semantische Datenbank genutzt. Hier wird die Medienontologie gespeichert.
- Die deutsche DBpedia als Wissensbasis über SPARQL-Endpoint unter `http://de.dbpedia.org/sparql`

Der Prototyp wurde zwar auf einem Windows-System entwickelt, ist aber durch die plattformunabhängige Programmiersprache und die Nutzung von Web-Services auch auf anderen Betriebssystemen, wie Ubuntu, lauffähig. Dies ermöglicht die Verwendung des Systems für eine breite Zahl von Nutzern und externen Systemen, die auf Daten zugreifen oder Medienobjekte einsenden wollen. Im folgenden Abschnitt werden die einzelnen Komponenten des Systems, auch in Bezug auf die hier vorgestellten Umgebungswerkzeuge, behandelt.

4.1.2 Komponenten

Das Klassendiagramm in Abbildung 4.1 zeigt die umgesetzte Architektur und verwendeten Klassen. Die Implementierung beschränkt sich hierbei nur auf das Backend der in Kapitel 3.3 vorgestellten Architektur. Als Service für eine semantische Wissensbasis dient der SPARQL-Endpoint der deutschen DBpedia. Für die Verwaltung der Daten in einer semantischen Datenbank wird OWLIM Lite mit der eingebundenen erweiterten Medienontologie verwendet. Dieser Triplestore wurde gewählt, da er beim Ausführen von Benchmarktests und realen Datensätzen eine gute Skalierbarkeit und Gesamt-Performanz gezeigt hat. Ein detaillierter Bericht dazu ist in [VMS12] zu finden. Im Backend werden zwei packages verwendet. Das `handler`-package beinhaltet alle Klassen und Schnittstellen, die zur Steuerung verschiedener Aktivitäten bzw. Nutzung des Services und der Datenbank notwendig sind. Diese Klassen sind statisch. Im `objects`-package befinden sich alle Klassen, die in der Ontologie vorhanden und instanziierbar sind. Für diesen Prototypen sind das lediglich Klassen für Kategorien, Entitäten und Medienobjekte. In den folgenden Abschnitten werden die Klassen und ihre Methoden bzw. Funktionalitäten im Backend erläutert.

Klassen im Package `objects`

Da die Klassen im Package `objects` eine fast identische Menge von Methoden anbieten, werden sie in diesem Abschnitt zusammengefasst. Sie sind instanziierbar und beinhalten alle bestimmten Zugehörigkeiten und deren berechneten Confidence-Werte. Die Algorithmen zur Bestimmung der Zugehörigkeiten zu Kategorien sind jeweils für die Klassen `Category` und `Entity` mit der `getCategoryCounts`-Methode umgesetzt worden. In der `calculateCategories`-Methode, welche von allen drei Klassen bereitgestellt wird, erfolgt die Berechnung der Confidence-Werte, für Entitäten und Kategorien mithilfe der zuvor bestimmten Zugehörigkeiten und für Medienobjekte anhand der erkannten Entitäten. Des Weiteren gibt es Methoden für das Lesen und Schreiben von Daten aus bzw. in die Ontologie, welche sich inhaltlich jedoch stark unterscheiden. Werden hier für die `Category`-Klasse nur Informationen über die Zugehörigkeiten zu anderen Kategorien betrachtet, so müssen bei der `Entity`-Klasse zusätzliche Informationen, hier in einer `additionalInformation`-Map gespeichert, einbezogen werden. Für Medienobjekte zählen hier sogar noch

die Zugehörigkeiten von erkannten Entitäten dazu. Für die Klasse `Media` werden außerdem noch Methoden für das Hinzufügen von Entitäten und zusätzlicher Informationen bereitgestellt.

