

INSTITUT FÜR INFORMATIK

Universität Heidelberg

Einsatz einer Ontologie als Datenmodell zur Datenaggregation

Bachelorarbeit

24.03.2016

Benedikt Tröster

Matrikelnummer: 2897356

Betreuung: Frau Prof. Dr. Barbara Paech

Herr Dipl.-Inform.(FH) Christian Kücherer

Danksagungen

Ohne die hier genannten Personen wäre die Bachelorarbeit in dieser Form nicht möglich gewesen, deshalb geht an dieser Stelle besonderer Dank an:

Zunächst meinen Betreuer, Christian Kücherer, der mit hilfreichem Feedback und zahlreichen Tipps zur Gestaltung und zum Aufbau dieser Bachelorarbeit beigetragen hat.

Frau Prof. Dr. Barbara Paech, Michael Schaaf und Kais Tahar, die ebenfalls mit Feedback und Verbesserungsvorschlägen geholfen haben.

Außerdem danke ich meinen Eltern, Pamela und Helmut Tröster, für die Unterstützung bei der Rechtschreibkorrektur und Fehlersuche.

Meinem Chef, Matthias Luft, der mir in der Entstehungsphase dieser Arbeit den Rücken frei gehalten hat.

Und den Entwicklern von Ontop und Teiid für ihre Unterstützung bei der Einarbeitung in die Software.

Vielen Dank!

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die Bachelorarbeit mit dem Titel „Einsatz einer Ontologie als Datenmodell zur Datenaggregation“ selbstständig verfasst, noch nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel genutzt, wörtlich und sinngemäße Zitate als solche gekennzeichnet, sowie die „Sicherung guter wissenschaftlicher Praxis“ der Universität Heidelberg beachtet habe.

Heidelberg, den 24.03.2016

Benedikt Tröster

1 Inhaltsverzeichnis

Danksagungen	3
Selbstständigkeitserklärung	5
1 Einleitung.....	13
2 Grundlagen	15
2.1 Begriffe	15
2.1.1 Datenmodell	15
2.1.2 Datenaggregation	16
2.1.3 Datenbank	16
2.1.4 Virtuelle Datenbank.....	17
2.1.5 JDBC.....	17
2.1.6 Ontologie	17
2.1.7 Resource Description Framework (RDF)	20
2.1.8 Zuordnung (Mapping).....	20
2.1.9 Schlussfolgern (Reasoning).....	22
2.1.10 Web Ontology Language (OWL)	23
2.1.11 SPARQL	24
2.1.12 Ontology Based Data Access (OBDA)	25
2.1.13 Query Rewriting (Abfragen-Umschreiben).....	26
2.2 Datenintegration und deren Problematik.....	27
2.2.1 Problematik	27
2.2.2 Datenintegration	30
2.3 SNIK-Projekt.....	33
2.3.1 Das SNIK-Projekt und CION	33
2.3.2 Die CIOx-Ontologie	34
3 Literaturrecherche.....	37
3.1 OBDA Ansätze.....	46
3.2 Bewertungskriterien der Frameworks	52
3.3 Bestandsaufnahme gefundener Frameworks	53
3.3.1 Apache Jena.....	54
3.3.2 OBDA-Frameworks	55
3.3.3 Teiid	60
4 Planung und Gestaltung einer Lösung.....	62
4.1 Software Anforderungs Spezifikation (SAS)	62
4.1.1 Scope	62

4.1.2	Allgemeine Beschreibung (des Softwareprodukts)	62
4.1.3	Spezifische Anforderungen	63
4.1.4	externe Schnittstellen	65
4.1.5	Design-Beschränkungen	65
4.1.6	Sonstige Anforderungen.....	65
4.2	Ergebnisse der Betrachtung der OBDA Frameworks.....	68
4.3	Lösungskonzeption.....	74
4.4	Manuelle Lösung	74
5	Umsetzung und Qualitätssicherung	81
5.1	Umsetzung.....	81
5.1.1	Prototyp.....	82
5.1.2	Teiid	87
5.1.3	Protégé-Ontop-Bundle	89
5.2	Qualitätssicherung.....	92
5.2.1	Zusammenfassung der Testergebnisse	93
6	Fazit	97
6.1	Vorteile der Lösung	97
6.1.1	Prototyp.....	97
6.1.2	Teiid	97
6.1.3	Ontop-Protégé-Bundle	97
6.2	Aktuell ungelöste Schwierigkeiten	98
6.2.1	Prototyp.....	98
6.2.2	Teiid	98
6.2.3	Ontop-Protégé-Bundle	99
7	Zukünftige Arbeiten.....	100
8	Anhang.....	101
8.1	Programmcode der startReasoner-Methode.....	101
8.2	Programmcode der executeQuery-Methode.....	102
8.3	Programmcode der showQueryResult-Methode.....	103
8.4	Schritt für Schritt Anleitung.....	104
8.4.1	Voraussetzungen	104
8.4.2	Installation und Erst-Konfiguration	104
8.4.3	Inhalt der gepackten Version der Lösung.....	108
8.4.4	Überblick über die WildFly/Teiid Weboberfläche	108
8.4.5	Erstellen der Zuordnungen über Protégé.....	114

8.4.6	Hinzufügen/Ändern von Datenquellen in Teiid.....	117
9	Literaturverzeichnis	130
10	Abbildungs- und Tabellenverzeichnis.....	132

Abstract

Ontologien enthalten eine semantische Beschreibung von Konzepten und deren Relationen aus der realen Welt und liegen in maschinenlesbarer Form vor. Im Rahmen des DFG geförderten Projekts SNIK (Semantisches Netz des Informationsmanagement im Krankenhaus) wird eine Ontologie erstellt, die Aufgaben, Rollen und Objekttypen des Informationsmanagements abbildet.

In der IT Umgebung des Krankenhauses fallen unzählige Daten an, die für Entscheidungen des IT-Leiters oder auch Chief Information Officer (CIO) relevant sind. Um den CIO bei seinen Entscheidungen zu unterstützen, wird der CIO-Navigator (CION) im SNIK Projekt entwickelt. Der fertige CION unterstützt den CIO durch die Erstellung, Aggregation und Darstellung von relevanten Informationen aus der IT-Abteilung im Krankenhaus, und ermöglicht so das Treffen objektiver Entscheidungen. So entstand die Idee, die Ontologie als Datenmodell mit semantischer Beschreibung zur Datenaggregation zu verwenden. Dieser Ansatz wird in der wissenschaftlichen Literatur als Ontology Based Data Access (OBDA) bezeichnet und lässt sich unter anderem in Frameworks wiederfinden.

Diese Bachelorarbeit zeigt den aktuellen Forschungsstand des Einsatzes einer Ontologie als Datenmodell und zum Zugriff auf Daten im Bereich OBDA. Verschiedene Ansätze und Frameworks werden gegenübergestellt und hinsichtlich der Anforderungen an eine an das Projekt angepasste Lösung, in Form eines Prototyps, verglichen.

Der in dieser Arbeit entstandene Prototyp zeigt auf, wie die vorhandene Ontologie mit den verschiedenen Datenquellen (z.B. Excel) verknüpft und so als eine semantische Beschreibung der Daten verwendet werden kann. Für die Entwicklung werden dessen Anforderungen erhoben und dann im Prototyp umgesetzt. Der Prototyp baut auf einem in der Literaturrecherche evaluierten Framework namens Ontop auf und greift über dieses auf die Daten zu. Die Daten werden über einen Föderationsserver (Teiid) als eine virtuelle Datenbank an Ontop exponiert. Um sicher zu stellen, dass der Prototyp den Anforderungen gerecht wird, werden manuelle- und Unit-Tests eingesetzt. Zur Dokumentation der Anforderungen wird UNICASE verwendet.

1 Einleitung

Diese Bachelorarbeit befasst sich mit der Fragestellung, in welcher Form Ontologien als semantische Datenmodellbeschreibung und zum Zugriff auf Datenquellen verwendet werden können. Ontologien haben ihren Ursprung in der Philosophie und werden in der Informatik dazu verwendet, Begriffe und deren Beziehungen in einer bestimmten Domäne abzubilden [1]. Die Modellierung der Informationen erfolgt durch eine festgelegte Syntax, durch die ein Computer in der Lage ist, die Daten zu verarbeiten. Eine Verarbeitung der Daten kann dazu eingesetzt werden, um Inhalte der Ontologie zu interpretieren oder aus dieser Schlussfolgerungen zu ziehen (siehe Kapitel Schlussfolgern (Reasoning)). Reasoning ermöglicht das Extrahieren von implizierten Informationen aus einer Ontologie, die sich aus modellierten Zusammenhängen und Aussagen ergeben. Für diese Arbeit wichtig ist zudem die Fragestellung, wie eine Verwendung einer Ontologie als ein gemeinsames Datenmodell der Datenquellen umgesetzt werden kann. Konkret geht es in dieser Arbeit um den Einsatz einer vorhandenen Ontologie, die innerhalb des DFG-Projektes „Semantisches Netz des Informationsmanagements im Krankenhaus“ (SNIK) entsteht. Im Rahmen des Projekts entsteht eine Ontologie, die die Aufgaben des Chief Information Officer (CIO) enthält, der/die für das Informationsmanagement und Projektmanagement im Krankenhaus zuständig ist. Der CIO muss mit den in der IT Umgebung des Krankenhauses anfallenden Informationen projektrelevante Entscheidungen treffen. Die Datenquellen, aus denen die Informationen kommen (wie z.B. eine Projektwarteliste in Form einer Excel-Datei), sind heterogen und semantisch weder beschrieben, noch miteinander verknüpft. Das Zusammenführen der Daten sowie Anwenden des Wissens über die Daten ist eine manuelle Aufgabe des CIO, die durch eine Software erleichtert werden soll.

Im engen Zusammenhang mit dem hier zusammengefassten Weg zur Lösung der beschriebenen Aufgabenstellung steht der Begriff des Ontology Based Data Access (OBDA) [2]. Er beschreibt den Prozess des Zugriffs auf Daten über eine Ontologie als Datenmodell: Das Auslesen von Daten liefert keine Information dazu, in welchem semantischen Zusammenhang die bezogenen Daten (z.B. aus einer Datenbank) mit anderen Daten stehen. So ist es erst durch die Nutzung einer Ontologie möglich, deren zusätzliche Semantik mit den Daten zu verknüpfen.

Ziel dieser Arbeit ist es durch den Einsatz einer Ontologie als Datenmodell die verschiedenen Daten aus unterschiedlichen, heterogenen Datenquellen semantisch zu beschreiben und zu aggregieren. Die Ontologie soll also ein gemeinsames Datenmodell der verschiedenen Daten sein, welches ein semantisches „Verständnis“ der Daten mit sich bringt. Dies ist wichtig, da der semantische Zusammenhang der Daten dem/der NutzerIn eine weitere Ebene der Information bringt. Durch das gemeinsame Datenmodell der Ontologie lassen sich außerdem die verschiedenen Datenquellen zentral zusammenführen, wodurch der manuelle Schritt des Betrachtens und Zusammenfügens von verschiedenen Daten eingespart wird. Die Daten sind zwar vorhanden (z.B. in Form einer Zelle in einer Excel Mappe), was aber fehlt sind Informationen darüber, was diese Daten repräsentieren und in welchem Zusammenhang sie mit anderen Daten stehen. Genau das soll eine Software erfüllen, um so Entscheidungsprozesse im Umfeld des CIO zu erleichtern. Dazu trägt vor allem bei, dass es mit der neuen Software möglich wird, durch schlichtes Zuordnen von Datenquellen zu Elementen der Ontologie komplexe Zusammenhänge, die in der Ontologie beschrieben sind, mit den tatsächlichen Daten zu kombinieren.

Im Rahmen der Arbeit wird hierzu eine Lösung in Form eines Software-Prototyps entwickelt, welcher OBDA-Konzepte einsetzt, um über eine gegebene Ontologie auf verschiedene Datenquellen (wie z.B. Excel und Word) zuzugreifen. Auf diese Weise wird die Lücke zwischen den heterogenen Datenquellen und deren jeweiligem semantischem Verständnis aus der Ontologie geschlossen.

Die vorliegende Arbeit ist dabei wie folgt strukturiert: Kapitel 3 beschreibt die durchgeführte Literaturrecherche zur Begutachtung des aktuellen Forschungsstandes, sowie eine Analyse und Suche verfügbarer OBDA Frameworks. Dabei wird auch die Frage beantwortet, welche OBDA Ansätze es gibt und wie diese umgesetzt sind. Das Kapitel Planung und Gestaltung einer Lösung dokumentiert die an den Prototyp gestellten Anforderungen und dessen Aufbau. Die Anforderungen spielen eine primäre Rolle bei der Planung und Entwicklung des Prototyps. Diese werden für die Bewertung der Frameworks, in Bezug auf deren Eignung zur Umsetzung des Prototyps, herangezogen. Unter der Berücksichtigung der Ergebnisse des ersten Teils wird der Prozess beschrieben, und die für die Gestaltung der Software getroffenen Entscheidungen werden begründet und diskutiert. Außerdem wird eine Beispieldlösung entwickelt an der sich der Prototyp orientiert. Kapitel 5 beschreibt die tatsächliche Umsetzung des gewählten Ansatzes in einem Prototyp, sowie die notwendige Qualitätssicherung desselben. Für die Umsetzung und Qualitätssicherung liegt der Fokus auf der Beschreibung und Dokumentation der Entwicklung und besonders der Qualitätssicherung. Nach der Planung der Softwaretests, welche sicherstellen, dass die Anforderungen erfüllt sind, werden diese erstellt und anschließend durchgeführt. Bei der Darstellung der entsprechenden Ergebnisse werden die Existenz und der Zweck der Tests begründet. Abschnitt 6 fasst dann die Vor- und Nachteile der Lösung, inklusive des Prototyps, zusammen und bewertet diese. Der letzte Abschnitt (7) gibt einen Ausblick für zukünftige Verbesserungen. Dazu werden mögliche Ansätze, mit denen sich der Prototyp schließlich in eine vollwertige Software umwandeln lässt, besprochen. Aber auch Verbesserungsmöglichkeiten bezüglich des OBDA Ansatzes allgemein und den Prozessen rund um den Prototyp werden thematisiert.

2 Grundlagen

Dieses Kapitel beschreibt wichtige Grundlagen und Begriffe, die in dieser Bachelor Arbeit verwendet werden und für das Verständnis benötigt werden. Im Unterkapitel „Begriffe“ werden allgemeine Themen wie z.B. „Datenbank“ erklärt. Dann folgt eine Beschreibung der Begriffe Ontologie und verwandter Themen zu OBDA. Das Unterkapitel „Datenintegration und deren Problematik“ beschreibt Probleme, die aus dem Umfeld der Datenintegration bekannt sind, da es Ähnlichkeiten zu den Schwierigkeiten im Umgang mit OBDA gibt. Im Unterkapitel „SNIK-Projekt“ wird abschließend das SNIK-Projekt kurz vorgestellt.

2.1 Begriffe

2.1.1 Datenmodell

Das Datenmodell einer Datei beschreibt deren Aufbau. Laut Sommerville [3] werden Datenmodelle sehr ähnlich zu Klassenmodellen dargestellt. In ihnen werden die Zusammenhänge und Attribute der Daten Entitäten modelliert. Die Beschreibung kann hierbei über UML stattfinden: Die Datenentitäten werden als Objekt-Klassen, die Attribute als Objekt-Klassen-Attribute und die Relationen zwischen ihnen als Assoziationen umgesetzt. Zum besseren Verständnis ein Beispiel Datenmodell in UML:

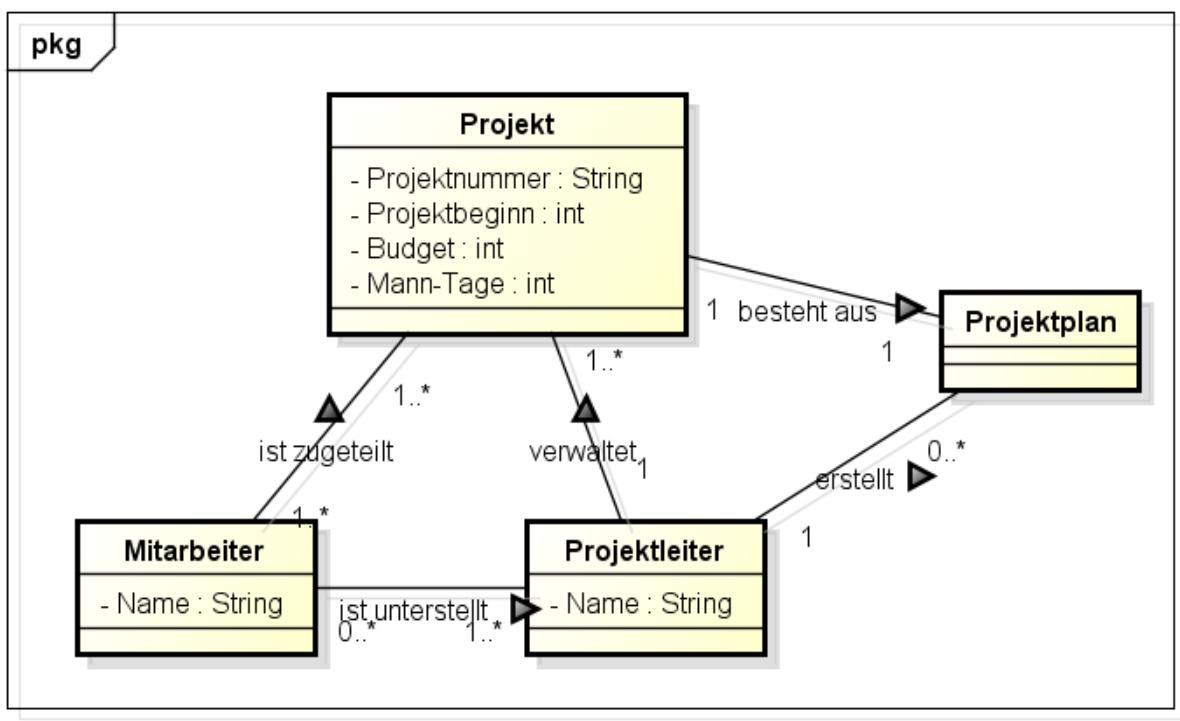


Abbildung 1 Beispiel Vereinfachtes Datenmodell

Im Zusammenhang mit Datenmodellen, wird (falls benötigt) auch die physische Repräsentation der Daten in einem sogenannten „physischen Datenmodell“ modelliert. Der physische Aufbau beschreibt, **wie** die Daten in eine Datei geschrieben werden. Hierbei spielt vor allem die technische Umgebung der Datei eine Rolle. Außerdem beeinflusst der physische Aufbau, wie die Datei am effizientesten ausgelesen werden kann.

Abbildung 2 zeigt den Prozess, der in die Modellierung einer Datei fließt, schematisch auf. Dieser Prozess wird im Allgemeinen dazu verwendet, um Daten mit dem Datenmodell auszustatten, welches der jeweiligen Situation angepasst ist.

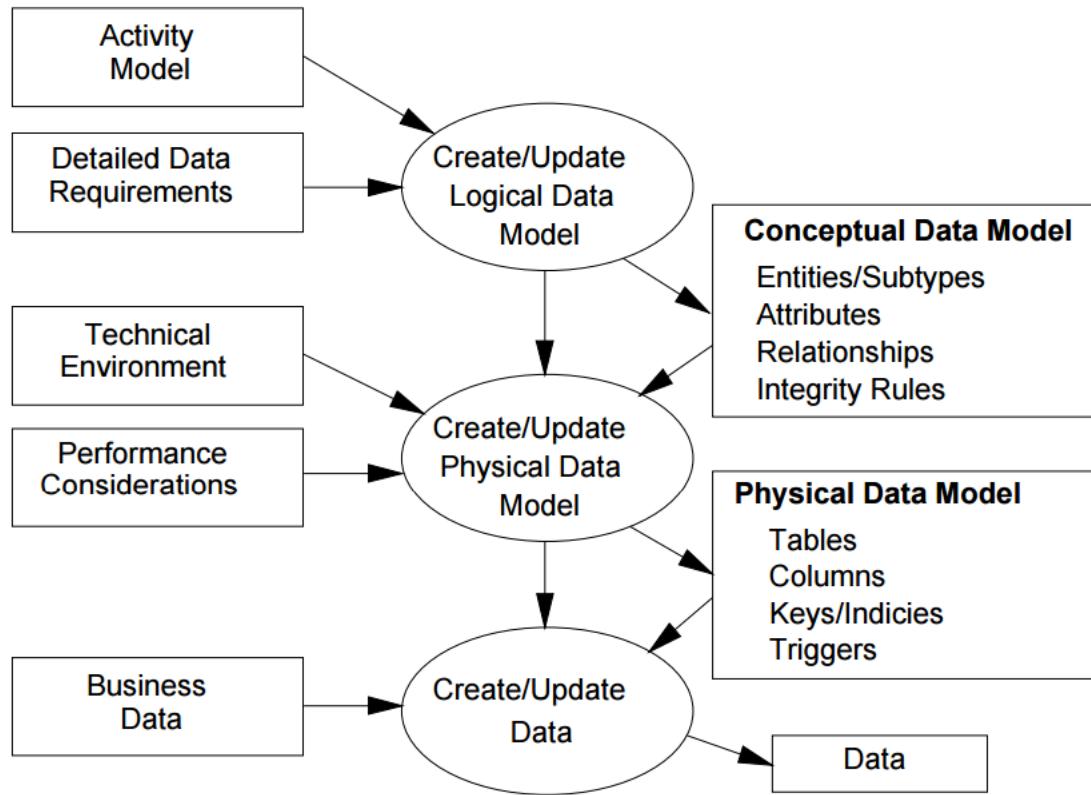


Abbildung 2 Datenmodellierungsprozess. Quelle: [4]

2.1.2 Datenaggregation

Der Begriff der Datenaggregation beschreibt den Vorgang des Zusammenföhrens und Ordnens von Daten. Das kann virtuell, aber auch physisch passieren. Virtuell bedeutet, dass die Ursprungsdaten nicht verändert werden, sondern z.B. nur zur Laufzeit in aggregierter Form vorhanden sind. Physisch würde bedeuten, dass eine tatsächliche Veränderung der Daten stattfindet.

2.1.3 Datenbank

Eine Datenbank stellt eine in einer bestimmten Art und Weise geordnete Ansammlung von Daten dar. Eine Datenbank zeichnet sich durch ein Schema aus, welches definiert, wie und in welcher Form die Daten in Tabellen abgelegt werden können. Eine Tabelle beinhaltet Spalten, in denen alle einzelnen Daten eines bestimmten Typs vorhanden sind (z.B. Spalte „Name“). Außerdem gibt es Zeilen, die wiederum einen zusammenhängenden Datensatz (z.B. 1,Luke,20.1.2000) enthalten. Eine Zeile ist wiederum aus einzelnen Daten (Felder oder Zeile X Spalte) aufgebaut (z.B. „Leia“). Tabelle 1 zeigt eine Beispiel-Tabelle.

Tabelle 1 Tabellen-Beispiel

ID	Name	Geburtstag
1	Luke	20.1.2000
2	Leia	13.2.1990

Das Beziehen der Daten erfolgt über Abfragen (Queries) an ein Datenbank-Management System (DBMS) in einer herstellerspezifischen Abfragesprache. Bei den meisten heutigen DBMSs wird dazu SQL verwendet. Im Kontext von Datenbanken wird meist von relationalen Datenbanken gesprochen.

2.1.4 Virtuelle Datenbank

Im Kontext dieser Arbeit wird von virtuellen Datenbanken (VDBs) gesprochen. Diese unterscheiden sich in ihrer Funktionalität nur kaum von einer klassischen relationalen Datenbank. Der Unterschied liegt darin, dass es sich bei der Datenbank nicht um eine echte und persistente Datenbank handelt, sondern diese nur in einer Software existiert und wohl möglich von Daten abgeleitet wird, die in ihrem Aufbau keiner relationalen Datenbank entsprechen. Eine VDB wird also durch Abstraktion von vorhandenen Daten erzeugt.

2.1.5 JDBC

JDBC¹ ist eine Abkürzung für „Java Database Connectivity“. Bei JDBC handelt es sich um eine Java-Schnittstelle, die für die Kommunikation mit relationalen Datenbank-Systemen verschiedenster Hersteller verwendet wird. Hersteller von DBMS bringen produktsspezifische JDBC-Treiber für ihr DBMS mit, um eine standardkonforme Kommunikation zwischen JDBC-Schnittstelle und Datenbank zu ermöglichen. Java Applikationen greifen über diese JDBC Schnittstelle und den passenden Treiber auf Datenbanken zu.

2.1.6 Ontologie

Ursprünglich ist der Begriff der Ontologie (wie vieles aus der Mathematik und Informatik) der Philosophie entnommen. Dort handelt es sich um ein Teilgebiet der theoretischen Philosophie, die sich mit der Kategorisierung von Entitäten (z.B. abstrakte aber auch konkrete Dinge) und deren Beziehungen befasst. Nach Aussage von Guarino und Giaretta [5], sind Ontologien in der Informatik klar definiert. Die Ontologie wird demnach als eine formale Art der Wissensdarstellung verstanden. Wissen ist hier nicht zwangsläufig als Allgemeinwissen, sondern auch als Wissen über spezielle Vorgänge zu verstehen. Wie auch in der Philosophie werden hier Entitäten und deren Beziehungen zueinander in einem Netzwerk dargestellt. Das Besondere an Ontologien ist, dass diese Axiome enthalten, die es möglich machen, Schlussfolgerungen aus den repräsentierten Informationen zu ziehen. Dies wird im Abschnitt „Schlussfolgern (Reasoning)“ noch etwas genauer erläutert. Auf Grund der Möglichkeit des Reasonings über vorhandenes Wissen, sind Ontologien im Web der Daten²

¹ <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>

² <https://www.w3.org/2013/data/>

(ehemals „semantisches Web³“) von großer Bedeutung. Ontologien bestehen meist aus den folgenden Bestandteilen:

- **Klassen**, welche Dinge oder Mengen von Objekten bzw. Instanzen sind. Zum Beispiel eine Stadt.
- **Instanzen** oder Objekte, welche die konkreten Elemente der Klassen sind. Beispiel: Heidelberg (als eine Instanz einer Stadt).
- **Eigenschaften**, die Parameter, Eigenschaften oder Attribute von Objekten oder Klassen sein können.
- **Ereignisse**, die Veränderungen von Eigenschaften oder Relationen sind.
- **Relationen**, die bestimmen, wie Klassen und Instanzen miteinander und untereinander in Verbindung stehen.
- **Axiome**, welche wie in der Mathematik auch, allgemeingültige, oft globale Regeln darstellen.

Zur besseren Anschauung der Elemente einer Ontologie, hier ein Auszug aus der Ontologie des SNIK-Projekts für den Bereich Projektmanagement (Abbildung 3):

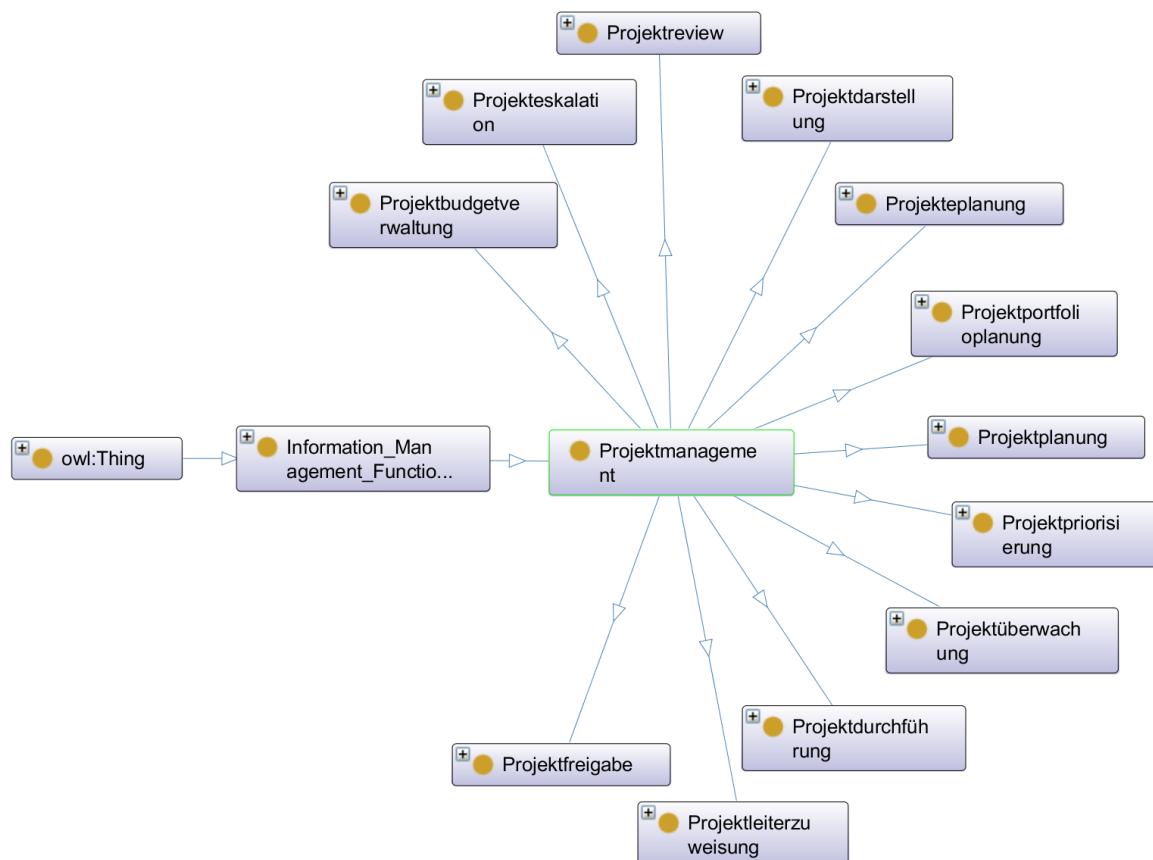


Abbildung 3 Ausschnitt "Projektmanagement" der CIOx Ontologie des SNIK-Projekts
(OntoGraf Export)

³ <https://www.w3.org/2001/sw/>

Die in Abbildung 3 in Gelb dargestellten Elemente sind Klassen, die über Relationen (is-a) miteinander verknüpft sind. Die Ontologie des SNIK-Projekts folgt einem Meta-Modell, welches genauer im Abschnitt „Die CIOx-Ontologie“ (2.3.2) beschrieben wird. Die Relation (is a) ist im Kontext dieser Ontologie als „ist eine Aufgabe von“ zu lesen. Beispiel: „Projektfreigabe ist eine Aufgabe des Projektmanagements“.

Die nachfolgende Abbildung 4 zeigt die Elemente Instanz (siehe Projektpriorität „hoch“, „mittel“ und „gering“) auf:

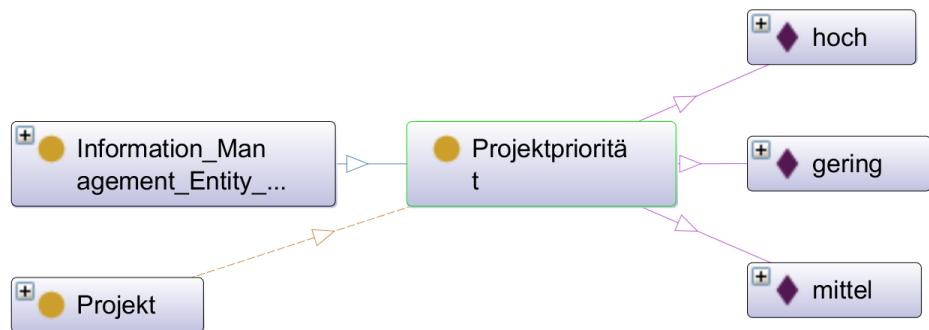


Abbildung 4 Ausschnitt Projektmanagement der CIOx-Ontologie mit Fokus auf "Projektpriorität" (OntoGraf Export)

Jeder dieser Bestandteile Klasse, Objekt, usw. soll in einer für die Beschreibung der Ontologie verwendeten Sprache vorhanden sein, um garantieren zu können, dass auch das Modellieren von komplexen Zusammenhängen möglich ist. Ontologien können deshalb verwendet werden um semantische Zusammenhänge zu beschreiben. Es gibt verschiedene Beschreibungssprachen, um Inhalte in Ontologien zu beschreiben. Der de-facto Standard zur Beschreibung ist die „Web Ontology Language (OWL)“, es gibt aber eine Vielzahl anderer Sprachen zur Beschreibung. Persistiert werden Ontologien meist im XML-Format. Je nach Beschreibungssprache kann die Struktur der XML-Datei allerdings variieren. Es gibt einige offene Frameworks zum Bearbeiten und Lesen von Ontologien. Ein prominentes Beispiel für ein solches Framework ist Apache Jena⁴, das im Bereich des semantischen Webs häufig verwendet wird und in Java implementiert ist.

Auch komplette Werkzeuge zur Bearbeitung von Ontologien sind verfügbar. Ein gängiges Programm zum Verarbeiten von Ontologien ist Protégé⁵, welches auch eine grafische Oberfläche bereitstellt und somit nicht nur zum Bearbeiten eingesetzt wird sondern auch beim Betrachten unterstützt.

Sowohl Apache Jena als auch Protégé ermöglichen das Abfragen von Informationen aus der Ontologie über eine eigene Abfragesprache (SPARQL), welche in ihrer Syntax SQL ähnelt.

Zur Vervollständigung ist anzumerken, dass es einige öffentliche Ontologien gibt, die Wissen aus verschiedensten Bereichen abbilden. Dazu zählen hauptsächlich medizinische und biologische, aber

⁴ <https://jena.apache.org/>

⁵ <http://Protégé.stanford.edu/>

auch literarische Bereiche der Forschung. Einen Einblick gewährt die Bibliothek in der Wiki-Seite von Protégé⁶.

2.1.7 Resource Description Framework (RDF)

Beim Resource Description Framework, welches von der RDF Arbeitsgruppe des World Wide Web Consortium (W3C)⁷ entwickelt wurde, handelt es sich um ein Standard-Modell für Datenaustausch im Web. Genauer wird es heute dazu eingesetzt, um logische Aussagen über Dinge in einer bestimmten Form zu dokumentieren. Die Form lässt sich dabei als ein gerichteter Graph beschreiben. Dieser entsteht durch das Zusammenfügen der einzelnen Tripel. Jedes Tripel ist eine Aussage in der Form: Subjekt, Prädikat, Objekt. Hierdurch können Informationen dargestellt werden.

Tabelle 2 Beispiele für valide Tripel:

Subjekt	Prädikat	Objekt
Vater	hat	Kind
Vater	ist	Männlich
Kind	hat	Vater

Subjekt und Prädikat stellen Ressourcen dar, die eindeutig sind. Zur Identifikation von RDF-Ressourcen werden diese mittels URIs bezeichnet, die auch in Ontologien vorkommen. URIs sind ähnlich zu URLs in ihrem Aufbau, stellen aber nicht zwangsläufig einen aus dem Web-Umfeld bekannten „Link“ z.B. zu einer Webseite oder allgemein einer Netzwerkressource dar. Durch die Nutzung von URIs sind Ressourcen eindeutig (auch über die aktuell betrachtete Datei hinaus) identifiziert und können ggf. mit anderen RDF-Daten zusammengefasst werden, wenn auf gleiche URLs verwiesen wird. Anders als Subjekt und Prädikat verhält sich das Objekt, welches in einer Aussage referenziert wird: Dabei kann es sich um eine Ressource oder einfach nur ein Attribut in der Form eines Strings oder anderen Daten (Zahlen, Datum, etc.) handeln.

Wie jedes Format muss auch RDF in einer bestimmten Form als Datei gespeichert werden. Hierzu wird unter anderem XML verwendet, um die Informationen kodiert abspeichern zu können. Auf Grund des Aufbaus von RDF-Graphen bietet sich das Speichern in Form einer relationalen Datenbank an. Da das reine Ablegen der Tripel in einer relationalen Tabelle aber nicht für effiziente Abfragen geeignet ist, werden RDF-Daten meist in einem sog. „Triplestore“ gespeichert. Ein Triplestore ist eine spezielle Datenbank, die auf das Speichern von Tripeln ausgelegt ist und so schnelleren Datenzugriff über Abfragen erlaubt.

2.1.8 Zuordnung (Mapping)

Generell bezeichnet eine Zuordnung im Kontext dieser Arbeit immer eine Verbindung von einem Element der Ontologie zu Daten. Diese Zuordnung kann z.B. zwischen einer Instanz einer Klasse einer Ontologie und einer Datenquelle erfolgen, aber auch zwischen mehreren Ontologien. Zuordnungen

⁶ http://Protégéwiki.stanford.edu/wiki/Protégé_Ontology_Library

⁷ <http://www.w3.org/>

können dann als eine Art Referenz verstanden werden. Bei der Zuordnung von zwei Ontologien geht es aber auch darum, die semantischen Konzepte der Beiden Ontologien miteinander vergleichbar zu machen, dazu eine Erklärung aus [6]:

Eine Ontologie sei ein Paar $O \equiv (C, A)$, sei C eine Menge von Konzepten und A eine Menge von Axiomen. Um zwei Ontologien C_1 und C_2 miteinander zu verknüpfen, brauchen wir eine Abbildung $f: C_1 \rightarrow C_2$, die die beiden semantischen Konzepte in Relation stellt. Und zwar so, dass die Axiome von C_1 und C_2 nicht verletzt werden. Es kann auch partielle Zuordnungen geben, welche eine Zuordnung zwischen einer Teil-Ontologie von C_1 und der Ontologie C_2 darstellen.

Für Zuordnungen zwischen Ontologien und relationalen Datenquellen gibt es den vom W3C verabschiedeten Standard R2RML⁸. Mit Hilfe von R2RML und anderen Zuordnungssprachen können relationale Datenquellen zu RDF-Datenmengen zugeordnet werden. Eine solche Zuordnung ist auf einer schematischen Ebene wie folgt dargestellt:

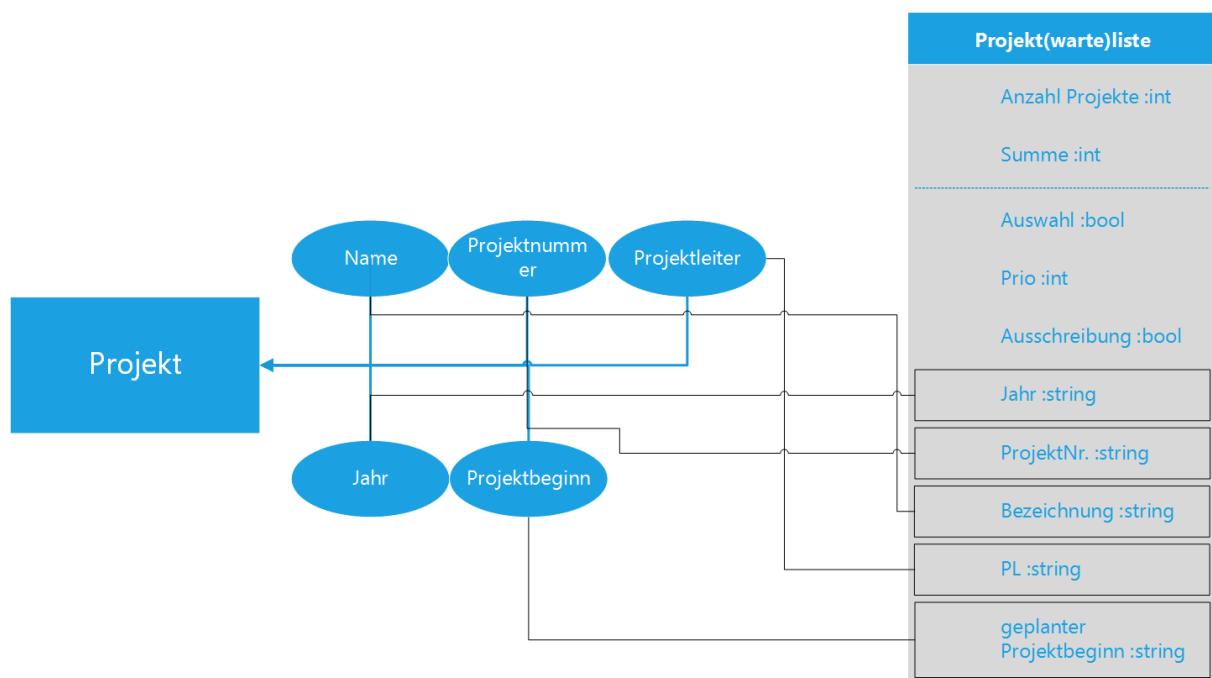


Abbildung 5 Beispiel Zuordnung zwischen Ontologie-Element "Projekt" und dessen Eigenschaften mit der Datenquelle "Projekt(warte)liste"

Die in Abbildung 5 dargestellte Zuordnung geschieht von der Datenquelle „Projektwarteliste“ auf das Ontologie-Element „Projekt“. Die Projekt(warte)liste ist in Abbildung 6 als SQL-Datenquelle eingebunden. Die Blauen Pfeile zeigen an, dass die Eigenschaften Name, Jahr, Projektbeginn, usw. zu Projekt gehören. Hingegen zeigen die grauen Pfeile die Zuordnung zwischen der Datenquelle und der Ontologie auf.