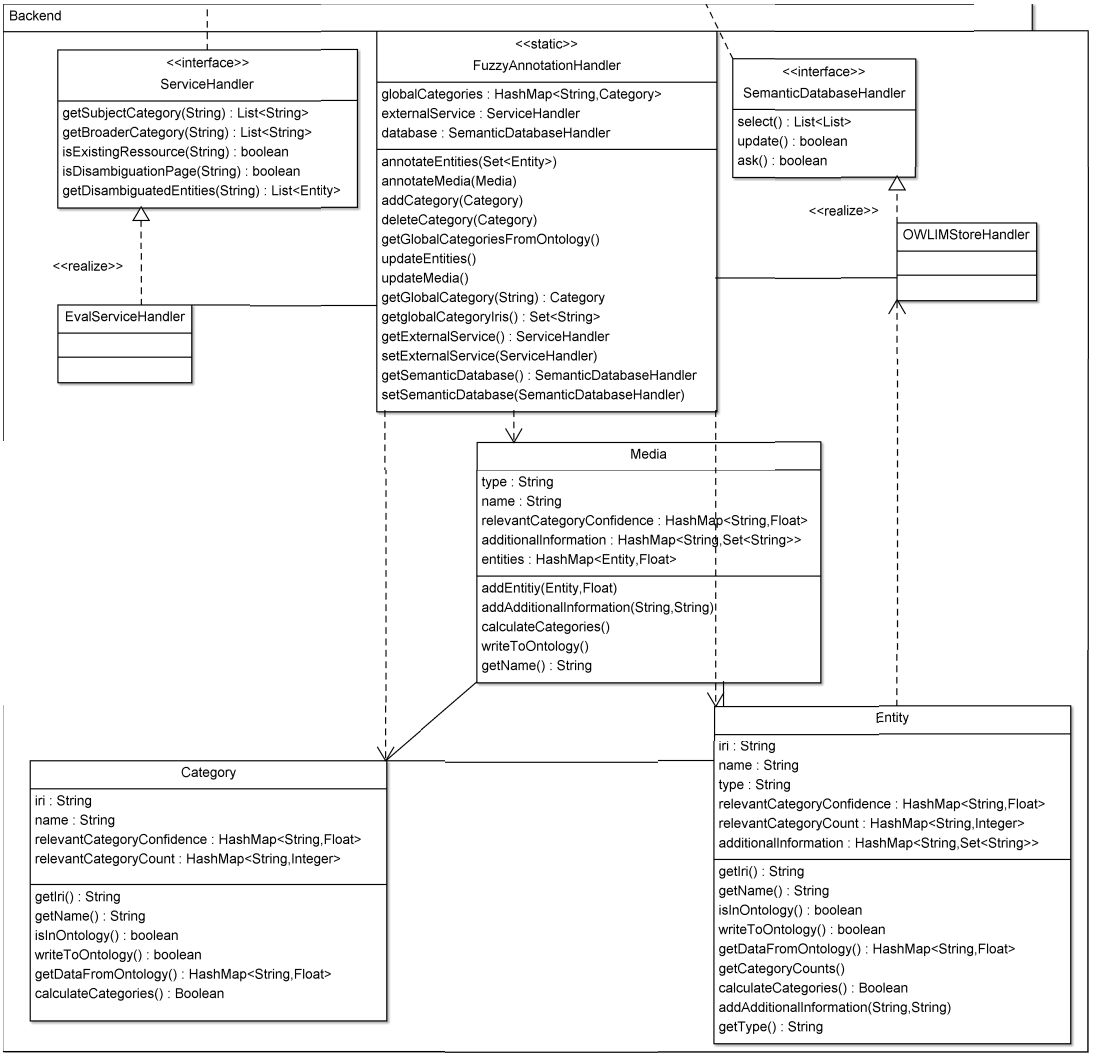


Abbildung 4.1: Klassendiagramm für die Umsetzung des Prototypen

FuzzyAnnotationHandler

Als Hauptkomponente des Systems stellt der `FuzzyAnnotationHandler` Methoden bereit, um Aktionen zum Annotieren von Daten zu steuern. Sie ist außerdem die Komponente, die hauptsächlich vom Frontend genutzt wird. In dieser statischen Klasse werden außerdem die Handler für den Service der Wissensbasis und der semantischen Datenbank registriert. Die wichtigsten Methoden behandeln das Einfügen und Löschen von Kategorien und das damit verbundene Aktualisieren der Entitäten und Medienobjekte sowie das Hinzufügen und Annotieren der beiden Letztgenannten. Das Hinzufügen von Objekten bzw. deren Annotation unscharfer Aussagen der Zugehörigkeiten dieser Objekte zu Kategorien wurde strikt nach den im Kon-

zept vorgestellten Algorithmen implementiert. Für die Methoden `updateEntities` und `updateMedia` werden derzeit nur alle in der Ontologie existierenden Objekte aktualisiert ohne Bezug auf zeitliche Abhängigkeiten zu nehmen, wie das priorisierte Bearbeiten kürzlich oder im jüngsten Zeitraum am häufigsten benutzter Entitäten und der neuesten Medienobjekte. Diese Vorgänge können demnach noch optimiert werden, um als Hintergrundprozesse den Datenbestand aktuell zu halten.

ServiceHandler und DbPediaHandler

In Kapitel 3.3 wurde bereits auf die Architektur eingegangen, welche Schnittstellen für den `ServiceHandler` und den `SemanticDatabaseHandler` vorsieht. Für diesen Prototypen wurde dementsprechend eine Klasse erstellt, welche die `ServiceHandler`-Schnittstelle implementiert. Diese Klasse heißt `DbPediaHandler` und hält Methoden bereit, um Anfragen an die deutsche DBpedia stellen zu können. Im speziellen sind dies Methoden, die direkte Kategorien für Entitäten (`getSubjectCategory`) und Oberkategorien (`getBroaderCategory`) in Form einer Liste zurückliefern und bei der Bestimmung von Zugehörigkeiten von Entitäten und Kategorien genutzt werden. Des Weiteren gibt es Methoden, um die Existenz einer Ressource, von Begriffserklärungen sowie Weiterleitungen zu überprüfen und zu behandeln. Für diesen Prototypen werden diese, bis auf das Prüfen der Existenz, nicht verwendet, da hier zusätzlich Betrachtungen zur Bestimmung des Typs einer Entität getroffen werden müssen.

SemanticDatabaseHandler und OWLIMStoreHandler

Bei der Implementierung der Schnittstelle für die semantische Datenbank sind Methoden zum Stellen von Anfragen mit ASK- und SELECT-Konstrukten sowie von SPARQL-Update umzusetzen. Daher sind hier die Methoden `select`, `ask` und `update` vorgesehen. Mit der Klasse `OWLIMStoreHandler` wurde die `SemanticDatabaseHandler`-Schnittstelle für OWLIM Lite, aufgesetzt als Endpoint auf einem Tomcat, programmiert. Der Endpoint wird hier mithilfe des `java.net.*`-packages angefragt. Für SELECT-Anfragen wird eine Ergebnisliste zurückgegeben, welche für jedes Ergebnistupel eine eigene Liste enthält. Beim Endpoint wird dies durch einen GET-Request mit einer URL, welche die Anfrage enthält, und dem Auslesen der Ergebnisdaten aus der Response des Servers umgesetzt. Dies gilt auch für die `ask`-Methode, welche allerdings einen booleschen Wert zurückgibt. Für die SPARQL-Update Anfragen muss hingegen, hier im Falle von OWLIM Lite, ein POST-Request genutzt werden. Die Implementierung der Schnittstelle ist also abhängig davon, ob die Datenbank, wie in diesem Fall, extern genutzt wird und über HTTP angesprochen werden kann und wie der benutzte Endpoint verschiedene Anfragetypen entgegennimmt, hier am Beispiel des POST-Requests für SPARQL-Update.

Man sieht, dass die Architektur, welche in der Konzeption vorgestellt wurde, entsprechend umgesetzt worden ist. Bei der Implementierung traten jedoch auch Probleme auf, welche im nächsten Abschnitt vorgestellt werden.

4.2 Probleme während der Implementierungsphase

Während der Umsetzung des Konzepts als Prototyp kamen verschiedene Probleme auf, welche die Performanz der Bestimmung der Zugehörigkeiten negativ beeinflussen, aufgrund von Datenstrukturen Fehler aufrufen und zuletzt auch die Kodierung von Schriftzeichen betreffen. Im Folgenden werden diese Probleme und die angewandte Lösung näher erläutert.

4.2.1 Performanz bei Nutzung von Services und Verbesserung des Algorithmus

Das erste Problem, welches bei der Umsetzung des Konzepts schnell sichtbar wurde, war die lange Dauer der Bestimmung von Zugehörigkeiten einer Kategorie oder einer Entität zu einer relevanten Kategorie. Die Begründung hierfür ist, dass das Stellen einer Anfrage an den Service für die Wissensbasis und Warten bzw. Verarbeiten der Antwort bis zu einige hundert Millisekunden dauerte. Daher musste ein Weg gefunden werden, welcher die Zahl der Anfragen an den Service so effektiv wie möglich reduziert und einen Großteil der Bestimmung der Zugehörigkeiten auf den lokalen Speicher verlagert. Da bei den Anfragen nur Strings verarbeitet werden, wird, je nach Zahl eingepflegter Kategorien im genutzten Service, der lokale Speicher unterschiedlich stark belegt, wobei er aber nicht überlastet wird.
Konkret wurde diese Verlagerung durch Nutzen zweier Maps realisiert, wobei eine Map den Vorgänger einer erreichten relevante Kategorie und die andere die Zahl der Treffer für diese Kategorie speichert. Dies betrifft sowohl relevante als auch für den Nutzer irrelevante Kategorien, welche nicht in der Medienontologie aufgenommen wurden. In der Map, in welcher Vorgänger protokolliert werden, entsteht somit eine Art Baumstruktur, welche anschließend, ausgehend von jeder erreichten relevanten Kategorie, abgelaufen und die Zahl der Treffer pro Kategorie für die relevante Kategorie addiert wird. Durch dieses Verfahren können die Pfade bis zu zehn mal schneller abgearbeitet werden.
Eine weitere Möglichkeit langsame, externe Services performanter zu machen, wäre das lokale Aufsetzen eben dieser. Im Rahmen dieser Arbeit konnte dies aufgrund fehlender Rechnerressourcen nicht durchgeführt und getestet werden. Jedoch ist die lokale Verlagerung auch hier noch schneller, da alle unterschiedlichen Pfade, die zu einer relevanten Kategorie führen nur einmal abgearbeitet werden. Bei einem lokal aufgesetzten Service werden ähnliche Teilpfade mehrmals abgesucht, was bei dem hier vorgestellten Ansatz eine weitere Zeitersparnis bedeutet.

4.2.2 Dynamisches Erweitern der `remainingCategories`-Liste

Ein weiteres Problem während der prototypischen Implementierung war das dynamische Ändern von Listen. Dies trat speziell bei der `addCategory`-Methode des `FuzzyAnnotationHandler` auf, wenn man über die `remainingCategories`-Liste der noch zu bearbeitenden Kategorien iteriert und dabei Elemente einfügen möchte. Das Ergebnis war eine `ConcurrentModificationException`, welche es zu lösen galt.
Hierzu wurden zwei Werte festgelegt: einen Wert, der den aktuellen Index in der Liste repräsentiert, damit wirklich immer das nächste Element bis zum Erreichen

des Endes der Liste bearbeitet wird und einen Wert, der den aktuellen Wert des letzten Index der Liste repräsentiert. Wenn ein Element der Liste hinzugefügt wird, so wird der `maxIndex` inkrementiert. Hingegen bei jeder Abarbeitung eines Elements der Liste wird der aktuelle Index, der `currentIndex` um 1 erhöht. Der Sachverhalt ist in Listing 4.1 als Auszug des Quelltextes dargestellt. Durch die Nutzung dieser zwei Hilfswerte ist es also möglich, gleichzeitig über die Liste zu laufen und sie zu erweitern.

```
1  public static void addCategory(Category newCat){
2
3  LinkedList<Category> remainingCategories = new LinkedList<Category>();
4  remainingCategories.add(newCat);
5
6  if(!newCat.isInOntology()){
7
8         if(!globalCategories.isEmpty()){
9
10        int currentIndex = 0;
11        int maxIndex = 0;
12               while(currentIndex <= maxIndex){
13
14               Category nextCat = remainingCategories.get(currentIndex);
15               nextCat.getCategoryCounts();
16               globalCategories.put(nextCat.getIri(), nextCat);
17
18               for(String globalCat:FuzzyAnnotationHandler.globalCategories.
                     keySet()){
19
20               if((!nextCat.relevantCategoryCount.containsKey(globalCat))){
21
22                              if(!remainingCategories.contains(
                                    FuzzyAnnotationHandler.getCategory(globalCat))){
23
24                                  remainingCategories.add(
                                        FuzzyAnnotationHandler.getCategory(
                                        globalCat));
25                                  maxIndex++;
26
27                              }
28               }
29               }
30               currentIndex++;
31               }
32        ...
```

Listing 4.1: Quellcode: Iterieren über zu ändernde Liste

4.2.3 UTF-8 Kodierung

Ein letztes Problem, welches hier vorgestellt wird, betrifft die Zeichensatzkodierung in Bezug auf Anfragen über HTTP. UTF-8 ist mit seinem Umfang von Zeichen als de-facto-Standard im Internet etabliert. Jedoch wird er nicht von jeder Software oder jedem Server standardmäßig verwendet. Bei der Verarbeitung der deutschen Sprache mit ihren Umlauten und anderen Sonderzeichen ist es jedoch wichtig, dass diese

Zeichen ohne Fehler im System verarbeitet werden. Zur Verarbeitung der Input- und Outputstreams für HTTP Anfragen an Service und Datenbank müssen daher alle Zeichenketten in UTF-8 kodiert werden. Dies ist Java möglich, indem bei der Definition des verwendeten Streams die Zeichenkodierung übergeben wird. Als Beispiel dient ein Auszug des Quellcodes in Listing 4.2, welcher aus den `select`- und `update`-Methoden der Klasse `OWLIMStoreHandler` entnommen wurde. Es zeigt die Initialisierung eines `BufferedReader`, welcher den Inputstream in UTF-8 kodiert, sowie das Setzen des UTF-8 Zeichensatzes im Request-Header einer POST-Methode.

```
1 URL url = new URL(storeUrl + URLEncoder.encode(query,"UTF-8"));
2 ...
3 //http - GET
4 BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream(), "
      UTF-8"));
5 ...
6 //http - POST
7 HttpURLConnection conn = (HttpURLConnection) url.openConnection();
8 ...
9 conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded;
      charset=UTF-8");
```

Listing 4.2: Quellcode: UTF-8 Kodierung von Input- und Outputstream

Zusätzlich ist es hierbei auch nötig den Server, auf dem die semantische Datenbank aufgesetzt ist, auf den UTF-8 Zeichensatz festzulegen. In Listing 4.3 ist ein Beispiel für eine Einstellung eines Tomcats auf UTF-8. Hierzu muss der sich in der `server.xml`-Datei befindliche `Connector` für den Port 8080 erweitert werden.

```
1
2 <!-- $tomcat_home$\conf\server.xml -->
3 <Connector port="8080" protocol="HTTP/1.1"
4               connectionTimeout="20000"
5               redirectPort="8443"
6                             URIEncoding="UTF-8" />
```

Listing 4.3: Änderung der Zeichenkodierung bei einem Tomcat

4.3 Zusammenfassung

In diesem Kapitel wurden die Ergebnisse der Implementierung behandelt. Dabei wurde auf die Architektur des Konzepts eingegangen und deren konkrete Umsetzung gezeigt. Es wurde erläutert, welche Komponente welche Funktionen umfasst und nutzt. Außerdem wurden eine Reihe von Problemen und ihre Lösungsansätze erläutert. Hierbei sticht besonders die Problematik der Performanz des Systems durch Nutzung eines Services heraus. Durch die Nutzung von Web-Services über HTTP Schnittstellen können außerdem Probleme in Bezug auf die Zeichenkodierung entstehen, welche man für die korrekte Darstellung und Verarbeitung der Daten in der Ontologie beheben sollte.