Ein Beispiel für eine Zuordnung zwischen der CIOx-Ontologie des SNIK Projekts und einer SQL-Datenquelle, die in dieser Arbeit verwendet werden, sieht wie folgt aus:

⁸ <http://www.w3.org/TR/r2rml/>

```

target          :Projekt/{PN}/ a :Projekt ; :Projektnummer {PN} ; :Name
{Beschreibung} ; :Projektbeginn {Beginn} ; :Jahr
{Jahr} ; :isAssociatedWith :Controllingliste_Invest/{PN}/ , :Controllingl
iste_Budget/{PN}/ , :Projektpriorität/{Prio}/ , :Projektleiter/{PL}/ .

source         select "ProjektNr" as PN, "Beschreibung und Bemerkung"
as Beschreibung, "PL" , "geplanter Projektbeginn" as Beginn, "Prio",
"Jahr" from "projLst"."Tabelle1"

```

Abbildung 6 Zuordnung zwischen CIOx-Ontologie und SQL-Datenquelle in OBDA-Syntax

Bei der Syntax handelt es sich um die Syntax eines betrachteten Frameworks (Ontop).

Die unter „Target“ angegebene Information repräsentiert die zu erstellende Struktur der Ontologie für die Klasse „Projekt“. Die nach dem „::“ folgenden Elemente sind Klassen der Ontologie. Jede in „Target“ angegebene Information ist durch „;“ vom folgenden Teil getrennt. Der Text innerhalb der geschweiften Klammern (z.B. „{PN}“) markiert jeweils die Stelle, die mit den Daten aus der SQL-Abfrage befüllt wird. Die Bezeichnung (PN) innerhalb der Klammern und in der SQL-Abfrage müssen hierfür identisch sein.

Die erste Information (:Projekt/{PN}/ a :Projekt ; :Projektnummer {PN} ;) gibt zunächst eine eindeutige Bezeichnung (in diesem Fall: :Projekt/{PN}/) für das in der Ontologie virtuell zu erzeugende Element an. Zusammen mit dem darauf folgenden a und dem angegebenen Typ („:Projekt“) ergibt sich der folgende Datensatz für ein Projekt:

<<http://example.org/Projekt/2007-08-001>> rdf:type :Projekt.

Die nachfolgenden Eigenschaften (:Name) und Relationen (:isAssociatedWith) sind in der gleichen Art und Weise angegeben.

Eine detailliertere Erklärung der Zuordnungssyntax, die von der in der Arbeit erzeugten Lösung verwendet wird, ist im Anhang in Abschnitt 8.4.5 zu finden.

2.1.9 Schlussfolgern (Reasoning)

In einer Ontologie sind Beziehungen und Regeln hinterlegt. Nicht immer ist eine sich aus den Eigenschaften der Ontologie ergebende Konsequenz explizit angegeben. Implizit angegebene Eigenschaften müssen aus den Kombinationen vorhandener Eigenschaften erschlossen werden. Der Vorgang des Erschließens dieser Eigenschaften wird „Reasoning“ genannt. Um das zu veranschaulichen, hier ein Beispiel:

In einer Ontologie ist folgende Beziehung hinterlegt (in beispielhafter Syntax):

Vater = **Mann** AND **hatEin**(**Kind**)
Mann = **Mensch** AND **ist**(**Männlich**)

Wobei orange eingefärbte Elemente eine Klasse, lila eingefärbte Elemente eine Instanz und blau eingefärbte Elemente eine Relation darstellen. Die gleiche Farbkodierung gilt auch für Abbildung 7. In der Abbildung sind die Relationen (Pfeile) zusätzlich mit verschiedenen Farben eingezeichnet:

Pink für „has“ (Instanz), **Blau** für „has“ (Klasse), **Gelb** für „hatEin“ (Klasse) und **Rot** für „ist“ (Klasse).

Betrachten wir die Instanz vom Typ `Vater` mit dem Namen „Bob“, die der Relation `hatEin()` ein Kind „Junior“ zugewiesen hat.

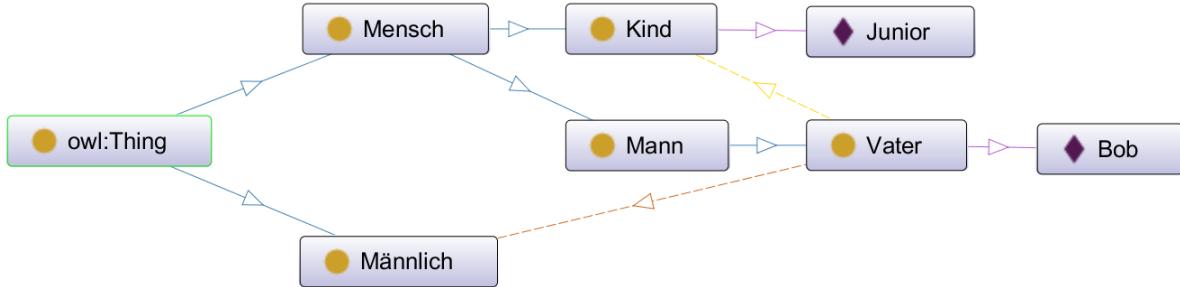


Abbildung 7 Schema Beispiel-Ontologie

Stellt man nun eine Abfrage an den Reasoner kann dieser aus der gegebenen Information mit logischem Reasoning generelle Fragen beantworten. Eine Abfrage an den Reasoner kann z.B. die Frage „Welche Menschen haben Kinder?“ sein. Diese kann beantwortet werden, wenn alle Subklassen und deren Subklassen von „Mensch“ betrachtet und diese nach der Relation „hatEin“ auf die Klasse „Kind“ durchsucht werden. Das Ergebnis der Abfrage ist in diesem Fall die Klasse „Vater“. Falls die Ontologie größer modelliert wäre, würde z.B. auch „Mutter“ zurückgeliefert werden.

Jede Beschreibungssprache der Ontologie verlangt einen passenden Reasoner, da der Reasoner von der Syntax der Sprache abhängig ist und sie verarbeiten müssen.

2.1.10 Web Ontology Language (OWL)

OWL⁹ ist eine Sprache zur Beschreibung von Ontologien. Die im SNIK Projekt verwendete Ontologie ist in OWL beschrieben. Aus diesem Grund wird Web Ontology Language (OWL) und dessen Bedeutung genauer erklärt.

OWL ist nicht genau eine Sprache, sondern eine Familie von Sprachen, die in ihrer Art sehr ähnlich zueinander sind. Unterschiede gibt es vor allem dahingehend, wie ausdrucksstark die Sprachen sind und weshalb sie für verschiedene Einsatzzwecke besser geeignet sein können.

Ab OWL 2 werden die Untersprachen als Sprach-Profile bezeichnet, die sich auch in ihrem Einsatzziel unterscheiden¹⁰:

- OWL 2 EL: Dieses Profil wird häufig für Ontologien verwendet, die eine große Zahl an Eigenschaften und Klassen besitzen. Die Ausdrucksmöglichkeiten sind an solche Ontologien angepasst und Reasoning ist möglich.
- OWL 2 QL: Das Profil ist darauf ausgelegt, mit großen Mengen von Instanzen umzugehen und die Abfragezeit so gering wie möglich zu halten.

⁹ <https://www.w3.org/TR/owl2-overview/>

¹⁰ <https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/#Introduction>

- OWL 2 DL: Ist darauf ausgelegt, die komplette Ausdrucksstärke von OWL 2 für bessere Leistung zu tauschen, was für große Ontologien hilfreich ist.

Im Vergleich zu RDF ist OWL eine mächtigere Sprache dahingehend, dass sie z.B. nicht nur die von RDF gegebene Syntax bereitstellt, sondern auch komplexere Konzepte wie Vereinigungen und logische Verknüpfungen zwischen Objekten. OWL ist daher als eine Untermenge von RDF zu verstehen. Außerdem ist es möglich, durch die Auswahl einer bestimmten OWL Sub-Sprache zu beeinflussen, wie hoch der Rechenaufwand für Abfragen oder Reasoning ist. Man tauscht auf diese Art Teile der Beschreibungsfreiheit gegen eine höhere Geschwindigkeit beim Berechnen ein. Ein Reasoner muss deshalb mit der von der Sprache bereitgestellten Syntax kompatibel sein. Zur Abfrage von Informationen aus Ontologien kann (sofern vom Reasoner unterstützt) SPARQL verwendet werden.

2.1.11 SPARQL

Bei SPARQL handelt es sich um eine vom W3C¹¹ entwickelte Abfragesprache zum Zugriff auf RDF- und OWL-Daten. Die Syntax ist ähnlich zu SQL, was eine Konvertierung von SPARQL nach SQL ermöglicht.

Ein Beispiel zur Anschauung:

Gegeben sei folgendes Datentripel im RDF-Format:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title>
"SPARQL Tutorial" .
```

Das Tripel ist dabei wie folgt zu verstehen:

Subjekt	Prädikat	Objekt
Buch [book1]	Titel [title]	Daten ["SPARQL Tutorial"]

Die in SPARQL Syntax verfasste Abfrage zur Extraktion des Elements „title“ sieht wie folgt aus:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> ?title .
```

Die Bearbeitung dieser Anfrage durch den Reasoner führt zu folgendem Ergebnis:

title
"SPARQL Tutorial"

Dieser Datensatz kann je nach Wunsch in den folgenden maschinenlesbaren Formaten ausgegeben werden:

- Extensible Markup Language (XML¹²)

¹¹ <http://www.w3.org/>

- JavaScript Object Notation (JSON¹³)
- Comma Separated Values (CSV¹⁴)
- Tab Separated Values (TSV¹⁵)

Durch den UPDATE Befehl ist es zudem möglich, wie auch in einer Datenbank, Daten zu modifizieren.

2.1.12 Ontology Based Data Access (OBDA)

Der Begriff des Ontologie-basierten Datenzugriffs, engl. Ontology Based Data Access, (OBDA) bezeichnet Vorgehensweisen zum Zugriff auf Daten über Ontologien.

Bei OBDA [2] geht es darum, durch die Verwendung einer Ontologie ihr zugeordnete Daten semantisch zu beschreiben und auf diese zu zugreifen. Der prinzipielle Aufbau eines OBDA Systems in einer vereinfachten Darstellung ist in Abbildung 8 dargestellt.

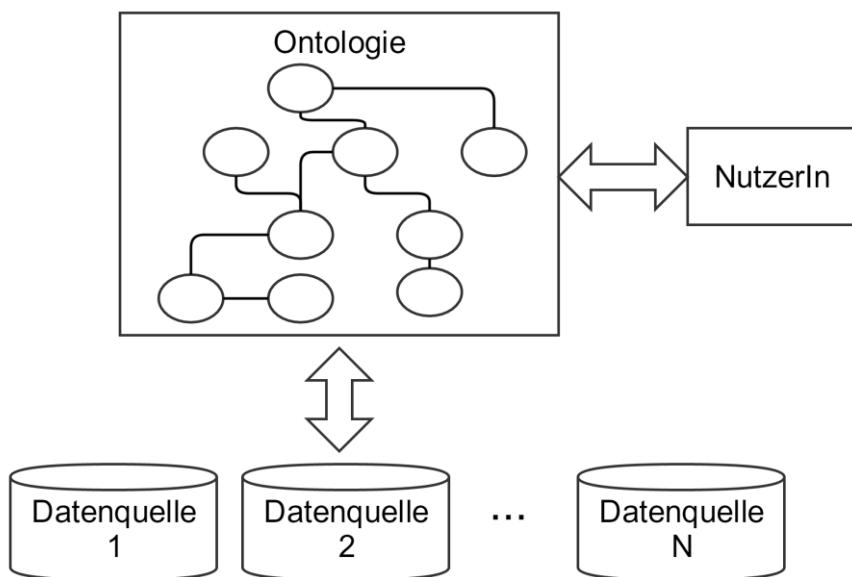


Abbildung 8 Schematischer Aufbau OBDA

Der generelle Aufbau eines OBDA Systems ist wie folgt:

Es gibt eine Ontologie, die als semantisches Datenmodell dient und mit der die Daten aus verschiedenen Datenquellen verknüpft werden. Die Verknüpfung der Ontologie zu den Daten kann z.B. mittels Zuordnungen erfolgen, in denen angegeben ist wie die Elemente der Ontologie mit den Daten der Datenquellen in Verbindung stehen. Stellt der/die NutzerIn eine Abfrage zum Beziehen von Ontologie-Daten an eine Ontologie, so muss die Abfrage unter der Berücksichtigung der Verknüpfung auf die Datenquellen angepasst werden.

¹² <http://www.w3.org/XML/>

¹³ <https://www.ietf.org/rfc/rfc4627.txt>

¹⁴ <https://www.ietf.org/rfc/rfc4180.txt>

¹⁵ <http://www.iana.org/assignments/media-types/text/tab-separated-values>

Nun gibt es prinzipiell zwei Ansätze zum Einbeziehen der Zuordnungen zwischen Ontologie und Datenquellen:

- Statische, bei denen z.B. aus der Ontologie ein festes Datenbankschema erzeugt wird, in welches die Daten aus den Datenquellen übertragen werden.
- Dynamische, bei denen die Zuordnungen zur Laufzeit evaluiert und Abfragen auf die Ontologie in Abfragen auf die Datenquellen übersetzt werden. Dieser Ansatz wird von allen in 3.3.2 vorgestellten Frameworks verwendet.

Ergebnis ist bei beiden Ansätzen, dass der NutzerIn die der Ontologie zugeordneten Daten aus den Datenquellen erhält. Auf diese Weise kann also über eine Ontologie auf Datenquellen zugegriffen werden, was der Vorgehensweise des OBDA entspricht.

Im Abschnitt 3.1 werden weitere Ansätze zur Vorgehensweise OBDA vorgestellt.

2.1.13 Query Rewriting (Abfragen-Umschreiben)

Die Technik des Query Rewriting wird von OBDA Lösungen eingesetzt, die einen direkten und dynamischen Zugriff auf die Datenquellen benötigen. Dabei wird eine Abfrage der Ontologie in SPARQL mittels einer Übersetzungssoftware, die oft in den Reasoner integriert ist, in eine Abfrage der angebundenen Datenquelle, z.B. SQL, übersetzt. Diese Übersetzung setzt voraus, dass Zuordnungen vorgenommen wurden, die es dem Reasoner ermöglichen die passende Datenquelle mit der richtigen Abfrage anzusprechen. Die in den Zuordnungen angegebenen allgemeinen SQL-Abfragen werden dann der SPARQL-Abfrage entsprechend durch verschiedene Algorithmen [2], [6] umgewandelt, um die gewünschten Informationen bereitzustellen. Das nachfolgende Schaubild (Abbildung 9) zeigt den Vorgang schematisch am Beispiel einer Zuordnung zu einer SQL Datenquelle auf.

Der NutzerIn sendet eine SPARQL-Abfrage an das OBDA-System. Dieses evaluiert die Abfrage und zieht die Zuordnungen heran, um eine passende Abfrage an eine SQL-Datenquelle zu senden. Das Ergebnis wird dann wieder an den NutzerIn zurückgegeben.

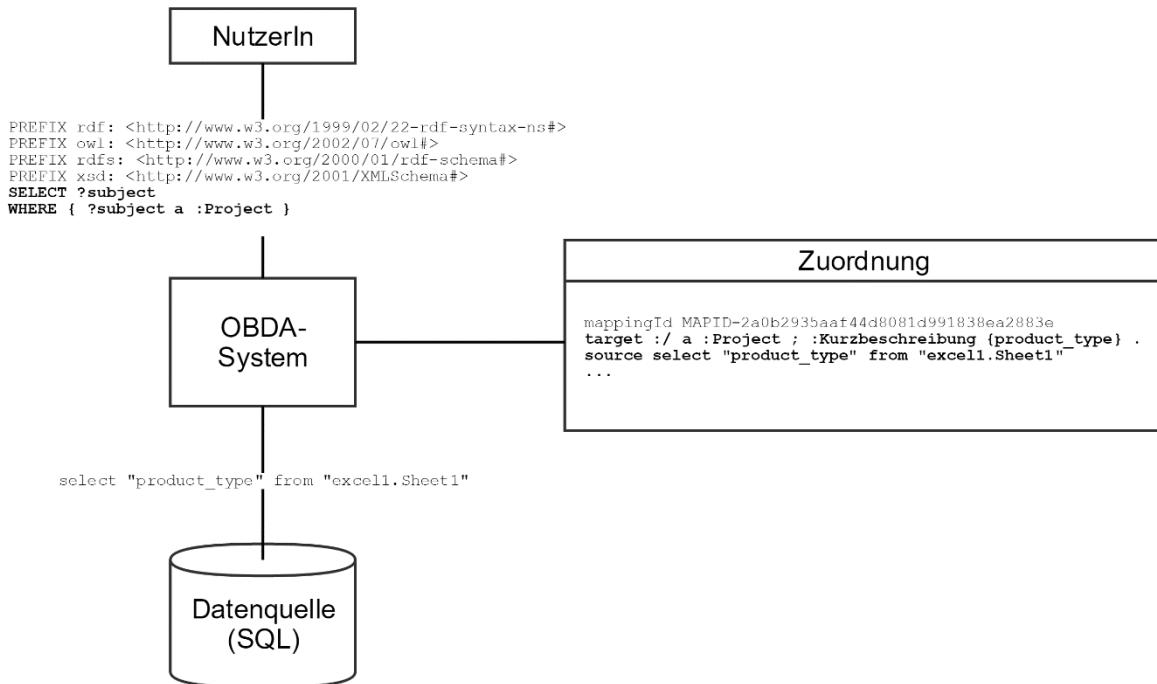


Abbildung 9 Schema Query Rewriting

2.2 Datenintegration und deren Problematik

Der Wunsch nach einer Möglichkeit, Daten semantisch zu beschreiben, reicht schon viele Jahre zurück. Ursprünglich werden Daten in relationalen Datenbanken hinterlegt, von wo sie dann über eine Anfragesprache, wie z.B. SQL wieder beschafft werden können. Die Einschränkung dieser Methode liegt darin, dass sich natürlich sprachliche Zusammenhänge auf diese Art kaum beschreiben lassen. Zudem ist es nicht ohne weiteres möglich, Meta-Informationen, die sich aus den gewonnenen Daten ergeben, zu erheben.

Eine Ontologie hilft hier weiter und ermöglicht es prinzipiell, implizierte Informationen, die sich aus den Zusammenhängen zwischen den Elementen der Ontologie ergeben, durch den Einsatz von Reasoning aufzudecken. Damit Ontology Based Data Access (OBDA) funktioniert, ist es notwendig, eine Verknüpfung zwischen den Daten(quellen) und der Ontologie herzustellen. Diese Verknüpfung muss zu einer großen Zahl unterschiedlicher Technologien kompatibel sein, z.B. zu Datenbanktechnologien wie SQL und zu unterschiedlichen Datenstrukturen wie Text- und Binär-Daten. Außerdem müssen die Daten so integriert werden, dass diese von den Werkzeugen der Ontologie verarbeitet werden können. Hierbei kann es zu vielerlei Schwierigkeiten kommen, die in den kommenden Abschnitten beschrieben werden.

2.2.1 Problematik

Der Datenzugriff auf verschiedene, meist heterogene Datenquellen stellt ein Problem dar, da es die Heterogenität notwendig macht, ein gemeinsames „Verständnis“ der Daten zu entwickeln, um diese vergleichbar zu machen. Heterogenität im Zusammenhang mit Daten bedeutet, dass sich die Daten in ihrer Struktur, ihrem Inhalt, dem Speicherort und der Zugriffsmethode unterscheiden. Bei der

Datenintegration müssen diese Unterschiede der Datenquellen aufgelöst werden, da sonst kein effektives Vergleichen und Zusammenführen möglich ist.

Laut Winter et al. [8] gibt es die folgenden Integrationsarten, die in einem Informationssystem wichtig sind:

- Datenintegration liegt vor, wenn die Daten, die bereits vorhanden sind, überall dort zur Verfügung stehen, wo sie gebraucht werden, ohne dass sie vorher erneut eingegeben werden müssen. Die Daten müssen also nur einmal aufgenommen oder bearbeitet werden und das unabhängig davon, welche Softwarekomponente eines Informationssystems dies durchführt.
- Semantische Integration liegt vor, wenn verschiedene Softwarekomponenten auf den gleichen oder zusammenhängenden Konzepten basieren. Hierdurch werden Daten auf die gleiche Art interpretiert.
- Zugriffsintegration liegt vor, wenn die Softwarekomponenten dort verfügbar sind, wo sie für eine bestimmte Aufgabe benötigt werden.
- Präsentationsintegration liegt vor, wenn die Arbeitsprozesse eines Informationssystems von geeigneten Interaktionsmöglichkeiten in der Applikation unterstützt werden.
- Kontextuelle Integration liegt vor, wenn der Kontext eines Informationssystems erhalten bleibt, auch wenn in eine andere Komponente gewechselt wird. Eine Aufgabe, die bereits für einen bestimmten Zweck durchgeführt wurde, soll nicht noch einmal durchgeführt werden müssen. Ein Beispiel hierfür ist das aktiv halten einer Login-Session.
- Funktionale Integration liegt vor, wenn Funktionen, die in verschiedenen Komponenten der Applikation gebraucht werden, nur einmal implementiert und von dort für andere Komponenten abrufbar sind.
- Prozessintegration liegt vor, wenn die Arbeitsprozesse von Interaktionsmöglichkeiten zwischen den Komponenten unterstützt werden. Ein Beispiel hierfür ist im Krankenhaus die Notwendigkeit, auf verschiedene Komponenten zugreifen zu müssen, wenn man einen Report erzeugen möchte.

In [9] wird der aktuelle Stand der Datenintegration beschrieben. Die Autoren stellen zunächst eine Studie vor, in der verschiedene Unternehmen nach den Mengen ihrer Datenquellen befragt wurden. Auffällig ist hier, dass fast 80% der Unternehmen angaben, mehr als zwei Dokumenten-Speicher zu verwenden. Das ist zunächst wenig überraschend, viel schwerwiegender ist, dass 25% der Unternehmen über 15 verschiedene Dokumenten-Speicher einsetzen. Die Studie gibt einen guten Einblick, wie viele heterogene Daten in Unternehmen erhoben werden. Dem gegenüber steht der Wunsch des Managements, schnell und effizient auf Veränderungen zu reagieren. Dazu ist eine Voraussetzung, die für Entscheidungen notwendigen Daten zu verarbeiten und entsprechend verfügbar zu machen. Heute werden meist so viele Daten an verschiedenen Stellen erzeugt, dass die viel größere Aufgabe, die Nutzbarmachung der Daten geworden ist. Es ist schlicht sehr schwer geworden, die vielen Quellen zu verwalten bzw. zu aggregieren. Auch das SNIK Projekt mit der Software des CION verfolgt dieses Ziel.

Der hierfür notwendige Prozess der Datenintegration lässt sich laut einer Publikation [9] methodisch in folgende Teilschritte unterteilen:

- **Verstehen.** Zunächst müssen die zu integrierenden Daten verstanden werden. Dazu werden die Daten auf ihre Qualität und statistische Eigenschaften analysiert. Dabei können z.B.

Verknüpfungen zwischen den Daten aufgedeckt werden, die im späteren Verlauf des Prozesses wichtig sein können.

- **Standardisierung.** Hierbei wird die vorher gewonnene Information dazu verwendet, um den besten Weg der einheitlichen Repräsentation der Daten auszuwählen. Dazu wird festgelegt, wie die unterliegenden Daten behandelt, angepasst oder repariert werden. So wird versucht, Probleme mit inkompletten, doppelten sowie Daten, die die gleichen Objekte referenzieren, zu entfernen.
- **Spezifikation.** In diesem Schritt werden Spezifikationen zur eigentlichen Ausführung der Integration erstellt. Spezifikationen können z.B. Konfigurations-Daten für Zuordnungswerzeuge sein, die den Zusammenhang zwischen Quellen und Zielen beschreiben, um diese in das gewünschte Format umwandeln zu können.
- **Ausführung.** In diesem Schritt wird die eigentliche Integration mittels der Zuordnungs- und Umwandlungsregeln durchgeführt. Es gibt dazu mehrere Ansätze:
 - **Materialisierung.** Dabei werden die Daten an einer Stelle eingelesen, verarbeitet und dann in einer vereinten Datenbank zusammengeführt.
 - **Föderalisierung** ist im Gegensatz zur Materialisierung nur eine virtuelle Form des Zusammenführens in einer Datenbank. Bei der Föderalisierung werden Anfragen zur Laufzeit an einen Mediator gesendet, der diese wiederum für das unterliegende Format umwandelt, um an die gewünschten Daten in der gewünschten Form heran zu kommen.
 - Die **Suche** wiederum erstellt einen gemeinsamen Index, über die zu integrierenden Daten. Dieser Ansatz wird meistens für unstrukturierte Daten verwendet und ist als eine partielle Materialisierung anzusehen.

Allgemein können bei der Integration von Daten auf verschiedenen Ebenen Probleme auftreten. Schwierigkeiten im Zusammenhang mit heterogenen Daten sind in der Vergangenheit besonders im Bereich der Datenbanken aufgetreten, die sich auch auf das Problem des OBDA übertragen lassen. Im Fall OBDA treten diese Probleme genauso auf: Es gibt Unterschiede bei den Datenquellen, die angesprochen werden müssen. Zum einen werden die Datenquellen auf unterschiedliche Art und Weise zur Verfügung gestellt, und deren Daten sind nicht immer direkt miteinander vergleichbar. All diese und die nachfolgenden Schwierigkeiten sind daher beim Erstellen der Zuordnungen und der Konfiguration des Frameworks relevant. Bei der Planung der Zuordnungen und der Durchführung werden die oben genannten Schritte berücksichtigt.

Die folgenden Abschnitte zeigen die Probleme der Datenintegration im Allgemeinen laut [6] auf.

Man teilt die Konflikte, die auftreten können, grob in drei Kategorien ein:

1. **Syntaktisch:** Da es unterschiedliche Beschreibungen für semantisch gleiche Daten geben kann, treten Probleme der Vergleichbarkeit bei unterschiedlichen Datenmodellen oder - Typen auf (z.B.: Entity-Relationship Schema im Vergleich zu relationaler/objektorientierter Modellbeschreibung oder unterschiedliche Abfragesprachen).

2. **Strukturell:** Unterschiede in der Art und Weise, wie Daten in den Systemen abgelegt werden. Beispiel: XML-Datei und SQL-Datenbank.
3. **Semantisch:** Bezieht sich auf den Inhalt der Daten und deren Bedeutung, die zwischen den Systemen klar sein muss, um den Austausch zwischen Systemen zu ermöglichen.
Semantische Interoperabilität muss sowohl in der Applikation als auch beim Verständnis der Daten beachtet werden.
Semantische Heterogenität kann verschiedene Gründe haben, die die Interoperabilität beeinträchtigen:
 1. **Verwechslung:** Liegt vor, wenn zwei Informationen scheinbar die gleiche Bedeutung haben, in Wirklichkeit allerdings (z.B. auf Grund von anderen Zeitepochen) eine andere Bedeutung haben.
 2. **Skalenunklarheit:** Diese liegt vor, wenn bei der Erhebung der Daten verschiedene Messmethoden oder Skalen verwendet wurden, die die Daten wiederum nicht miteinander vergleichbar machen.
 3. **Benennungskonflikte:** Treten auf, wenn sich Namenskonventionen voneinander unterscheiden. Dazu gehören die Wortgruppen der Homonyme und Synonyme.

Damit in einer Applikation die Integration gelingt, müssen Konflikte gelöst bzw. vermieden werden, nur dann kann die Integration reibungslos vonstattengehen. Gerade das Lösen solcher Konflikte ist immer noch ein Prozess, der sich kaum automatisieren lässt und daher manuell durchzuführen ist. Es ist essenziell, die vorliegenden Daten genau zu kennen und einen geeigneten Plan zur Lösung solcher Konflikte zu erstellen. In diesem Plan geht es primär um die Frage, auf welchem Weg die Konflikte und Unterschiede gelöst werden. Über die Problematik der Datenkonflikte hinaus ist es sinnvoll, bei der Wahl eines Integrationskonzepts für die Daten deren Einsatzzweck zu berücksichtigen.

2.2.2 Datenintegration

Konzeptionell haben sich mehrere Ansätze, die in [6] beschrieben werden, zur Umsetzung von Datenintegration herauskristallisiert, die sich in heutigen Frameworks wiederfinden. Im Folgenden werden diese Ansätze kurz, mit ihren Vor- und Nachteilen, erläutert.

2.2.2.1 Multibank System

Ein Multibank System erlaubt es NutzerInnen, durch ein gemeinsames globales Schema die Datenbank wie eine einzelne föderierte Datenbank zu betrachten. Hierbei wird allerdings nicht versucht, die Semantik der Daten zu vereinen. Ein Multibank System basiert auf zwei Komponenten: Ein Werkzeug zur Schemaerstellung und ein Anfragebearbeitungssystem. Das Schema-Erstellungs-Werkzeug wird von Datenbank-Entwicklern eingesetzt, um Zuordnungen zwischen den lokalen Datenbanken und dem globalen Schema zu definieren. Diese Zuordnungen werden dann wiederum vom Abfragesystem verwendet, um globale Anfragen in lokale umzuwandeln. Zur Illustration kann Abbildung 10 betrachtet werden.

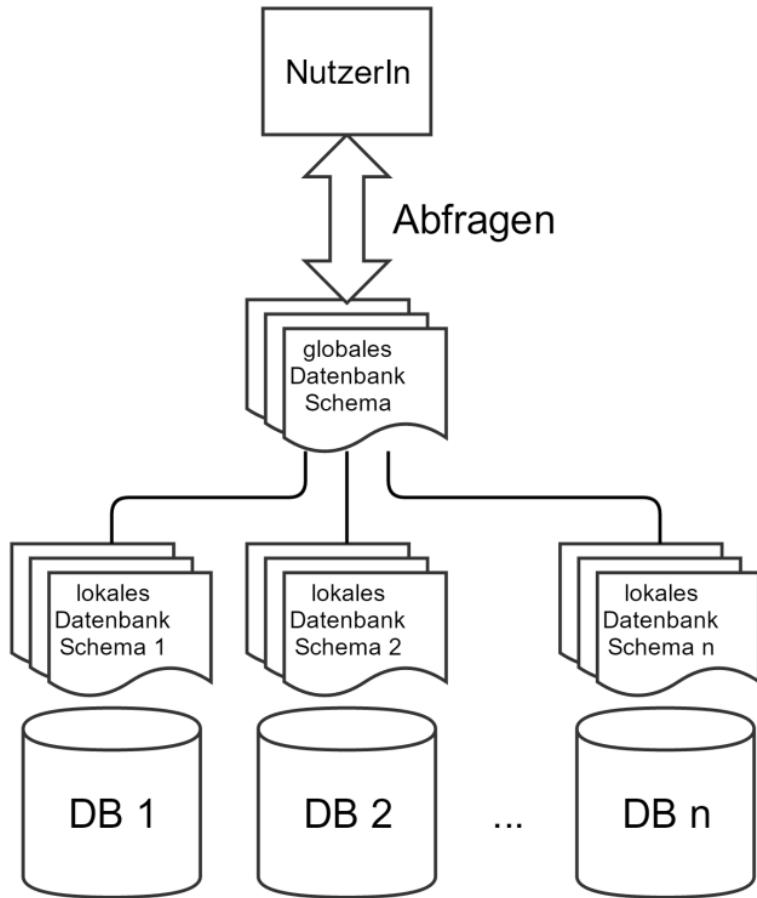


Abbildung 10 Schematische Darstellung eines Multibank Systems

Ein großer Vorteil des Modells ist, dass hier mit den Quelldaten gearbeitet wird. Schwierigkeiten ergeben sich vor allem zu Beginn der Umsetzung des Systems, da die verschiedenen Schemata mit Hilfe des Werkzeugs erstellt werden müssen und diese recht statisch sind. Daher ist das System nicht geeignet für Anwendungen, in sich stetig verändernden Umgebungen. Schwierig bzw. unmöglich ist es ebenso, Datenbanksysteme von verschiedenen Herstellern auf Basis dieses Ansatzes miteinander zu integrieren, da diese im Zweifelsfall nicht durch ein gemeinsames Schema abgebildet werden können und sich die Abfragemethoden unterscheiden.

Der Multibank-Ansatz ist ähnlich zu OBDA, wie es über die Frameworks umgesetzt wird. An Stelle einer Datenbank, wie im Multibank System, kann aber in einem OBDA-Framework auch ein föderiertes System angesprochen werden, welches mehrere Datenquellen zusammenführt. Der große Unterschied zu OBDA ist, dass statt eines statischen Schemas bei den meisten OBDA Frameworks dynamisches Query Rewriting eingesetzt wird und der Zugriff auf die Daten über eine Ontologie erfolgt.

2.2.2.2 Föderiertes System

Bei einem föderierten System [10] findet die Integration auf der Schema-Ebene jeder Datenquelle statt. Bei der Gestaltung eines föderierten Schemas wird versucht die Quellschemata zu vereinen und Heterogenität dadurch zu vermeiden. Dabei ist es notwendig genau diese Problematiken mit dem

neuen Schema zu beheben. Diese Form der Integration wird als statisch bezeichnet, da die Zuordnungen zwischen den lokalen und dem globalen Schema statisch sind.

Ein föderiertes System ist im Vergleich zu einem Multibank System zusätzlich darauf ausgelegt, semantische Unterschiede zwischen den Datenquellen zu entfernen. Die Zuordnungen zu den verschiedenen Quellen können mittels Regeln, bestimmten Sprachen, aber auch über eine Ontologie erfolgen, die die Aufgabe einer Schema-Übersetzung zwischen verschiedenen lokalen Datenbank-Schemata erfüllen kann.

2.2.2.3 *Mediations-System*

Ein Mediator ist eine Software, die kodiertes Wissen über eine (Unter-)Menge von Daten dazu verwendet, Informationen für eine höhere Schicht der Applikation herzustellen. Ein Mediator ist eine Art Vermittler zwischen dem/der NutzerIn und den tatsächlichen Datenquellen. Die Daten werden dafür nicht im Mediator gespeichert, sondern verbleiben an ihrem ursprünglichen Ort. Beim Zugriff auf die Datenquellen werden Wrapper eingesetzt, die die Anfrage auf das darunterliegende System anpassen und die gewonnenen Daten wiederum an den/die NutzerIn weitergegeben.

Je nachdem, wie viele unterschiedliche Technologien für die verschiedenen Datenquellen eingesetzt werden, ist es dementsprechend aufwändig, die Mediatoren uniform zu gestalten. Ein großer Vorteil liegt aber darin, dass die Daten, die bezogen werden, stets aktuell sind, da direkt auf den Quelldaten operiert wird.

2.2.2.4 *Data Warehouse (DWH)*

Data Warehouses allgemein ermöglichen einheitliche Abfragen auf importierten Daten aus unterschiedlichen Quellen.

Dieses Konzept hat den großen Vorteil, dass nur relevante Daten im Data Warehouse in genau der Form, in der sie benötigt werden, verfügbar sind. Das bringt erhöhte Geschwindigkeit bei Zugriffen auf Daten des DWH mit sich, da diese bereits in der Form, in der sie von der Applikation benötigt werden, hinterlegt sind. Dadurch kann ein DWH für den Einsatz in bestimmten Situationen optimiert werden. Allerdings hat das Modell auch durchaus seine Nachteile: Der Prozess der Extraktion und die Verarbeitung benötigen eine gewisse Zeit, in der keine neuen Daten mehr in das System eingespielt werden können. Daher muss bei der Verwendung klar sein, dass mit veralteten Daten gearbeitet wird, bis die Änderungen an das Datawarehouse propagiert wurden.

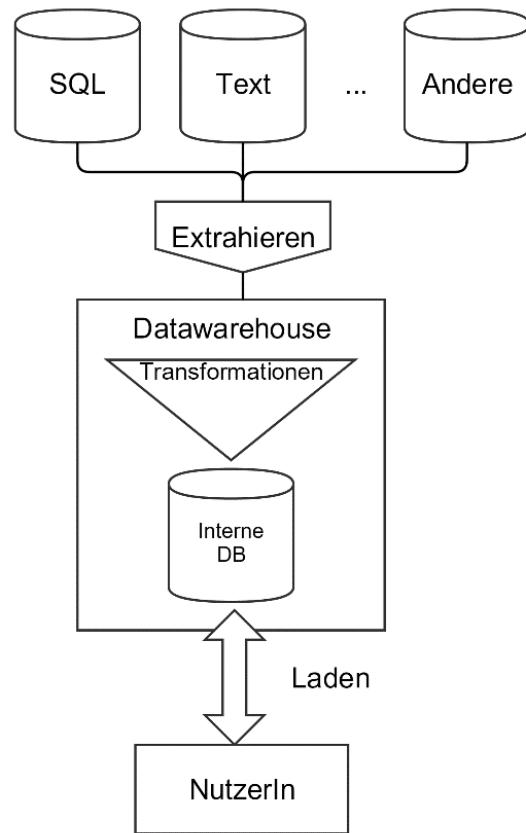


Abbildung 11 Schematisches Beispiel eines Data Warehouse

Solche Szenarien würden dem Ziel eines Systems zur Entscheidungsunterstützung entgegenwirken, da die Möglichkeit besteht unabsichtlich mit nicht aktuellen Daten zu arbeiten.

Im Vergleich zu OBDA gibt es nämlich einen dedizierten Extraktions-, Transformations- und Lade-Prozess (ETL-Prozess). Dabei werden die Daten aus der ursprünglichen Quelle (z.B. SQL oder Log-Dateien) extrahiert, in die gewünschte Form transformiert und in eine Datenbank des DWH geladen. Erst dann stehen die Daten dem/der NutzerIn zur Verfügung. Dieser Aufbau ist in Abbildung 11 schematisch dargestellt. Bei OBDA erfolgt auf Basis der Abfrage des/der NutzerIn ein direkter Zugriff auf die darunter liegenden Daten.

2.3 SNIK-Projekt

2.3.1 Das SNIK-Projekt und CION

Das SNIK-Projekt ist ein DFG gefördertes Forschungsprojekt und eine Kooperation der Universität Heidelberg und dem IMISE der Universität Leipzig¹⁶. Primäres Ziel des Projektes ist die Entwicklung einer Ontologie des Informationsmanagements im Krankenhaus, die dessen Bestandteile aus Literatur und realer Umgebung abbildet. Ein weiteres Ziel ist die Entwicklung eines entscheidungsunterstützenden Systems CIO-Navigator (CION), das den Krankenhaus-CIOs bei seinen Aufgaben unterstützt.

¹⁶ <http://www.snik.eu/>

2.3.2 Die CIOx-Ontologie

Die im Rahmen des SNIK-Projekts entstandene Ontologie, die die Aufgaben des Informationsmanagements in einem großen Krankenhaus in Deutschland modelliert, ist eine wichtige Komponente dieser Arbeit. Sie entsteht im bisherigen Verlauf des Projekts und wird als das Datenmodell verwendet, auf Basis dessen die Zuordnungen gemacht werden verwendet. Genauer handelt es sich um eine Ontologie, die durch die Betrachtung des Arbeitsbereichs des CIO in einem großen Deutschen Krankenhaus entstanden ist. Diese Ontologie wird im Folgenden als die „CIOx-Ontologie“ bezeichnet. Beschrieben ist die Ontologie in OWL. Die nachfolgende Tabelle zeigt einige Metriken auf:

Tabelle 3 CIOx-Ontologie Metriken

Axiome	907
Klassen	186
Objekteigenschaften	11
Dateneigenschaften	29
Individuen	28

Auf Grund ihrer Größe, kann die Ontologie hier nicht in ihrer Gänze beschrieben werden. Hierfür wird auf [11] verwiesen. Das Metamodell der Ontologie ist in Abbildung 12 beschrieben.

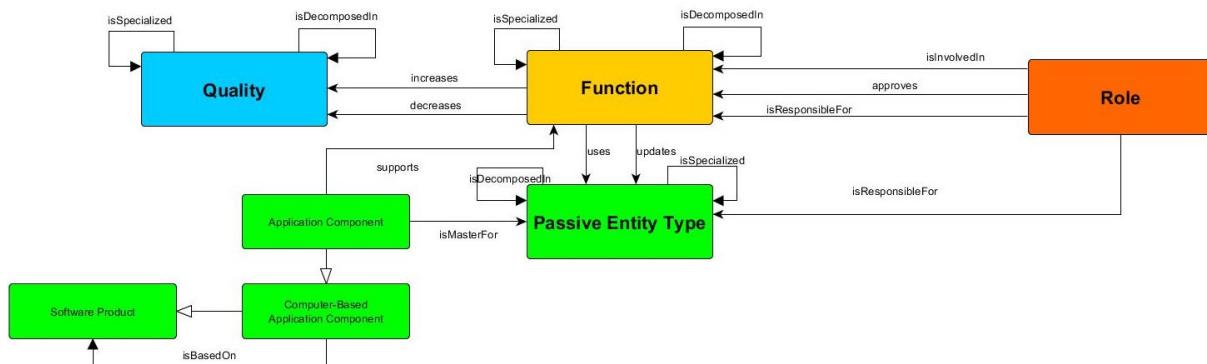


Abbildung 12 Schema des Metamodells der CIOx-Ontologie. Quelle: SNIK Projekt, IMISE Leipzig

Das hier dargestellte Metamodell der Ontologie wird mit Hilfe der klassischen Ontologieelemente, wie z.B. Klassen, aufgebaut (siehe 2.1.10).

Die im Metamodell definierten „Funktionen“ (Function), „Rollen“ (Role), „Passive Entitäten Typen“ (Passive Entity Types) und „Qualitäten“ (Quality) sind, in der Ontologie selbst, alle als Klassen modelliert.