5 Evaluierung

In diesem Abschnitt wird das Konzept mit Hilfe des umgesetzten Prototypen und einem Testszenario evaluiert. Dies soll die erfolgreiche Umsetzbarkeit des Konzepts beweisen und zeigen, dass alle angestrebten Ziele dieser Bachelorarbeit umgesetzt wurden.

5.1 Testszenario

In Anlehnung an das Referenzszenario wird das Testszenario für diese Evaluierung in vier Teilaufgaben gesplittet. Diese Aufgaben decken die Kategorisierung sowie Annotation der erhaltenen Daten für ein Medienobjekt und seine Entitäten, das Einfügen sowie das Löschen einer Kategorie und die daraus erfolgenden Neuberechnungen der Daten ab. Zwischen jedem dieser Schritte werden die Daten durch eine passende Anfrage an die Ontologie überprüft.
Um die Ergebnisse mit eigenen Berechnungen überprüfen zu können, wird eine sehr kleine Evaluierungsontologie eingesetzt, die als Wissensbasis eines Services dient. Sie besteht aus drei Entitäten und sieben Kategorien, welche in Relationen zueinander mithilfe der Propertys `skos:broader` und `dcterms:subject` stehen. Zusätzlich wurde ein Zyklus mit der Pseudokategorie `Zyklus` eingebaut. Die Ontologie ist in Abbildung 5.1 skizziert.

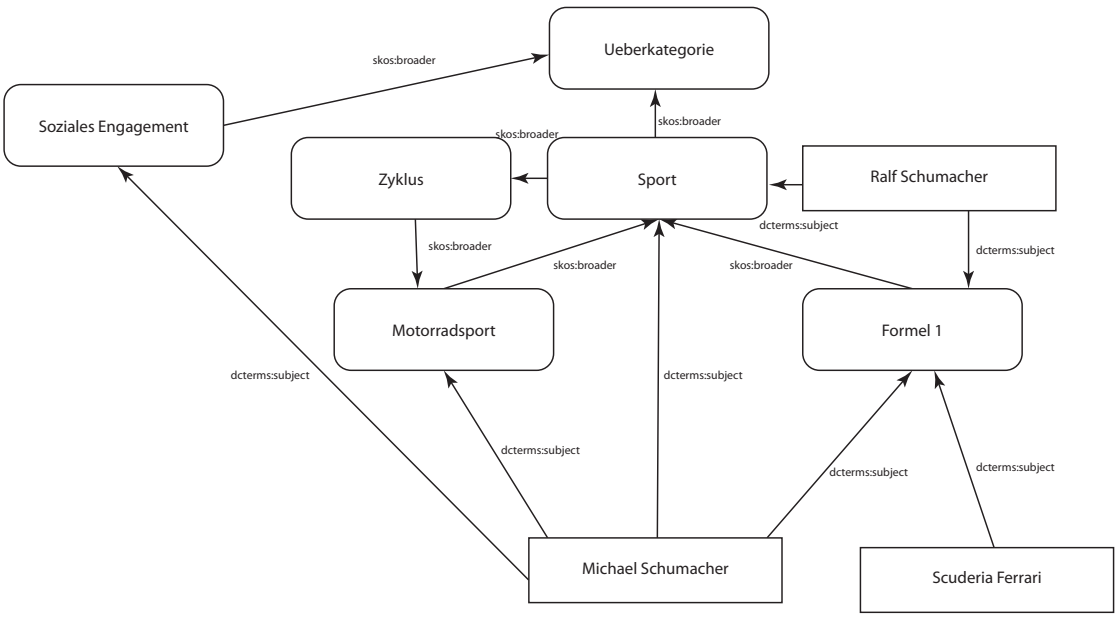


Abbildung 5.1: Evaluierungsontologie für das Testszenario

Für das Testszenario wird, wie im Referenzszenario, ein Artikel als Medienobjekt angenommen, der über die Formel 1-Karrieren von Michael und Ralf Schumacher berichtet. Zu diesem Artikel werden die in Tabelle 5.1 gelisteten Entitäten mit ihren Typen und Confidence-Werten übergeben. Außerdem werden auch zusätzliche Informationen wie Autor und Datum übermittelt.

Entität	Typ	$\text{conf}_{\text{MO ENT}}$
Michael Schumacher	foaf:Person	0.9
Ralf Schumacher	foaf:Person	0.5
Scuderia Ferrari	foaf:Organization	1.0

Tabelle 5.1: Liste von Entitäten und Confidence-Werten, die mit dem Medienobjekt übergeben werden

Die SPARQL-Anfrage, welche nach jedem der in diesem Testszenario behandelten Teilschritte ausgeführt wird, ist in Listing 5.1 dargestellt. Sie liefert alle Objekte, deren Typ und zugehörige Kategorien sowie die errechneten Confidence-Werte für diese Zugehörigkeiten. Das Ergebnis ist dabei absteigend nach Confidence-Werten geordnet.

```
select ?objectName ?categoryName ?confidence ?type
where{
       ?object tpcs:hasFuzzyNode ?fuzzynode.
       ?object rdfs:label ?objectName.
       ?object a ?type.
       ?fuzzynode tpcs:hasSemItem ?category.
       ?category a tpcs:Category.
       ?category rdfs:label ?categoryName.
       ?fuzzynode tpcs:hasConfidence ?confidence.
              filter (
              ?type = event:Event ||
              ?type = foaf:Person ||
              ?type = foaf:Organization ||
              ?type = schema:Place||
              ?type = tpcs:Media||
              ?type = tpcs:Category
              )
}order by DESC (?confidence)
```

Listing 5.1: SPARQL-Anfrage zur Überprüfung der berechneten Zugehörigkeiten von Objekten zu Kategorien

Bestimmen der relevanten Kategorie

Es werden nun die zu testenden Funktionen nacheinander abgearbeitet. Hierbei werden im ersten Schritt die Kategorien hinzugefügt, die für einen Benutzer relevant sein sollen. Für die Menge der in der Evaluierungsontologie befindlichen Kategorien sind

dies `Sport`, `Formel_1` und `Motorradsport`. Die vom Programm zu berechnenden Zugehörigkeiten sind in Tabelle 5.2 dargestellt. In Abbildung 5.2 ist ein Screenshot der Ergebnismenge in OWLIM als Antwort auf die in Listing 5.1 dargestellte SPARQL-Anfrage, welche eben dieses Ergebnis bestätigen soll. Auf Basis dieser Daten sind nun die Berechnungen für den Artikel und die extrahierten Entitäten möglich.