Um die Modellierung besser verständlich zu machen, wird diese anhand des Beispiels aus dem Bereich des Projektmanagements erklärt. Hierfür kann Abbildung 3, die die Klassen im Bereich Projektmanagement darstellt, betrachtet werden. Die als Unterklassen des „Projektmanagements“ modellierten Klassen sind als Funktionen zu verstehen, die im Zusammenhang mit dem Projektmanagement stehen. Diese Funktionen können wiederum von verschiedenen Rollen durchgeführt werden. Eine Rolle kann z.B. der Projektleiter sein, der über die

Relation „isInvolvedIn“ mit der Funktion „Wartungsvertragserstellung“ verbunden ist (siehe Abbildung 13). Auf diese Weise wird der reale Umstand modelliert, dass der Projektleiter bei der Erstellung des Wartungsvertrags beteiligt ist. Passive Entitäten Typen sind Elemente wie z.B. „Projekt“, welches über die Relation „isAssociatedWith“ einen Projektleiter zugewiesen hat (siehe Abbildung 14). Passive Entitäten stellen Elemente dar, die für Funktionen verwendet werden. Auch hier ist die Modellierung Realität nachempfunden: Ein Projektleiter ist für ein Projekt zuständig.

Description: Projektleiter

Equivalent To +

SubClass Of +

- **Information_Management_Staff**
- **isInvolvedIn some Wartungsvertragserstellung**
- **isResponsibleFor some Projekt**
- **Name some xsd:string**
- **Ursachen some xsd:string**

Abbildung 13 Projektleiter Details (Screenshot Protégé)

Die in Abbildung 14 dargestellten Elemente, mit Datentypen (z.B. Projektnummer), sind Eigenschaften der Klasse. In diesem Fall handelt es sich um eine im Metamodell als Passive Entität modellierte Klasse mit den Eigenschaften „Jahr“, „Name“, „Projektbeginn“ und „Projektnummer“.

Description: Projekt

Equivalent To +

SubClass Of +

- **Information_Management_Entity_Type**
- **isAssociatedWith exactly 1 Controllingliste_Dienstleistungen**
- **isAssociatedWith exactly 1 Controllingliste_Invest**
- **isAssociatedWith exactly 1 Geplante_Aktivierbare_Dienstleistungsmittel**
- **isAssociatedWith exactly 1 Geplanter_Personalaufwand**
- **isAssociatedWith exactly 1 Plansumme_Dienstleistungsmittel**
- **isAssociatedWith exactly 1 Plansumme_Fremdmittel**
- **isAssociatedWith exactly 1 Plansumme_Investitionsmittel**
- **isAssociatedWith exactly 1 Projektpriorität**
- **isAssociatedWith exactly 1 Projektrisiko**
- **isAssociatedWith some Projektleiter**
- **isAssociatedWith some Projektwebsite**
- **isDecomposedIn some Projektdefinition**
- **Jahr exactly 1 xsd:dateTime**
- **Name some xsd:string**
- **Projektbeginn exactly 1 xsd:dateTime**
- **Projektnummer some xsd:int**

Abbildung 14 Projekt Details (Screenshot Protégé)

Bei den in Abbildung 12, links von den Passiven Entitäten Typen, grün dargestellten Elementen handelt es sich um Software Komponenten. Diese unterstützen bei den jeweiligen Funktionen und haben damit Einfluss auf die Passiven Entitäten Typen. Ein Beispiel hierfür ist die Funktion

„Personnel_Planning“, welche von einem Personaleinsatzplanungssystem unterstützt wird (siehe Abbildung 15).

Description: Personnel_Planning

Equivalent To +

SubClass Of +

- **isSupportedBy some Personaleinsatzplanungssystem**
- **Personalmanagement**

Abbildung 15 Personell_Planning Details (Screenshot Protégé)

Die hier aufgeführten Elemente sind alle Elemente des Typs „Function“ des Metamodells. Das ist daran erkennbar, dass die Funktion, bzw. Aufgabe „Projektmanagement“ als eine Subklasse der Klasse „Information Management Function“ modelliert ist. Somit sind die hier aufgeführten Klassen im Metamodell als Funktionen zu verstehen. Genauso verhält es sich bei den Rollen:

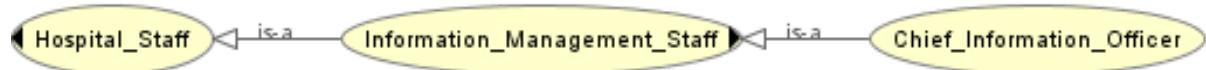


Abbildung 16 Ausschnitt Rollen der Ontologie (OWLWiz export)

Auch die Rollen sind als Klasse modelliert, allerdings als Subklasse der Rolle „Hospital_Staff“.

3 Literaturrecherche

Dieses Kapitel beschreibt Vorgehen und Ergebnisse der Literaturrecherche. Die Recherche besteht aus zwei Teilen. Der erste Teil beschäftigt sich mit der Fragestellung, welche Ansätze es zur Verwendung von Ontologien als Datenquellen und als Datenmodell gibt. Im zweiten Teil, welcher ab Abschnitt 3.3 beginnt, wird die Frage beantwortet welche Frameworks für OBDA es gibt und welche Eigenschaften diese Frameworks besitzen.

Dabei werden zuerst die folgenden konkreten Fragestellungen beantwortet:

- „Welche Ansätze zur Verwendung von Ontologien als Datenquelle gibt es?“
- „Welche Ansätze zur Verwendung von Ontologien als Datenmodell gibt es?“

Zur Recherche werden Suchen auf verschiedene Literatur-Quellen ausgeführt.

- IEEE¹⁷
- Heidi¹⁸
- ACM¹⁹
- Google Scholar²⁰ (GooSch)

Das Vorgehen bei der Literaturrecherche wurde nach den Richtlinien der Arbeitsgruppe Software Engineering der Universität Heidelberg [12] durchgeführt. Zusammenfassend verlief die Recherche wie folgt:

Zur Einarbeitung in die Fragestellungen werden allgemeinere Suchbegriffe abgeleitet:

- Ontology
- Data model
- Data source

Deren Ergebnisse werden zunächst dafür verwendet, um einen Überblick über die Thematik zu erhalten. Hierbei kommt schnell der Begriff „OBDA“ auf, welcher mit beiden Fragen eng in Verbindung steht, da bei OBDA eine Ontologie sowohl als Datenmodell als auch Datenquelle für den Zugriff auf Daten zum Einsatz kommt. Als nächstes werden daher spezifischere Suchen, die sich mit OBDA und verwandten Themen beschäftigen, abgeleitet:

- Ontology based access
- ontology based data access
- ontology as data source
- OBDA
- Ontology based database access

Diese werden dazu verwendet um oben genannte Fragen zu beantworten und gleichzeitig konkrete Anwendungsbeispiele für OBDA zu finden. Auf diese Weise wird auch die Fragestellung nach den existierenden OBDA-Ansätzen beantwortet.

Die o.g. Literaturquellen werden jeweils mit diesen Suchbegriffen befragt.

¹⁷ <https://www.ieee.org/index.html>

¹⁸ <http://www.ub.uni-heidelberg.de/helios/kataloge/heidi.html>

¹⁹ <https://www.acm.org/>

²⁰ <https://scholar.google.de/>

Um die Relevanz der Ergebnisse zu verbessern, sind folgende Exklusions-Kriterien definiert:

- Die gefundenen Ergebnisse werden auf die letzten 10 Jahre (2005-2015) beschränkt, da mehr als 10 Jahre in die Vergangenheit gerade in der Informatik eine enorme Zeitspanne ist, in der sich viel verändert.
- Außerdem werden nur die ersten fünfzig Treffer pro Suchanfrage betrachtet, da eine Sichtung sonst zu viel Zeit in Anspruch nimmt.

Als relevant wird eine Publikation eingestuft (siehe „# relevante Ergebnisse“ in Tabelle 4), wenn das Thema eine Anwendung oder Forschung zu Prinzipien des OBDA in Abstract und Einleitung beschreibt. Dabei wird darauf geachtet, dass sowohl Inhalte die sich mit allgemeineren Fragestellungen wie z.B. „Wie kann OBDA durchgeführt werden?“, aber auch spezielleren Fragestellungen, wie OBDA für ein bestimmtes Einsatzgebiet eingesetzt werden kann, betrachtet werden. Falls die gefundene Publikation eine potenzielle Quelle ist, so wird es dem Mendeley²¹ Katalog hinzugefügt.

Tabelle 4 zeigt die Ergebnisse der im Zeitraum November 2015 durchgeföhrten Literaturrecherche:

Tabelle 4 Suchergebnisse Literaturrecherche

Such Ort	Suchdatum	Suchbeschränkung	Suchanfrage	# Ergebnisse	# relevante Ergebnisse
IEEE	16.11.2015	2005-2015	ontology based data access	555	5
GooSch	16.11.2015	2005-2015	ontology as data source	301.000	2
GooSch	16.11.2015	2005-2015	OBDA	4900	2
GooSch	16.11.2015	2005-2015	Ontology based database access	62.100	6
IEEE	28.11.2015	2005-2015	ontology as data source	1.232	3
IEEE	28.11.2015	2005-2015	OBDA	6	3
IEEE	28.11.2015	2005-2015	Ontology based database access	252	4
GooSch	28.11.2015	2005-2015	Ontology based access	410.000	4
ACM	28.11.2015	2005-2015	Ontology based access	120.906	2
ACM	16.11.2015	2005-2015	ontology based data access	159.866	3
ACM	28.11.2015	2005-2015	ontology as data source	173.732	1
ACM	28.11.2015	2005-2015	OBDA	7	4
ACM	28.11.2015	2005-2015	Ontology based database access	127.505	4
Heidi	28.11.2015	2005-2015	Ontology based access	11.384	5
Heidi	16.11.2015	2005-2015	ontology based data access	5.416	5
Heidi	28.11.2015	2005-2015	ontology as data source	6.338	5
Heidi	28.11.2015	2005-2015	OBDA	211	5
Heidi	28.11.2015	2005-2015	Ontology based database access	2.577	1

Insgesamt sind so 23 relevante Publikationen gefunden worden.

Nach der Identifikation der relevanten Publikationen, wurden alle Publikationen gelesen und zusammengefasst. Dabei werden auch Publikationen identifiziert, die weniger relevant sind, z.B. weil es keine besonderen Hinweise in Bezug auf die Verwendung einer Ontologie als Datenmodell/-Quelle

²¹ <https://www.mendeley.com/>

gibt. Wenn mehrere Publikationen ein bestimmtes Framework vorstellen wird die aktuellste Publikation verwendet. Gibt es mehrere Publikationen, die das gleiche Framework nutzen, so wird betrachtet ob es durch eines der beiden Publikationen einen besonderen Aspekt gibt, der von anderen Publikationen nicht betrachtet wird. Ist das nicht der Fall, so wird die Publikation nicht verwendet.

Für diese Arbeit werden 14 der 23 Publikationen verwendet. Tabelle 5 zeigt die gelesenen Quellen mit Bemerkungen und einer kurzen Zusammenfassung auf. Die Spalten ganz rechts geben Aufschluss darüber ob und warum eine Quelle (nicht) verwendet wurde. Die wesentlichen Ergebnisse der Literatursuche und den ausgewählten Publikationen sind im anschließenden Kapitel 3.3 beschrieben.

Tabelle 5 Übersicht gefundener Publikationen

Autoren, Titel, Veröffentlichungsjahr	kurze Zusammenfassung	Bemerkung	Verw.	Begründung	# in Literaturverz.
Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. (2010).	Problematik: Datenströme von verschiedenen Medien sind heterogen und sollen auf Ontologie-Level beschreibbar sein. Vorstellung eines Erweiterten SPARQL-Abfrage-Sets für Streaming Quellen (SPARQLStream). Beschreibung der Zusammenführung der heterogenen Datenquellen	Spezielle Datenquellen	X	Konzeptionell im Vergleich zu anderen Publikationen nicht besonders und außerdem auf Datenströme ausgelegt	
Calvanese, D., Cogrel, B., Komla-ebri, S., Kontchakov, R., Lanti, D.: Ontop : Answering SPARQL Queries over Relational Databases. (2015).	Vorstellung der Funktionsweise und Umfang des Ontop-Frameworks. Detaillierte Beschreibung des Aufbaus und möglicher Einsatzziele.	Aktuell (2015)	✓	Umfangreiche Publikation mit konkreter Vorstellung eines Frameworks und dessen Aufbau. Allgemeine Hilfreiche Erklärungen	[13]
Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. (2011).	Vorstellung der Funktionsweise und Umfang der Software "Mastro". Mit dem Fokus auf den Reasoner. Detaillierte Beschreibung des Aufbaus.		✓	Umfangreiche Publikation mit konkreter Vorstellung eines Frameworks und dessen Aufbau.	[14]
Calvanese, D., Giacomo, G. De, Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Ontology-based database access. (2007).			X		

	Calvanese, D., Mosca, A., Remesal, J., Rezk, M., Rull, G.: A "Historical Case" of Ontology-Based Data Access. (2015).	Beschreibt den Einsatz von OBDA für die Aggregation von Geschichtsdaten an einem konkreten Beispiel		X	Konzeptionell im Vergleich zu anderen Publikationen nicht besonders, aber dennoch interessantes Beispiel.	
	Gagnon, M.: Ontology-based integration of data sources. 2007 10th Int. Conf. Inf. Fusion. 1–8 (2007).	Generelle Publikation, welche die Problematiken und Lösungsansätze bei OBDA aufzeigt.	2007 und daher etwas älter, aber konzeptionell immer noch passend	✓	Stellt wichtige Punkte um OBDA dar. Ist vor allem für die Grundlagen relevant.	[6]
	Giacomo, G. De, Lembo, D., Lenzerini, M.: Mastro: A reasoner for effective ontology-based data access. (2012).	Vorstellung der Funktionsweise und Umfang der Software "Mastro". Detaillierte Beschreibung des Aufbaus.		✓	Umfangreiche Publikation mit konkreter Vorstellung eines Frameworks und dessen Aufbau.	[15]
41 von 133	Haas, L.: Beauty and the Beast : The Theory and Practice of Information Integration. (2007).	Vorstellung des Standes der aktuellen OBDA und Datenintegration in 2007. Die Beispiele sind nicht mehr aktuell, die Problematiken und Lösungsansätze aber dennoch passend.	Vergleiche aus der Vergangenheit und Entwicklungen der letzten Jahre möglich	✓	Dient als Referenz für Vergangenen Zustand und heute. Die beschriebenen Problematiken sind auch heute noch relevant, daher wird es verwendet. Ist hauptsächlich für die Grundlagen relevant.	[9]

Park, J., Lee, E.K., Wang, Q., Li, J., Lin, Q., Pu, C.: Health-connect: An ontology-based model-driven information integration framework and its application to integrating clinical databases. (2012).	Vorstellung einer eigens entwickelten Lösung, die den Zugriff auf die Daten über das automatische Ableiten von Informationen aus Ontologie und gegebenen Transformationsmodellen durchführt. Gibt einen Einblick in mögliche Richtung der OBDA Frameworks zu mehr Automation	Vorgestellte Lösung nicht offen zugänglich.	✓	Bietet eine detaillierte Beschreibung der Lösung. Ist außerdem ein gutes Beispiel für kommende Entwicklung zu mehr Automation.	[16]
Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. (2008).	Vorstellung der Funktionsweise und Umfang der Software "QuOnto", die mittlerweile "Mastro" genannt wird. Detaillierte Beschreibung des Aufbaus und verwendeter Konzepte	Nicht mehr ganz aktuell	X	Vorläufer zu Mastro. Die beschriebenen Konzepte unterscheiden sich nicht von der Vorgehensweise die in den Mastro Publikationen vorgestellt wird, mit dem Unterschied dass die Publikation weniger aktuell ist.	
Poggi, A., Rodriguez-muro, M., Ruzzi, M.: Ontology-based database access with DIG-M ASTRO and the OBDA Plugin for Protégé. (2008)	Vorstellung der Funktionsweise und Umfang der Software "DIG-Mastro", die mittlerweile "Mastro" genannt wird. Detaillierte Beschreibung des Aufbaus und verwendeter Konzepte	Nicht mehr ganz aktuell	X	Vorläufer zu Mastro. Die beschriebenen Konzepte unterscheiden sich nicht von der Vorgehensweise die in den Mastro Publikationen vorgestellt wird, mit dem Unterschied dass die Publikation weniger aktuell ist.	

Pokharel, S., Sherif, M.A., Lehmann, J.: Ontology Based Data Access and Integration for Improving the Effectiveness of Farming in Nepal. (2014).	Beschreibt den Einsatz von OBDA für die Aggregation von Landwirtschafts-, Geographie- und Wetter-Daten an einem konkreten Beispiel: Verbesserung der Landwirtschaft in Nepal.		X	Gutes Beispiel für den Einsatz von OBDA für ein bestimmtes Einsatzziel. Allerdings nichts Neues.	
Rodríguez-Muro, M., Calvanese, D.: Quest, an OWL 2 QL reasoner for ontology-based data access. (2012).	Vorstellung der Funktionsweise und Umfang des im Ontop-Framework eingesetzten Reasoners "Quest". Detaillierte Beschreibung des Aufbaus.		✓	Genauere Beschreibung des in Ontop verwendeten Reasoners: "Quest"	[17]
Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. (2013).	Vorstellung der Funktionsweise und Umfang des Ontop-Frameworks. Detaillierte Beschreibung des Aufbaus.		X	Weniger aktuelle Publikation zu Ontop.	
Rodriguez-Muro, M., Lubyte, L., Calvanese, D.: Realizing Ontology Based Data Access: A plug-in for protégé. (2008).	Vorstellung der Funktionsweise und Umfang eines OBDA-plugins für Protégé, welches für Zuordnungen in Mastro eingesetzt werden kann. Eher kurze Beschreibung.		X	Vorläufer zu Mastro. Die beschriebenen Konzepte unterscheiden sich nicht von der Vorgehensweise die in den Mastro Publikationen vorgestellt wird, mit dem Unterschied dass die Publikation weniger aktuell ist.	
Xiao, G.: Ontology Based Data Access. (2014).	Umfangreiche Präsentation eines Ontop-Entwicklers zum Thema OBDA allgemein. Bietet eine umfangreiche Basis der verschiedenen Komponenten und verwandten Themen		X	Gute Quelle und Umfangreiche Präsentation, allerdings kein wissenschaftlicher Text.	

Guarino, N., Oberle, D., Staab, S.: What Is an Ontology? Handbook on Ontologies. In: Handbook on Ontologies SE - International Handbooks on Information Systems. pp. 1–17 (2009).	Viele Grundlagen und Beschreibungen zum Thema Ontologien.		✓	Wird in den Grundlagen verwendet.	[1]
Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 4900 LNCS, 133–173 (2008).	Definiert OBDA ausführlich. Zuordnungen, Reasoning über Ontologien mit Zuordnungen usw.		✓	Wird in den Grundlagen verwendet.	[2]
Guarino, N., Giaretta, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. Towar. Very Large Knowl. Bases Knowl. Build. Knowl. Shar. 1, 25–32 (1995).	Viele Grundlagen und Beschreibungen zum Thema Ontologien.		✓	Wird in den Grundlagen verwendet.	[5]
Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. Iswc. 8218 LNCS, 558–573 (2013).	Beschreibt das Ontop-Framework auf einem etwas technischeren Level		✓	Wird in Bezug auf die Framework-Recherche zu Ontop verwendet.	[7]
Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. 22, 183–236 (1990).	Definiert viele Begriffe rund um Datenbanken. Auch das Föderierte Datenbanksystem wird hier ausführlich erklärt		✓	Wegen der Grundlagen in Föderierten Datenbanksystemen verwendet	[10]

<p>Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information-a survey of existing approaches. IJCAI Work. Ontol. Inf. Shar. 108–117 (2001).</p>	<p>Publikation zum Thema der Datenintegration mittels Ontologien.</p>		✓	<p>Wird für den Ansatz mehrerer Ontologien verwendet.</p>	[18]
<p>Erling, O., Mikhailov, I.: RDF support in the virtuoso DBMS. Stud. Comput. Intell. 221, 7–24 (2009).</p>	<p>Umfangreiche Publikation zu Virtuoso.</p>		✓	<p>Wird für das Framework Virtuoso benötigt.</p>	[19]

3.1 OBDA Ansätze

Bisher ist nur die Integration der Daten allgemein beleuchtet worden, nicht aber, wie es sich mit Ontologien in diesem Zusammenhang verhält. Beim OBDA steht/t/en über den verschiedenen Datenquellen eine gemeinsame (oder mehrere lokale) Ontologie(n), die die Daten in der darunter befindlichen Ebene durch Zuordnungen miteinander vergleichbar machen und in semantischen Kontext stellen. Die Daten sind z.B. die Instanzen der in der Ontologie modellierten Klassen. Aus diesem Grund sind die Herausforderungen, die sich bei der Integration von Daten auf Datenbankebene ergeben, auch analog für OBDA relevant: In beiden Fällen muss klar sein, wie die Daten untereinander, z.B. zu den in der Ontologie hinterlegten Klassen, in Verbindung stehen. Beim OBDA gilt es auch die folgenden Aspekte bei der Integration von Daten in die Ontologie zu lösen. Auftretende Konflikte (siehe 2.2.2) werden vom OBDA Framework durch passende Zuordnungen gelöst:

1. **Syntaktisch:** Im Allgemeinen ist die Syntax der Datenquellen häufig heterogen. Es ist daher nicht ohne weiteres möglich, z.B. Daten aus einer Excel-Datei mit Daten aus einer Datenbank zusammen zu führen. Die Syntax der Formate muss vom Framework normiert werden. In den aktuellen Frameworks (wie z.B. Ontop, Stardog und Mastro) wird das durch den Einsatz von JDBC-Treibern bewerkstelligt, die eine JDBC-Schnittstelle anbieten, an die SQL-Abfragen gestellt werden können. Auf diese Weise sind die Daten, die in das Framework eingebunden werden, in ihrer Syntax auf SQL normiert.
2. **Strukturell:** In welcher Form die Daten auf Speichermedien abgelegt werden, spielt für OBDA eine untergeordnete Rolle. Es ist lediglich wichtig, dass das Framework auf die Daten zugreifen kann. Auch hier erfolgt der Zugriff wieder indirekt über einen JDBC-Treiber, welcher die Schnittstelle zwischen der Datenquelle und dem OBDA Framework ist. Aktuelle JDBC-Treiber unterstützen Datenquellen auf entfernten Systemen.
3. **Semantisch:** Diese Probleme sind zumeist manuell zu lösen. Bei der Zuordnung von Ontologie und Datenquelle ist manuelles Eingreifen notwendig, um zu bestimmen, wie die Daten aus der Quelle interpretiert werden sollen. Hier wird auch der Typ festgelegt. An dieser Stelle ist es daher wichtig, dass die Daten, die man den Elementen der Ontologie zuordnet, auch semantisch zu einander passen.

Nach Beachtung obiger Aspekte werden Daten miteinander vergleichbar und in Kontext zueinander gebracht. Wo genau in einer Softwarelösung die Ontologie eingesetzt wird, spielt dabei auch eine Rolle. In der Publikation [6] gehen die Autoren auf die Frage ein, wie nun Daten mit Ontologien verbunden werden können. Eine Möglichkeit besteht darin, ein globales Datenbankschema zu erstellen, um den/den NutzerInnen einen zentralen Zugriffspunkt zu geben. Dazu wurde von den Autoren die Umsetzung über Ontologien als sinnvoll befunden.

Allerdings bestehen Unterschiede in der Erstellung von Datenbank-Schemata und der Erzeugung und Verwendung von Ontologien. Im Vergleich zu Schemata sind Ontologien durch z.B. Relationen darauf ausgelegt, reichere Semantik bereitzustellen. Durch den Einsatz von Ontologien und passenden Zuordnungen ist es außerdem möglich, Integrationsprobleme zu lösen und die Daten mit einander in Verbindung zu bringen.

Es lassen sich zwei große Trends beim Einsetzen von Ontologien für OBDA feststellen. Beim Einen werden die Ontologien verwendet, um Anfragen oder Ergebnisse zu übersetzen, z.B. SPARQL nach SQL, was besonders für sich häufig ändernde Schemata nützlich ist. Beim Anderen werden Ontologien dazu benötigt, globale Schemata abzuleiten, was wiederum sinnvoll ist, wenn es selten zu Änderungen am Schema kommt und ein eher statischer Ansatz unkritisch ist. Auf Basis der abgeleiteten Schemata ist es dann möglich, die Datenquellen z.B. über SQL direkt abzufragen. Die Ontologie wird nach der Erstellung des Schemas nicht mehr zwangsweise benötigt, folglich ist ein Zugriff auf die Daten dann auch ohne sie möglich.

Das Zusammenspiel zwischen Ontologie und Schema wird generell durch Zuordnungen ermöglicht. In der Publikation von Wache et al. [18] kommen die Autoren zu dem Schluss, dass mehrere Datenmodelle von verschiedenen Applikationen verarbeitet werden können. Die Repräsentation dieser Daten durch zugeordneten Ontologien, als eine Art gemeinsamer Ebene, dient dazu, die Information semantisch korrekt austauschbar zu machen. Es geht in diesem Ansatz also darum, durch den Einsatz mehrerer Ontologien die mit ihnen verbundenen Datenquellen miteinander vereinbar zu machen. Aus diesem Grund ist in dieser Ebene eine Zuordnung zwischen Ontologien notwendig. In [6] werden drei Ansätze bei der Integration von Ontologien vorgestellt:

1. **Der Globale Ontologie-Ansatz**, wie in Abbildung 17(a) dargestellt, verfügt über eine integrierte Ontologie, die alle Datenquellen beschreibt. Jegliche Anfragen werden über diese globale Ontologie abgewickelt.
2. **Der Multiple Ontologien-Ansatz**, wie in Abbildung 17(b) dargestellt sieht folgendermaßen aus: Jede Datenquelle wird durch ihre eigene lokale Ontologie dargestellt. Zuordnungen zwischen den Ontologien müssen aufgestellt werden. Anfragen über die integrierten Daten geschehen über die Zuordnungen auf den lokalen Ontologien.
3. **Der Hybrid-Ontologie-Ansatz**, wie in Abbildung 17(c) dargestellt, in dem die Daten jeder Quelle durch eine lokale Ontologie beschrieben werden. Es existiert ein gemeinsames Vokabular zwischen den Ontologien, um Begriffe zwischen den Ontologien austauschbar zu machen.

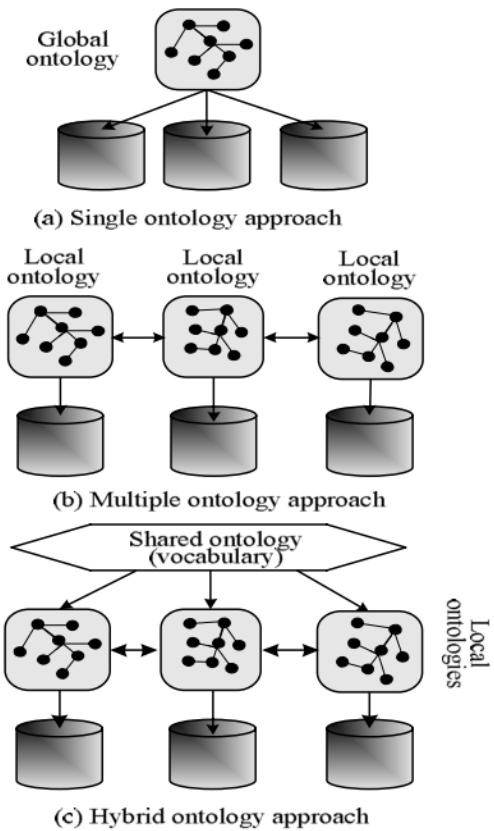


Abbildung 17 Ansätze der Integration von Ontologien. Quelle: [6]

Für die Ansätze 2. und 3. ist es notwendig, Ontologien miteinander zu verknüpfen, wie in Abbildung 17(b, c) dargestellt. Deshalb gibt es hier einige Bestandteile die beachtet werden müssen, um die Kompatibilität zwischen diesen zu gewährleisten: der Kontext der Ontologien bzw. deren Domänen, das Maß der Allgemeinheit, Abstraktion bzw. Genauigkeit und die Ontologie-Sprache, in der sie beschrieben sind.

Aktuelle kommerzielle Lösungen zur Integration und Aggregation von verschiedenen Daten sind meistens „Spezialanfertigungen“, die für spezifische Aufgaben verwendet werden. Allgemeine Lösungen wie die Frameworks, die im Abschnitt 3.3.2 vorgestellt werden, sind eher selten. Die Software ist zwar für den jeweiligen Einsatzzweck gut geeignet, möchte man das System allerdings anpassen oder umbauen, stößt man schneller an dessen Grenzen. In diesen Spezialanfertigungen wird zwar oft Wissen aus Ontologien verwendet, über die auf Daten zugegriffen wird, dieses Wissen wird aber von den Entwicklern fest in die Software eingebaut. Auch in diesem Fall kann man daher von OBDA sprechen – sofern die Software auf einer Ontologie basiert. Nachteil beim Einbauen des Zugriffs auf Datenquellen in eine Software ist, dass es sich um einen sehr statischen Ansatz handelt. Das hat zur Folge, dass Änderungen an den Datenquellen oder der Ontologie, die als Schablone für die Software verwendet wird, nur mit großem Aufwand vorgenommen werden können.

Es besteht bei allen Ansätzen des OBDA eine Gemeinsamkeit darin, dass die Zuordnungen zwischen Ontologie und Datenquellen einen großen Aufwand bedeuten. Gerade die Ableitung von Schemata aus Ontologien ist bei einem statischen Ansatz zeitaufwändig. Es gibt erste Ansätze, OBDA weiter zu automatisieren. Ein solches System zur Verringerung des Aufwands wird in der Publikation [16] vorgestellt. Die Idee beruht auf der Integration der verschiedenen Datenquellen durch automatisches Ableiten von Schemata und Datenbankskripten aus einer Ontologie. Durch die Nutzung von

Algorithmen lassen sich viele Schritte, die bei der Integration vorher mühsam manuell von einem Experten umgesetzt wurden (z.B. Datenbank-Schema-Entwicklung), auf Basis eines einmalig erstellten Modells automatisieren. Die Autoren stellen hierfür ein Framework mit dem Namen „Health-Connect“ vor, welches auf Basis von manuell erstellter ER-Modelle der unterliegenden Daten und deren Zusammenhänge, automatisiert, die benötigten Schema-Übersetzungen und die Extraktion erstellt und durchführt. Ein solches System hat den großen Vorteil, dass es zwar statisch ist, aber dennoch leicht an veränderte Daten angepasst werden kann. Gerade das ist im Klinik-Bereich ein durchaus wichtiger Punkt, da sich durch neue Technologien immer mehr Daten erheben lassen. Das wiederum bedeutet auch, dass diese neue Datenquelle mit bestehenden Systemen und deren Daten kompatibel gemacht werden muss. Die dafür notwendigen Schritte können mit dem von den Autoren vorgestellten Werkzeug mit vergleichsweise wenig Aufwand eingepflegt werden. Manuelle Zuordnungen werden auf ein Minimum reduziert. Der nun notwendige Prozess zur Integration lässt sich in vier Punkte aufteilen:

1. Datenmodellierung
2. Datenbank-Schema Erzeugung
3. Zuordnungs-Regel Erzeugung
4. Abfrage Transformation

In der Datenmodellierung wird ein konzeptionelles Datenbank-Schema auf Basis einer klinischen Ontologie in Form eines ER-Modells manuell erzeugt. Das ER-Modell stellt dabei den gesamten Aufbau und die Relationen zwischen den Daten auf einer hohen Ebene dar - dem Aufbau der Ontologie nachempfunden). Ergebnis der Modellierung ist die Repräsentation des ER-Modells als XML-Datei. Mit Hilfe der Extensible Stylesheet Language (XSLT²²), einer Turing-vollständigen Sprache, die mit Hilfe von sog. XSLT-Stylesheets aus einem XML-Dokument Text- oder sogar Binärdateien erzeugen kann, wird dann aus der XML-Datei des ER-Modells ein relationales Datenbank-Modell erzeugt. So wird aus dem Datenmodell ein relationales Modell, welches später zur Ableitung von datenbankspezifischen Schema-Skripten herangezogen werden kann. Im nächsten Schritt, der Zuordnungs-Regel-Erzeugung, wird die Unified Medical Language System (UMLS)²³ verwendet, um Verbindungen zwischen der Ontologie und den Daten zu finden. Damit die Anfragen korrekt verarbeitet werden können, ist es notwendig, aus dem erzeugten relationalen Datenbank-Modell ein herstellerspezifisches Datenbank-Skript (z.B. für MySQL) zu generieren. Dazu bedienen sich die Autoren eines weiteren XSLT Stylesheets, je nach Hersteller, und erhalten so ein Skript um die Datenbank aufsetzen zu können.

²² <https://www.w3.org/TR/xslt>

²³ <https://www.nlm.nih.gov/research/umls/>

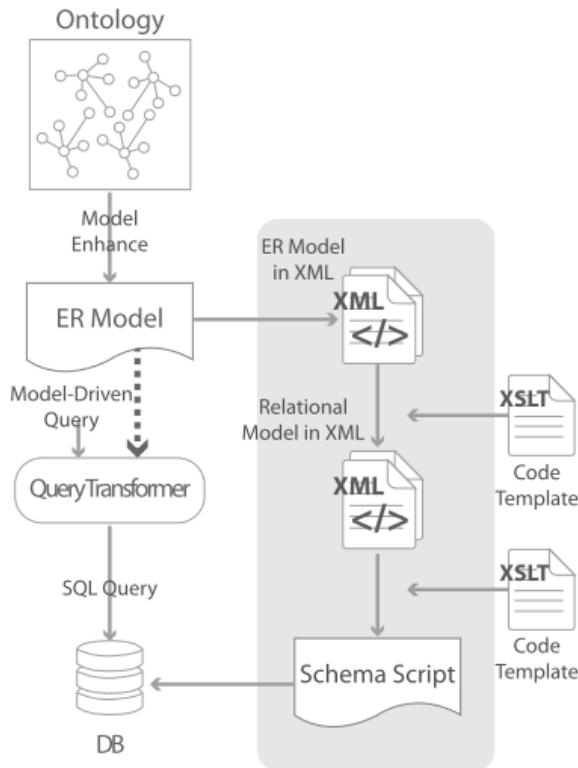


Abbildung 18 Schematische Darstellung des modellgetriebenen Ansatzes zur Datenintegration mittels Ontologien. Quelle: [16]

Wird nun eine Anfrage an das System gestellt, so wird aus dieser zunächst eine herstellerspezifische Anfrage abgeleitet, die dann wiederum auf das DBMS angewendet wird und so die benötigten Daten abgreift. Zum besseren Verständnis wird der beschriebene Prozess in Abbildung 18 dargestellt.

Das von den Autoren vorgestellte Framework, Health Connect, zur Integration von Daten ist nicht öffentlich verfügbar. Dennoch ist es ein gutes Beispiel für die Verwendung einer Ontologie und das Ableiten von passenden Übersetzungs-Schemata. Die Autoren zeigen jedoch mit ihrem Beitrag auf, wie sich die Landschaft der Datenintegration über Ontologien in den kommenden Jahren verändern könnte. In der Zukunft könnte ein solches System viele der aktuellen Probleme (z.B. Heterogenität und Komplexität bei den Zuordnungen) lösen. Gerade in einer umfangreichen Applikation könnte so für mehr Dynamik und weniger Zeitverlust durch Wartungsarbeiten gesorgt werden, da es die notwendigen Prozesse zur Integration der Datenquellen auf einer höheren Ebene abstrahiert und es damit Experten Arbeit abnimmt.

Ein weiterer OBDA-Ansatz ist es, die notwendigen Datenquellen vollständig in die Ontologie einzubetten. Auch hier müssen wieder Zuordnungen verwendet werden um die Ontologie mit den Datenquellen in Verbindung zu bringen. Aus dieser Information kann dann eine Ontologie erzeugt werden, die Ontologie und Datenquellen vereint. Das hat zur Folge, dass sich die Ontologie durch das Einbauen von zusätzlichen Informationen stark vergrößern kann. Außerdem ist der Ansatz als statisch zu bezeichnen, da Veränderungen an den Daten aus den Datenquellen erst explizit eingebaut werden müssen. Das direkte Einbauen von Daten in die Ontologie bringt so ein paar Schwierigkeiten mit sich,

die aber durch keine echten Vorteile ausgeglichen werden: Die Zuordnungen sind genauso notwendig wie bei anderen Ansätzen und die Daten in der Ontologie aber am Ende nicht aktuell.

Der häufigste Ansatz bei OBDA ist das Verknüpfen der Ontologie mit einer relationalen Datenbank (z.B. SQL) als Datenquelle. Um hier die Verbindung zwischen Ontologie und Datenquelle herzustellen, wird Query Rewriting eingesetzt. Dabei wird aus einer Abfrage über die Ontologie eine der Datenquelle angepasste Abfrage (meist SQL) erzeugt. Dieser Ansatz wird auch in allen im Abschnitt 3.3.2 beschriebenen Frameworks verwendet. Der Ansatz ist schematisch wie folgt zu verstehen (Abbildung 19):

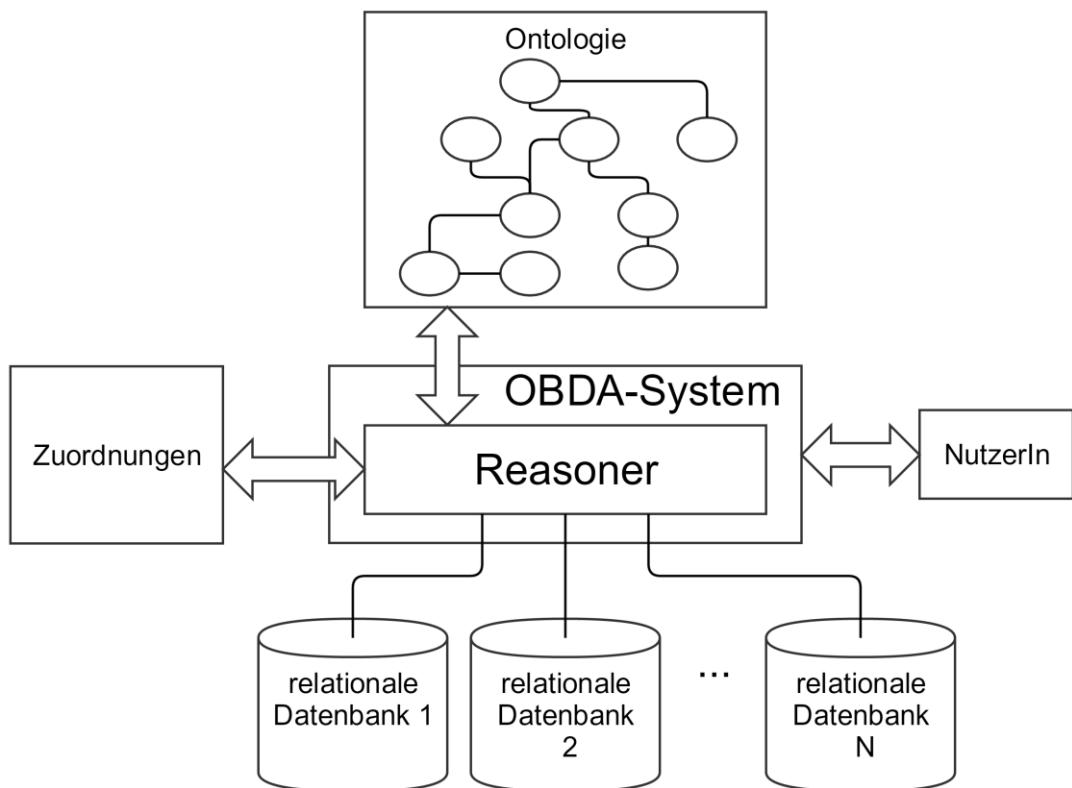


Abbildung 19 OBDA über Reasoner mit Query Rewriting

Wieder ist eine Ontologie gegeben, die mit verschiedenen Datenquellen verbunden werden soll. Zuordnungen, die separat erstellt werden müssen, stellen eine Verbindung zwischen den Datenquellen und der Ontologie selbst her. Stellt der/die NutzerIn eine SPARQL-Abfrage zum Beziehen von Daten an das OBDA-System, erzeugt der Reasoner des Frameworks auf Basis der Zuordnungen zwischen Ontologie und Datenquellen eine für die Datenquelle angepasste SQL-Abfrage, um die Daten aus der Datenquelle (meist eine relationale Datenbank) zu extrahieren. Hierbei wird Query Rewriting verwendet, um SPARQL-Abfragen der Ontologie in SQL-Abfragen umzuwandeln und um passgenaue Abfragen zu generieren. Die für sich agierenden Datenbanksysteme verarbeiten dann die erhaltene SQL-Abfrage vom Reasoner und geben die abgefragten Daten zurück. Der Reasoner kann dann die Daten in SPARQL-Abfragen des/der NutzerIn

einbeziehen. Das Besondere an diesem Ansatz ist, dass die Abfragen und der Datenzugriff zum einen zur Laufzeit geschehen und die Zuordnungen und Datenquellen prinzipiell unabhängig voneinander sind. Es wird zum anderen durch das Umschreiben der Abfragen nur auf Informationen zugegriffen, die zum Beantworten der Abfrage des/der NutzerIn benötigt werden. Ein weiterer Unterschied zu manchen der vorangegangenen Ansätze besteht darin, dass die ursprünglichen Datenquellen in keiner Form verändert werden.

Weiterhin ist es möglich, eine eigene Umschreibungsfunktionalität im Reasoner zu entwickeln, durch die dann eine eigens entwickelte Datenquelle angebunden werden kann. Dadurch ist es möglich spezifische Datenquellen einzubinden, die z.B. nicht über SQL abgefragt werden.

3.2 Bewertungskriterien der Frameworks

Die in Kapitel 4.2 beschriebene Bewertung der Frameworks setzt die Definition der Bewertungskriterien der Frameworks voraus. Das Ziel der Bewertung ist die Feststellung der Eignung eines Frameworks für die Entwicklung von CION.

Tabelle 6 Bewertungskriterien für die Bewertung der Frameworks

ID	Kriterium	Begründung
K1	Das Framework wird aktuell aktiv weiterentwickelt	Bei Fehlern, die bei der Entwicklung auftauchen, ist so die Chance gegeben, dass diese in späteren Versionen beseitigt sind.
K2	Umfangreiche und ausführliche Dokumentation des Frameworks ist vorhanden.	Der Grad der Dokumentation spielt vor allem bei der Entwicklung des Prototyps und ggf. bei der späteren Entwicklung des CION eine Rolle. Ist das Framework kaum oder nur minimal beschrieben, ist es schwer, die vorhandenen Funktionen zu beurteilen bzw. bei auftretenden Unklarheiten oder Schwierigkeiten eine Lösung zu finden.
K3	Aktive Community	Dieser Punkt ähnelt dem Kriterium der aktiven Entwicklung, da durch eine aktive Community Fehler schneller beseitigt werden können.
K4	Das Framework ist ausgereift und stabil.	Bei dem Framework soll es sich nicht um eine frühe Version, sondern um eine möglichst erwachsene und ausgereifte Lösung handeln. Aufschluss darüber geben vor allem folgende Punkte: <ul style="list-style-type: none"> • Versionsnummer und Beschreibung • Codezeilen • Vorhandensein von Unit Tests • Beginn der Programmierung und letzte Aktualisierung • Informationen zum Framework in Publikationen oder der offiziellen Webseite
K5	Das Framework greift zur Laufzeit auf Daten zu.	Die Daten im Prototyp sollen so aktuell wie möglich sein, ein Zwischenspeichern ist unerwünscht.
K6	Es ist möglich das Framework über eine Schnittstelle (CLI, API, Endpoint) zu programmieren	Da zwischen der Nutzeroberfläche und dem Framework eine Verbindung vorhanden sein muss, um dieses aus der Oberfläche zu steuern, ist eine Schnittstelle gewünscht.

	bzw. Abfragen abzusetzen.	
K7	Das Framework unterstützt mehrere Datenquellen.	Mehrere unterschiedliche Datenquellen sind anzubinden, es muss daher möglich sein, diese auch alle integrieren zu können.
K8	Das Framework setzt auf Standards bei der Zuordnungssprache sowie den unterstützten Ontologie-Typen.	Standards sind wichtig für den weiteren Verlauf im Projekt und der möglichen Erweiterbarkeit des Systems, da so ermöglicht wird, dass die erstellten Konfigurationsdaten auch wiederverwendet werden können.
K9	Das Framework wird mit einer projektfreundlichen Lizenz (nicht-kommerziell) ausgegeben.	Um das Framework später auch produktiv einsetzen zu können, ist es wichtig, dass dessen Lizenz so ausgelegt ist.
K10	Der Quelltext des Frameworks ist öffentlich.	Damit im Fall von notwendigen Änderungen diese auch selbstständig vorgenommen werden können, ist ein offener Quelltext wünschenswert.
K11	Abgefragte Daten sind aktuell.	Bei der Abfrage sollen die Datenquellen angesprochen werden, um so die Gefahr zu vermeiden, dass die Daten nicht aktuell sind.
K12	Der vom Framework verwendete Reasoner unterstützt OBDA.	Ohne einen Reasoner, der OBDA unterstützt, ist es nicht möglich, die SPARQL-Abfragen mit den Daten zu beantworten.
K13	OWL Ontologien werden unterstützt.	Das Format der CIOx-Ontologie ist OWL, daher ist es sinnvoll, wenn dieses Format ohne weitere Konvertierung vom Framework unterstützt wird.
K14	SPARQL-Abfragen können verwendet werden.	Das Abfragen über SPARQL-Abfragen ist ein Kernpunkt, der benötigt wird, um Informationen aus der Ontologie abzurufen.
K15	Integration von Excel und Word als Datenquellen	Im Projekt geht es vorrangig um das Beziehen von Daten aus Excel und Word-Dokumenten. Es ist daher wichtig, dass deren Integration unterstützt wird.
K16	Zugriff auf Daten über eine Ontologie mittels Zuordnungen wird unterstützt.	Ein wichtiger Punkt, da gewünscht ist, dass es möglich ist, die Zuordnungen auf Wunsch schnell anzupassen.

3.3 Bestandsaufnahme gefundener Frameworks

Bei der Betrachtung der vorher besprochenen Ansätze sind vor allem die Erkenntnisse und Entwicklungen der letzten Jahre wichtig. Möchte man heute ein System, welches OBDA verwendet, aufbauen, so bestehen zwei Möglichkeiten: Ein vorhandenes Framework nutzen oder ein eigenes System entwickeln. Nicht alle in Publikationen beschriebenen Lösungen sind frei und offen zugänglich, manche sogar nur als Konzepte zu verstehen. Zur Identifikation von Frameworks zum Umsetzen eines OBDA Systems ist zunächst eine Bestandsaufnahme der verfügbaren Frameworks von Nöten.

In diesem Teil der Literaturrecherche werden die Fragestellungen

„Welche Frameworks existieren, um ein OBDA System zu entwickeln?“ und

„Welche Eigenschaften haben die gefundenen Frameworks?“ beantwortet.

Zum Auffinden möglicher OBDA Frameworks und dem Aufdecken der potenziellen Verwendbarkeit, werden unterschiedliche Quellen verwendet: Zum einen gibt es Referenzen und Beschreibungen zu Frameworks in den im ersten Teil der Literaturrecherche genannten Quellen, zum anderen dienen Maillisten, Foren und Google-Suchen als Quellen:

- Referenzen auf Frameworks aus Literatur:
 - Ontop
 - Mastro
 - Virtuoso
- Maillisten:
 - Apache Jena²⁴
 - Virtuoso-users²⁵
- Foren:
 - Stardog-users²⁶
 - Ontop²⁷

In der vorangegangenen Literatur-Suche sind viele Frameworks gefunden worden. Bei der jetzigen Recherche via IEEE, ACM und GoogleScholar und Google werden die folgenden Suchbegriffe verwendet.

- obda framework
- ontology based data access framework
- FRAMEWORK_NAME features

Diese haben das Ziel, zum einen passende Frameworks zu finden und zum anderen Informationen zu diesen zu finden:

Leider ist durch die Suche mit den Suchmaschinen IEEE, ACM und GoogleScholar kein neues Framework aufgetaucht. Zum Auffinden von Funktionen der Frameworks haben sich eine simple Google-Suche, sowie das genaue Lesen der Pages, als am aufschlussreichsten herausgestellt.

Das Ergebnis dieser Recherche ist eine Liste von OBDA Frameworks, die in Tabelle 9 und Tabelle 10 dargestellt sind.

Zunächst wird jedes der identifizierten Frameworks kurz vorgestellt und dann in Kapitel „Ergebnisse der Betrachtung der OBDA Frameworks“ in Bezug auf das Projekt bewertet.

Alle hier vorgestellten Frameworks sind in Java geschrieben, plattformübergreifend und quelloffen (wenn nicht ausdrücklich anders festgestellt).

3.3.1 Apache Jena²⁸

Das Apache Jena Framework ist ein weit verbreitetes Framework zum Verarbeiten von Ontologien und RDF-Daten. Die aktuellste Version, 3.0.1., wurde am 8. Dezember 2015 veröffentlicht.

²⁴ https://jena.apache.org/help_and_support/

²⁵ <https://lists.sourceforge.net/lists/listinfo/virtuoso-users>

²⁶ <https://groups.google.com/a/clarkparsia.com/forum/#!forum/stardog>

²⁷ <https://groups.google.com/forum/#!topic/ontop4obda>

²⁸ <https://jena.apache.org/>

Jena unterscheidet sich im Vergleich zu den anderen Frameworks dahingehend, als dass es nicht explizit für den Einsatz als ein OBDA System entwickelt wurde. Es ist vielmehr ein allgemeines Framework für den Umgang mit Ontologien und verwandten Technologien wie RDF. Der Grund warum es hier aufgelistet ist, ist der, dass es in Teilen Funktionalität besitzt die OBDA prinzipiell ermöglichen würden. So ist z.B. Datenintegration mittels Triplestores und SQL möglich. Gute Voraussetzungen für den Einsatz von Jena stellen die aktive Community und regelmäßige Aktualisierungen dar.

Für das Projekt wichtige Funktionen, die das Jena Framework bereitstellt, werden hier angesprochen. Zuerst sei die Fähigkeit von Ontologien erwähnt, mittels SPARQL abzufragen. Es besteht außerdem die Möglichkeit, externe Reasoner zu verwenden, wodurch eine bessere Anpassung an unterstützte Ontologien möglich ist. In Bezug auf die unterstützten Formate ist Jena sehr vielseitig: sowohl OWL, N3²⁹, RDF als auch Turtle³⁰ werden unterstützt. Praktisch ist auch, dass das Framework über ein Kommandozeileninterface (CLI) gesteuert werden kann. Hierdurch ist es möglich Jena auch außerhalb einer Java Applikation einzusetzen.

3.3.2 OBDA-Frameworks

Die Folgenden Frameworks können allgemein als OBDA-Frameworks bezeichnet werden, da diese explizit für den Umgang mit Ontologien zum Einsatz für OBDA entwickelt wurden.

3.3.2.1 D2rq³¹

Die Entwicklung von D2RQ begann im Jahr 2004 als eines der ersten Systeme für OBDA. Die FU Berlin und DERI sind die Hauptentwickler, aber auch andere Firmen haben zur Entwicklung beigetragen. D2rq stellt einen SPARQL-Endpoint zur Verfügung und kann über die Jena API und ein CLI gesteuert werden. Es wird kein Reasoner vom System unterstützt. Dafür ermöglicht das System über Query Rewriting einen Zugriff auf Daten relationaler Datenbanken. Dabei erfolgt das Mapping in den Sprachen D2RQ Mapping, R2RML.

Zum aktuellen Zeitpunkt ist die neuste Version 0.8.1, und es hat seit fast 4 Jahren keine Aktualisierung mehr gegeben. Die von der Software bereitgestellten Funktionen sind Query Rewriting und Zuordnungen.

3.3.2.2 Virtuoso³²

Bei Virtuoso [19] handelt es sich um einen Relationalen Datenbank Server, der zudem den Zugriff auf verschiedene Datenquellen über eine Ontologie unterstützt. Der Server ist für verschiedene Plattformen verfügbar, allerdings nicht in Java geschrieben. Daher sind das Ausrollen und die Konfiguration mancher Funktionen, wie der Zugriff auf Excel-Dateien auf anderen Plattformen als Windows unterschiedlich. Die aktuelle quelloffene Version 7.2 ist unter der GPL veröffentlicht und kann unter Github³³ eingesehen werden. Die quelloffene Version ist in ihren Funktionen eingeschränkt. Der volle Funktionsumfang ist den kommerziell verfügbaren „Enterprise“-Variante

²⁹ <https://www.w3.org/DesignIssues/Notation3.html>

³⁰ <https://www.w3.org/TR/turtle/>

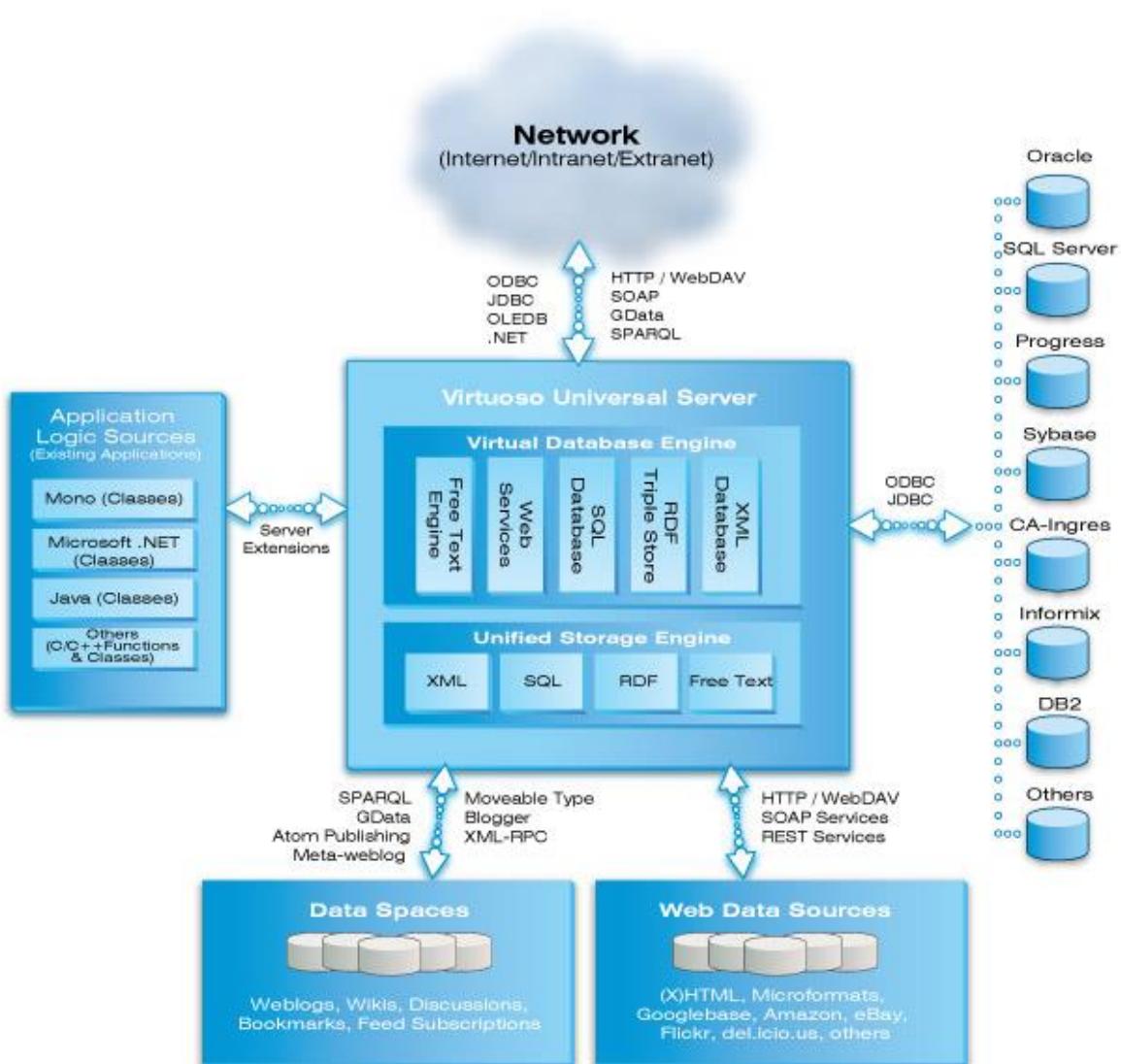
³¹ <http://d2rq.org/>

³² <http://virtuoso.openlinksw.com/>

³³ <https://github.com/>

vorbehalten. Das bezieht sich vor allem auf das Anbinden von JDBC-Quellen, für die in der freien Variante ein zusätzliches Modul gekauft werden muss. Zudem ist das Anbinden mehrerer Datenquellen nicht möglich, da die Föderationsfunktion ebenfalls nicht in der freien Version der Software enthalten ist. Bezuglich Standards ist Virtuoso gut ausgestattet: Als Schnittstellen werden eine JenaAPI, Sesame API und ein SPARQL-Endpoint angeboten. Außerdem werden Zuordnungen in der R2RML-Sprache unterstützt, die intern in eine gleichwertige Sprache übersetzt werden. Als Ontologie-Formate werden OWL und RDF unterstützt. In der kommerziellen Version werden Anbindungen an verschiedene und mehrfache Datenquellen unterstützt, das ist in der freien Version nur für bestimmte Quellen wie z.B. SQL und Triplestores möglich, nicht aber an Excel-Dateien.

Die nachfolgende Grafik (Abbildung 20) zeigt schematisch den Einsatz des Frameworks in einem Gesamtszenario:



*Abbildung 20 Schema Virtuoso Funktionen und Einsatzszenario. Quelle:
<http://virtuoso.openlinksw.com/>*

Rechts ist eine Auflistung der Komponenten, die über eine JDBC- oder ODBC-Schnittstelle, welche in ihrer Funktionsweise sehr ähnlich sind, mit Virtuoso verbunden werden können. Eine Anbindung der Komponenten über ein Netzwerk ist ebenfalls möglich (siehe oben). Unten werden Web-Datenquellen aufgezeigt, wie z.B. Wikis, oder Blogs, aber auch explizite Webseiten wie z.B. Amazon. Links sind Applikationsservererweiterungen aufgelistet. Im Inneren des Virtuoso Universal Servers sind die Komponenten „Virtual Database Engine“ sowie „Storage Engine“ angesiedelt, die steuern wie die bezogenen Daten gespeichert werden.

3.3.2.3 *Stardog*³⁴

Dieses Framework wird seit 2010 von Complexible Inc.³⁵ vertrieben. Es handelt sich primär um eine kommerzielle Lösung, die aber auch als eine in ihren Fähigkeiten reduzierte „Community“-Version angeboten wird. Konkret wird z.B. die Anzahl möglicher Nutzer auf 4 beschränkt, es dürfen nur 10 Datenbanken angebunden werden und eine OWL Ontologie darf nur 10.000 Axiome³⁶ enthalten. Erst die Enterprise-Lösung ist unbeschränkt. Das Framework erlaubt eine Anbindung an verschiedene relationale Datenbanksysteme. Die Zuordnungen können im R2RML Standard, aber auch im eigenen Format „Stardog Mapping Syntax³⁷“ erfolgen, welche in einander übersetzt werden können. Ein SPARQL Endpoint³⁸ ist vorhanden. Außerdem werden verschiedene APIs angeboten, unter anderem auch dotNetRDF, welches eine Verwendung in C# erlaubt. Es werden weiterhin ein CLI sowie Java APIs angeboten. Der Reasoner unterstützt OWL 2 QL, RL, EL sowie RDFS. Die Dokumentation ist umfangreich und die Community recht aktiv.

3.3.2.4 *Mastro*³⁹

Mastro [11], [12] ist zusammen mit Ontop das aktuellste OBDA System, das frei zugänglich ist. Dieses System wird von der Sapienza Università di Roma⁴⁰ seit 2006 entwickelt. Reasoning und SPAQRL Abfragen sind über Ontologien der OWL 2 QL Sprache möglich. Standardmäßig kann Mastro an MySQL, PostgreSQL und MS SQL Server angebunden werden. Sofern ein JDBC Treiber vorhanden ist, ist auch eine Anbindung an föderierte Datenbanksysteme möglich. Zuordnungen werden in der Standard-Sprache R2RML ausgedrückt und sind damit mit anderen Frameworks weitgehend austauschbar.

Obwohl das System sehr ausgereift ist, gibt es einige Nachteile: Die Lizenz für die Nutzung des Systems ist von den Entwicklern nicht weiter definiert. Außerdem fällt die Dokumentation mit Installationshinweisen und einem kleinen Beispiel sehr mager aus. Ein sinnvoller Einsatz des Systems über einen Prototyp hinaus ist daher fraglich.

³⁴ <http://stardog.com/>

³⁵ <http://complexible.com/>

³⁶ <http://stardog.com/#plans>

³⁷ http://docs.stardog.com/#_stardog_mapping_syntax

³⁸ http://docs.stardog.com/#_sparql_protocol

³⁹ <http://www.dis.uniroma1.it/~mastro/>

⁴⁰ <http://www.uniroma1.it/>

3.3.2.5 *Ontop*⁴¹

Das Framework [13], [7] wurde an der Freien Universität Bozen⁴² ab 2010 entwickelt. Es zeichnet sich durch Reasoning-Fähigkeiten über OWL 2 QL Ontologien und über den „Quest“ Reasoner [17] aus.

Der in Ontop eingesetzte Reasoner „Quest“ setzt den PerfectRef Algorithmus in einer veränderten und verbesserten Form [17] zum Umschreiben von Abfragen von SPARQL nach SQL ein. Die Verbesserungen (weniger verschachtelte SQL-JOINS⁴³) haben zur Folge, dass die erzeugten SQL-Abfragen wesentlich kürzer sind und von Datenbanksystemen besser verarbeitet werden.

Allgemein gesprochen erfolgt die Anbindung an die Datenbanken durch das Verknüpfen der Elemente in der Ontologie (z.B. Klassen und Eigenschaften) und deren Äquivalent in der Datenquelle. Hieraus entsteht ein virtueller RDF Graph, der der Ontologie mit den Daten aus der Quelle entspricht. Dieser wird als virtuell bezeichnet, da er nur zur Laufzeit mit Daten gefüllt ist und diese dynamisch aus den Quellen abgerufen werden. Die SPARQL-Abfragen, die zum Beziehen von Informationen aus der Ontologie dienen, werden dann, den Zuordnungen entsprechend, in SQL-Abfragen der angebundenen Quelle übersetzt.

Die in Abbildung 21 gezeigten Schichten des Ontop Frameworks werden im Folgenden von unten nach oben genauer erläutert:

3.3.2.5.1 Eingabe-Schicht (Inputs)

- Ontologien werden in den gängigen Sprachen OWL 2 QL und RDFS unterstützt.
- Zuordnungen werden mit R2RML definiert. Darüber hinaus kann eine native Sprache von Ontop verwendet werden, die von und in R2RML konvertiert werden kann. Eine Zuordnung ist dabei so aufgebaut, dass eine Quelle in Form einer SQL-Anfrage für eine Datenbank mit einem Ziel verbunden wird. Das Ziel ist wiederum ein aus Werten der Quelle definiertes RDF Tupel. Um Konflikte zu reduzieren, ist es möglich, Zuordnungen so zu definieren, dass gleiche Objekte vom System als identisch erkannt werden.
- Anfragen in SPARQL 1.0 sowie in Teilen 1.1 werden unterstützt.
- Datenbanken können, sofern ein JDBC Treiber vorhanden ist, von Ontop abgefragt werden. Sowohl gängige kommerzielle (DB2, Oracle, MS SQL Server), als auch quelloffene (PostgreSQL, MySQL, H2, HSQL) relationale Datenbanken werden dabei unterstützt. Auch föderierte Datenbanksysteme wie Teiid oder Exareme⁴⁴ können angebunden werden. Somit ist es möglich, auf Daten, die z.B. von Teiid verarbeitet werden können, zuzugreifen. Zum aktuellen Zeitpunkt gibt es bereits eine Anbindmöglichkeit an Excel-Dateien – für die anderen Daten (Word und SharePoint) ist die Implementierung eines eigenen Plug-Ins nötig. Mehrere Datenquellen können nur über eine Föderationssoftware angebunden werden, die diese zusammenführt, da Ontop lediglich eine Datenquelle erlaubt.

⁴¹ <http://ontop.inf.unibz.it/>

⁴² <http://www.unibz.it/de/>

⁴³ http://www.w3schools.com/sql/sql_join.asp

⁴⁴ <http://madgik.github.io/exareme/>

3.3.2.5.2 Ontop-Kern (Core)

Im Kern arbeitet der Reasoner[6] [10] [17], der die SPAQL Abfragen über die virtuellen RDF Graphen in SQL-Abfragen der relationalen Datenbanken übersetzt. Zudem ist dieser für Schlussfolgerungen aus den in der Ontologie gewonnen Zusammenhängen zuständig.

3.3.2.5.3 API Schicht (API Layer)

Die API-Schicht erlaubt es Entwicklern, Ontop wie eine Bibliothek zu verwenden. Hierfür werden vom Framework zwei Java APIs angeboten:

1. OWL API, welche eine Implementierung zum Bearbeiten von Ontologien ist. Die von Ontop hinzugefügten Fähigkeiten sind in der `OWLReasoner` Schnittstelle zu finden.
2. Sesame Storage And Inference Layer (SAIL), welches Reasoning und Abfragen der Ontologien ermöglicht.

3.3.2.5.4 Applikations-Schicht (Application Layer)

Nicht nur auf API-Ebene kann mit Ontop interagiert werden, sondern auch über ein schlichtes CLI in der Applikationsschicht. Außerdem sind in das Framework bereits mehrere Werkzeuge integriert, die den Aufbau eines OBDA Systems erleichtern. Hierzu zählen:

- Ein OWL API basiertes Protégé Plug-In, mit dem essenzielle Schritte, wie das Bearbeiten von Zuordnungen, das Ausführen von SPARQL-Abfragen, Konsistenz-Prüfungen der Ontologie sowie softwaregestützte Zuordnungen zwischen Ontologie und Datenbanken in einer grafischen Oberfläche ermöglicht werden. Da Protégé auch im Rahmen des SNIK-Projekts eingesetzt wird, verändert sich der Arbeitsfluss bei der Arbeit mit Ontop kaum, da man mit gewohnter Software arbeitet.
- Die Webapplikation Sesame OpenRDF Workbench ermöglicht das Administrieren von Sesame Repositories. Diese können dann als ein SPARQL Endpunkt zum Absetzen von Abfragen verwendet werden.

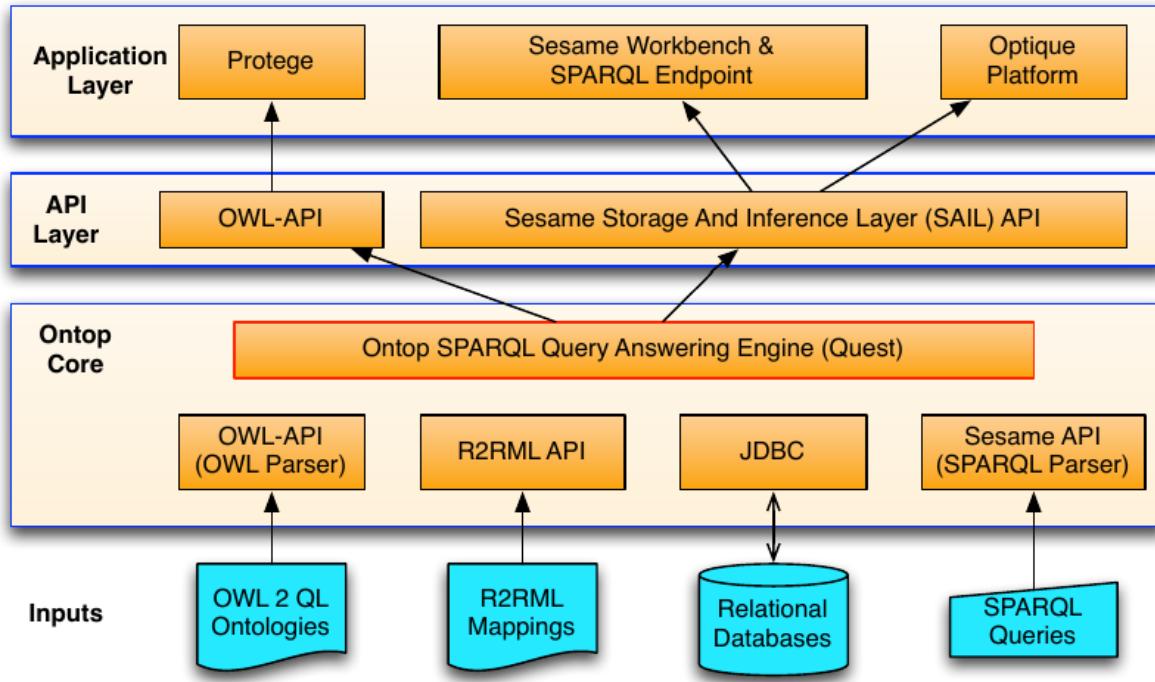


Abbildung 21 Aufbau der Schichten des Ontop Frameworks. Quelle: [13]

Die Vorteile von Ontop liegen hier auf der Hand: Es handelt sich um ein gut dokumentiertes und aktuelles Framework, das viele Funktionen bereits mitbringt. Das Plug-In würde es den NutzerInnen später mit wenig Aufwand für Umgewöhnung ermöglichen, Zuordnungen vorzunehmen und Ontop zu verwenden.

3.3.3 Teiid⁴⁵

Da im Zusammenhang mit Ontop auch das Datenföderationssystem Teiid erwähnt wird [13] und dieses von Relevanz ist, da hierdurch der Zugriff auf mehrere Datenquellen ermöglicht wird, wird an dieser Stelle separat darauf eingegangen. Bei der Entwicklung des CION und des Prototyps kann Teiid dazu eingesetzt werden einheitlichen Zugriff auf die verschiedenen Datenquellen (Word, Excel und SharePoint) zu ermöglichen.

Teiid ist ein Teil des JBoss Projekts und wird quellöffent unter der LGPL 2.1⁴⁶ vertrieben. Wie die anderen Frameworks ist es in Java geschrieben und zur Laufzeit vom JBoss Application Server (AS) abhängig, der ebenfalls quellöffent und frei zugänglich ist.

Teiid vereint die Funktionalität eines Mediations- und Föderationssystems. Im Grunde handelt es sich um eine relationale Abstraktion von beliebigen Daten. Um Datenquellen an Teiid anzubinden zu können, muss ein Connector vorhanden sein, der die Abstraktion vom Ursprungsmodell der Daten in ein relationales Modell durchführt. Ist ein Connector für eine Datenquelle entwickelt, kann Teiid diese als eine relationale virtuelle Datenbank (VDB) an eine andere Applikation weiterreichen. Das geschieht über die JDBC API.

Abbildung 22 zeigt den Aufbau schematisch:

⁴⁵ <http://teiid.jboss.org/>

⁴⁶ <http://www.gnu.de/documents/lgpl-2.1.de.html>

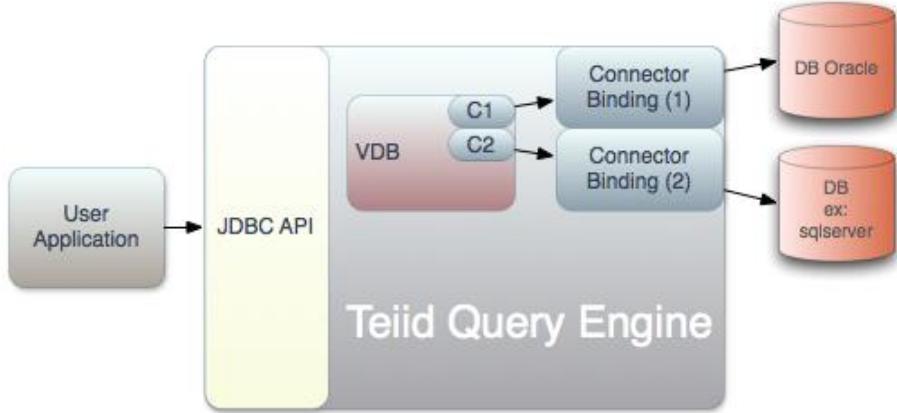


Abbildung 22 Schematische Darstellung einer auf Teiid basierten virtuellen Datenbank (VDB)
Quelle: <http://teiid.jboss.org/basics/virtualdatabases/>

Sind die Datenquellen über eine VDB an die darüber liegende Applikation (im Bild links) exponiert, können die darin enthaltenen Daten mittels SQL-Abfragen extrahiert werden. Zur Umwandlung der Abfragen muss ein Translator (zu Deutsch „Übersetzer“) entwickelt werden. Dieser führt eine Übersetzung von SQL in die Abfragesprache bzw. Zugriffsmethode der Quelle (z.B. Oracle, Excel, Cassandra, Amazon SimpleDB, etc.) durch.

Auch die physische Datenquelle kann je nach Umgebung eine andere sein. Es ist offen, ob die Daten, auf die zugegriffen wird, auf einer Festplatte, einem NAS, einem Webdienst o. Ä. abgerufen werden müssen. Um den Zugriff zu erlauben, kann für diese Situationen ein eigens angepasster Resource Adaptor (zu Deutsch „Ressourcen Anpasser“) entwickelt werden.

Es ist möglich die Datenmodelle der Datenquellen selbst zu verändern: Hierfür kann der Teiid Designer⁴⁷ verwendet werden. Durch den Designer ist es möglich aus den Datenquellen virtuelle Datenbanken nach eigenen Vorstellungen zu erzeugen. Das kann hilfreich sein, wenn z.B. statt vielen kleinen Tabellen in einer Datenbank nur eine gemeinsame Tabelle entstehen soll, die die kleinen Tabellen zusammenfasst.

⁴⁷ <http://teiddesigner.jboss.org/>

4 Planung und Gestaltung einer Lösung

Im Folgenden wird auf den Konzeptions- und Entscheidungsprozess eingegangen, der die Entwicklungen eines Prototyps beschreibt.

Um ein geeignetes Konzept für die Umsetzung eines Prototyps erstellen zu können, müssen zunächst die Anforderungen genauer spezifiziert werden, die dieser erfüllen muss. Aus den dokumentierten Anforderungen kann dann ein geeignetes Anwendungsdesign abgeleitet werden, das anschließend implementiert werden kann.

4.1 Software Anforderungs Spezifikation (SAS)

In diesem Kapitel wird das SAS-Dokument [20] beschrieben, welches sich in seiner Gliederung weitgehend an den Empfehlungen des IEEE orientiert. Die Überschriften sind den Empfehlungen entnommen: „Produkt“ ist in diesem Fall mit „Prototyp“ gleichzusetzen. Da es bei dem Projekt eine überschaubare Menge an Anforderungen gibt, die für ein solches Dokument verwendbar sind, ist eine genaue Abarbeitung des IEEE Standards nicht gerechtfertigt.

4.1.1 Scope

Innerhalb des DFG-Projektes „Semantisches Netz des Informationsmanagements im Krankenhaus (SNIK)“ soll die auf einem semantischen Netz basierende entscheidungsunterstützende Software CIO-Navigator (CION) für Krankenhaus-CIOs entwickelt werden. Zur Entscheidungsunterstützung soll CION Informationen aus verschiedenen Systemen aggregieren und vernetzt in Form eines Dashboards darstellen. Als wesentliche Vorarbeit wurde im vorliegenden Projekt eine Ontologie entwickelt, die Konzepte und Relationen aus dem Informationsmanagement-Bereich des Gesundheitswesens enthält.

4.1.2 Allgemeine Beschreibung (des Softwareprodukts)

4.1.2.1 *Produktfunktionen*

Der Prototyp soll folgende Funktionen besitzen:

- Es soll möglich sein, Informationen aus der Ontologie per SPARQL abzufragen.
- Es soll möglich sein vom CIO benötigte Informationen, die in Datenquellen hinterlegt sind, zu akquirieren, aggregieren und darzustellen.
- Es soll möglich sein, weitere Datenquellen hinzufügen zu können.
- Es soll möglich sein, Datenquellen (z.B. Word, Excel, Text, etc.) mit Konzepten der Ontologie zu verknüpfen.
- Es soll durch die Verknüpfung der Ontologie mit den Daten möglich sein, Antworten auf komplexe Fragestellungen zu bekommen (Aggregation).

4.1.2.2 *NutzerInmerkmale*

Der CIO eines Krankenhauses wird der/die HauptnutzerIn des Prototyps sein. Daher kann ein akademischer Bildungsgrad vorausgesetzt werden. Zudem kann die Kompetenz im Umgang mit Softwaresystemen als überdurchschnittlich bezeichnet werden.

Eine weitere NutzerInnen-Gruppe sind diejenigen, die für die Erstellung der Zuordnungen zwischen Ontologie und Datenquellen zuständig sind. Das kann zwar auch vom CIO übernommen werden, es ist aber denkbar dafür eigene Experten zu nominieren, die mit der Aufgabe betraut sind.

4.1.2.3 Einschränkungen

Die folgenden Aspekte sind Einschränkungen für die Entwicklung, die festgelegt wurden:

- Die Nutzbarkeit spielt eine geringe Rolle, es soll lediglich ein Prototyp erstellt werden, der die prinzipielle Machbarkeit des Datenzugriffs über die Ontologie zeigt.
- Die Lösung muss nicht mit großen Ontologien (mehr als 5000 Konzepte) umgehen können.
- Performanz und Reaktionszeiten der Lösung spielen eine geringe Rolle.

4.1.2.4 Annahmen und Abhängigkeiten

Betriebssystem: Microsoft Windows

Microsoft Software im Einsatz: Microsoft Office und SharePoint

Außerdem ist auch indirekt eine Abhängigkeit zum SMB-Protokoll (Server Message Block) von Microsoft gegeben, bei dem es sich um ein Protokoll zum Zugriff auf Netzwerk-Freigaben (z.B. auf einem Server oder NAS) handelt. Des Weiteren wird über eine Netzwerkverbindung der Zugriff auf eine Webseite und SharePoint per HTTP zugegriffen.

4.1.3 Spezifische Anforderungen

Im Folgenden werden die Anforderungen an die Softwarelösung genauer spezifiziert. Bei der nachfolgenden Auflistung handelt es sich um das Anforderungsdokument, welches in UNICASE⁴⁸ erstellt wurde.

4.1.3.1 Funktionale Anforderungen

4.1.3.1.1 Funktion zum Anlegen/Löschen/Bearbeiten von Datenzuordnungen

Datenzuordnungen sollen angelegt, bearbeitet oder gelöscht werden können. Hierfür ist eine textuelle Darstellung der Zuordnungen ausreichend.

4.1.3.1.2 Darstellung von Verknüpfungen (Konzept und Datenquelle - Textuell)

Verknüpfungen zwischen Ontologie-Elementen und den Datenquellen sollen von der Applikation aus dargestellt werden können. Eine textuelle Darstellung ist hierfür ausreichend.

4.1.3.1.3 Hinzufügen/Löschen/Bearbeiten von Datenquellen

Datenquellen und deren Zuordnungen, sollen in der Applikation veränderbar, hinzufügbar und lösbar sein. Bei den Datenquellen handelt es sich um einzelne Dateien.

⁴⁸ <http://se.ifi.uni-heidelberg.de/research/projects/unicase.html>

4.1.3.1.4 Aggregation von Daten aus verschiedenen Quellen und Quelltypen

Das Aggregieren und Zusammenführen von Daten aus verschiedenen Datenquellen soll möglich sein. Dabei soll es auch möglich sein, Daten aus verschiedenen Quelltypen (Excel, Word, usw.) miteinander zu vereinen.

4.1.3.1.5 Beantwortung von Fragestellungen mittels SPARQL-Abfragen

Es sollen bestimmte Fragestellungen, die im Umfeld des CIO wichtig sind mittels SPARQL-Abfragen beantwortet werden können. Eine Übersicht über diese Fragestellungen ist in Tabelle 7 gegeben.

4.1.3.1.6 Darstellung der Aggregierten- und Rohdaten (Textuell)

Sowohl Rohdaten, als auch aggregierte Daten sollen im Prototyp textuell dargestellt werden können.

4.1.3.1.7 Unterstützung von Zugriff auf Netzwerk und SharePoint für Datenquellen

Der Zugriff auf entfernte Datenquellen, die über das Netzwerk erreichbar sind (z.B. über URLs, Netzwerk-Pfade oder in einem SharePoint) soll möglich sein.

4.1.3.2 *Nicht-Funktionale Anforderungen*

4.1.3.2.1 Fehlertoleranz gegenüber falschen Datentypen

Der Prototyp soll gegenüber falschen Datentypen, die in den Datenquellen hinterlegt sind, fehlertolerant sein. Das heißt, dass dieser nicht unkontrolliert abstürzen, sondern Ausnahmen behandeln soll.

4.1.3.2.2 Fehlertoleranz gegenüber falschen Datenzuordnungen

Der Prototyp soll gegenüber syntaktisch falschen Datenzuordnungen fehlertolerant sein. Die Datenzuordnungen werden dafür verwendet, um den Inhalt der Datenquellen zu beschreiben und einer Klasse in der Ontologie zuzuweisen. Der Prototyp soll nicht unkontrolliert abstürzen, wenn diese nicht korrekt sind, sondern die Ausnahmen behandeln.

4.1.3.2.3 Fehlertoleranz gegenüber fehlerhaften Ontologien

Der Prototyp soll gegenüber syntaktisch falschen Ontologien fehlertolerant sein. Das heißt, dass dieser nicht unkontrolliert abstürzen, sondern Ausnahmen behandeln soll.

4.1.3.2.4 Schließen der Verbindungen nach dem Schließen

Beim Beenden der Applikation sollen offene Verbindungen zu Datenquellen und von der Applikation verwendete Dateien automatisch und ordnungsgemäß geschlossen werden, um Datenverlust zu vermeiden.

4.1.3.2.5 Automatisches öffnen der Verbindungen beim Starten

Beim Starten der Applikation sollen automatisch alle benötigten Dateien und Verbindungen zu Datenquellen geöffnet werden.

4.1.3.2.6 Modularer Aufbau (für leichte Erweiterbarkeit)

Der Aufbau des Prototyps soll modular sein, um eine spätere Verwendung und Erweiterbarkeit der Komponenten im weiteren Projektverlauf (SNIK) zu ermöglichen.

4.1.3.2.7 Beschaffung der Daten findet zur Abfrage-Zeit statt

Die Datenquellen sollen nicht zwischengespeichert werden, sondern zur Laufzeit abgefragt werden, um aktuellste Daten zu garantieren.