Kategorie	conf $_{\text{CAT Sport}}$	conf $_{\text{CAT Formel1}}$	conf $_{\text{CAT Motorradsp.}}$
Sport	-	0.0	0.0
Formel 1	1.0	-	0.0
Motorradsport	1.0	0.0	-

Tabelle 5.2: Zu erwartende Ergebnisse der Berechnung von Zugehörigkeiten von Kategorien

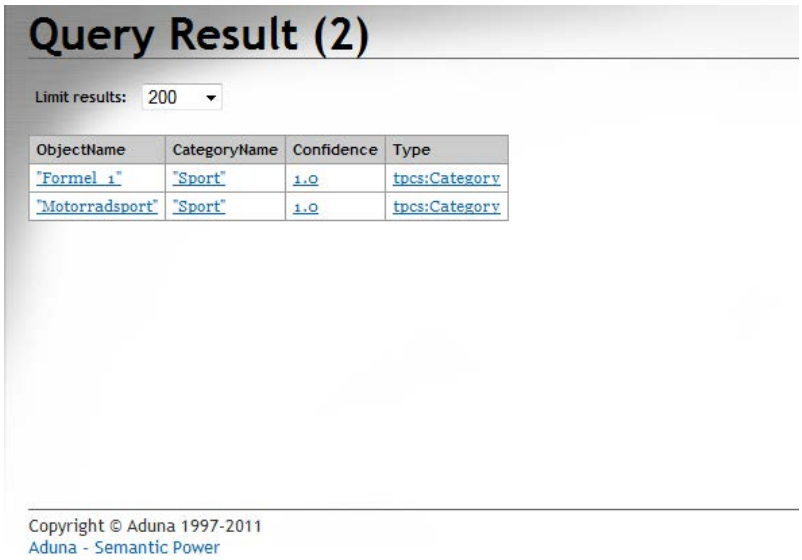


Abbildung 5.2: Ermittelte Ergebnisse der Berechnung von Zugehörigkeiten von Kategorien

Annotation eines Medienobjektes und seiner Entitäten

Nachdem nun die für den Nutzer relevanten Kategorien festgelegt und deren Zugehörigkeiten berechnet wurden, wird das Medienobjekt `Testszenario` an das System übergeben. Zunächst werden die Confidence-Werte der zugehörigen Entitäten zu den Kategorien berechnet. Für das Medienobjekt muss nun, unter Berücksichtigung der in Tabelle 5.1 beschriebenen Confidence-Werte über die Zuordnung Entitäten, die Zugehörigkeit zu den relevanten Kategorien bestimmt werden. Hier dient Tabelle 5.3 als Referenzergebnis. Abbildung 5.3 zeigt die Ergebnisse, der mit dem System berechneten Werte für das Medienobjekt.

Objekt	conf $_{\text{OBJ Sport}}$	conf $_{\text{OBJ Formel1}}$	conf $_{\text{OBJ Motorradsp.}}$
Michael Schumacher	1.0	0.333	0.333
Ralf Schumacher	1.0	0.5	0.0
Scuderia Ferrari	1.0	1.0	0.0
Testszenario	0.8	0.516	0.1

Tabelle 5.3: Zu erwartende Ergebnisse der Berechnung von Zugehörigkeiten des Medienobjekts und seiner Entitäten zu relevanten Kategorien

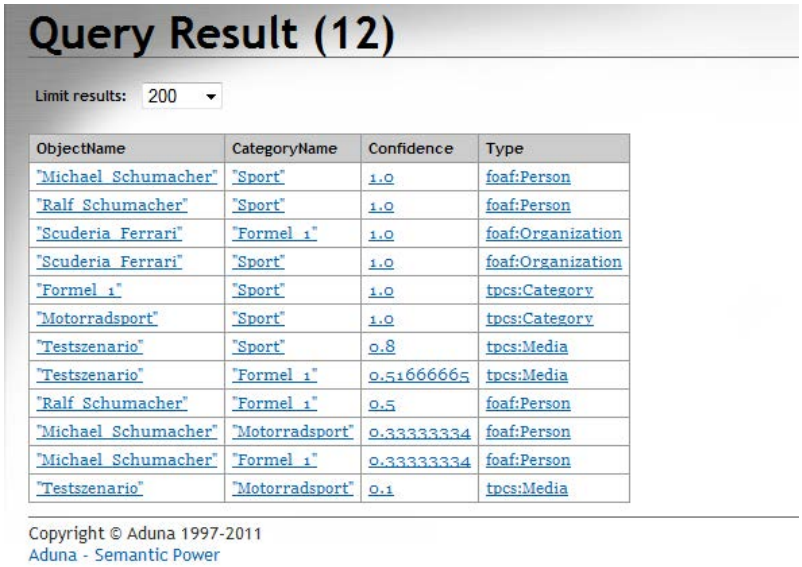


Abbildung 5.3: Ermittelte Ergebnisse der Berechnung von Zugehörigkeiten des Medienobjekts und seiner Entitäten zu relevanten Kategorien

Einfügen und Löschen einer neuen Kategorie

Als letzte Teilschritte wird nun eine neue Kategorie namens `Soziales Engagement` eingefügt und später wieder gelöscht. Bei dem Einfügen werden die Zugehörigkeiten zwischen der neuen und den bisher relevanten Kategorien bestimmt, was die Aktualisierung des gesamten Datensatzes, also für Entitäten und Medienobjekte nach sich zieht. Die neuen berechneten Werte sind in Tabelle 5.4 und Abbildung 5.4 zu vergleichen.

Objekt	conf $_{\text{OBJ Sport}}$	conf $_{\text{OBJ Formel1}}$	conf $_{\text{OBJ Motorradsp.}}$	conf $_{\text{OBJ Soz.Engagem.}}$
Sport-	0.0	0.0	0.0	
Formel 1	1.0	-	0.0	0.0
Motorradsport	1.0	0.0	-	0.0
Soz. Engagement	0.0	0.0	0.0	-
Michael Schumacher	0.75	0.25	0.25	0.25
Ralf Schumacher	1.0	0.5	0.0	0.0
Scuderia Ferrari	1.0	1.0	0.0	0.0
Testszenario	0.725	0.492	0.075	0.075

Tabelle 5.4: Zu erwartende Confidence-Werte aller Kategorien, Entitäten und des Medienobjekts

Query Result (14)

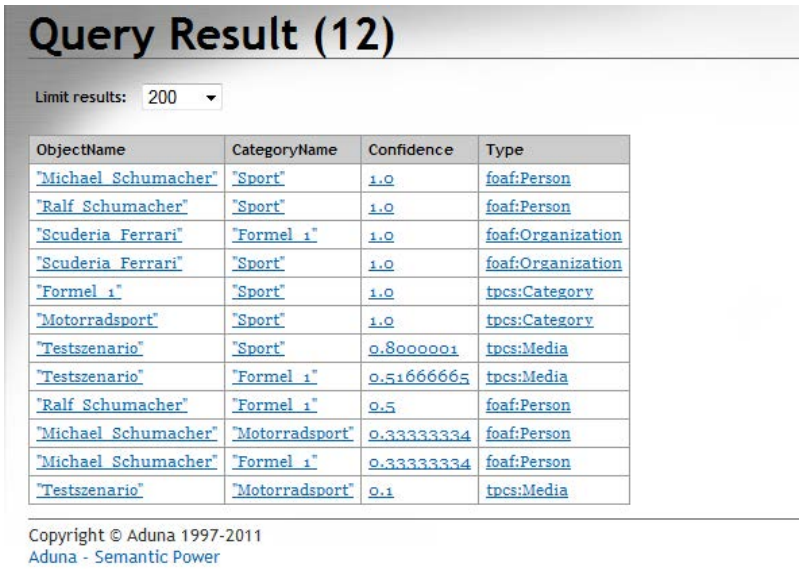
Limit results: 200

ObjectName	CategoryName	Confidence	Type
"Ralf_Schumacher"	"Sport"	1.0	foaf:Person
"Scuderia_Ferrari"	"Formel_1"	1.0	foaf:Organization
"Scuderia_Ferrari"	"Sport"	1.0	foaf:Organization
"Formel_1"	"Sport"	1.0	tpcs:Category
"Motorradsport"	"Sport"	1.0	tpcs:Category
"Michael_Schumacher"	"Sport"	0.75	foaf:Person
"Testszenario"	"Sport"	0.725	tpcs:Media
"Ralf_Schumacher"	"Formel_1"	0.5	foaf:Person
"Testszenario"	"Formel_1"	0.4916667	tpcs:Media
"Michael_Schumacher"	"Soziales_Engagement"	0.25	foaf:Person
"Michael_Schumacher"	"Motorradsport"	0.25	foaf:Person
"Michael_Schumacher"	"Formel_1"	0.25	foaf:Person
"Testszenario"	"Soziales_Engagement"	0.075	tpcs:Media
"Testszenario"	"Motorradsport"	0.075	tpcs:Media

Abbildung 5.4: Ermittelte Confidence-Werte aller Kategorien, Entitäten und des Medienobjekts

Wird die gerade hinzugefügte Kategorie gelöscht, so müssen die in den Tabellen 5.2 und 5.3 errechneten Ergebnisse wieder ermittelt werden. Das Ergebnis ist in Ab-

bildung 5.5 dargestellt. Man erkennt hier nur minimale Abweichungen der vorherigen Ergebnisse, die auf das Runden von Werten zurückzuführen sind.



Query Result (12)

Limit results: 200

ObjectName	CategoryName	Confidence	Type
"Michael_Schumacher"	"Sport"	1.0	foaf:Person
"Ralf_Schumacher"	"Sport"	1.0	foaf:Person
"Scuderia_Ferrari"	"Formel_1"	1.0	foaf:Organization
"Scuderia_Ferrari"	"Sport"	1.0	foaf:Organization
"Formel_1"	"Sport"	1.0	tpcs:Category
"Motorradsport"	"Sport"	1.0	tpcs:Category
"Testszenario"	"Sport"	0.8000001	tpcs:Media
"Testszenario"	"Formel_1"	0.51666665	tpcs:Media
"Ralf_Schumacher"	"Formel_1"	0.5	foaf:Person
"Michael_Schumacher"	"Motorradsport"	0.33333334	foaf:Person
"Michael_Schumacher"	"Formel_1"	0.33333334	foaf:Person
"Testszenario"	"Motorradsport"	0.1	tpcs:Media

Copyright © Aduna 1997-2011
Aduna - Semantic Power

Abbildung 5.5: Ermittelte Ergebnisse der Berechnung von Zugehörigkeiten nach Löschen einer Kategorie mit minimalen Abweichungen

Durch Vergleichen der zu erhaltenen Ergebnisse in den Tabellen und den Abbildungen erkennt man, dass im System die für das Konzept entwickelten Lösungsansätze für die in Kapitel 1.1 beschriebenen Probleme erfolgreich umgesetzt wurden.

5.2 Typische SPARQL-Anfragen bei der Nutzung Systems

In diesem Abschnitt werden SPARQL-Anfragen behandelt, die oft bei der Verwendung des Systems von einem Nutzer gestellt werden könnten. Diese Anfragen können als Grundlage für ein zu entwickelndes User Interface genutzt werden, um die Bedienung des Systems für Nutzer mit fehlender oder keiner Erfahrung in Bezug auf SPARQL zu erleichtern bzw. zu ermöglichen.

Suche nach allen Medienartikeln einer bestimmten Kategorie

Einer der wichtigsten Anwendungsfälle durch einen Nutzer ist die Suche nach allen Medienobjekten, die einer bestimmten Kategorie zugehörig sind. Das in Listing 5.2 dargestellte Beispiel lässt sich um mehrere Kategorien, die ebenfalls oder gar nicht abgedeckt werden sollen, erweitern. Analog lassen sich so auch Entitäten finden, welche einer bestimmten Kategorie zugehörig sind.

```
select ?media
where {
       ?media a tpcs:Media.
       ?media tpcs:hasFuzzyNode ?fuzzynode.
       ?fuzzynode tpcs:hasSemItem tpcs:Sport.
}
```

Listing 5.2: Suche nach Medienobjekten, welche die Kategorie `tpcs:Sport` abdecken

Suche nach allen Medienartikeln mit einer maximalen und minimalen Kategoriezugehörigkeit

Neben der reinen Zugehörigkeit kann auch bestimmt werden, welchen Confidence-Wert diese Zugehörigkeit maximal, minimal oder genau haben darf. Die beiden meist genutzten Anwendungsfälle wären hier die Suche mit der Angabe eines minimalen oder in einem Intervall befindlichen Confidence-Werts. In Listing 5.3 ist das Beispiel für eine Suche nach Medienobjekten, deren Zugehörigkeit zu einer Kategorie in einem Intervall liegt, dargestellt.

```
select ?media ?confidence
where {
       ?media a tpcs:Media.
       ?media tpcs:hasFuzzyNode ?fuzzynode.
       ?fuzzynode tpcs:hasConfidence ?confidence.
       ?fuzzynode tpcs:hasSemItem tpcs:Sport.
              filter (?confidence > 0.6 && ?confidence < 0.7)
}
```