4.1.3.2.8 Zugriff/Aggregation von Word-, Excel- und Webseiten-Daten (lesend)

Der lesende Zugriff auf die folgenden Datenquellen soll möglich sein:

Excel-Dateien (xls, xlsx)

Word-Dateien (doc, docx)

Webseiten-Daten (html)

4.1.3.2.9 Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)

Der Prototyp soll gegenüber fehlerhaften Dateien (z.B. syntaktisch inkorrekte Dateien), sowie nicht vorhandenen Dateien und Daten in den Dateien fehlertolerant sein. Das heißt, dass dieser nicht unkontrolliert abstürzen, sondern Ausnahmen behandeln soll, falls Dateien oder Daten nicht vorhanden sind, bzw. ungewöhnlichen Inhalt haben.

4.1.3.2.10 Kontinuierliches Logging von Abläufen und Fehlern für bessere Wartbarkeit

Ein fortlaufendes Logging der Abläufe im Prototyp und OBDA Framework soll eine bessere Wartbarkeit bei Fehlern ermöglichen.

4.1.3.2.11 Warnung des Nutzers, bzgl. auftauchender Fehler (z.B. unbehandelte Ausnahmen)

Der/die NutzerIn soll über unbehandelte Ausnahmen informiert werden um diese ggf. beseitigen zu können.

4.1.4 externe Schnittstellen

1. API zur Kommunikation mit dem Framework und der GUI
2. JDBC API zwischen OBDA-Framework und Datenquelle

4.1.5 Design-Beschränkungen

Bei der Entwicklung der Kommunikation mit Microsoft Office und dem SharePoint muss .NET verwendet werden, um reibungslose Funktionalität zu gewährleisten. Daher wird auf eine eigene Implementierung dieser Anbindung verzichtet.

4.1.6 Sonstige Anforderungen

Der Prototyp soll so dokumentiert werden, dass dieser auch ohne Experten-Wissen wieder aufgesetzt und konfiguriert werden kann. Außerdem soll ein modularer Aufbau verwendet werden, damit der Prototyp zu einem späteren Zeitpunkt um neue Funktionen erweitert werden kann.

Die folgende Liste enthält die genauen Fragen, die der Prototyp beantworten können soll.

Tabelle 7 Vom Prototyp zu beantwortende Fragestellungen und deren Formulierung als SPARQL Request (falls möglich)

Inhaltliche Fragestellung	Konkrete SPARQL-Abfragen, um an das Objekt zu gelangen
„Welche Projekte gibt es?“	PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled-ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?x where {?x rdf:type :Projekt.}
„Zeige Projektplan zu Projekt x“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Wieviel Budget ist für Projekt X übrig?“	PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled-ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?z where {?x rdf:type :Projekt. ?x :Projektnummer "2007-08-003". ?x :isAssociatedWith ?y. ?y rdf:type :Controllingliste_Dienstleistungen. ?y :Budgetstand ?z}
„An welchen Projekten nimmt Person X teil?“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Wann ist das nächste Projekt voraussichtlich fertig?“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Wann werden neue Ressourcen frei?“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige den Projekt Status Report des Projekts X“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige den Projekt Fertigstellungs-Report des Projekts X“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige das Übergabe-Protokoll des Projekts X“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige die IT-Strategie“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige die Projekt-Warteliste“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige den Ressourcen Plan“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige die Projekt Webseite des Projekts X“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Zeige das Projekt Portfolio“	Abfrage kann mit den gegebenen Test-Daten nicht erzeugt werden.
„Für welche Projekte ist Projektleiter X zuständig?“	PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled-ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?x ?z where {?x rdf:type :Projekt. ?x :Projektnummer ?y. ?x :isAssociatedWith ?z. ?z

	<pre> rdf:type :Projektleiter. ?z :Name "Hans" } PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled- ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?inhalt where {?x rdf:type :Controllingliste_Dienstleistungen_Position. ?x :Inhalt ?inhalt. ?x :isAssociatedWith ?y. ?y :Name "Hans"} </pre>
„Zeige Daten aus Controllingliste_Invest für Projekt X“	<pre> PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled- ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?nr ?verbrauch ?inhalt ?lieferant ?plname where {?x rdf:type :Projekt. ?x :Projektnummer "2007-08- 003". ?x :isAssociatedWith ?y. ?y rdf:type :Controllingliste_Invest. ?y :isDecomposedIn ?z. ?z rdf:type :Controllingliste_Invest_Position. ?z :Verbrauch ?verbrauch. ?z :I nhalt ?inhalt. ?z :Lieferant ?lieferant. ?z :Nr ?nr. ?z :isAssociatedWith ?pl. ?pl :Name ?plname} </pre>
„Zeige Daten aus Controllingliste_Dienstleistungen für Projekt X“	<pre> PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled- ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?nr ?inhalt ?lieferant ?plname where {?x rdf:type :Projekt. ?x :Projektnummer "2007-08- 003". ?x :isAssociatedWith ?y. ?y rdf:type :Controllingliste_Dienstleistungen. ?y :isDecomposedIn ?z. ?z rdf:type :Controllingliste_Dienstleistungen_Position. ?z :Inhalt ?inhalt. ?z :Lieferant ?lieferant. ?z :Nr ?nr. ?z :isAssociatedWith ?pl. ?pl :Name ?plna me} </pre>
„Welche Projektleiter gibt es?“	<pre> PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled- ontology-20#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> select distinct ?y where {?x rdf:type :Projektleiter. ?x :Name ?y } </pre>

4.2 Ergebnisse der Betrachtung der OBDA Frameworks

Als ein Ergebnis der Evaluation von OBDA Frameworks sind Tabelle 8, Tabelle 9 und Tabelle 10 entstanden, die die wichtigsten Punkte zusammenfassen und die Frameworks dadurch miteinander vergleichbar machen. In Tabelle 9 werden die Frameworks in Bezug auf Entwicklungsaktivitäten (z.B. Codezeilen, letzte Commits oder Umfang der Kommentare) ihrer Repositories verglichen. In den Tabellen Tabelle 9 und Tabelle 10 werden die Frameworks auf ihre Aktualität, unterstützte Ontologie-Formate, Reasoner und Datenbanksysteme, eingesetzte Zuordnungs-Sprachen, exponierte Schnittstellen (z.B. CLI) und vorhandene Dokumentation verglichen.

Die Frameworks unterscheiden sich oberflächlich untereinander nur wenig. Wichtige Funktionen und Standards sind eingebaut und die Frameworks sind als weitgehend stabil zu bezeichnen. Erst im Detail sind Unterschiede erkennbar. So ist ein wichtiger Faktor für die Nutzung als Produktivsystem die Dokumentation und die Community, die das Produkt entwickelt – schließlich muss über längere Zeit sichergestellt sein, dass das System Aktualisierungen erhält und bei Problemen leicht gewartet werden kann. Diese Punkte sind auch schon während der Entwicklung relevant, da hier immer wieder Schwierigkeiten auftreten können, die bei unzureichender Dokumentation erschwert lösbar sind. Die Analyse der Repositories (dargestellt in Tabelle 8) lässt dabei den Grad der Ausgereiftheit und den Umfang der verschiedenen Frameworks erahnen.

Tabelle 8 Repository Vergleich der Frameworks

Projekt	Jena	Ontop	D2RQ und D2R Server	OpenLink Virtuoso (Open-Source Edition)
<i>Projekt Aktivität</i>	Sehr Hoch	Hoch	Sehr Niedrig	Hoch
<i>Beteiligte</i>	23 Entwickler	33 Entwickler	17 Entwickler	12 Entwickler
<i>Commits (Insgesamt)</i>	4,550 commits	4,023 commits	1,025 commits	12,866 commits
<i>erster Commit</i>	Vor über 3 Jahren	Vor über 5 Jahren	Vor über 11 Jahren	Vor fast 10 Jahren
<i>letzter Commit</i>	Vor 3 Tagen	Vor ca. 1 Monat	Vor über 1 Jahr	Vor 6 Tagen
<i>Beteiligte (letzte 12 Monate)</i>	15 Entwickler	16 Entwickler	Keine Aktivität	5 Entwickler
<i>Commits (letzte 12 Monate)</i>	1,491 commits	678 commits	Keine Aktivität	837 commits
<i>Dateien verändert (letzte 12 Monate)</i>	9,874 Dateien	1,150 Dateien	Keine Aktivität	7,579 Dateien
<i>Zeilen hinzugefügt (letzte 12 Monate)</i>	2,069,544 Zeilen	965,359 Zeilen	Keine Aktivität	217,300 Zeilen
<i>Zeilen entfernt (letzte 12 Monate)</i>	1,975,800 Zeilen	230,469 Zeilen	Keine Aktivität	59,578 Zeilen
<i>Year-Over-Year Commits</i>	Steigend	Fallend	Fallend	Stabil
<i>Beteiligte (letzte 30 Tage)</i>	6 Entwickler	Keine Aktivität	Keine Aktivität	2 Entwickler
<i>Commits (letzte 30 Tage)</i>	126 commits	Keine Aktivität	Keine Aktivität	57 commits

<i>Dateien verändert (letzte 30 Tage)</i>	582 Dateien	Keine Aktivität	Keine Aktivität	56 Dateien
<i>Zeilen hinzugefügt (letzte 30 Tage)</i>	25,361 Zeilen	Keine Aktivität	Keine Aktivität	711 Zeilen
<i>Zeilen entfernt (letzte 30 Tage)</i>	21,272 Zeilen	Keine Aktivität	Keine Aktivität	267 Zeilen
<i>Hauptsächlich geschrieben in</i>	Java	Java	Java	C
<i>Kommentare</i>	Durchschnittlich	Durchschnittlich	Sehr Niedrig	Niedrig
<i>Code-Zeilen</i>	567,809 Zeilen	872,935 Zeilen	47,327 Zeilen	2,387,967 Zeilen

Zur Erstellung dieser Tabelle werden die durch die Openhub⁴⁹ Plattform gegebenen Werkzeuge eingesetzt, um den Code in den Repositories der Frameworks zu analysieren. Openhub bezieht den gesamten Code eines Github-Repositories und errechnet Metriken, die dann dazu verwendet werden, um z.B. die Anzahl der Kommentare, beteiligte Entwickler usw. zu errechnen. Aus diesem Grund können in dieser Tabelle auch nur die Frameworks aufgelistet werden, deren Quelltext öffentlich ist.

Das Ergebnis ist eine Gegenüberstellung der Frameworks in Bezug auf deren Code Details. Diese Informationen sind zusätzliche Indikatoren für den Entwicklungsstand. Es ist erkennbar, dass Virtuoso und Ontop in dieser Gegenüberstellung die größte Aktivität und die meisten Code-Zeilen aufweisen. Auch wenn die Anzahl an Kommentaren durchschnittlich bis niedrig ist, so lässt sich unter Anbetracht aller Informationen feststellen, dass diese beiden Projekte in Bezug auf ihre Ausgereiftheit und Nutzbarkeit in einem Produktivsystem gut dastehen.

Auch mit Blick auf Tabelle 9 und Tabelle 10 lässt sich feststellen, dass Ontop zusammen mit Mastro und Virtuoso gute Kandidaten für den Einsatz in einem Produktivsystem sind, wenn es darum geht, deren OBDA-Funktionen zu nutzen. Was die Frameworks voneinander unterscheidet, ist vor allem der Grad der Dokumentation und deren Lizenz. Die Dokumentation fällt bei Mastro nur gering aus, da lediglich Installationshinweise gegeben werden.

Während man für Ontop Anwendungs- und Konfigurationsbeispiele sowie generelle Dokumentation in Bezug auf Entwicklung findet, fehlt diese bei Mastro. Virtuoso hingegen ist sehr gut dokumentiert. Weniger, aber dennoch negativ ins Gewicht fällt, dass die Lizenz von Mastro nicht klar definiert und nur für Forschungszwecke gedacht ist. Die Nutzung außerhalb des SNIK Projekts ist daher kaum möglich. Ontop ist, wie Virtuoso auch, in seiner Lizenz sehr frei.

D2RQ ist nicht so ausgereift wie Ontop oder Virtuoso, und auch hier fehlt es an Dokumentation. Zudem ist D2RQ in seinen Funktionen nicht weiter fortgeschritten als andere Frameworks wie Ontop, Virtuoso oder Mastro. Auf Grund der nun mehr 4 Jahre andauernden Inaktivität der Entwicklung ist es fraglich, ob neue Entwicklungen in der Forschung im Projekt D2RQ zukünftig berücksichtigt werden. Gerade was das Umschreiben von Abfragen von SPARQL zu SQL angeht, hat es Verbesserungen gegeben, die in den Frameworks Ontop und Mastro eingebaut sind.

⁴⁹ <https://www.openhub.net/>

Wenn es um die Umsetzung einer eigenen Lösung für OBDA geht, so kann Jena dafür eingesetzt werden, da es in der Theorie verschiedene Reasoner unterstützt und durch JDBC Treiber an verschiedene Datenbanksysteme angebunden werden kann. So ist theoretisch auch eine Anbindung an Teiid möglich, welches einen JDBC Treiber zur Verfügung stellt. In Diskussionen mit anderen Jena NutzerInnen wurde allerdings klar, dass die Umsetzung mit Jena eine Modifikation des Quelltextes voraussetzt und daher unnötig kompliziert ist. Eine eigene Implementierung kann also auf Grund der Qualität der bereits vorhandenen expliziten OBDA Frameworks vernachlässigt werden.

Die in Tabelle 9 und Tabelle 10 gelisteten Funktionen der Frameworks dienen zusammen mit der Repository Analyse in Tabelle 8 als Basis für die nachfolgende Bewertung der Frameworks nach deren Erfüllung der Kriterien.

Die Skala für die Bewertung reicht von 0 bis 2, wobei wie folgt bewertet wurde:

- bedeutet: Kriterium wurde nicht erfüllt.
 - Punkte: 0
- bedeutet: Kriterium wurde teilweise erfüllt.
 - Punkte: 1
- bedeutet: Kriterium wurde voll erfüllt.
 - Punkte: 2

Ergebnis des Vergleichs und der Bewertung ist folgende Top3-Belegung (für Details siehe Tabelle 11):

1. Ontop (27 Punkte)
2. Virtuoso (26 Punkte)
3. Stardog (25 Punkte)

Durch eine zusätzliche Gewichtung der Fragen wäre der Unterschied stärker ausgefallen, da Ontop das einzige Framework ist, welches offiziell über Teiid das Einbinden von Excel-Dateien erlaubt. Zwar wäre eine Anbindung an andere Frameworks mittels eines JDBC-Treibers möglich (z.B. bei Mastro), allerdings wird Teiid hier nicht offiziell unterstützt. Bei Virtuoso gibt es außerdem das Problem, dass wichtige Funktionen der Enterprise-Edition vorbehalten sind, die für das Projekt wichtig sind: Es ist nicht möglich mehrere Datenquellen anzubinden und ein Anbinden mehrerer Datenquellen mit Teiid über dessen JDBC-Treiber ist auch nicht möglich. Hierfür muss eine zusätzliche Komponente vom Hersteller gekauft werden um JDBC-Treiber zu unterstützen.

Bei der genaueren Betrachtung des Siegers Ontop, werden aber dennoch ein paar Punkte aufgezeigt, die nicht optimal sind:

- Mehrere Datenquellen werden nicht unterstützt (nur indirekt über Teiid)
- Die Anbindung von Excel-Daten über Teiid ist möglich, allerdings kein Word
- Die Community ist nicht so aktiv wie bei anderen Frameworks (z.B. Virtuoso).

Folgernd aus den ersten beiden Punkten ist also der Einsatz von Teiid als ein Föderationsserver notwendig, damit Ontop über Teiid auf die verschiedenen Datenquellen zugreifen kann.

Tabelle 9 Vergleichstabelle verschiedener Frameworks (Teil 1)

Feature/Name	Jena	Ontop	Mastro	Kriterium
SPARQL	✓(ARQ)	✓	✓	K14
Reasoner	Mehrere (OBDA?)	Quest (OWL 2 QL)	OWL 2 QL	K12
Rewriting	X	Automatisch (SQL)	Automatisch	K5, K11
Mapping	X	Ontop Mapping, R2RML	R2RML	K8, K16
OBDA mit...	SQL; Triple Store;	PostgreSQL, MySQL, H2, DB2, SQL Server, Teiid, Oracle	MySQL, PostgreSQL, SQL Server	K15
letzte Akt.	Commits ~Jeden Tag	17.11.2015	Um 2013 (Publikation)	K1, K3
Stabil	✓	✓	✓	K4
Akt. Ver.	3.0.1	1.17.1	unklar	
Ontologie Formate	OWL,N3,rdf,Turtle	RDF,OWL	OWL	K8, K13
SPARQL-Endpoint	✓Fuseki	✓	✓	
API	✓(Jena)	✓(OWL API, SAIL)	X	K6
CLI	✓	✓	✓	K6
non-rel. OBDA	X nicht implementiert	✓ indirekt über Teiid	inoffiziell über JDBC	K15
Mehrere Datenquellen	X nicht implementiert	✓ indirekt über Teiid	✓	K7
Dokumentation	Sehr gut (Setup, viele Beispiele, verschiedene Erklärungen, aktive community)	Gut (Setup, Beispiele, versch. Erklärungen)	Knapp (Setup, kurze Erklärung)	K2
Lizenz	Apache 2.0	Apache 2.0	Unklar (Akademisch)	K9
OpenSource	✓	✓	X	K10

71 von 133

Tabelle 10 Vergleichstabelle verschiedener Frameworks (Teil 2)

<i>Feature/Name</i>	d2rq	Stardog	Virtuoso (Quelloffen)	Kriterium
SPARQL	✓	✓	✓	K14
Reasoner	✓	✓	✓	K12
Rewriting	Automatisch (SQL)	Automatisch (SQL)	Automatisch	K5, K11
Mapping	D2RQ Mapping, R2RML	R2RML, SMS (Stardog)	R2RML und proprietär	K8, K16
OBDA mit...	Unklar	MySQL, PostgreSQL, Microsoft SQL Server, Oracle, DB2, H2, SAP HANA	SQL, XML, RDF	K15
letzte Akt.	12.03.2012	14.01.2016	Commits alle paar Tage	K1, K3
Stabil	Unklar	✓	✓	K4
Akt. Ver.	0.8.1	4.0.3	7.2.2	
Ontologie Formate	OWL, RDF	OWL, RDF	OWL, RDF	K8, K13
SPARQL-Endpoint	✓	✓	✓	
API	✓ (Jena API)	✓ (.NET, Java,...)	✓(Jena)	K6
CLI	✓(Queries und mapper)	✓	X	K6
non-rel. OBDA	Nicht direkt, ggf. JDBC	unklar	X (JDBC, andere - nur kommerziell)	K15
Mehrere Datenquellen	Nicht direkt, ggf. JDBC	✓	X (nur kommerziell)	K7
Dokumentation	Knapp (Setup, kurze Erklärung)	Gut (Setup, Beispiele, versch. Erklärungen)	Nur Installationshinweise	K2
Lizenz	Apache 2.0	kommerziell; eingeschränkt privat	GPL 2.0	K9
OpenSource	✓	X	✓	K10

Tabelle 11 Übersicht über die Bewertungen der Frameworks

ID	Kriterium	Jena	Ontop	Mastro	d2rq	Stardog	Virtuoso (Quelloffen)
K1	Das Framework wird aktuell aktiv weiterentwickelt	✓	✓	○	X	✓	✓
K2	Umfangreiche und ausführliche Dokumentation des Frameworks ist vorhanden.	✓	○	○	○	✓	✓
K3	Aktive Community	✓	○	X	X	○	✓
K4	Das Framework ist ausgereift und stabil.	✓	✓	○	○	✓	✓
K5	Das Framework greift zur Laufzeit auf Daten zu.	X	✓	○	○	✓	✓
K6	Es ist möglich das Framework über eine Schnittstelle (CLI, API, Endpoint) zu programmieren bzw. Abfragen abzusetzen.	✓	✓	○	○	✓	○
K7	Das Framework unterstützt mehrere Datenquellen.	X	X	✓	X	✓	X
73 von 133	K8	○	✓	✓	✓	✓	✓
	K9	✓	✓	○	✓	X	○
	K10	✓	✓	X	✓	X	✓
	K11	X	✓	✓	✓	✓	✓
	K12	X	✓	✓	○	✓	✓
	K13	✓	✓	✓	✓	✓	✓
	K14	✓	✓	✓	✓	✓	✓
	K15	X	○	○	X	X	X
	K16	X	✓	✓	✓	✓	✓
	Punkte	19	27	21	19	25	26

4.3 Lösungskonzeption

In den vorangegangenen Kapiteln wurden die Anforderungen an eine Softwarelösung für das Projekt formuliert. Aus diesen lassen sich zwei übergeordnete Nutzungsszenarien ableiten: ein Konfigurationsvorgang und ein Nutzungsvorgang. Der Nutzungsvorgang setzt eine erste Konfiguration als abgeschlossenen voraus. Es ist sinnvoll, den Prototypen so zu gestalten, dass die beiden Vorgänge getrennt voneinander sind. Das lässt sich damit begründen, dass der Vorgang der Konfiguration, im optimalen Fall, nur ein Mal durchgeführt werden muss und später nur noch kleinere Änderungen vorgenommen werden. Ontop wird in einem „Protégé-Bundle“ ausgegeben, welches eine Kombination von Protégé und einem dazu passenden Ontop-Plug-In ist. So ist es möglich Konfigurationsänderungen und damit die Erstellung der Zuordnungen über Protégé durchzuführen. Das hat den Vorteil, dass zum einen das gewohnte Programm Protégé verwendet werden kann und die direkte Einsicht der Ontologie in Protégé das Erstellen der Zuordnungen erleichtert. Die eigentliche Nutzung des Ontop Frameworks kann dann wiederum über den Prototyp erfolgen. Ein föderiertes System wie Teiid wird dann eingesetzt. Hierdurch ist es möglich die heterogenen Datenquellen, die im Krankenhaus vorhanden sind, darüber miteinander vereinen zu können, ohne dabei zwangsweise die Datenmodelle der Datenquellen zu ändern.

4.4 Manuelle Lösung

Zum besseren Verständnis wird eine manuelle Lösung entwickelt. Die manuelle Lösung ist vorrangig als eine Erklärung der Vorgänge im Prototyp auf einer höheren Ebene zu verstehen: Die Zuordnungen, die in dieser Lösung gemacht werden, werden im Prototyp nachgeahmt. Die manuelle Lösung hat zum Ziel ein besseres Verständnis für die notwendigen Änderungen an der Ontologie oder den Datenquellen zu bekommen. Mit Bezug auf den im Kapitel „Datenintegration“ beschriebenen Prozess werden auch hier die verschiedenen Daten darüber analysiert und diskutiert, wie diese über Zuordnungen in die Ontologie integriert werden können.

Somit ist die manuelle Zuordnung eine Vorarbeit für die Umsetzung des eigentlichen Prototyps. Der Unterschied zum fertigen Prototyp ist aber, dass die manuelle Lösung lediglich statische Zuordnungen zu Elementen vorsieht, während im fertigen Prototyp die Zuordnungen zur Laufzeit dynamisch umgeschrieben werden (siehe 2.1.13).

Die Ontologie wird dafür zunächst als Graph wie Abbildung 24 zeigt, dargestellt.

Tabelle 12 Übersicht der Ontologie-Elemente

Ebene	# Konzepte	# Informationsobjekte
1	12	0
2	21	12
3	5	0

Tabelle 12 ist eine Übersicht der Unterklassen der verschiedenen Ebenen, sowie die Anzahl möglicher Informationsobjekte (IO). Zum besseren Verständnis der Ontologie kann Abbildung 24 betrachtet werden. Die Informationsobjekte stellen eine tatsächliche Datenquelle in Form einer Datei dar.

Es folgt eine Übersicht der Informationsobjekte mit deren in der Ontologie angegebenen Datenquellen:

- Excel-Datei
 - Ressourcenplan
 - Projektewarteliste
 - Projektbudgetliste
 - Projekteliste
- Word-Datei
 - Projektplan
 - IT-Strategie
 - Projektübergabeprotokoll
 - Projektstatusbericht
 - Projektabschlussbericht
 - Wartungsvertrag
- Webseite
 - Projektwebsite
- Andere
 - Projektportfolio

Aus diesen Quellen mit den dazugehörigen Klassen der Ontologie lässt sich eine vereinfachte Zuordnung der Informationsobjekte der Ontologie zu den jeweiligen Datenquellen erzeugen.

Hierzu wird folgende Beispiel-Syntax eingeführt um Zuordnungen zu beschreiben:

QUELLE-->ZIEL

Quelle ist ein Objekt der Klasse *Informationsobjekt* in der Ontologie und Ziel ist eine konkrete Datenquelle. Die Beschreibung der Daten aus der Datenquelle für die Zuordnung folgt folgender Syntax:

EXTRAKTIONS-PUNKT : DATENTYP

Tabelle 13 Erklärung der Zuordnungs-Syntax

Syntax-Komponente	Beschreibung
EXTRAKTIONS-PUNKT	Stellt (je nach Dateityp) etwas anderes dar, da die Extraktion der Daten je nach Dateityp anders verläuft. <ul style="list-style-type: none"> • Word/Webseite/Text: Regular-Expression zur Extraktion des gewünschten Texts (z.B.: hier zwischen (.*)) steht der Text) • Excel: Name einer Spalte, definiert durch den darüber liegenden Text • SharePoint: SharePoint Abfrage
DATENTYP	Gibt den Datentyp der extrahierten Daten an (z.B.: Integer, String, String List)

Sind die Zuordnungen im System hinterlegt und die Quellen verfügbar, kann auf diese zugegriffen werden. Der Zugriff erfolgt aus Sicht des Nutzers über eine Oberfläche auf der dann entweder SPARQL-Abfragen direkt in der Nutzeroberfläche eingegeben oder durch Betätigung von Knöpfen ausgelöst werden.

Stellt der/die NutzerIn nun eine solche SPARQL-Abfrage, werden die hierfür notwendigen Klassen der Ontologie aus der Abfrage evaluiert und dann deren Zuordnungen mit den Datenquellen aufgelöst. Dann werden die Daten aus der Datenquelle extrahiert und der Antwort beigefügt, diesen Schritt übernimmt in der fertigen Lösung das OBDA-Framework Ontop. Das zurückgegebene Tripel ist die mit den Daten aus den Quellen erweiterte Abfrage, die auf der Nutzeroberfläche dargestellt wird.

Abbildung 23 veranschaulicht diesen Vorgang in Form eines UML-Sequenzdiagramms.

Die Zuordnungen die zwischen den Datenquellen bzw. Daten und den Klassen der Ontologie gemacht werden, werden am Beispiel von einigen Klassen und der Datenquelle „Controllingliste Invest für ein Projekt“ dargestellt. Ein Ausschnitt aus dem Bereich des Projektmanagements zeigt die zum Verständnis notwendigen Elemente in Abbildung 25. Da die Zuordnungen trotz der kleinen Anzahl an Datenquellen und Ontologie-Elementen unübersichtlich sind, wird nur dieser Teil grafisch betrachtet. Als Legende kann dient die Legende der Abbildung 24. Ein Beispiel für eine Zuordnung:

Der Zusammenhang zwischen den Klassen „Projekt“ (blauer Kasten, links) und „Controllingliste _Invest“ (blau gestrichelter Kasten rechts von Projekt), welcher im Bild durch einen blauen Pfeil angezeigt ist, wird über das Datenfeld „ProjektNr“ der Datenquelle „Controllingliste Invest für ein Projekt“ (grauer Kasten, rechts) hergestellt.

Die in dem Schaubild gestrichelt eingezeichneten Elemente der Ontologie sind in der ursprünglichen Ontologie nicht vorhanden und sind im Zuge der manuellen Zuordnung zwischen den Datenquellen und der CIOx-Ontologie entstanden. Die Ontologie wurde dementsprechend ergänzt. Es gibt eine vollständige grafische Darstellung der Zuordnungen zwischen den Datenquellen, deren Daten und den zugehörigen Elementen der Ontologie. Diese Darstellung ist allerdings zu groß für diese Arbeit, kann aber digital betrachtet werden. Die nachfolgenden Listen (Tabelle 14, Tabelle 15, Tabelle 16 und Tabelle 17) zeigen die Zuordnungen tabellarisch auf:

Tabelle 14 Zuordnungen für Controllingliste invest

Ontologie-Element	Daten aus Datenquelle
Projekt.isAssociatedWith.Controllingliste_Invest	ProjektNr. :string
Controllingliste_Invest.Budgetstand	Budgetstand zu DATUM :float
Controllingliste_Invest.Planbudget	Budget :float
Controllingliste_Invest.Auftragsvolumen	Auftragsvolumen :float
Controllingliste_Invest.Rest	Rest :float
Controllingliste_Invest_Position.Nr	Nr :int
Controllingliste_Invest_Position.Lieferant	Lieferant :string
Controllingliste_Invest_Position.Inhalt	Inhalt :string
Controllingliste_Invest_Position.isAssociatedWith.Projektleiter	Projektleiter :string
Controllingliste_Invest_Position.Auftragsvolumen	Verbrauch :float

Tabelle 15 Zuordnungen für Controllingliste Budget

Ontologie-Element	Daten aus Datenquelle
Projekt.isAssociatedWith.Controllingliste_Dienstleistungen	ProjektNr. :string

Controllingliste_Dienstleistungen.Budgetstand	Budgetstand zu DATUM :float
Controllingliste_Dienstleistungen.Planbudget	Budget :float
Controllingliste_Dienstleistungen.Auftragsvolumen	Auftragsvolumen :float
Controllingliste_Dienstleistungen.Rest	Rest :float
Controllingliste_Dienstleistungen_Position.Nr	Nr :int
Controllingliste_Dienstleistungen_Position.Lieferant	Lieferant :string
Controllingliste_Dienstleistungen_Position.Inhalt	Inhalt :string
Controllingliste_Dienstleistungen_Position.IsAssociatedWith.Projektleiter	Projektleiter :string

Tabelle 16 Zuordnungen für Projektwarteliste

Ontologie-Element	Daten aus Datenquelle
Projekt.isAssociatedWith.Projektpriorität	Prio :int
Projekt.Jahr	Jahr :string
Projekt.Projektnummer	ProjektNr. :string
Projekt.Name	Bezeichnung :string
Projekt.isAssociatedWith.Projektleiter	PL :string
Projekt.Projectbeginn	geplanter Projektbeginn :string
Plansumme_Dienstleistungsmittel	DL geplant :string
Geplante_Aktivierbare_Dienstleistungsmittel	aDL geplant :string
Plansumme_Investitionsmittel	INV geplant :string

Tabelle 17 Zuordnungen für Projektbudgetliste Sachkonto

Ontologie-Element	Daten aus Datenquelle
Projektbudgetliste.ProjektNr.	Projekt-Nr. :string
Projektbudgetliste.Budgetstand	Budgetstand zu DATUM :float
Projektbudgetliste.Restbudget	Restbudget :float

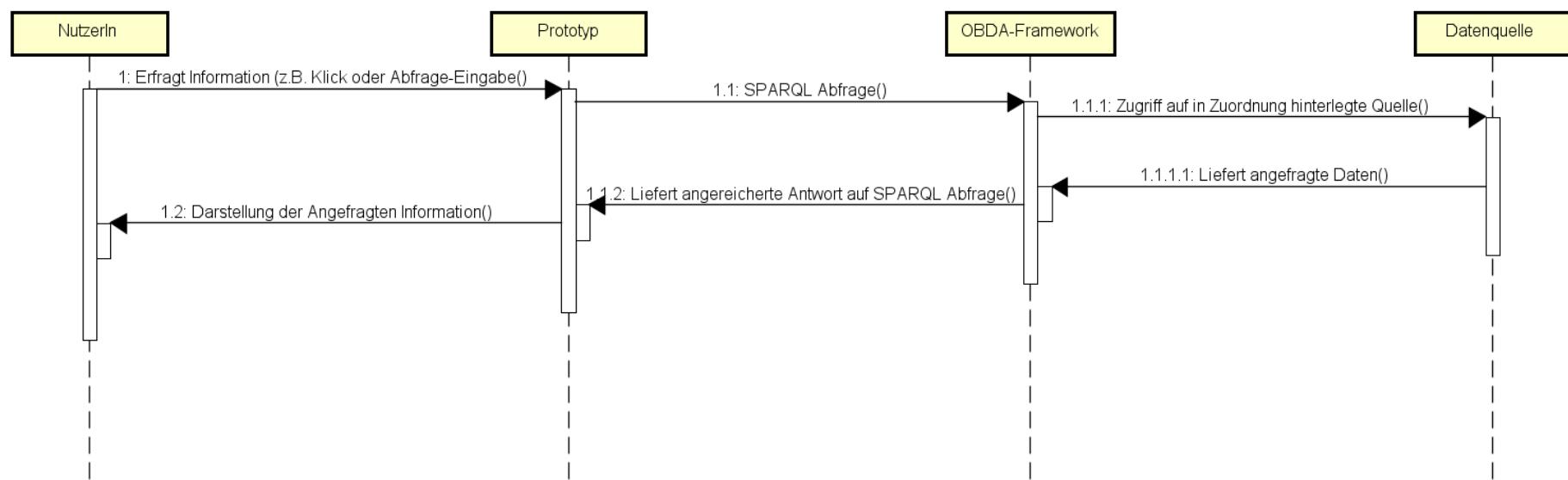


Abbildung 23 Sequenzdiagramm der Nutzung der manuellen Lösung

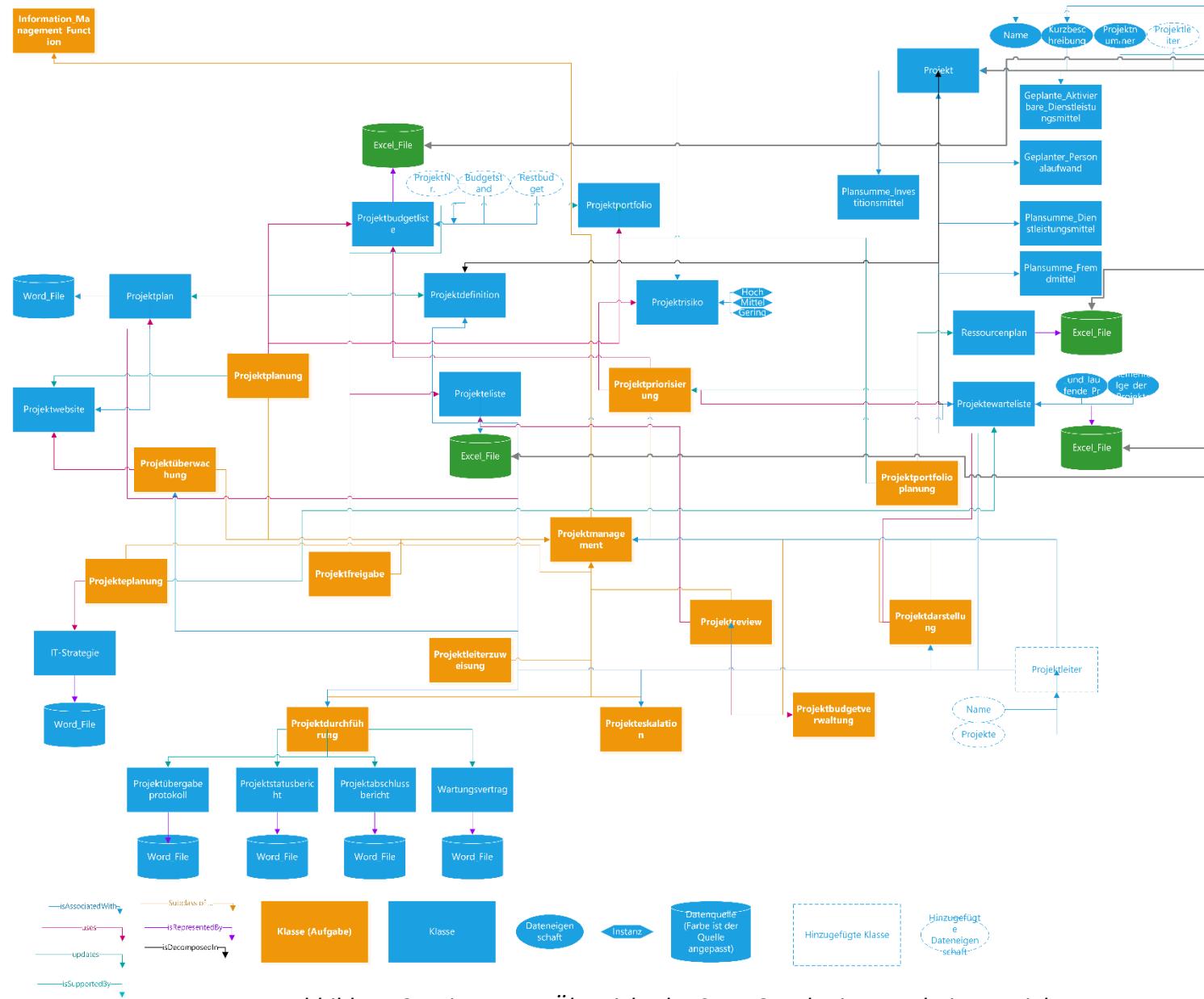


Abbildung 24 Diagramm-Übersicht der SNIK-Ontologie Ausschnitt „Projektmanagement“

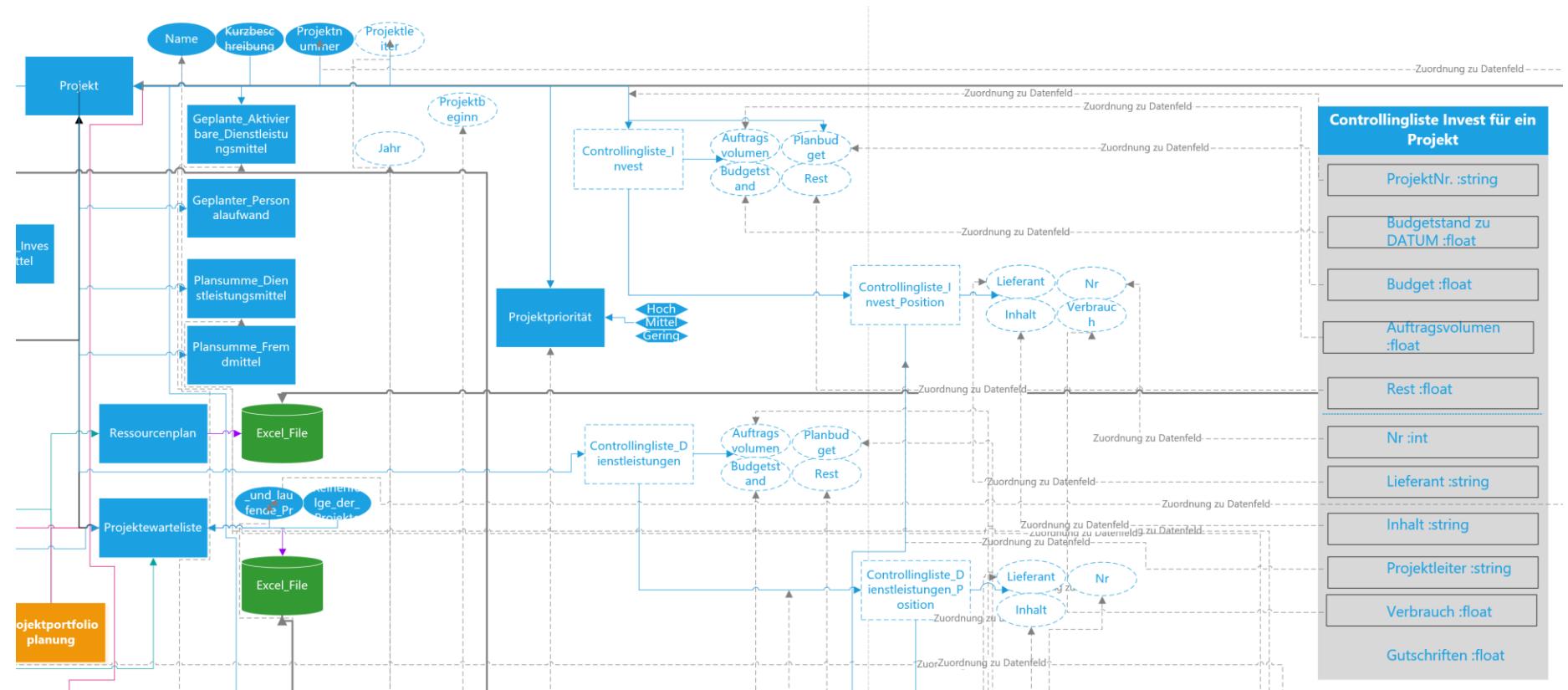


Abbildung 25 Ausschnitt Darstellung Ontologie und Datenquellen Zuordnung

5 Umsetzung und Qualitätssicherung

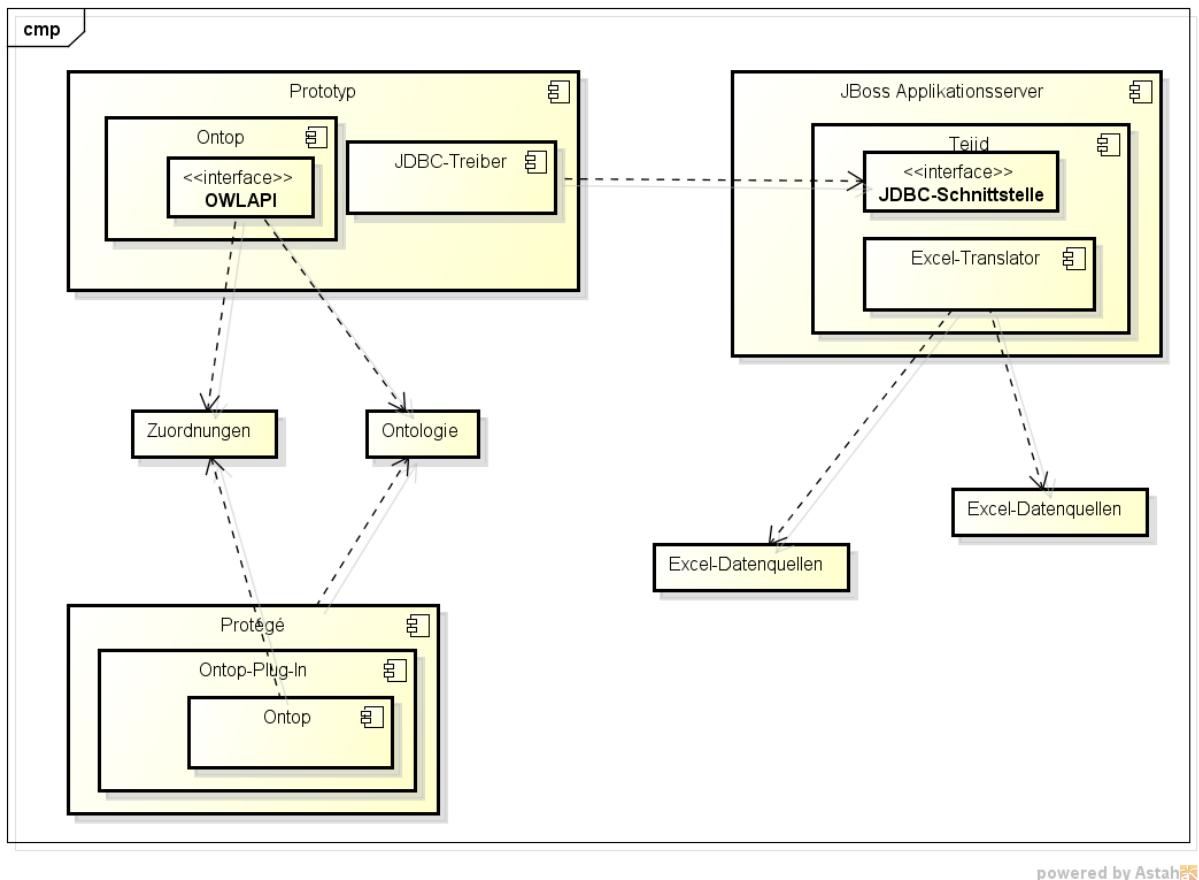
In diesem Kapitel wird auf die tatsächliche Umsetzung des Prototyps und Details der mit ihm in Verbindung stehenden Komponenten (konkret: Teiid und Protégé) eingegangen. Dazu wird die Umsetzung zunächst in ihrer Gänze beschrieben und dann komponentenweise detaillierter vorgestellt.