Listing 5.3: Suche nach Medienobjekten und ihrer Confidence-Werte zur Kategorie `tpcs:Sport` mithilfe eines Intervalls

Suche nach allen Medienartikeln mit bestimmten Entitäten

Bei der Suche nach Entitäten muss beachtet werden, dass sowohl Entitäten als auch Kategorien eine Unterklasse des `tpcs:SemItem` sind. Aufgrund der verschiedenen Arten von Entitäten, wie Personen, Orte etc., müssen diese explizit gefiltert werden. In Listing 5.4 wurde eine Suche nach allen Entitäten eines Medienobjekts durchgeführt, welche deren Typen und Confidence-Wert zurückliefert.
Diese Anfrage könnte beispielsweise ausgelöst werden, wenn ein Nutzer auf die Details eines Artikels klickt und seine Entitäten aufgelistet werden sollen. Die Filterung der Entitäten wurde durch eine Anfrage aller semantischen Items abzüglich der Kategorien realisiert. Es können natürlich vorangegangene Anfragen mit dieser verknüpft werden, sodass eine Filterung nach Entitäten und Confidence-Werten vorgenommen werden kann.

```
select ?entity ?type ?confidence
where {
       ?media a tpcs:Media.
       ?media tpcs:hasFuzzyNode ?fuzzynode.
       ?fuzzynode tpcs:hasSemItem ?entity.
       ?entity a ?type.
       ?fuzzynode tpcs:hasConfidence ?confidence.
              filter( ?media = tpcs:Medienobjekt && (
              ?type = event:Event ||
              ?type = foaf:Person ||
              ?type = foaf:Organization ||
              ?type = schema:Place )
              )
}
```

Listing 5.4: Suche nach Entitäten, ihren Typen und Confidence-Werte zu einem Medienobjekt

5.3 Zusammenfassung

In diesem Kapitel wurde ein Testszenario vorgestellt, welches die Grundfunktionen des entwickelten Konzepts auf seine Korrektheit testet. Dabei wurden die vom System berechneten Ergebnisse mit zu erwartenden Ergebnissen verglichen, wodurch gezeigt wurde, dass bei der Implementierung des Prototyps das Konzept korrekt umgesetzt wurde und automatisierte Kategorisierungen von eingehenden Medienobjekten mithilfe einer Wissensdatenbank, hier am Beispiel einer minimalen Testontologie, durchgeführt werden können. Ausgehend von diesem Stand können nun weitere Betrachtungen vorgenommen werden, wobei einige davon im nächsten Kapitel als Ausblick vorgestellt werden.

6 Zusammenfassung der Bachelorarbeit und Ausblick

Diese Bachelorarbeit behandelt die Problematik der Verwendung von Unschärfe in semantischen Modellen am Beispiel des derzeit noch in der Entwicklung befindlichen Topic/S-Projekts. Zu Beginn wurde ein Überblick über die Problemstellung und Ziele, die mit dieser Arbeit erreicht werden sollen, gegeben. Hierbei wurden drei Hauptziele hervorgehoben: Das erste Ziel war die automatisierte unscharfe Kategorisierung von Medienobjekten mithilfe von Services, die den Zugriff auf Wissensdatenbanken anbieten. Ein weiteres Ziel war die Modellierung dieser Daten, sodass die Aussagen mit den berechneten Confidence-Werten als Gewichtungen beschrieben werden können. Des Weiteren war ein Ziel, die Möglichkeit diese Daten auch abfragen zu können.
Zum Erfüllen des ersten Ziels wurden effiziente Algorithmen entwickelt, die es ermöglichen, die Zugehörigkeiten von Kategorien, Entitäten und Medienobjekten zu relevanten Kategorien performant zu bestimmen und zu berechnen. Hierfür wurden Grundprobleme, wie das Erkennen und Auflösen von Zyklen, erkannt und Lösungsansätze umgesetzt. Dabei wurde darauf geachtet, dass die, mithilfe des in Kapitel 2.1 vorangestellten Referenzszenarios klar strukturierten Anforderungen erfüllt bzw. nicht verletzt werden. Für das Erreichen des zweiten Ziels, musste eine bestehende Medienontologie so erweitert werden, dass die Modellierung der Unschärfe ermöglicht wird. Hierbei wurde die Modellierung mit n-ären Relationen gewählt und als eleganter Lösungsansatz angewandt. Durch das Verwenden dieser Art der Modellierung erhält man eine gut strukturierte Ontologie, welche mit einfachen Mitteln durchsuchbar ist. Daten lassen sich ebenso einfach hinzufügen, wie auch entfernen. Gleichzeitig wurde eine robuste und generische Architektur vorgestellt, welche es ermöglicht, das hier entwickelte Teilsystem entweder als Bibliothek in einem Projekt oder als Service einzubinden. Es werden Schnittstellen zur Verfügung gestellt, die es erlauben unterschiedlichste Services von Wissensdatenbanken und Triplestores zum Verwalten der Daten einzubinden und somit einfach in bestehende Systeme zu integrieren. Um das Konzept zu evaluieren, wurde ein Prototyp mit den wichtigsten Funktionen umgesetzt und die Korrektheit des Konzepts anhand eines Testszenarios und mithilfe einer minimalen Wissensontologie gezeigt. Für diesen Prototyp wurde der Algorithmus nochmals verbessert, indem die Bestimmung und Berechnung von Zugehörigkeiten auf eine lokale Ebene verlagert wurde. Außerdem wurden verschiedene Beispiele für wichtige SPARQL-Anfragen für die Verwendung des Systems vorgestellt, mit denen die Daten einfach zu durchsuchen sind.
Im Folgenden werden nochmals alle in Kapitel 2 aufgestellten funktionalen und nichtfunktionalen Anforderungen aufgezeigt und ein Überblick gegeben, mit welchen Mitteln diese erfüllt worden sind.

Automatisierte Verarbeitung

Es wurden drei Algorithmen entwickelt, welche diese Automatisierung ermöglichen. Die einzige Einschränkung ist das manuelle Hinzufügen und Entfernen von Kategorien, was hingegen aber für eine Entlastung des Systems spricht.

Gewichtung

Mithilfe von Services von Wissensdatenbanken ist es möglich Aussagen automatisiert unscharf zu gewichten.

Services und Datenbanken nutzen

Bestehende Services und Datenbanken können durch generische Schnittstellen beliebig genutzt und ausgetauscht werden.

Erweiterte Medienontologie

Eine bestehende Medienontologie wurde erweitert bzw. strukturell minimal verändert. Für die Modellierung der Unschärfe wurden hierbei n-äre Relationen, eine Modellierung mit Hilfsknoten, welche die Relation selbst repräsentieren, gewählt und angewandt. Sie ist einfach umzusetzen und ebenso abzufragen.

Suche von unscharf gewichteten Daten

Mithilfe der gewählten Modellierung ist die Suche von gewichteten Daten problemlos. Es wurden außerdem typische Beispielanfragen vorgestellt, die allgemein gültig auf diese Ontologie anwendbar sind.

Reasoning

Ein unscharfes Reasoning wurde in dieser Arbeit nicht in dem Sinne betrachtet, dass es zur Zeit der Anfrage ausgeführt wird. Allerdings erhält man durch Vorberechnungen aller Zugehörigkeiten zwischen relevanten Kategorien selbst alle möglichen und nötigen Werte für eine umfassende Kategorisierung von Objekten. Diese Variante ist, wenn man davon ausgeht, dass die Menge der relevanten Kategorien sehr selten geändert wird, effizienter als eine Berechnung zur Zeit der Anfrage.

Kategorisierung

Die entwickelten Algorithmen decken jede Art der Kategorisierung ab. Sie können analog auf weitere Objekte der Topic/S-Ontologie angewendet werden.

Einfachheit

Sowohl die Modellierung der Daten mithilfe der n-ären Relation, als auch die Suche nach gewichteten Daten ist sehr einfach. Dies wurde durch das Nutzen von RDF Schema möglich, welches ausreichend Sprachmittel bei vergleichsweise wenig Komplexität mit sich bringt.

Standardkonformität

Durch das Nutzen empfohlener Standards des W3C kann bei der Modellierung der Daten auf zusätzliche Frameworks verzichtet werden. Es werden außerdem viele Funktionen des neuen SPARQL 1.1 Standards verwendet, wie z. B. SPARQL Update, was die Nutzung von bestehenden Triplestores und damit die Datenverwaltung erleichtert.

Performanz

Die Performanz des entwickelten Prototyps ist durch effiziente Algorithmen und vor allem die Verlagerung der Zugehörigkeitsbestimmung und -berechnung auf lokale Ebene sehr hoch. Ebenso sind Anfragen, durch den Verzicht auf Berechnungen während der Suche, sehr effizient.

Wiederverwendbarkeit

Das gesamte Konzept ist im Rahmen des Topic/S-Projektes wiederverwendbar. Dies wird durch definierte Schnittstellen für die Services und Triplestores ermöglicht.

Es konnte mit dieser Arbeit erfolgreich ein Konzept aufgestellt werden, welches beschreibt, wie man nur mithilfe von gut strukturierten Services Medienobjekte automatisiert kategorisieren kann. Bei dieser Bestimmung werden unscharfe Aussagen getroffen und deren Confidence-Werte berechnet. Hervorzuheben ist hier, dass hierfür Hierarchien von Kategorien einfließen und somit eine Einordnung sowohl in sehr spezielle als auch allgemeine Themengebiete vorgenommen wird. Dieser wissenschaftliche Nutzen ermöglicht es, noch genauere Aussagen über die Einordnung solcher Objekte in Kategorien zu treffen und zuverlässiger, z. B. im Rahmen des Topic/S-Projekts für das Verlagswesen, zu nutzen.
Mithilfe der in dieser Arbeit erreichten Ziele und umgesetzten Lösungen ist eine sehr gute Grundlage für weitere Arbeiten entstanden. Weitere Untersuchungen können beispielsweise zu den Aktualisierungen der Datensätze gemacht werden. Der bisherige Lösungsansatz sieht hierfür eine Neuberechnung aller Entitäten und Medienobjekte vor, wenn die Menge der relevanten Kategorien so geändert wird, dass eine Aktualisierung der Zugehörigkeiten im Datenbestand notwendig ist. Dies kann durch Einbeziehen von zeitlichen Informationen optimiert werden, z. B. wenn aktuell häufig benutzte Entitäten und Medienobjekte bei der Aktualisierung der Daten bevorzugt werden.

Des Weiteren ist es noch nicht möglich, über ein einfach zu bedienendes User Interface alle Funktionen des Prototypen zu nutzen. Da die Entwicklung einer guten Oberfläche ein sehr komplexes Thema darstellt, konnte sie im Rahmen dieser Arbeit nicht betrachtet werden. Als Alternative dazu wurden jedoch häufige SPARQL-Anfragen vorgestellt, die eine gute Grundlage bei der Entwicklung einer UI darstellt.
Aufgrund der komplexen Struktur und einer riesigen Menge von Kategorien und Zyklen in sehr großen Wissensontologien, wie der englischen DBpedia, war es bisher nicht möglich erfolgreiche Tests mit diesem Service durchzuführen. Daher müssen Untersuchungen angestellt werden, ob die Performanz bei solch großen Datenbanken erhalten bleibt. Sollte dies nicht zutreffen, müssen die bisherigen Lösungsansätze dementsprechend erweitert werden.
Es müssen außerdem Tests mit realen Daten unter realen Bedingungen durchgeführt werden. Da zum Testen oder für die Überlegungen des Konzepts keine realen Daten vorlagen, konnten keine bestehenden Metadaten eines Medienobjektes, wie vordefinierte Kategorien, berücksichtigt werden. Diese Daten könnten neben der automatisierten Bestimmung mithilfe der Services zur effektiven Kategorisierung beitragen.
Ein weiterhin komplexes Thema, welches im Rahmen dieser Arbeit nicht betrachtet werden konnte, ist das Erkennen von aktuellen Trends. Diese Problematik des »Hot Topic« setzt wiederum zeitliche Informationen voraus. Jedoch können mithilfe der unscharfen Aussagen ähnliche Medienobjekte bzw. Gruppierungen von ähnlichen Zugehörigkeiten zu relevanten Kategorien gefunden und damit Themen sehr genau definiert und zum Erkennen von Trends genutzt werden.

Literaturverzeichnis

[Hol09] Markus Holi. »Crisp, Fuzzy, and Probablistic Faceted Semantic Search«. Dissertation. School of Science und Technology, Aalto University, 2009 (siehe Seite 1).

[VMS12] Martin Voigt, Annett Mitschick und Jonas Schulz. »Yet Another Triple Store Benchmark? Practical Experiences with Real-World Data«. In: *Proceedings of the 2nd International Workshop on Semantic Digital Archives*. 2012 (siehe Seite 43).