5.1 Umsetzung

Für die Umsetzung wird Ontop eingesetzt, da dieses Framework die gewünschten Kriterien am besten erfüllt (siehe 4.2). Ontop kann nicht mit mehreren Datenquellen umgehen und ist daher auf einen Föderationsserver angewiesen. Aus diesem Grund wird Teiid eingesetzt um die verschiedenen Datenquellen als eine Datenquelle an das Framework zu exponieren. Die Datenquellen (Excel-Dateien) werden dabei als unterschiedliche Tabellen über einen JDBC-Treiber für Ontop verfügbar gemacht. Das Übersetzen von Excel in eine von Teiid als virtuelle Datenbank exponierbare Form erfolgt über einen Translator. Eine wichtige Eigenschaft ist, dass Teiid keine Veränderungen am Datenmodell der Quelldaten vornimmt und daher deren Aufbau erhalten bleibt. Ontop selbst kommt sowohl im Ontop-Plug-In von Protégé zum Einsatz, welches zum Erstellen der Zuordnungen verwendet wird und im Prototyp, um OBDA durchzuführen. Der Prototyp ist in seiner Funktionalität flexibel erweiterbar, da dieser auf der von Ontop exponierten OWLAPI aufbaut. Es ist möglich die über die Ontologie erhaltenen Daten innerhalb des Prototyps weiter zu verarbeiten, da diese dort als vom Programmcode nutzbare Objekte vorliegen (dazu mehr in 5.1.1.1). Der/die NutzerIn hat, nachdem die Zuordnungen über Protégé erstellt wurden, die Möglichkeit über den Prototyp die Datenquellen über eine SPARQL-Abfrage der Ontologie zu beziehen.

Die zukünftigen Nutzungsmöglichkeiten und weitere Forschung wird im Kapitel 7 genau dargelegt.

Das folgende Schaubild gibt einen Überblick über den oben beschriebenen Aufbau:



powered by Astah

Abbildung 26 Komponentendiagramm der Umsetzung

Eine detaillierte Anleitung zur Installation, Konfiguration und Nutzung aller Komponenten findet sich im Anhang (8.1).

5.1.1 Prototyp

5.1.1.1 Funktionsweise auf Code-Ebene

Die Nutzeroberfläche ist mittels Swing⁵⁰ Komponenten erstellt. Der Aufbau der grafischen Oberfläche wird in Kapitel 5.1.1.2 genauer beschrieben.

Die wesentlichen Methoden des Prototyps sind `startReasoner` und `executeQuery`. Diese beiden Methoden sind getrennt voneinander, da die Initialisierung des Reasoners vergleichsweise lange dauert, je nach Menge der Zuordnungen und Größe der Ontologie. So muss der Reasoner nur einmal gestartet werden, um dann unterschiedliche SPARQL-Abfragen bearbeiten zu können. Ändern sich Zuordnungen oder Ontologie, während der Prototyp läuft, muss der Reasoner neu gestartet werden, um die Änderungen zu übernehmen.

Zunächst ein Überblick über die `startReasoner` Methode (für detaillierteren Code siehe Kapitel 8.1):

Die `startReasoner` Methode führt die notwendigen Anweisungen und Vorarbeiten aus, um den Reasoner zu starten. Dazu sind zwei Parameter notwendig. Erstens, ein Pfad zu einer OWL-Ontologie und zweitens, ein Pfad zu einer Zuordnungsdatei im OBDA-Format, die beide als String übergeben

⁵⁰ <https://docs.oracle.com/javase/7/docs/api/javax/swing/JComponent.html>

werden. Zuerst wird über einen `OWLManager`⁵¹ eine `OWLOntology`⁵² erzeugt. Der `OWLManager` ist eine Management-Klasse der OWLAPI für die Ontologie, über die z.B. Änderungen an der Ontologie vorgenommen werden können. Da Änderungen im Beispiel des Prototyps nicht relevant sind wird der `OWLManager` nur dafür verwendet um das `OWLOntology` Objekt zu erzeugen. Das `OWLOntology` Objekt repräsentiert eine OWL 2 Ontologie, in der alle Strukturen aus der zum Erzeugen verwendeten Datei enthalten sind (z.B. Axiome, Annotations). Als nächstes werden die Zuordnungen aus der über den Pfad angegebenen Datei geladen. Das Ergebnis ist ein Objekt des Typs `OBDAModel`⁵³, welches sämtliche Relationen zwischen der Ontologie und den Zuordnungen zu den SQL-Abfragen, sowie die JDBC Datenbankverbindungen enthält. Die beiden Objekte `OWLOntology` und `OBDAModel` werden mittels einer `OWLReasonerFactory` dazu eingesetzt, um eine `QuestOWL`⁵⁴ Instanz zu erzeugen, die den initialisierten Reasoner darstellt. Dies geschieht indem aus dem `OBDAModel` eine nicht veränderliche Konfiguration (`QuestOWLConfiguration`⁵⁵) erzeugt wird, die dann mit der `OWLOntology` zusammengeführt wird. Ergebnis der Anweisungen ist ein `QuestOWL` Objekt, welches den Reasoner mit den konfigurierten Werten für Ontologie und Zuordnungen darstellt. Der Reasoner ist somit erzeugt und im Speicher des Prototyps vorhanden. Jetzt können Abfragen an den Reasoner gestellt werden.

Das passiert über die `executeQuery` Methode (siehe 8.2), welche die SPARQL-Abfrage entgegen nimmt. Die SPARQ-Abfrage kann entweder als Dateipfad oder als String übergeben werden.

Zunächst wird geprüft, ob die Abfrage in Form einer Datei oder eines Strings erfolgt. Ist ersteres der Fall, wird die Datei zunächst eingelesen und dann als String an den Reasoner übergeben. Falls die Abfrage in Form eines Strings vorhanden ist, wird die Anfrage direkt an den Reasoner geleitet. Hierfür wird zunächst eine Verbindung zum Reasoner aufgebaut. Dann muss ein `QuestOWLStatement`⁵⁶ aus dieser Verbindung erzeugt werden, welches benötigt wird um die eigentlichen Abfragen geordnet auszuführen. In diesem Fall ist es nur eine Abfrage in der Form des zuvor ausgelesenen oder übergebenen Strings, welche über die Methode `executeTuple()` ausgeführt wird. Ergebnis dieses Vorgangs ist dann eine Antwort auf die Abfrage in der Form einer `QuestOWLResultSet`⁵⁷. Dieses Objekt enthält die Ergebnisse der Abfrage und wird benötigt, um den Inhalt weiter zu verarbeiten. Das Verarbeiten ist im Prototyp zum einen das direkte Ausgeben der Ergebnisse in einem Text-Bereich und zum anderen das Befüllen einer Combobox, für weitere Abfragen (diese wird im Abschnitt 5.1.1.2 genauer beschrieben).

Abschließend wird die Ergebnismenge über die Methode `showQueryResult` (siehe 8.3) in einem Text-Bereich auf der Nutzeroberfläche ausgegeben.

⁵¹ <http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/apibinding/OWLManager.html>

⁵² <http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLOntology.html>

⁵³ <https://github.com/ontop/ontop/wiki/ontopOBDAModel>

⁵⁴ <http://mavenbrowse.pauldo.com/central/it/unibz/inf/ontop/ontop-quest-owlapi3/1.12.0/ontop-quest-owlapi3-1.12.0-javadoc.jar/-/it/unibz/krdb/obda/owlrefplatform/owlapi3/QuestOWL.html>

⁵⁵ <https://github.com/ontop/ontop/blob/21c7ea2d9dc6af860291b74a6adec59712b59869/quest-owlapi3/src/main/java/it/unibz/krdb/obda/owlrefplatform/owlapi3/QuestOWLConfiguration.java>

⁵⁶ <http://mavenbrowse.pauldo.com/central/it/unibz/inf/ontop/ontop-quest-owlapi3/1.12.0/ontop-quest-owlapi3-1.12.0-javadoc.jar/-/it/unibz/krdb/obda/owlrefplatform/owlapi3/QuestOWLStatement.html>

⁵⁷ <http://grepcode.com/file/repo1.maven.org/maven2/it.unibz.inf.ontop/ontop-quest-owlapi3/1.14.0/it/unibz/krdb/obda/owlrefplatform/owlapi3/QuestOWLResultSet.java>

Diese Methode ist simpel: über das übergebene `QuestOWLResultSet` wird solange iteriert, bis alle darin enthaltenen `OWLObject`⁵⁸ Objekte verarbeitet wurden. Die Ergebnisse werden in der Schleife in das Ausgabe Text-Feld geschrieben.

Zusätzlich zu diesen sehr wichtigen Methoden gibt es im Prototyp diverse andere Methoden, die hauptsächlich dazu verwendet werden, um die Nutzeroberfläche korrekt und funktional zu halten.

Zur Veranschaulichung der Möglichkeit des Verarbeitens der Abfrage-Ergebnisse, über das bloße Ausgeben in einem Textfeld hinaus, existieren zwei Buttons deren Funktion nun auf Code-Ebene beschrieben wird. Beim Klick auf den Button „Lade Projektleiter“ in der Nutzeroberfläche wird eine für diesen Button geschriebene Abfrage ausgeführt, die in der Datei „loadProjectleaders.q“ hinterlegt ist. Diese Abfrage bezieht alle Namen der Projektleiter. Die über die Abfrage bezogenen Namen werden dann nicht nur in einem Textfeld ausgegeben, sondern in eine Combobox (auch Dropdown-Menü genannt) geladen. Daraufhin kann der Code eines zweiten Buttons aufgeführt werden, welcher auch eine speziell für diesen Button erstellte Abfrage-Datei lädt („showProjectsOf.q“). Diese Datei enthält einen Ersetzungspunkt der durch zwei Dollar-Zeichen angegeben wird:

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2015/5/untitled-ontology-20#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select distinct ?y where {?x
  rdf:type :Projekt. ?x :Projektnummer ?y. ?x :isAssociatedWith ?z. ?z
  rdf:type :Projektleiter. ?z :Name $$ }
```

Abbildung 27 SPARQL-Abfrage "showProjectsOf.q"

Bei der oben gezeigten Abfrage wird die Variable `$$` mit einer in der Nutzeroberfläche ausgewählten Information ersetzt (Konkret: den Namen der Projektleiter).

5.1.1.2 Funktionsweise aus der Sicht des Anwenders

Um den Prototyp vollständig zu verstehen ist es notwendig diesen auch aus der Sicht des Anwenders zu betrachten. Abbildung 28 zeigt die Nutzeroberfläche nach dem ersten Ausführen des Prototyps. Grafisch ist der Prototyp in zwei Hälften aufgeteilt: Die linke Hälfte für die Konfiguration und das Anstoßen von Funktionen und die rechte Hälfte für die Ausgabe der Funktionen und Status Informationen.

Als erstes fallen die drei großen Textfelder auf: Zwei blaue und ein weißes. Das weiße Textfeld wird zum Anzeigen von Status- und Fehlerinformationen verwendet (sei im Nachfolgenden Info-Textfeld genannt). Hier werden Informationen über den aktuellen Vorgang und auch Fehler in Textform dargestellt. Das blaue, linke Textfeld ist, je nach Situation, eine Vorschau für den Dateiinhalt der aktuell ausgewählten SPARQL-Abfrage-Datei, oder es beinhaltet die SPARQL-Abfrage direkt in Textform (sei im Nachfolgenden Abfrage-Textfeld genannt). Durch das Anwählen von einer der beiden darüber liegenden Optionen verändert sich das Verhalten:

Im Fall „Datei-Abfrage“ ist das Textfeld eine nicht editierbare Vorschau. Durch das Auswählen der Option „Text-Abfrage“ ist das Textfeld editierbar, und dessen Inhalt wird als Abfrage verwendet. Das rechte, blaue Textfeld wird für die Ausgabe der Ergebnisse der Abfrage verwendet (sei im

⁵⁸ <http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLObject.html>

Nachfolgenden Ausgabe-Textfeld genannt). An dieser Stelle werden die Ergebnisse der SPARQL-Abfragen dargestellt.

Links oben sind drei Textfelder in welche der/die NutzerIn manuell die Speicherorte der jeweiligen Dateien angeben kann. Die vom Prototyp benötigten Dateien sind, von oben nach unten: Eine Ontologie im OWL-Format, eine Zuordnungs-Datei im OBDA-Format und eine Abfrage-Datei in Text-Form, bzw. Text aus dem linken, darunter liegenden Abfrage-Textfeld. Die jeweils rechts der Textfelder für Pfade gesetzten Schaltflächen (Buttons) zeigen beim Klick einen Dateiauswahl-Dialog an, durch den die jeweiligen Dateien einfach ausgewählt werden können.

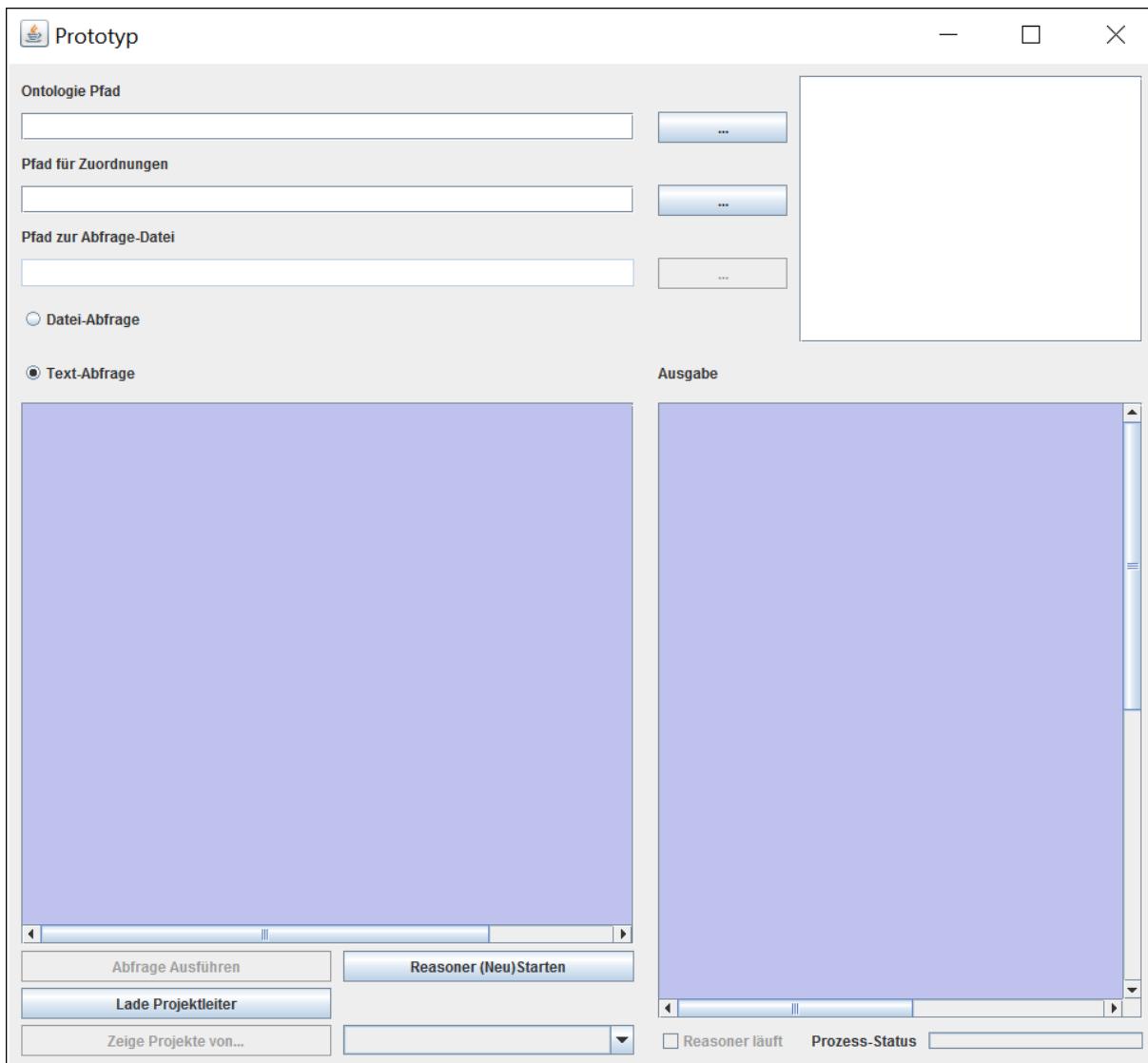


Abbildung 28 Nutzeroberfläche des Prototyps nach dem ersten Starten

Die Buttons unterhalb des Abfrage-Textfelds erfüllen die folgenden Funktionen:

- „Reasoner (Neu)Starten“
 - Initialisiert den Reasoner mit den in den Textfeldern angegebenen Informationen (Ontologie-Pfad und Zuordnungs-Pfad) über die `startReasoner` Methode.

- Erneutes Drücken initialisiert den Reasoner neu. Bei Änderungen an den ausgewählten Daten sollte der Reasoner neugestartet werden.
- „Abfrage Ausführen“
 - Je nachdem welche Option bei den beiden Auswahlpunkten (Datei- oder Text-Abfrage) ausgewählt ist, wird die Abfrage entweder aus dem Abfrage-Textfeld oder aus dem Abfrage-Pfad verwendet. Die Abfrage wird dann an den Reasoner übergeben (`executeQuery` Methode wird ausgeführt).
- „Lade Projektleiter“
 - Dieser Button wird dafür verwendet, um die Möglichkeit der Datenverarbeitung anschaulich demonstrieren zu können: Ein Klick darauf führt eine Abfrage aus, die unter dem Dateinamen „`loadProjectleaders.q`“ gespeichert ist, um das daneben befindliche Dropdownmenü mit den aus der Abfrage erhaltenen Daten zu befüllen.
- „Zeige Projekte von...“
 - Dieser Button wird erst aktiv, wenn zuvor der „Lade Projektleiter“-Button betätigt wurde. Zusammen mit dem aktuell ausgewählten Inhalt im Dropdownmenü rechts daneben, wird die Abfrage in der Datei „`showProjectsOf.q`“ ausgeführt, in der vorher der ausgewählte Inhalt der eingefügt wird.

Die beiden letzten Elemente rechts unten sind am besten nach der Nutzung zu erkennen (siehe Abbildung 29). Die Häkchenbox mit dem Text „Reasoner läuft“ zeigt den aktuellen Status des Reasoners an. Läuft dieser, ist ein Häkchen gesetzt, andernfalls nicht. Der rechts daneben liegende Fortschrittsbalken zeigt den Fortschritt des aktuellen Vorgangs (z.B. Reasoner Start oder Abfrage-Ausführung) an.

In Abbildung 29 ist anschaulich erkennbar, wie die Nutzeroberfläche nach dem Befüllen mit Daten und Ergebnissen aussieht.

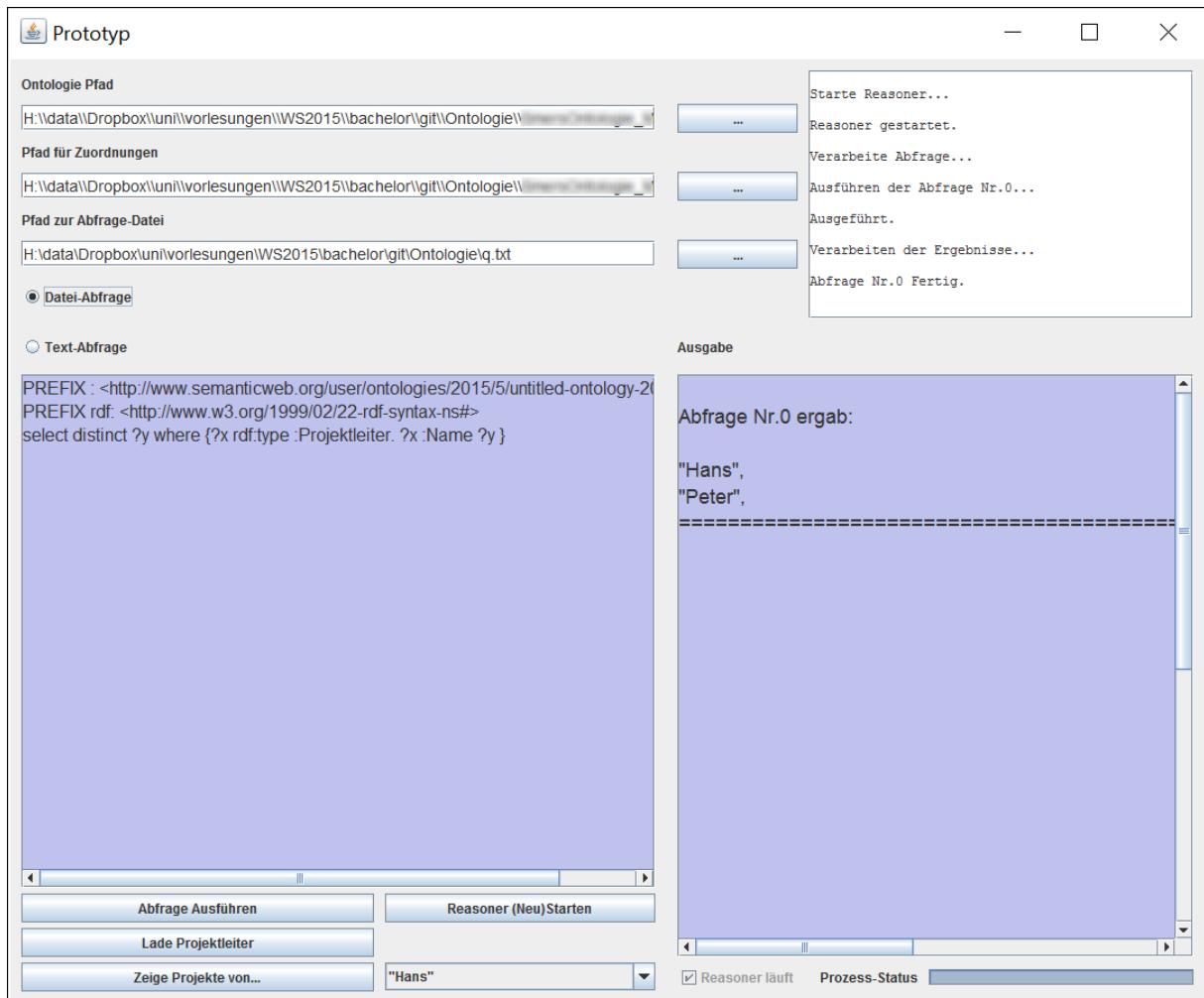


Abbildung 29 Nutzeroberfläche des Prototyps nach Nutzung

Die Ansicht in Abbildung 29 ergibt sich, wenn der Reasoner mit den in der Abbildung gezeigten Dateien für Ontologie und Zuordnungen initialisiert wird. Bei den Datenquellen handelt es sich um Test-Daten. Die Ontologie ist dem Projekt entnommen, und deren Zuordnungen sind in dieser Arbeit entstanden. Im Abfrage-Textfeld kann die SPARQL-Abfrage betrachtet werden, die zu dem rechts stehenden Ergebnis geführt hat. Die hier gezeigten Daten stammen aus einer Excel-Datei, die Teiid als virtuelle Datenbank exponiert. Zudem ist erkennbar, dass der Button „Lade Projektleiter“ betätigt wurde, da das Dropdownmenü mit der Option „Hans“ gefüllt ist. Im Info-Textfeld werden die ausgeführten Prozesse dokumentiert. Abbildung 29 zeigt also, wie die Nutzeroberfläche sich durch die Nutzung verändert und wie die Informationen angezeigt werden.

Über die in der Nutzeroberfläche angezeigten Möglichkeiten hinaus, kann der/die NutzerIn das Log-Level durch das Bearbeiten der Datei „logback.xml“ im log-Verzeichnis des Prototyps fein granular einstellen.

5.1.2 Teiid

Teiid stellt einen wichtigen und notwendigen Bestandteil der gesamten Lösung dar. Durch Teiid wird der Zugriff auf die Datenquellen ermöglicht. Außerdem ist Teiid dazu in der Lage, Daten aus vielen verschiedenen Datenquellen (Excel, SQL, Text) zu beziehen und an den Prototyp weiterzuleiten.

Teiid ist so konfiguriert, dass die Datenquellen (Excel-Dateien) als virtuelle Datenbanken exponiert werden. Dazu ist es notwendig, diese über eine Konfigurations-Datei einzubinden. Die in dieser Arbeit eingesetzte Konfiguration für die virtuelle Datenbank mit dem Namen „snik“ ist vollständig im Anhang zu finden (8.4.6).

Zum besseren Verständnis wird das Auslesen der Datenquellen an einem Beispiel veranschaulicht.

Gegeben sei die folgende Definition für eine virtuelle Datenbank:

```
<vdb name="snik" version="1">
    <model visible="true" name="contrInv1">
        <property name="multisource" value="false"/>
        <property name="importer.headerRowNumber" value="3"/>
        <property name="importer.ExcelFileName" value="Controllingliste Invest fur ein Projekt.xlsx"/>
        <source name="xls-snik-budgetlisten" translator-name="excel" connection-jndi-name="java:/xls-snik-budgetlisten"/>
    </model>
</vdb>
```

Abbildung 30 Ausschnitt der Konfiguration für SNIK-VDB

Die erste und letzte Zeile werden benötigt um zu definieren unter welchem Namen die VDB von Teiid exponiert werden soll. In diesem Fall lautet der Name „snik“.

Der Parameter `name` innerhalb der `model`-Tags gibt an, unter welchem Namen die Datei als eine Tabelle aufrufbar sein soll. In diesem Beispiel ist der Tabellenname also: `contrInv1`. Die Spaltennamen der virtuellen Tabelle ergeben sich aus den durch den `importer.headerRowNumber` angegebenen Parameter und dessen „`value`“ (in diesem Fall 3). Dadurch wird definiert, dass die in Zeile 3 gefundenen Elementen als die Spaltennamen zu verstehen sind. Abbildung 31 zeigt eine Datei mit Testdaten. Gelb eingefärbt sind die Spaltennamen, die sich durch den Parameter `name="importer.headerRowNumber" value="3"` ergeben. Die eigentlichen Daten die am Ende in den Zeilen der Tabelle zu finden sind, blau eingefärbt.

A	B	C	D	E	F	G	H	I	J	K	L
1 Kto:698140	2007-08-003	Budget Stand X	Budget	Auftragsvolumen	Rest						
2 Hans		10.000 €		60000	52000	8000					
3 Lieferant	Angebot-Nr.	Inhalt	Projektleiter	SAP-Nr.	Auftragsvolumen	Auftragsdatum	Kostenstelle	KostenAnteil	KostenAnteil	KostenAnteil	MVZ
4 LieferantA	37812	test1	Hans	3819	1000	14.01.2015	a	0	0	0	
5 LieferantB	326319	test2	Peter	5349	20000	13.04.2015	a	0	0	0	
6 LieferantA	3718793	test3	Peter	47283	1000	05.03.2014	a	0	0	0	
7 LieferantB	789349	test4	Hans	47328	30000	18.02.2015	a	0	0	0	
8											
Zusammenfassung Projekte											

Abbildung 31 Excel-Datenquelle Controlling-Liste Sachkonto mit Test-Daten

Standardmäßig werden die Zeilen unter der angegebenen Zeile (in diesem Fall Zeile 3) als Daten verwendet.

Jede Datenquelle (Excel-Datei) muss einzeln eingebunden werden. Manche Datenquellen müssen doppelt mit unterschiedlicher Bezeichnung eingebunden werden (z.B. `contrInv1` und `contrInv1s`). Das doppelte Einbinden mancher Dateien ist notwendig, um die singulären Datenfelder in der Excel-Dateien, einzubinden. So sind z.B. in Abbildung 31 in Zeile 1 die Daten Kto:698140, 2007-08-003, Budget Stand X, Budget, Auftragsvolumen und Rest enthalten, die für sich

alleine stehen müssen. Um diese in Ontop verwenden zu können, muss die Datei in der virtuellen Datenbank-Konfiguration ein weiteres Mal mit den folgenden Parametern eingebunden werden:

```
<property name="importer.headerRowNumber" value="1"/>
<property name="importer.dataRowNumber" value="1"/>
```

Dadurch wird die erste Zeile als Spaltennamen und als Datenzeile verwendet. Abbildung 32 zeigt das anschaulich:

A	B	C	D	E	F	G	H	I	J	K	L
1	Kto:698140	2007-08-003	Budget Stanc	Budget	Auftragsvolum	Rest					
2		Hans	10.000 €	60000	52000	8000					
3	Lieferant	Angebot-Nr.	Inhalt	Projektleiter	SAP-Nr.	Auftragsvolum	Auftragsdatu	Kostenstelle	KostenAnteil	Kostenanteil	Kostenanteil MVz
4	LieferantA	37812	test1	Hans	3819	1000	14.01.2015	a	0	0	0
5	LieferantB	326319	test2	Peter	5349	20000	13.04.2015	a	0	0	0
6	LieferantA	3718793	test3	Peter	47283	1000	05.03.2014	a	0	0	0
7	LieferantB	789349	test4	Hans	47328	30000	18.02.2015	a	0	0	0
8	Zusammenfassung Projekte										

Abbildung 32 Excel-Datenquelle Controlling-Liste Sachkonto mit Test-Daten (zweite Konfiguration)

Über diese Art der Einbindung als Tabelle, ist es später über SQL-Abfragen möglich, genau eine Zeile mit singulären Feldern (in diesem Fall Zeile 2) zu erhalten. Das kann durch eine einfache Einschränkung auf die zweite Zeile in der SQL-Abfrage zum Beziehen der Daten erfolgen.

Alle weiteren Konfigurations-Details werden im Anhang 8.4.6 beschrieben.

5.1.3 Protégé-Ontop-Bundle

Das Protégé-Ontop-Bundle wird in der Lösung dafür verwendet, um die Zuordnungen zwischen der Ontologie und der von Teiid exponierten virtuellen Datenbank, die die eigentlichen Datenquellen enthält, zu erstellen. Dazu sind in Protégé zusätzlich der „Datasource Manager“ im Reiter „Ontop Mappings“ und der „Query Editor“ im Reiter „Ontop SPARQL“ notwendig. Diese Reiter werden durch das Ontop-Plug-In in Protégé eingefügt.

Abbildung 33 zeigt den Datasource Manager im Reiter „Ontop Mappings“. In diesem Reiter wird die Konfiguration der Datenquelle (in diesem Fall Teiid) eingestellt. Da die Verbindung über JDBC erfolgt, müssen hier die JDBC-typischen Parameter angegeben werden:

- Connection URL:
 - Hierüber wird dem JDBC-Treiber mitgeteilt, mit welcher virtuellen Datenbank eine Verbindung aufgebaut werden soll.
- Database Username:
 - Die Login-Details für den Teiid Server werden in Form von Username und Passwort übergeben. In dieser Arbeit: „user“.
- Database Password
 - Die Login-Details für den Teiid Server werden in Form von Username und Passwort übergeben. In dieser Arbeit: „b4chelor!“
- Driver class
 - Die Treiber Klasse gibt an, unter welchem Namen der JDBC-Treiber in Java geladen wird. Der Name ist der gleiche wie auch bei Imports, wenn man den Treiber in einer Entwicklungsumgebung nutzen möchte.

Ist der JDBC-Treiber korrekt konfiguriert, kann über den „Test Connection“-Button die Verbindung zu Teiid geprüft werden.

Es kann außerdem notwendig sein den JDBC-Treiber von Teiid beim ersten Starten von Protégé hinzufügen zu müssen (siehe 8.4.2.1.3)

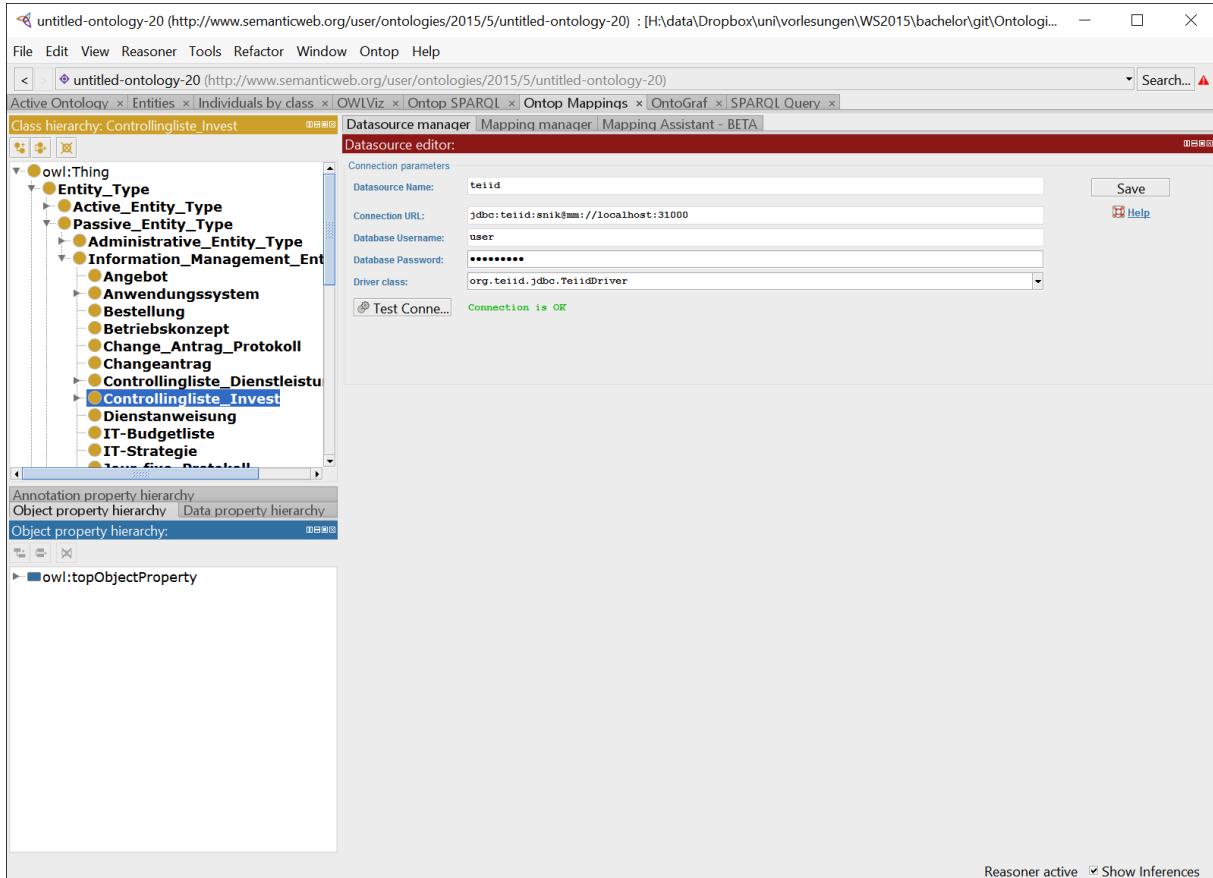


Abbildung 33 Screenshot Protégé Reiter Ontop Mappings → Datasource Manager

Als nächstes werden die eigentlichen Zuordnungen zwischen virtueller Datenbank und der Ontologie erzeugt. Das wird über den Reiter „Mapping Manager“ gemacht (siehe Abbildung 34). Die Zuordnungen sind wie folgt aufgebaut:

```

mappingId      MAPID-42485f535ea04ff8ac7489df1750b0eb
target          :Projekt/{PN}/ a :Projekt ; :Projektnummer {PN} ; :Name
{Beschreibung} ; :Projektbeginn {Beginn} ; :Jahr
{Jahr} ; :isAssociatedWith :Controllingliste_Invest/{PN}/ , :Controllinglis
te_Budget/{PN}/ , :Projektpriorität/{Prio}/ , :Projektleiter/{PL}/ .
source          select "ProjektNr" as PN, "Beschreibung und Bemerkung" as
Beschreibung, "PL" , "geplanter Projektbeginn" as Beginn, "Prio", "Jahr"
from "projLst"."Tabelle1"
  
```

Die **mappingId** ist ein eindeutiger Bezeichner der Zuordnung. Das **target** (Ziel) gibt die Klassen und Relationen an, die von den Zuordnungen aufgebaut werden sollen. In den eckigen Klammern (z.B. {Beschreibung}) sind die Stellen angegeben die später mit den Daten aus der Quelle gefüllt werden. Die **source** (Quelle) ist dann eine SQL-Abfrage, die die jeweiligen Daten mit gleichem Namen enthält. Das bedeutet, dass z.B. die Zeilen aus der Spalte „Beschreibung und Bemerkung“, aus

der Datenquelle „projLst.Tabelle1“ (entspricht der Projektliste Arbeitsmappe „Tabelle1“), der Dateneigenschaft „Name“, der Klasse „Projekt“, zugeteilt werden. Da Ontop Query-Rewriting einsetzt, ist das Framework in der Lage, aus den allgemeinen SQL-Abfragen der Quelle spezifische SQL-Abfragen zu schreiben, wenn sie benötigt werden. Falls der/die NutzerIn also eine SPARQL-Abfrage abschickt, um aus der Ontologie Informationen zum Projektnamen eines Projekts zu erhalten, so wird nicht die komplette SQL-Abfrage an Teiid abgeschickt, wie in der Zuordnung angegeben, sondern nur der notwendige Teil.

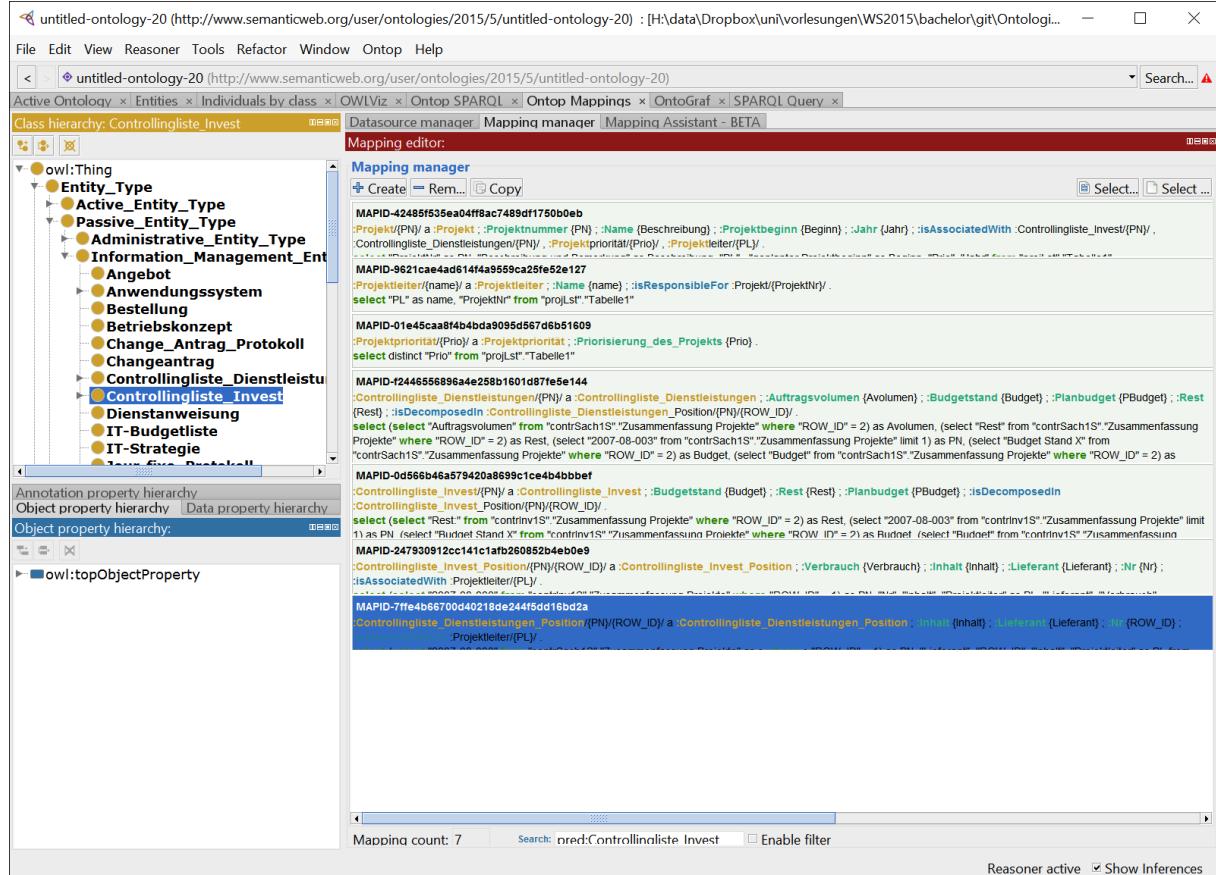


Abbildung 34 Screenshot Protégé Reiter Ontop Mappings → Mapping Manager

Sind die Zuordnungen fertig erstellt, können diese für den Zugriff über die Ontologie verwendet werden. Eine genauere Beschreibung der Zuordnungen ist im Anhang (8.4.5) zu finden.

Auch das Absenden und Testen von SPARQL-Abfragen mit den Zuordnungen ist über Protégé möglich. Dazu kann im Reiter „Ontop SPARQL“ eine SPARQL-Abfrage eingefügt und bei Bedarf gespeichert werden. Die Ergebnisse werden dann im darunterliegenden Bereich angezeigt. Abbildung 35 zeigt die genannten Funktionen auf.

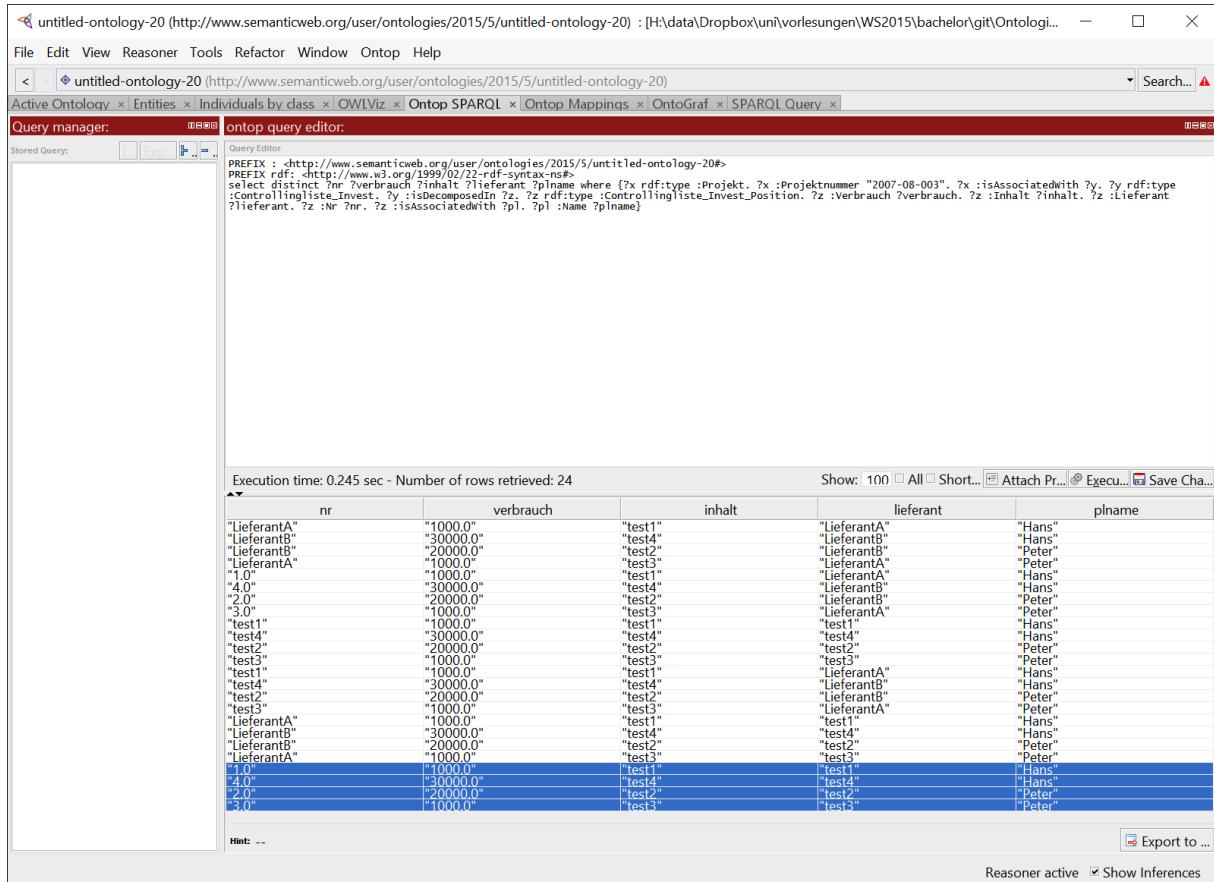


Abbildung 35 Screenshot Protégé Reiter OnTop SPARQL

5.2 Qualitätssicherung

Sowohl Teiid als auch Ontop sind getestete Systeme, für die Unit-Tests vorhanden sind. Dennoch werden Teiid, und die im Prototyp angesprochene API und die Nutzeroberfläche des Prototyps auf ihr Verhalten überprüft.

Da die Tests selbst für die Arbeit weniger relevant sind, sind die Testfälle in einer separaten Excel-Datei zu finden und die Testergebnisse in der im Anhang befindlichen Tabelle 21 aufgelistet.

Die Ergebnisse der Testfälle und deren Zusammenhang mit den Anforderungen werden in der anschließenden Tabelle 18 zusammengefasst.

5.2.1 Zusammenfassung der Testergebnisse

Zur Prüfung der Funktion und der Einhaltung der definierten Anforderungen wurden Tests durchgeführt. Um sicher zu stellen, dass die Nutzeroberfläche des Prototyps ordnungsgemäß funktioniert wurden manuelle Tests durchgeführt, bei denen die Nutzeroberfläche in unterschiedlicher Reihenfolge sowie allgemein auf deren Funktion geprüft wurde. Hierbei sollte vor allem herausgefunden werden, ob der/die NutzerIn in einen Zustand der Nutzeroberfläche geraten kann, in dem eine weitere Verwendung nicht mehr möglich ist. Die automatischen Tests werden eingesetzt, um sicher zu stellen, dass der Prototyp mit defekten Daten umgehen kann und nicht komplett abstürzt. Ergebnis dieser Tests sind meist genaue Fehlerbilder bei der Verwendung bestimmter Eingabedaten. Beispielsweise führt das Laden einer binären Datei an Stelle einer Ontologie zu einer Exception, die vom Prototyp abgefangen und dem/der NutzerIn präsentiert wird. Zudem wurde eine statische Analyse des Codes, auf Fehler durchgeführt, die geholfen hat Fehler im Code schon vor der Kompilierung aufzudecken. Die Prüfung der Komponenten Teiid und Protégé beschränken sich auf manuelle Tests, in denen das Verhalten durch Eingaben in der Nutzeroberfläche geprüft wird. Die Tests sind größtenteils wie erwartet ausgefallen. Probleme sind vor allem bei der Verarbeitung von Inhalten aus Excel-Daten durch Teiid vorhanden:

Datei-Pfade dürfen keine Sonderzeichen oder Umlaute enthalten, obwohl diese in der Praxis vorkommen. Dies muss beim Einbinden der Datenquellen berücksichtigt werden.

Außerdem werden von Teiid bisher keine Word-Dateien unterstützt. Damit wird nur ein Datei-Typ von den zuvor festgelegten unterstützt.

In Bezug auf die Erweiterbarkeit und Modularität der gesamten Lösung lässt sich festhalten, dass durch die verwendeten Komponenten eine leichte Austauschbarkeit und Erweiterbarkeit möglich ist:

Der Föderationsserver Teiid wird über eine standardisierte JDBC-Schnittstelle eingebunden, dadurch ist ein Tausch durch simples ersetzen der JDBC-Konfiguration unter der Annahme, dass sich die Tabellen-Namen, Spalten und Zeilen nicht ändern, möglich. Die Methoden des Prototyps selbst sind so geschrieben, dass sie in wichtige Schritte aufgeteilt sind: Das Ausführen einer Abfrage wird unterteilt in das Starten des Reasoners, Ausführen der Abfrage und das Verarbeiten der Ergebnisse. Zudem wurde die Methode zum Befüllen von GUI-Komponenten durch die Abfrage-Ergebnisse bewusst nicht mit der Methode zur Ausgabe der Daten zusammengeführt (obwohl diese nahezu identisch sind). Das hat den Grund, dass so die Lesbarkeit und der Modulare Aufbau besser erhalten bleibt.

11 der 19 zuvor definierten Abfragen, die der Prototyp beantworten soll, können mit den gegebenen Daten oder wegen nicht erfüllten Anforderungen (z.B. Einbindung von Word-Dateien) nicht oder nur teilweise beantwortet werden. Die Tabelle 19 zeigt den Status der Abfragen und die dazugehörigen Begründungen tabellarisch auf.

Tabelle 18 Anforderung und deren Status (mit Begründung ob erfüllt/nicht erfüllt)

Anforderung	Status	Begründung
Funktion zum Anlegen/Löschen/Bearbeiten von Datenzuordnungen	Erfüllt	Diese Funktionalität wird vom Ontop-Protégé-Bundle zur Verfügung gestellt.
Darstellung von Verknüpfungen (Konzept und Datenquelle - Textuell)	Erfüllt	Diese Funktionalität wird vom Ontop-Protégé-Bundle zur Verfügung gestellt. Sowohl textuelle als auch grafische Beschreibungen werden unterstützt.
Hinzufügen/Löschen/Bearbeiten von Datenquellen	Erfüllt	Diese Funktionalität wird vom Ontop-Protégé-Bundle zur Verfügung gestellt (Teiid als Datenquelle); sogar mehr als textuell. In Teiid können die verschiedenen Datenquellen in Form von Excel-Dateien hinzugefügt oder gelöscht werden.
Aggregation von Daten aus verschiedenen Quellen und Quelltypen	Teilweise erfüllt	Das Aggregieren von Daten aus mehreren Excel-Dateien ist kein Problem. Es können auch Daten aus SQL und Excel zusammengeführt werden. Word-Dateien werden nicht unterstützt.
Beantwortung von Fragestellungen mittels SPARQL-Abfragen	Erfüllt	Die mit den gegebenen Test-Daten beantwortbaren Fragestellungen können beantwortet werden.
Darstellung der Aggregierten- und Rohdaten (Textuell)	Erfüllt	Diese Funktionalität wird vom Ontop-Protégé-Bundle zur Verfügung gestellt (Teiid als Datenquelle). Im Prototyp werden nur die aggregierten Ergebnisse dargestellt, aber keine Rohdaten (Excel-Daten).
Fehlertoleranz gegenüber falschen Datentypen	Erfüllt	Der Prototyp ist umfassend auf dessen Umgang mit defekten, invaliden oder binären Daten (Ontologie-, Zuordnungs-, Abfrage-Dateien) geprüft. Außerdem ist der Umgang mit falschen Pfaden getestet. Die entsprechenden Fehler, die im Zusammenhang mit diesen Dateien auftreten, werden vom Prototyp an den/die NutzerIn weitergemeldet. Hierbei wird außerdem ein möglicher Grund angegeben, der dem/der NutzerIn hilft, das Problem zu beheben. Teiid gibt Fehler mit Dateien im Log aus und zwingt den/die NutzerIn dazu, diese zu beheben. Problematisch sind außerdem Umlaute in den Dateipfaden, da diese nicht korrekt verarbeitet werden. Protégé verhält sich wie der Prototyp.
Fehlertoleranz gegenüber falschen Datenzuordnungen	Erfüllt	Siehe oben. (Teiid ist hier ausgenommen)
Fehlertoleranz gegenüber fehlerhaften Ontologien	Erfüllt	Siehe oben. (Teiid ist hier ausgenommen)
Schließen der Verbindungen nach dem Schließen	Erfüllt	Das Start-Skript für den Prototyp startet Teiid automatisch und beendet Teiid korrekt nach dem Schließen.
Automatisches öffnen der Verbindungen beim Starten	Erfüllt	Das Start-Skript für den Prototyp startet Teiid automatisch und beendet Teiid korrekt nach dem Schließen.
Modularer Aufbau (für	Erfüllt	Bei der Entwicklung des Prototyps ist darauf geachtet

leichte Erweiterbarkeit)		worden, die Komponenten austauschbar und erweiterbar zu halten. Außerdem wurde der Quellcode umfangreich dokumentiert.
Beschaffung der Daten findet zur Abfrage-Zeit statt	Erfüllt	Die Daten werden zur Laufzeit vom Prototyp eingelesen und repräsentieren immer den Stand während der Ausführung der Abfrage. Änderungen der Dateien werden berücksichtigt, sobald sie auf das Speichermedium geschrieben sind.
Zugriff/Aggregation von Word-, Excel- und Webseiten-Daten (lesend)	Teilweise Erfüllt	Das Aggregieren von Daten aus mehreren Excel-Dateien ist kein Problem. Es können auch Daten aus SQL und Excel zusammengeführt werden. Word-Dateien werden nicht unterstützt.
Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)	Erfüllt	Der Prototyp ist umfassend auf dessen Umgang mit defekten, invaliden oder binären Daten (Ontologie-, Zuordnungs-, Abfrage-Dateien) geprüft. Außerdem ist der Umgang mit falschen Pfaden getestet. Die entsprechenden Fehler die im Zusammenhang mit diesen Dateien auftreten werden vom Prototyp an den/die NutzerIn weitergegeben. Hierbei wird außerdem ein möglicher Grund angegeben, der dem/der NutzerIn hilft, das Problem zu beheben. Teiid gibt Fehler mit Dateien im Log aus und zwingt den/die NutzerIn dazu, diese zu beheben. Problematisch sind außerdem Umlaute in den Dateipfaden, da diese nicht korrekt verarbeitet werden.
Kontinuierliches Logging von Abläufen und Fehlern für bessere Wartbarkeit	Erfüllt	Das Logging im Prototyp erfolgt getrennt von Teiid und dem Ontop-Protégé-Bundle. Es ist möglich, das Logging zu konfigurieren und kontinuierliche Logs mit granularem Log-Level zu erstellen.
Unterstützung von Zugriff auf Netzwerk und SharePoint für Datenquellen	Nicht getestet	Zum Zeitpunkt der Durchführung der Arbeit gibt es keine Testumgebung, in der diese Anforderung geprüft werden kann. Generell sind aber alle Komponenten in der Lage zumindest mit Netzwerk-Pfaden umzugehen. Generell ist Teiid, welches für den Zugriff auf die Datenquellen zuständig ist in der Lage, auf Daten eines SharePoint Servers zu zugreifen.
Warnung des Nutzers, bzgl. auftauchender Fehler (z.B. unbehandelte Ausnahmen)	Erfüllt	Alle Fehler im Prototyp werden dem/der NutzerIn in Form einer Fehlermeldung mit Lösungsvorschlägen präsentiert. Genauso verhält sich das Ontop-Protégé-Bundle. Fehler in Teiid werden nur auf der Konsole oder im Web-Interface angezeigt.

Tabelle 19 Status der geforderten SPARQL-Abfragen

SPARQL-Abfrage	Status	Begründung
„Welche Projekte gibt es?“	Erfüllt	Wird erfüllt.
„Zeige Projektplan zu Projekt x“	Erfüllt	Wird erfüllt.
„Wieviel Budget ist für	Erfüllt	Wird erfüllt. Das Budget ist aber als ein String

Projekt X übrig?		eingebunden. Manuelles Verändern des Datenmodells der Excel-Datei ist notwendig, um auch Vergleiche (z.B. A < B) durchführen zu können.
„An welchen Projekten nimmt Person X teil?“	Erfüllt	Wird erfüllt.
„Wann ist das nächste Projekt voraussichtlich fertig?“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden.
„Wann werden neue Ressourcen frei?“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden.
„Zeige den Projekt-Status Report des Projekts X“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden.
„Zeige den Projekt Fertigstellungs-Report des Projekts X“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden. Außerdem werden Word-Dateien nicht unterstützt.
„Zeige das Übergabe-Protokoll des Projekts X“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden. Außerdem werden Word-Dateien nicht unterstützt.
„Zeige die IT-Strategie“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden. Außerdem werden Word-Dateien nicht unterstützt.
„Zeige die Projekt-Warteliste“	Teilweise Erfüllt	Kann auf Grund fehlender Daten nicht komplett beantwortet werden.
„Zeige den Resourcen Plan“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden. Außerdem werden Word-Dateien nicht unterstützt.
„Zeige die Projekt Webseite des Projekts X“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden.
„Zeige das Projekt Portfolio“	Nicht erfüllt	Kann auf Grund fehlender Daten nicht beantwortet werden.
„Für welche Projekte ist Projektleiter X zuständig?“	Erfüllt	Wird erfüllt.
„Welche Dienstleistungen hat Projektleiter X bewilligt?“	Erfüllt	Wird erfüllt.
„Zeige Daten aus Controllingliste_Invest für Projekt X“	Erfüllt	Wird erfüllt.
„Zeige Daten aus Controllingliste_Dienstleistungen für Projekt X“	Erfüllt	Wird erfüllt.
„Welche Projektleiter gibt es?“	Erfüllt	Wird erfüllt.

6 Fazit

6.1 Vorteile der Lösung

OBDA unter der Verwendung von Query Rewriting ist ein zukunftsträchtiger Ansatz: Statt die Daten in die Ontologie zu integrieren oder statische Übersetzungs-Schemata zu entwickeln, um OBDA durchzuführen, kann durch den Einsatz von Query-Rewriting das gleiche Ergebnis erzielt werden. Der Vorteil besteht darin, dass die Zuordnungen und Daten klar von der Ontologie getrennt sind und nur zur Laufzeit virtuell zusammengeführt werden. Das macht es möglich, die Ontologie, Zuordnungen und Datenquellen leichter auszutauschen. Außerdem werden auf diese Weise die Datenquellen zur Laufzeit ausgelesen, was zur Folge hat, dass die Daten immer aktuell sind, im Gegensatz zum Ansatz des Einbindens der Daten in die Ontologie selbst. Beim Aufbau der Lösung wurde außerdem ausschließlich auf freie und quelloffene Software gesetzt, was die Nutzung in der Zukunft des Projekts ermöglicht. Hierbei ist es auch hilfreich, dass die Community um Protégé, Ontop und Teiid recht aktiv ist und alle Komponenten gut dokumentiert sind.

Durch die Verwendung von OBDA kann außerdem der semantische Zusammenhang zwischen den Datenquellen und der Ontologie mittels der Zuordnungen hergestellt werden.

6.1.1 Prototyp

Der Prototyp zeigt das Potenzial der Technologie deutlich auf: Die Chancen, die sich durch OBDA über Ontop ergeben, sind groß. Besonders deshalb trifft dies zu, weil Ontop als ausgereift zu bezeichnen ist und zuverlässig funktioniert. Durch die Kommunikation mit Ontop, über die standardisierte OWLAPI, wird außerdem leichtere Entwicklung und Handhabung von über Abfragen bezogene Daten ermöglicht, welche im Prototyp weiter verarbeitet werden können. Antworten auf Abfragen sind immer aktuell, da die Datenquellen zur Laufzeit ausgelesen werden.

Bei der Entwicklung des Prototyps wurde darauf geachtet, dass der Code wiederverwendbar und gut kommentiert ist. Das Erweitern des Prototyps ist also für den weiteren Projektverlauf eine Option. Dafür bringt der Prototyp schon erste nützliche Funktionen mit. Zum einen ist das Log-Level fein einstellbar, zum anderen werden die in der Nutzeroberfläche getätigten Eingaben gespeichert, und schließlich ist die Oberfläche übersichtlich und leicht zu bedienen.

6.1.2 Teiid

Teiid ist in seiner Standard-Installation bereits in der Lage, viele verschiedene Datenquellen in unterschiedlichen Formaten einzubinden und als VDB zu exponieren. Für Teiid werden ständig neue Translatoren und Funktionen entwickelt, die es ermöglichen, zuvor nicht erschließbare Datenquellen mit Teiid einzubinden. Ist Teiid einmal aufgesetzt, funktioniert die Software zuverlässig und beantwortet Abfragen schnell.

6.1.3 Ontop-Protégé-Bundle

Dadurch, dass Ontop durch ein Plug-In in Protégé eingebunden ist, kann der bekannte Arbeitsfluss in Protégé erhalten bleiben. Es kommen lediglich neue Funktionen von Ontop hinzu. Mittels der besseren farblichen Darstellung der Zuordnungen und der Möglichkeit des Testens der SQL-Abfragen auf die Datenquelle, wird das Erstellen der Zuordnungen erleichtert. Dabei hilft auch, dass sowohl die Ontologie als auch die Zuordnungen beide in Protégé vorliegen und somit verglichen werden können.

6.2 Aktuell ungelöste Schwierigkeiten

Die gesamte Lösung funktioniert prinzipiell. Die Komponenten konnten aber im Berichtszeitraum nicht perfekt aufeinander abgestimmt werden. Im Prinzip handelt es sich um drei Komponenten, die für sich stehen und erst im reibungslosen Zusammenspiel ihr volles Potenzial ausschöpfen. Ein gemeinsames Konfigurationsprogramm zur Erstellung der Zuordnungen zwischen Ontologie und der Integration der Datenquellen wäre wünschenswert.

6.2.1 Prototyp

Der Prototyp ist nicht viel mehr als ein Proof-of-Concept. Beispielsweise sind Funktionen für die Erstellung von Zuordnungen oder die Unterstützung für das Einbinden von Datenquellen nicht im Prototyp selbst, sondern in anderen Komponenten enthalten. Deshalb ist es z.B. notwendig, das Ontop-Protégé-Bundle für die Erstellung von Zuordnungen zu verwenden. Die Datenquellen in Teiid müssen manuell, über das Erstellen von Konfigurations-Dateien, eingebunden werden.

Es gibt Software-Fehler und Beschränkungen in Ontop, z.B. dass kein voller SPARQL 1.1 Support besteht und verschachtelte SQL-Abfragen nicht optimierbar sind, die durchblicken lassen, dass Ontop sich noch in einem Entwicklungsstadium befindet. Ungünstig ist speziell, dass Fehlermeldungen, die von Ontop kommen, teilweise wenig hilfreich sind. Erst durch das Einstellen von Debug-Level Logs kann man Rückschlüsse auf die zugrundeliegenden Probleme ziehen.

Als Nachteil zu nennen ist auch, dass es zum jetzigen Zeitpunkt keine Prüfung seitens des Prototyps gibt, ob Teiid korrekt läuft und aktiv ist.

6.2.2 Teiid

Von den ursprünglich gewünschten Datenquellen wird nur Microsoft Excel unterstützt. Für das Einbinden von Word-Dateien ist ein weiterer Translator notwendig, der noch nicht entwickelt ist. Außerdem konnte das Beziehen von Daten über das Netzwerk oder einen SharePoint nicht getestet werden. Obwohl diese Funktionen von Teiid unterstützt werden, ist also nicht klar, welche Probleme hierbei aufkommen.

Das Einbinden von Datenquellen ist ein zeitaufwändiger und umständlicher Prozess:

- Es ist notwendig, die Datenquellen in gewisser Form an Teiid zu übergeben:
 - Sonderzeichen in Spaltennamen sind für Teiid ein Problem, da diese nicht richtig verarbeitet werden. Es ist also besser, keine Sonderzeichen in den Spaltennamen zu verwenden.
 - Singuläre Felder machen es notwendig, die Datenquelle mehrfach einzubinden. (Details siehe 5.1.2). Das ist beim Erstellen der Zuordnungen umständlich und gleichzeitig doppelte Arbeit.
- Jede Datenquelle, also jede Excel-Datei, muss einzeln eingebunden werden. Zwar erfolgen das Auslesen der Informationen und des Datenmodells automatisch, es ist aber dennoch umständlich. In der Zukunft könnte dieses Problem allerdings gelöst werden, da bereits eine Anfrage für die entsprechende Funktion beim Hersteller von Teiid existiert⁵⁹. Das automatische Einbinden würde aber dennoch nur dann funktionieren, wenn die Datenquellen in ihrem Aufbau alle identisch wären.

⁵⁹ https://issues.jboss.org/browse/TEIID-3599#_=_

Aus diesen Punkten ergibt sich, dass Umstrukturierungen an den Dateien notwendig sind, um diese im vollen Umfang verwendbar machen zu können.

6.2.3 Ontop-Protégé-Bundle

Ebenfalls von Nachteil ist aktuell die Mühe, die das Erstellen von Zuordnungen bereitet. Es besteht keine genaue Syntax-Prüfung während des Schreibens. Meistens wird beim Erstellen der Zuordnung nur die Information angezeigt, dass es einen Fehler in der Zuordnung gibt. Wo er aufgetreten ist und wie der Fehler aussieht, ist oft nicht klar. Kleine Schreibfehler in den Ontologie-Elementen der Zuordnungen machen Schwierigkeiten. Diese werden von Ontop akzeptiert, weil es keine Prüfung gibt, ob die erzeugten Objekte überhaupt existieren. Das erschwert den ohnehin schon mühseligen Prozess des Erstellens der Zuordnungen. Zwar ist in Ontop ein Assistent zur erleichterten Erstellung von Zuordnungen enthalten, dieser funktioniert aber nur sehr unzuverlässig und ist daher nicht wirklich einsetzbar. Hinzu kommt, dass SQL-Abfragen, die über die „Test-SQL“-Funktion korrekte Informationen ausgeben, bei der Verwendung des Reasoners gegebenenfalls trotzdem nicht funktionieren. Das kann passieren, wenn Spaltennamen oder Tabellenbezeichnungen nicht in Anführungszeichen gesetzt sind, weil der Reasoner, in diesem Fall, Veränderungen an den Bezeichnungen vornimmt. Es ist deshalb extrem wichtig, auf die Schreibweise der Spalten- und Tabellennamen zu achten und diese in Anführungszeichen („) zu setzen, um zu vermeiden, dass diese vom Reasoner verändert werden. Singuläre Datenfelder in Datenquellen machen es notwendig, umständliche und für Ontop schwer zu verarbeitende SQL-Abfragen zu erzeugen, die verschachtelt sind und über eine WHERE-Anweisung⁶⁰ auf eine Zeile beschränkt sind. Das hat Einfluss auf die Geschwindigkeit, in der Abfragen beantwortet werden. Aktuell ist das zwar kein Problem, es kann sich aber zu einem Problem entwickeln, wenn die Zuordnungen mehr und komplizierter werden. Ungünstig ist auch, dass manche SQL-Funktionen werden vom Reasoner nicht unterstützt werden (z.B. „Limit“).

Kurzum ist die Lösung zunächst als ein Proof-of-Concept zu verstehen, welches klar die Vor- und Nachteile der Verwendung von Ontop und Teiid aufzeigt. Auch wenn das Ergebnis noch kein fertiges Endprodukt darstellt, handelt es sich um einen Ansatz, der in der Zukunft und, mit Einschränkungen auch jetzt schon, viel Potenzial und Vorteile gegenüber anderen Ansätzen mit sich bringt.

⁶⁰ http://www.w3schools.com/sql/sql_where.asp

7 Zukünftige Arbeiten

Insgesamt handelt es sich bei der Lösung um eine spannende und top aktuelle Technologie, die für zukünftige Arbeiten verwendet werden kann. Der Weg, weiterhin auf OBDA durch Query-Rewriting zu setzen, ist wegweisend und birgt viele Entwicklungsmöglichkeiten für die Zukunft. Zukünftige Arbeiten sind vor allem im Hinblick auf die Zuverlässigkeit von Ontop und das Zusammenspiel mit Teiid interessant. Hier besteht ein großes Potenzial, da Teiid ein sehr mächtiger Föderationsserver ist, der es ermöglicht, Datenquellen verschiedenster Typen mit einem JDBC-Treiber als eine virtuelle Datenbank zu exponieren. Eine Vereinfachung des Einbindens von Datenquellen ist in dessen Zusammenhang durchaus ein wichtiges Thema. Deshalb wird es in naher Zukunft wichtig sein, eine geeignete Umstrukturierung der Datenquellen und der Ontologie vorzunehmen, um den Einbindungs- und Zuordnungs-Prozess in Teiid und Ontop zu erleichtern bzw. überhaupt erst zu ermöglichen. Zudem ist es sinnvoll, Zuordnungen und Einbindungen aller wichtigen Bestandteile des Informationsmanagements im Krankenhaus vorzunehmen. Aktuell sind nur Teile davon in Form von Test-Daten und entsprechenden Zuordnungen vorhanden.

Um dem Ziel eines fertigen Produkts (z.B. CION) näher zu kommen, ist es sinnvoll, folgendes zu implementieren:

- Stärkere Integration der drei Komponenten Ontop, Teiid und Protégé. Sind das Erstellen von Zuordnungen zwischen Ontologie und Datenquelle und das Einbinden von Datenquellen in Teiid in einer einzigen Applikation möglich, wird die Handhabung erleichtert und Protégé gegebenenfalls komplett überflüssig. Außerdem ist eine Prüfung sinnvoll, ob Teiid läuft, um Fehler beim Datenzugriff zu vermeiden.
- Verbesserung der Nutzeroberfläche in Bezug auf dessen Nutzerfreundlichkeit und Aufbau.
- Erleichtertes Erstellen der Zuordnungen, durch folgenden Ansatz. Es ist prinzipiell möglich, die Zuordnungen auf der Basis grafischer Methoden zu erzeugen, z.B. durch Verbinden von Elementen in einem Graph, mittels Linien. Eine solche Zuordnungsmethode würde zunächst alle Datenpunkte laden und dann alle Elemente der Ontologie. Der/die NutzerIn kann dann die zu verbindenden Elemente mit Mausklicks verbinden. Die Umsetzung einer solchen Funktion würde die Erzeugung von Zuordnungen beschleunigen und weniger Raum für Fehler lassen.
- Erleichtertes Erstellen der SPARQL-Abfragen. Auch hier wäre ein grafischer Ansatz denkbar, der durch das Anklicken der Elemente in der Ontologie automatisch eine SPARQL-Abfrage erzeugt, durch die das ausgewählte Element aus der Ontologie abgefragt werden kann.
- Eine weitere essentielle Fragestellung lautet, wie die Daten aus Word-Dateien und anderen Datenquellen des Informationsmanagements im Krankenhaus in Teiid verfügbar gemacht werden können. Die Entwicklung passender Translatoren bietet hier einen Ansatzpunkt. Hierfür könnten die Translatoren auf in Word-Dateien eingebaute, unsichtbare Markierungen verarbeiten, die mit einem Word-Add-in erzeugt wurden.

Spannend ist auch die Überlegung, eine umfassende Software über den CION hinaus zu entwickeln, die nur mit einer passenden Ontologie und Zuordnungen ausgestattet werden muss und sehr anpassungsfähig sein müsste. Nur durch Konfigurations-Aufwand wäre es dann möglich, OBDA in beliebigen Bereichen durchzuführen. Zum Erreichen eines solchen Ziels ist weitere Forschung in Bezug auf das Verarbeiten der erhaltenen Daten aus den SPARQL-Abfragen über die OWLAPI sinnvoll. Gerade für solche Anwendungen spielt Ontop einen großen Vorteil aus, der genutzt werden sollte.

8 Anhang

8.1 Programmcode der startReasoner-Methode

```
/*
 * Starten des Reasoners. Dieser Teil muss normalerweise (außer bei
Änderungen der Parameter)
 * nur einmal gemacht werden. Außerdem benötigt dieser Schritt etwas mehr
Zeit.
 * @param owlFile Pfad zu der OWL-Datei der Ontologie.
 * @param obdaFile Pfad zu der OBDA-Datei mit den Zuordnungen
 */
private boolean startReasoner(String owlFile, String obdaFile) {
    updateProgressbar(1, 4);
    logToInfoArea("Starte Reasoner...\\n");
    try {
        //Bereite Ontologie vor
        OWLOntology ontology = OWLManager.createOWLOntologyManager().
            loadOntologyFromOntologyDocument(new File(owlFile));
        //Lade Zuordnungen.
        OBDAModel obdaModel = new
MappingLoader().loadFromOBDAFile(obdaFile);
        updateProgressbar(2, 4);
        //Erstelle Quest Reasoner Factory
        QuestOWLFactory factory = new QuestOWLFactory();
        updateProgressbar(3, 4);

        //Erstelle Konfiguration aus Zuordnungen
        QuestOWLConfiguration config =
QuestOWLConfiguration.builder().obdaModel(obdaModel).build();

        //Erstelle Reasoner Instanz
        reasoner = factory.createReasoner(ontology, config);
        updateProgressbar(4, 4);
        logToInfoArea("Reasoner gestartet.\\n");
        return true;
    } catch (IOException ex) {
        //Falls Ontologie/OBDA-Datei nicht verarbeitet werden kann.
        JOptionPane.showMessageDialog(null,
            ex.getMessage() + "\\n" +
            "Mögliche Gründe hierfür sind: \\n" +
            "OBDA/Ontologie-Datei ist fehlerhaft oder nicht
unterstützt.\\n",
            "Fehler beim Lesen der Ontologie/Zuordnungen",
            JOptionPane.ERROR_MESSAGE);
        logToInfoArea(ex.getMessage());
    } catch
(NoClassDefFoundError|NoSuchMethodError|OWLOntologyInputSourceException ex)
{
    //gekürzt
} catch (ReasonerInternalException ex) {
    //gekürzt
} catch (OBDAException ex) {
    //gekürzt
} catch (Exception ex) {
    //gekürzt
}
updateProgressbar(0, 4);
return false;
}
```

8.2 Programmcode der executeQuery-Methode

```
/**  
 * Diese Methode führt die angegebene Abfrage (als Text oder Datei) aus und  
 gibt den Inhalt im Info-Fenster aus.  
 * @param sparqlFile Der Pfad zu der Abfrage-Datei.  
 * @param sparqlQuery Eine SPARQL-Abfrage als String.  
 * @param getQueryFromFile Der Modus dieser Methode. Ist dieser Parameter  
 auf wahr gesetzt, so wird der Datei-Pfad  
 *                         verwendet und der Abfrage-String ignoriert.  
 */  
private QuestOWLResultSet executeQuery(String sparqlFile, String  
sparqlQuery, boolean getQueryFromFile) {  
    logToInfoArea("Verarbeite Abfrage...\\n");  
    updateProgressbar(1, 4);  
  
    //Switch für den Modus (Datei oder String)  
    if (getQueryFromFile)  
        try {  
            sparqlQuery =  
Files.lines(Paths.get(sparqlFile)).collect(joining("\\n"));  
        } catch (Exception ex){  
            //Genereller catch Exception-block, da sich eine defekte Datei  
in versch. Exceptions äußern kann.  
            JOptionPane.showMessageDialog(null, ex.getMessage() + "\\n" +  
                "Mögliche Gründe hierfür sind:\\n" +  
                "Ausgewählte Datei ist eine binäre Datei,  
beschädigt oder nicht vorhanden."  
                , "Fehler beim Laden der Abfrage-Datei",  
                JOptionPane.ERROR_MESSAGE);  
            logToInfoArea(ex.getMessage() + "\\nAbfrage abgebrochen.\\n\\n");  
            return null;  
        }  
  
    try {  
        //Kommuniziere mit Reasoner, erstelle Abfrage und führe diese aus.  
        logToInfoArea("Ausführen der Abfrage Nr." + queryNumber.toString()  
+ "...\\n");  
        QuestOWLConnection conn = reasoner.getConnection();  
        QuestOWLStatement statement = conn.createStatement();  
        //Führe Abfrage aus  
        QuestOWLResultSet resultSet = statement.executeTuple(sparqlQuery);  
        updateProgressbar(2, 4);  
  
        //Debug-Details  
        logger.debug("Ausgeföhrte SPARQL-Abfrage:");  
        logger.debug(sparqlQuery);  
  
        //gibt Abfrage-Ergebnis zurück  
        return resultSet;  
    } catch (UncheckedIOException ex) {  
        //gekürzt  
    } catch (RuntimeException ex) {  
        //gekürzt  
    } catch (Exception ex) {  
        //gekürzt  
    }  
    return null;  
}
```

8.3 Programmcode der showQueryResult-Methode

```
/**  
 * Gibt das aus der Abfrage erhaltene Resultset in der UI aus.  
 * @param resultSet Abfrage-Antwort in Form eines resultsets  
 */  
private void showQueryResult (QuestOWLResultSet resultSet){  
    logToInfoArea("Ausgeführt.\n");  
    //Auslesen des Abfrage-Ergebnisses.  
    try {  
        int columns = resultSet.getColumnCount();  
        logToInfoArea("Verarbeiten der Ergebnisse...\n");  
        logToResultArea("Abfrage Nr." + queryNumber.toString() + "  
ergab:\n");  
        updateProgressbar(3, 4);  
  
        //Loop für die Ausgabe der einzelnen Ergebnisse im Ausgabe-Textfeld  
        while (resultSet.nextRow()) {  
            for (int i = 1; i <= columns; i++) {  
                //Hole OWL Objekt aus der Ergebnismenge  
                OWLObject owlObject = resultSet.getOWLObject(i);  
                logToResultArea(owlObject.toString() + ", ");  
            }  
        }  
        resultSet.close();  
  
        logToResultArea("=====---\n=====---\n");  
  
        //Erfolgreich  
        logToInfoArea("Abfrage Nr." + queryNumber.toString() + "  
Fertig.\n");  
        queryNumber += 1;  
        updateProgressbar(4, 4);  
    } catch (OWLEException ex) {  
        //falls Probleme bei Zusammenspiel von Abfrage und Ontologie  
        auftreten  
        JOptionPane.showMessageDialog(null, ex.getMessage() +  
            "\nMögliche Gründe: Abfrage-Datei nicht an  
Ontologie angepasst, " +  
            "Verarbeitung der Ergebnisse schlug fehl",  
            "Fehler beim Verarbeiten der Ergebnisse",  
            JOptionPane.ERROR_MESSAGE);  
        logToInfoArea(ex.getMessage() + "\nAbfrage abgebrochen.\n\n");  
    } catch (Exception ex) {  
        //gekürzt  
    }  
}
```

8.4 Schritt für Schritt Anleitung

8.4.1 Voraussetzungen

- Java 1.8
- Getestet wurde Windows (8.1 und 10 in 64-Bit). Es ist aber anzunehmen, dass generell die folgenden Betriebssysteme unterstützt werden:
 - Windows Vista und neuer
 - Linux
 - Mac OS

8.4.2 Installation und Erst-Konfiguration

Bevor die eigentliche Beschreibung erfolgt, zwei kurze Hinweise:

*Die hier vorgestellten Befehle beziehen sich auf Windows. Die Skript-Namen und Befehle können auf anderen Betriebssystemen unterschiedlich sein. Beispiel: *.sh statt *.bat.*

Zu der Arbeit gibt es ein fertiges ZIP-Paket (im Folgenden „Komplettpaket“ genannt), welches alle notwendigen Programme enthält, um die gesamte Lösung verwenden zu können. Das Komplettpaket muss nach der Installation nicht mehr konfiguriert werden. Zum besseren Verständnis und, falls die Lösung neu aufgesetzt werden soll, werden diese Schritte beschrieben. Kapitel 8.4.2.1 ist optional.

8.4.2.1 (Optional) Installation aktuellster Teiid und Protégé-Version

Falls es gewünscht ist, den Prototyp mit neuen Versionen von Protégé und Teiid zu verwenden, so muss beim Herunterladen auf Folgendes geachtet werden:

8.4.2.1.1 Download und Installation

1) Herunterladen der aktuellsten Teiid Version.

1. Die aktuellste Teiid Version kann auf der Teiid-Webseite⁶¹ heruntergeladen werden.
Wichtig ist, dass die folgenden Elemente heruntergeladen werden:

- „Teiid Runtime“ (der eigentliche Server) und der dazu passende JDBC-Treiber. Beim Download von Teiid ist darauf zu achten, dass das Paket mit WildFly⁶² oder EAP gewählt wird. Teiid ist im Prinzip nichts anderes, als eine spezielle Applikation, die vom Applikationsserver (JBoss EAP oder Wildfly) ausgeführt wird. Wildfly ist der neue Projektname für den Applikationsserver.
- (Optional) Falls der Teiid Designer verwendet werden soll, um eigene Datenmodelle der Datenquellen zu erzeugen, so muss darauf geachtet werden, dass die heruntergeladene Version kompatibel zum Teiid Designer ist. Die Faustregel lautet hier: die ca. 2 Versionsnummern ältere Version ist kompatibel. Für mehr Details kann die Kompatibilitätsliste auf der Teiid Designer Webseite⁶³ betrachtet werden.

2) Herunterladen des Ontop-Protégé-Bundles.

1. Zum Herunterladen wird die Projekt-Webseite auf Sourceforge⁶⁴ aufgerufen und die aktuellste Versionsnummer gewählt. Im Ordner befindet sich auch ein Download für

⁶¹ <http://teiid.jboss.org/downloads/>

⁶² <http://wildfly.org/>

⁶³ http://teiddesigner.jboss.org/designer_summary/downloads.html

⁶⁴ <https://sourceforge.net/projects/ontop4obda/files/>

ein „ontop-Protégé-bundle-X.XX.XX.zip“, wobei X.XX.XX die Versionsnummer darstellt. Diese Datei herunterladen.

3) Entpacken beider Downloads.

1. Erstellen eines neuen Ordners (z.B. mit dem Namen „Prototyp“).
2. Entpacken der in Schritt 1 und 2 heruntergeladenen Downloads in den erstellten Ordner (Prototyp).
 - Es sind nun 2 Ordner (Teiid und Ontop) sowie eine jar-Datei (der JDBC-Treiber) im Ordner.

8.4.2.1.2 Konfiguration von Teiid

1) Starten von Teiid.

1. Öffnen eines Konsolen-Fensters im ./bin Verzeichnis im Teiid-Ordner. Beispiel:
„D:\bachelor\tools\teiid-8.11.5\bin“
2. Der Befehl
.\\standalone.bat -c standalone-teiid.xml
startet Teiid.

2) Erstellen von Nutzerkonten für Teiid.

1. Teiid hat zum jetzigen Zeitpunkt noch keine Nutzerkonten, deshalb müssen diese erst angelegt werden.
2. Öffnen eines zweiten Konsolen-Fensters (das andere unbedingt offen lassen!), im ./bin-Verzeichnis.
3. Nun den folgenden Befehl eingeben:
.\\add-user.bat

Dieser öffnet die Nutzerkonten-Erstellung von Teiid:

```
What type of user do you wish to add?  
a) Management User (mgmt-users.properties)  
b) Application User (application-users.properties)
```

4. Es müssen nun zwei Nutzer erstellt werden: ein „Management User“ (also Administrator) und ein „Application User“ (normaler Nutzer).

- Eingaben für den normalen Nutzer:

Zunächst wird Option b gewählt.

(a) : **b**

Als nächstes muss ein Nutzernname angegeben werden:

Username : **nutzer**

Und dann dessen Passwort (Anforderungen beachten!):

Password :

Re-enter Password :

Auf die Frage

What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none) []:

(Ist der Standard in Ordnung, also) einfach Enter drücken.

Die darauf folgende Frage wird bejaht:

```
About to add user 'nutzer' for realm 'ApplicationRealm'  
Is this correct yes/no? yes
```

Und bei der letzten Frage wird „no“ ausgewählt, da es sich um eine lokale Installation handelt:

```
Is this new user going to be used for one AS process to  
connect to another AS process?  
e.g. for a slave host controller connecting to the master
```

- or for a Remoting connection for server to server EJB calls.
 yes/no? **no**
 - **Eingaben für den Administrator:**
 Zunächst wird Option a gewählt (oder Enter gedrückt, da es sich um den Standard-Wert handelt).
 - (a) : **a**
 - Als nächstes muss ein Nutzernname angegeben werden:
Username : manager
 - Und dann dessen Passwort (Anforderungen beachten!):
Password :
Re-enter Password :
 - Auf die Frage**
 What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none) []:
 (Ist der Standard in Ordnung, also) einfach Enter drücken.
 - Die darauf folgende Frage wird bejaht:
 About to add user 'manager' for realm 'ManagementRealm'
 Is this correct yes/no? **yes**
 - Und bei der letzten Frage wird "no" ausgewählt, da es sich um eine lokale Installation handelt:
 Is this new user going to be used for one AS process to connect to another AS process?
 e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
 yes/no? **no**
5. Die beiden Nutzerkonten sind nun erstellt und können für das Einloggen (z.B. über den JDBC-Treiber, oder die Web-Oberfläche) verwendet werden.
- 3) Hinzufügen des Translators
 1. Siehe Abschnitt 8.4.6
 - 4) Erstellen der VDB-Konfiguration
 1. Siehe Abschnitt 8.4.6
 - 5) Bereitstellen der VDB-Konfiguration
 1. Hierfür muss die erzeugte „name.vdb“ in den Ordner **TEIID_ORDNER\standalone\deployments** verschoben werden.
 2. Die Bereitstellung erfolgt innerhalb von Sekunden.
 3. Bei erfolgreicher Bereitstellung wird eine Datei mit dem gleichen Dateinamen wie die Konfiguration mit der Dateierweiterung „.deployed“ erzeugt. Bei Fehlern wird „.undeployed“ erzeugt.
 4. Durch das Löschen der „.deployed“ oder „.undeployed“-Datei kann der Bereitstellungsprozess erneut gestartet werden.

8.4.2.1.3 Konfiguration von Protégé

- 1) Starten von Protégé.
 1. Starte run.bat im Protégé-Ordner
- 2) Konfiguration des JDBC-Treibers von Teiid
 1. Klick auf File → Preferences...
 2. Wechseln zum Reiter „JDBC Drivers“

3. Klick auf „Add“
 4. Es öffnet sich ein Fenster, in welches die folgenden Informationen eingetragen werden müssen:
 - Description: Eine Beschreibung, für den JDBC-Treiber
 - Classname: Der Klassenname des JDBC-Treibers. Für Teiid lautet dieser „org.teiid.jdbc.TeiidDriver“
 - Driver File (jar): Der Pfad zu dem heruntergeladenen JDBC-Treiber (kann über „Browse“ ausgewählt werden)
 - Bestätigen mit „OK“
 5. Es sollte nun eine neue JDBC-Treiber-Definition vorhanden sein.
 6. Bestätigen mit „OK“
- 3) Hinzufügen einer Datenquelle
1. Öffnen des Reiters „Ontop Mappings“ → „Datasource manager“
 2. Unter „Datasource Name“ kann nun ein Name für die Verbindung angegeben werden.
 3. Als „Connection URL“ muss für Teiid die folgende URL angegeben werden:
 - jdbc:teiid:VDB_NAME@mm://localhost:31000
 - Als VDB_NAME muss der Name der in Teiid hinterlegten VDB angegeben werden. Statt localhost kann auch ein entferntes System angegeben werden.
 - Beispiel: jdbc:teiid:snik@mm://localhost:31000
 4. Unter „Database Username“ und „Password“ müssen die Login-Informationen des normalen Nutzers von Teiid angegeben werden.
 5. Die „Driver Class“ ist dann wieder die Klasse des JDBC-Treibers (in diesem Fall Teiid) und kann, sofern der JDBC-Treiber definiert wurde, über die Combobox ausgewählt, oder manuell eingegeben werden.
 6. Ein Klick auf „Test Connection“ gibt Aufschluss über die korrekte Konfiguration.
 7. Mit dem Klick auf „Save“ werden die Daten gespeichert.

8.4.2.2 Installation des Komplettpakets

Dieser Teil ist der nicht-optionale Teil, der in jedem Fall benötigt wird. Je nachdem, ob Teiid und Protégé aus dem Komplettpaket oder durch den vorher beschriebenen Prozess installiert wurden, können manche Befehle anders lauten. Das Start-Skript des Prototyps muss z.B. auf die neu installierten Versionen von Teiid und Protégé angepasst werden.

- 1) Extrahieren des Komplettpakets in einen separaten Ordner
- 2) Starten des Prototyps
 - a. Über einen Doppelklick auf „startePrototyp.bat“ wird der Vorgang angestoßen
 - b. Ein Konsolen-Fenster öffnet sich, und es soll gewartet werden, bis Teiid gestartet wurde
 - c. Teiid startet sich in einem weiteren Fenster
 - d. Nachdem Teiid gestartet wurde, öffnet sich der Prototyp
- 3) Verwenden der Nutzeroberfläche des Prototyps:
 - a. Auswahl der Ontologie
 - b. Auswahl der erstellten Zuordnungs-Datei
 - c. Eingabe/Laden einer Abfrage
 - d. Starten des Reasoners

e. Ausführen der Abfrage

8.4.3 Inhalt der gepackten Version der Lösung

Nach der Extraktion des Archivs der Lösung ist die folgende Struktur vorhanden:

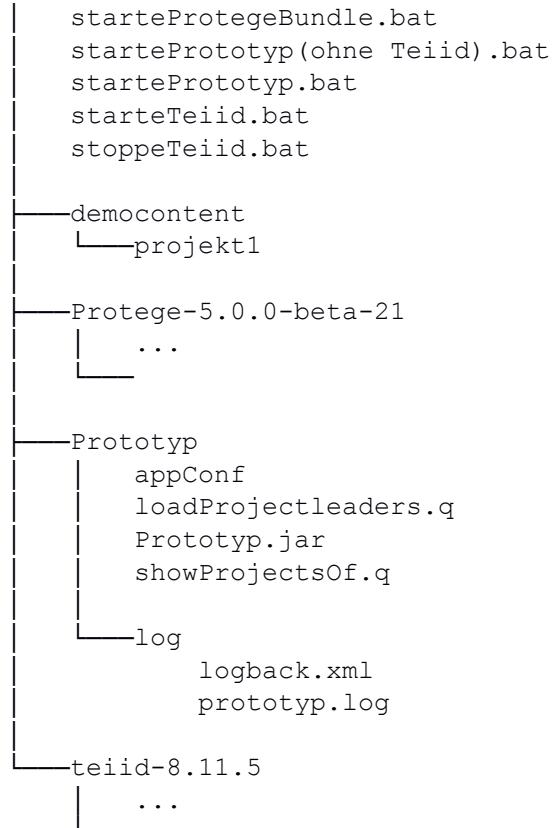


Abbildung 36 Ordner-Struktur der Lösung

Über die Start-Skripte können die einzelnen Komponenten gezielt gestartet oder gestoppt werden. Die Bezeichnungen der Skripte geben Aufschluss über deren Funktion. Falls Teiid separat gestartet wurde, muss Teiid über das stoppeTeiid-Skript beendet werden. Das kann der Fall sein, wenn Protégé und dann Teiid gestartet wurde, um Zuordnungen zu erstellen. Die Ordner „Protege-5.0.0-beta-21“ und „teiid-8.11.5“ beinhalten die Protégé- und Teiid-Installationen. Die Datenquellen liegen im Ordner „democontent“ in welchem die Testdaten abgelegt sind. Dort befindet sich auch der JDBC-Treiber welcher von Protégé verwendet wird, sowie ein weiterer Ordner „projekt1“, welcher projektspezifische Testdaten enthält. Im Ordner des Prototyps sind dessen Konfigurations-Datei „appConf“, der kompilierte Prototyp, sowie die vorgefertigten Abfragen, die für die Buttons benötigt werden, abgelegt. Logs werden in den dort enthaltenen log-Ordner gespeichert und das Log-Verhalten über die Datei „logback.xml“ gesteuert.

8.4.4 Überblick über die WildFly/Teiid Weboberfläche

Die Teiid Weboberfläche ist nach dem Starten von Teiid über „starteTeiid.bat“ standardmäßig unter der Adresse <http://localhost:9990/> erreichbar. Öffnet man diese URL im Browser, so erscheint ein Login-Fenster, in welches man die Login-Daten des Administrators eingeben muss (in dieser Arbeit:

„admin“ mit dem Passwort „b4chelor!“). Nach erfolgreichem Einloggen wird man zur Übersicht des Webservers weitergeleitet (siehe Abbildung 37).

The screenshot shows the WildFly + Teiid 8.11 administration interface. At the top, there is a navigation bar with links for Home, Deployments, Configuration, Runtime, and Administration. The Home link is currently selected. On the left, there is a sidebar titled "WildFly" with sections for "View and Manage Settings" (Deployments, Runtime), "Common Tasks" (Deploy an application, Create a datasource, Assign user roles), and a "Find More Resources" section with links to WildFly Home, Documentation, Admin Guide, Model Reference Documentation, Browse Issues, and Latest News. The main content area displays the "Find More Resources" sidebar.

Abbildung 37 Teiid Übersicht

Im Menü „Deployments“ (Abbildung 38) werden die im Teiid Server bereitgestellten Komponenten (z.B. Applikationen, VDBs, JDBC-Treiber) angezeigt. Von hier können diese auch de- oder aktiviert werden.

The screenshot shows the WildFly + Teiid 8.11 deployment management interface. The top navigation bar includes links for Home, Deployments (which is the active tab), Configuration, Runtime, and Administration. The Deployments section title is "Deployments" with the subtitle "Currently deployed application components." Below this, a table lists available deployments with columns for Name and Status (indicated by a checkmark icon). The table includes a "Filter:" input field and buttons for "Add", "Remove", "En/Disable", and "Replace". A specific deployment, "teiid-olingo-8.11.5-odata4.war", is selected and highlighted in blue. At the bottom, a "Deployment" tab is active, showing the deployment name "teiid-olingo-8.11.5-odata4.war".

Abbildung 38 Teiid "Deployments" Reiter

Der nächste Punkt heißt „Configuration“ (Abbildung 39). Hier können generelle Einstellungen am Server vorgenommen werden. Das sind z.B. Einstellungen an der Sicherheit, oder der Weboberfläche. Im Bereich „Connector“ können außerdem Datenquellen definiert bzw. eingesehen werden: Im Unterpunkt „Datasources“ ist eine Liste der JDBC-Datenquellen vorhanden, die mit Teiid eingebunden werden können. Unter „Resource Adapters“ sind die in Teiid integrierten Module aufgelistet. Hier findet sich z.B. der „fileQSEExcel“-Resource Adapter, der für den Zugriff auf Excel-Dateien verwendet wird.

The screenshot shows the WildFly + Teiid 8.11 configuration interface. The left sidebar has sections for Subsystems (Connector, JCA, Datasources, Resource Adapters), General Configuration (Interfaces, Socket Binding, Paths, System Properties), and a Runtime section. The main area is titled 'JCA Resource Adapters' and shows a table of available resource adapters:

Name	Connection Def.	Option
accumulo	0	View >
cassandra	0	View >
file	0	View >
fileQSEExcel	3	View >
google	0	View >

Buttons for 'Add' and 'Remove' are at the top right. Below the table are tabs for 'Attributes' and 'Properties', and a 'Need Help?' link.

Abbildung 39 Teiid "Configuration" Reiter mit Fokus auf "Resource Adapters"

Im Bereich „Runtime“ (siehe Abbildung 40) finden sich Informationen zum aktuellen Zustand von Teiid. Diese umfassen Statistiken zur Speichernutzung, aber auch eine Auflistung der bereitgestellten VDBs und deren Status.

The screenshot shows the WildFly + Teiid 8.11 runtime interface. The left sidebar has sections for Server (Overview, System Status), Platform (JVM, Environment, Log Viewer), Subsystems (Datasources, JPA, JNDI View), and a Virtual Databases section (Teiid, Transactions, Transaction Logs, Web, Webservices). The main area is titled 'VIRTUAL DATABASES' and shows 'Deployed Virtual Databases' with a table:

Name	Version	Dynamic	Status	Valid	Reload
ontology	1	true	ACTIVE	✓	Reload
sniK	1	true	ACTIVE	✓	Reload

Buttons for 'Refresh' and 'Summary', and tabs for 'Models', 'Overrides', 'Caching', 'Data Roles', 'Requests', and 'Sessions' are visible. Below the table are sections for 'Description:' and 'Errors'.

Abbildung 40 Teiid "Runtime" Reiter mit Fokus auf "Virtual Databases"

Um Details zu der VDB zu erhalten, muss die gewünschte VDB ausgewählt werden. Weiter unten im Fenster erscheinen dann die Details zu der Datenbank. Im Reiter „Models“ (Abbildung 41) werden die

erzeugten Modelle der Dateien angezeigt. Diese Übersicht ist sehr hilfreich, um fest zu stellen, wie die Excel-Dateien zugreifenden Applikationen zur Verfügung gestellt werden. In der Spalte „Name“ wird der Name, unter dem der Zugriff auf die Datei erfolgt, angezeigt. Dieser Name muss eindeutig sein. Der Zugriff auf eine Arbeitsmappe in der Datei „projLst“ würde dann über „projLst.Tabelle1“ (wobei „Tabelle1“ der Arbeitsmappen-Name ist) stattfinden.

The screenshot shows the Teiid interface with two main sections. The top section, titled 'Deployed VDBS', lists two entries: 'ontology' and 'snik'. Both are marked as 'ACTIVE' with a checkmark and have a 'Reload' button. The bottom section, titled 'Models', has tabs for 'Summary', 'Models' (which is selected), 'Overrides', 'Caching', 'Data Roles', 'Requests', and 'Sessions'. It lists five data sources: 'projLst', 'projBudLstInv', 'contrSach1', 'contrSach1S', and 'contrInv1'. Each entry includes columns for Name, Type (Physical), Visible?, Multi-Source?, Source Name, Translator Name, Datasource JNDI Name, Schema Status, and a small icon. The 'projLst' row is highlighted with a blue background. Navigation buttons at the bottom indicate '1-2 of 2' for the top section and '1-5 of 7' for the bottom section.

Name	Type	Visible?	Multi-Source?	Source Name	Translator Name	Datasource JNDI Name	Schema Status	
ontology	PHYSICAL	true	false	excelconnector	excel	java:/xls-snix	LOADED	
snik	PHYSICAL	true	false	excelconnector	excel	java:/xls-snix	LOADED	
projLst	PHYSICAL	true	false	xls-snix-budgetlisten	excel	java:/xls-snix-budgetlisten	LOADED	
projBudLstInv	PHYSICAL	true	false	xls-snix-budgetlisten	excel	java:/xls-snix-budgetlisten	LOADED	
contrSach1	PHYSICAL	true	false	xls-snix-budgetlisten	excel	java:/xls-snix-budgetlisten	LOADED	
contrSach1S	PHYSICAL	true	false	xls-snix-budgetlisten	excel	java:/xls-snix-budgetlisten	LOADED	
contrInv1	PHYSICAL	true	false	xls-snix-budgetlisten	excel	java:/xls-snix-budgetlisten	LOADED	

Abbildung 41 Übersicht der in der VDB "snik" bereitgestellten Datenquellen

Um das Modell einer Datei zu betrachten, muss auf den Button „DDL“ geklickt werden. Im Anschluss erscheint ein Detailfenster zur Schema-Definition der Datenquelle. Das in Abbildung 42 geöffnete Schema zeigt, dass aus der Arbeitsmappe der Datei eine Tabelle mit dem Namen „Tabelle1“ entstanden ist. Der Name der Tabelle wird aus dem Namen der Arbeitsmappe ausgewählt. Die darunter gelisteten Spalten sind die Spaltendefinitionen, die von Teiid durch die Eigenschaft `<property name="importer.headerRowNumber" value="3"/>` erzeugt wurden. Rechts neben deren Namen wird auch der gefundene Datentyp der Spalten-Daten angegeben. Durch die Verwendung von Teiid Designer oder einem Editor ist es möglich, dieses Modell manuell zu verändern und anzupassen. Das Ändern des Modells ist nicht Teil dieser Arbeit und wird daher nicht genauer beschrieben. Hilfreiche Information zu dem Thema kann aber auf der Teiid Webseite⁶⁵ gefunden werden.

⁶⁵ https://docs.jboss.org/author/display/TEIID/DDL+Metadata#_=_

```

Schema

SET NAMESPACE 'http://www.teiid.org/translator/excel/2014' AS teiid_excel;

CREATE FOREIGN TABLE Tabelle1 (
    ROW_ID integer OPTIONS (SEARCHABLE 'All_Except_Like', "teiid_excel:CELL_NUMBER" 'ROW_ID'),
    Auswahl string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '1'),
    Prio double OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '2'),
    Ausschreibung string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '3'),
    Jahr double OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '4'),
    ProjektNr string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '5'),
    Bezeichnung string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '6'),
    PL string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '7'),
    "geplanter Projektbeginn" string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '8'),
    "DL geplant" double OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '9'),
    "aDL geplant" string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '10'),
    "INV geplant" double OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '11'),
    "Speicher(INV)" string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '12'),
    "sonst verf Mittel" double OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMBER" '13'),
    "Beschreibung und Bemerkung" string OPTIONS (SEARCHABLE 'Unsearchable', "teiid_excel:CELL_NUMI
CONSTRAINT PK0 PRIMARY KEY(ROW_ID)
) OPTIONS (NAMEINSOURCE 'Tabelle1', "teiid_excel:FILE" 'projektwarteliste.xlsx', "teiid_excel:FIRST_DATA_

```

Close

Abbildung 42 Teiid Tabellen-Schema Details

Unter der Übersicht über die eingebundenen Datenquellen können Details darüber eingesehen werden, wie diese eingebunden sind. Dort wird unter anderem angezeigt, welche Datei eingebunden ist, und welche Zeile als die Zeile für die Definition der Spaltennamen verwendet wird (siehe Abbildung 43).

contrinv1	PHYSICAL	true	false	xls-snix-budgetlisten	excel	java:/xls-snix-budgetlisten	LOADED	DDL
« < 1-5 of 7 > »								
Path:								
Description:								
Properties								
Key	Value							
multisource	false							
importer.headerRowNumber	3							
importer.ExcelFileName	Controllingliste Invest fur ein Projekt.xlsx							
« < 1-3 of 3 > »								

Abbildung 43 Teiid Details zu einzelner Datenquelle

Im Reiter „Administration“ können Zugriffsrechte und Rollen verwaltet werden. Teiid ermöglicht eine sehr feine Einstellung der Nutzerrechte.

The screenshot shows the Teiid Administration interface. The top navigation bar includes Home, Deployments, Configuration, Runtime, and Administration. The Administration tab is selected. On the left, there's a sidebar with Access Control, Role Assignment (which is currently selected), Patching, and Patch Management. The main area is titled 'Users' and contains the sub-section 'Role Assignment'. It says 'A mapping of users to specific roles.' Below this is a table with two columns: 'User' and 'Roles'. A button 'Add' is at the top right of the table, and a message 'No Items!' is displayed below it. At the bottom left, there's a 'Selection' section with 'Edit' and 'User:' fields.

Abbildung 44 Teiid Reiter "Administration"

8.4.5 Erstellen der Zuordnungen über Protégé

Nachdem die Installation und Konfiguration (auch des JDBC-Treibers) abgeschlossen ist, kann Protégé dazu verwendet werden, um die Ontologie mit der Datenquelle (Teiid) zu verbinden. Zunächst sollte sichergestellt sein, dass eine Verbindung zu der Datenquelle vorhanden ist. Dazu muss im Reiter „Ontop Mappings“ und in dessen Reiter „Datasource manager“ auf den Button „Test Connection“ gedrückt werden. Erscheint ein grüner Schriftzug mit dem Text „Connection is OK“, wie in Abbildung 45 gezeigt, dann ist die Datenquelle richtig konfiguriert.

The screenshot shows the Protégé ontology editor interface. The title bar says 'untitled-ontology-20 (http://www.semanticweb.org/user/ontologies/2015/5/untitled-ontology-20) : [H:\data\Dropb...]' and the menu includes File, Edit, View, Reasoner, Tools, Refactor, Window, Ontop, Help. The tabs at the top are Active Ontology, Entities, Individuals by class, OWLViz, Ontop SPARQL, Ontop Mappings, OntoGraf, and SPARQL Query. The 'Ontop Mappings' tab is active. In the center, the 'Datasource manager' tab is selected. It shows a 'Datasource editor' with fields for Connection parameters: Datasource Name (teiid), Connection URL (jdbc:teiid:snik@mm://localhost:31000), Database Username (user), Database Password (redacted), and Driver class (org.teiid.jdbc.TeiidDriver). Below these fields is a 'Test Conn...' button and a status message 'Connection is OK'. Other tabs shown include Class hierarchy, Datasource manager, Mapping manager, and Mapping Assistant - BETA.

Abbildung 45 Protégé Datenquellen Test

Nun kann das eigentliche Zuordnen der Ontologie-Elemente zu den Daten stattfinden. Hierfür wird der „Mapping manager“ verwendet. Im Reiter „Mapping manager“ werden alle Zuordnungen aufgelistet. Ein Klick auf „New“ erstellt eine neue Zuordnung, die in einem neuen Fenster dargestellt

wird (Abbildung 46). Im Bereich „Target“ werden die der Ontologie zugehörigen Tripel eingetragen. In der Mitte liegt die Quelle „Source“, welche mit der hier eingetragenen SQL-Abfrage Daten ausliefert. Darunter liegt ein Bereich, der mit den aus der SQL-Abfrage erhaltenen Daten gefüllt wird, nachdem „Test SQL...“ gedrückt wurde. Ist die Zuordnung syntaktisch korrekt, ist der „Accept“-Button nicht ausgegraut und kann angeklickt werden, um die Zuordnung zu speichern.

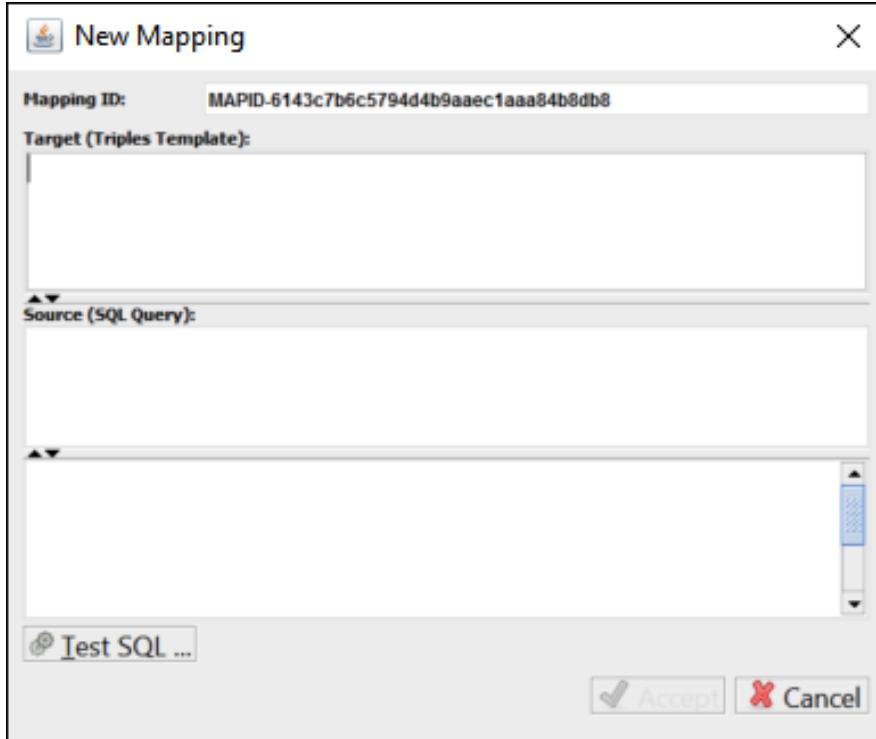


Abbildung 46 Fenster für eine neue Zuordnung

Das eigentliche Zuordnen der Daten zu der Ontologie lässt sich am besten an einem Beispiel erklären. Abbildung 47 zeigt eine fertige und funktionierende Zuordnung mit SQL-Daten. Die Angabe der Zuordnung im Ziel erfolgt über die von Ontop angebotene OBDA-Syntax⁶⁶. Die Syntax ist recht einfach zu lesen. Zum besseren Verständnis sei die folgende Ziel-Zuordnung gegeben:

```
:Projekt/{PN}/ a :Projekt ; :Projektnummer {PN} ; :Name
{Beschreibung} ; :Projektbeginn {Beginn} ; :Jahr
{Jahr} ; :isAssociatedWith :Controllingliste_Invest/{PN}/ , :Controllingliste_Budget/{PN}/ , :Projektpriorität/{Prio}/ , :Projektleiter/{PL}/ .
```

Jedes Tupel ist durch ein „;“ vom nächsten getrennt. Der Text innerhalb der geschweiften Klammern (z.B. „{PN}“) stellt die Punkte dar, die mit den Daten aus der SQL-Abfrage gefüllt werden. Die Bezeichnungen innerhalb der Klammern und in der SQL-Abfrage müssen hierfür identisch sein.

Das erste Tripel gibt zunächst eine IRI (in diesem Fall: :Projekt/{PN}/) für das zu erzeugende Element an. Zusammen mit dem darauf folgenden a und dem angegebenen Typ („:Projekt“) ergibt sich das folgende Tupel:

```
<http://example.org/Projekt/2007-08-001> rdf:type :Projekt.  
Die nachfolgenden Eigenschaften und Relationen sind in der gleichen Art und Weise angegeben.
```

⁶⁶ <https://github.com/ontop/ontop/wiki/ontopOBDAModel>

Mapping ID: MAPID-42485f535ea04ff8ac7489df1750b0eb

Target (Triples Template):

```
:Projekt/(PN)/ a :Projekt ; :Projektnummer {PN} ; :Name {Beschreibung} ; :Projektbeginn {Beginn} ; :Jahr {Jahr} ;
:isAssociatedWith :Controllingliste_Invest/{PN}/ , :Controllingliste_Budget/{PN}/ , :Projektpriorität/{Prio}/ ,
:Projektleiter/{PL}/ .
```

Source (SQL Query):

```
select "ProjektNr" as PN, "Beschreibung und Bemerkung" as Beschreibung, "PL", "geplanter Projektbeginn" as Beginn, "Prio", "Jahr" from "projLst"."Tabelle1"
```

PN	Beschreibung	PL	Beginn	Prio	Jahr
2007-08-001	Dies ist ein T...	Hans	Q1	1.0	2015.0
2007-08-002	PCs oeden n...	Peter	Q2,Q3	2.0	2016.0
2007-08-003	Update der ...	Peter	Q4	1.0	2014.0
2007-08-004	Neue Manag...	Hans	Q1	3.0	2017.0
2007-08-005	Neue Manag...	Hans	Q1	3.0	2018.0

Test SQL ... **Update** **Cancel**

Abbildung 47 Fenster mit Zuordnung und SQL-Daten (nach dem Klick auf "Test SQL...")

Das Ergebnis der Zuordnung mit den in Abbildung 47 angegebenen Daten lautet (für das erste Tupel):

```
<http://example.org/Projekt/2007-08-001> rdf:type :Projekt;
:Projektnummer „2007-08-001“;
:Name „Dies ist ein Test“;
:Projektbeginn „Q1“;
:Jahr 2015;
:IsAssociatedWith :Controllingliste_Invest/2007-08-001/;
:IsAssociatedWith :Controllingliste_Budget/2007-08-001/;
:IsAssociatedWith :Projektpriorität/1.0/
:IsAssociatedWith :Projektleiter/Hans/
```

Wichtig ist, dass bei der Referenz anderer Elemente wie z.B. eines Projektleiters, dem gleichen Schema folgt wie in den anderen Zuordnungen. In diesem Beispiel müssen Elemente des Typs Projektleiter also auch nach dem Schema :Projektleiter/{Projektleitername} / erzeugt werden. Ansonsten besteht kein korrekter Zusammenhang zwischen den Elementen vorhanden.

Bei der Erstellung der Zuordnungen ist besonders auf Folgendes zu achten:

- Syntax-Fehler werden nicht genauer beschrieben, was die Fehlersuche erschwert.
- Eine SQL-Abfrage kann nach dem Klick auf „Test SQL...“ die gewünschten Daten darstellen, der Reasoner selbst aber bei der Initialisierung Probleme mit der Abfrage haben.
 - „Test SQL...“ ist also nicht verlässlich.
 - Es ist darauf zu achten, dass die Spaltennamen (wie z.B. „ProjektNr“) immer in Anführungszeichen geschrieben werden, damit der Reasoner keine Änderungen an deren Schreibweise vornimmt. Außerdem muss die Angabe der Tabelle (also der eigentlichen Excel-Datei) wie folgt erfolgen:

- o „Dateibezeichnung“.“Arbeitsmappen-Name“
 - Wichtig ist, dass beide Begriffe separat von einander in Anführungszeichen geschrieben werden.

Sind die Zuordnungen erstellt, kann über das Menü durch „Reasoner“ → „Start Reasoner“ der Reasoner gestartet werden. Fehler bei den Zuordnungen werden spätestens hier dem/der NutzerIn mitgeteilt.

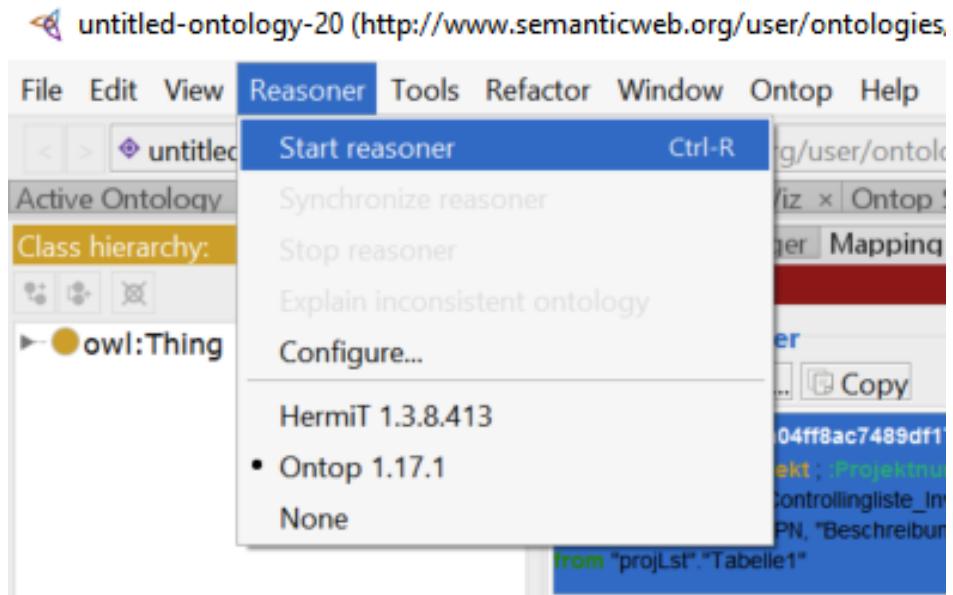


Abbildung 48 Start des Reasoners über Menü

Hinweis: Nach dem Ändern von Zuordnungen, muss der Reasoner neu gestartet werden. Dazu den Reasoner zunächst stoppen („Reasoner“ → „Stop Reasoner“) und dann wieder starten. Die Funktion „Synchronize Reasoner“ ist leider nicht zuverlässig.

8.4.6 Hinzufügen/Ändern von Datenquellen in Teiid

Jede neue Datei muss manuell hinzugefügt werden. Das Hinzufügen besteht aus zwei Schritten:

1. Zunächst muss ein Resource Adapter hinzugefügt werden, der den Zugriff auf die Excel-Dateien regelt.
2. Dann wird eine VDB-Konfiguration erstellt, die den Resource Adapter verwendet und in der beschrieben wird, wie die Dateien exponiert werden sollen.

Zu 1.:

Der Resource Adapter, der für Excel verwendet wird, benötigt als Parameter den Ordner der hinzuzufügenden Dateien. Das bedeutet, dass für jeden Ordner ein neuer Resource Adapter erzeugt werden muss. Die Definition des Resource Adapters erfolgt innerhalb der Datei „standalone-teiid.xml“, also der Serverkonfiguration. Die Datei liegt unter TEIID_ORDNER/standalone/configuration. Als erstes wird die Datei mit einem Texteditor geöffnet. Dann muss zum Eintrag `<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">` gesprungen werden. Im Resource Adapters Subsystem der Konfiguration werden die Resource Adapter verwaltet. Als nächstes wird der Eintrag `<resource-adapter id="fileQSEExcel">`

gesucht. Innerhalb der Resource-Adapter-Tags der Datei können nun neue Resource Adapter vom Typ „fileQSExcel“ hinzugefügt werden:

```
<resource-adapter id="fileQSExcel">
    <module slot="main" id="org.jboss.teiid.resource-adapter.file"/>
    <connection-definitions>
        <connection-definition class-
name="org.teiid.resource.adapter.file.FileManagedConnectionFactory" jndi-
name="java:/xls-snake" enabled="true" use-java-context="true" pool-
name="fileDS2">
            <config-property name="ParentDirectory">
                D:/bachelor/tools/democontent/
            </config-property>
            <config-property name="AllowParentPaths">
                true
            </config-property>
        </connection-definition>
    </connection-definitions>
</resource-adapter>
```

Dazu muss ein neuer „connection-definition“-Block innerhalb der „resource-adapters“-Tags eingefügt werden. Dieser kann wie folgt aussehen:

```
<connection-definition class-
name="org.teiid.resource.adapter.file.FileManagedConnectionFactory" jndi-
name="java:/xls-snake-budgetlisten" enabled="true" use-java-context="true"
pool-name="fileDS3">
    <config-property name="ParentDirectory">
        D:/bachelor/tools/democontent/projekt1/
    </config-property>
    <config-property name="AllowParentPaths">
        true
    </config-property>
</connection-definition>
```

Ein paar Anmerkungen dazu:

- Der „jndi-name“ und „pool-name“ müssen eindeutig sein.
- Der „jndi-name“ wird später wieder beim Erstellen der VDB wiederbenötigt.
- Innerhalb der „config-property“-Tags kann unter „ParentDirectory“ ein Pfad zum Ordner, der die Excel-Dateien enthält, angegeben werden.
- Immer auf vollständige Tags achten! Jeder Start-Tag (<tag>) benötigt auch einen Schließ-Tag (</tag> - erkennbar am /).

Zu 2.:

Sind die notwendigen Resource Adapter erstellt, können diese in der VDB-Definition verwendet werden. Man muss diese über eine Konfigurations-Datei einbinden. Die im Verlauf dieser Arbeit eingesetzte Konfiguration für die virtuelle Datenbank mit dem Namen „snik“ lautet wie folgt:

```
<vdb name="snik" version="1">
    <model visible="true" name="projLst">
        <property name="importer.headerRowNumber" value="4"/>
```

```

        <property name="importer.ExcelFileName"
value="projektwarteliste.xlsx"/>
        <source name="excelconnector" translator-name="excel" connection-
jndi-name="java:/xls-snix"/>
    </model>
    <model visible="true" name="projBudLstInv">
        <property name="importer.headerRowNumber" value="2"/>
        <property name="importer.ExcelFileName"
value="projektbudgetliste_invest.xlsx"/>
        <source name="excelconnector" translator-name="excel" connection-
jndi-name="java:/xls-snix"/>
    </model>
    <model visible="true" name="contrSach1">
        <property name="multisource" value="false"/>
        <property name="importer.headerRowNumber" value="3"/>
        <property name="importer.ExcelFileName" value="Controllingliste
Sachkonto Dienstleistungen fur ein Projekt.xlsx"/>
        <source name="xls-snix-budgetlisten" translator-name="excel"
connection-jndi-name="java:/xls-snix-budgetlisten"/>
    </model>
    <model visible="true" name="contrSach1S">
        <property name="multisource" value="false"/>
        <property name="importer.headerRowNumber" value="1"/>
        <property name="importer.dataRowNumber" value="1"/>
        <property name="importer.ExcelFileName" value="Controllingliste
Sachkonto Dienstleistungen fur ein Projekt.xlsx"/>
        <source name="xls-snix-budgetlisten" translator-name="excel"
connection-jndi-name="java:/xls-snix-budgetlisten"/>
    </model>
    <model visible="true" name="contrInv1">
        <property name="multisource" value="false"/>
        <property name="importer.headerRowNumber" value="3"/>
        <property name="importer.ExcelFileName" value="Controllingliste
Invest fur ein Projekt.xlsx"/>
        <source name="xls-snix-budgetlisten" translator-name="excel"
connection-jndi-name="java:/xls-snix-budgetlisten"/>
    </model>
    <model visible="true" name="contrInv1S">
        <property name="multisource" value="false"/>
        <property name="importer.headerRowNumber" value="1"/>
        <property name="importer.dataRowNumber" value="1"/>
        <property name="importer.ExcelFileName" value="Controllingliste
Invest fur ein Projekt.xlsx"/>
        <source name="xls-snix-budgetlisten" translator-name="excel"
connection-jndi-name="java:/xls-snix-budgetlisten"/>
    </model>
    <model visible="true" name="projBudLst">
        <property name="multisource" value="false"/>
        <property name="importer.headerRowNumber" value="2"/>
        <property name="importer.ExcelFileName" value="Projektbudgetliste
Sachkonto Dienstleistungen.xlsx"/>
        <source name="xls-snix-budgetlisten" translator-name="excel"
connection-jndi-name="java:/xls-snix-budgetlisten"/>
    </model>
</vdb>
```

Die gleichen Regeln des Hinzufügens von neuen „model“-Tags, die eine Datei repräsentieren, gelten auch hier:

- Immer auf vollständige Tags achten! Jeder Start-Tag (`<tag>`) benötigt auch einen Schließ-Tag (`</tag>` - erkennbar am `/`).

Zum besseren Verständnis werden an dieser Stelle noch einmal die Details der Konfiguration zusammengefasst:

Tabelle 20 Bedeutung der VDB-Konfiguration

Zeile	Bedeutung
<code><vdb name="snik" version="1"></code>	Parameter „name“ gibt den Namen an, unter dem die VDB von Teiid exponiert wird. Dieser Name wird bei der JDBC-Verbindung in Ontop benötigt.
<code><model visible="true" name="projLst"></code>	Parameter „visible“ kann dazu verwendet werden, um die Tabelle nicht verfügbar zu machen. Der Parameter „name“ muss eine eindeutige Bezeichnung sein. Dieser Name wird später in Ontop als Tabellenname, zusammen mit dem Arbeitsmappen-Name, verwendet.
<code><property name="importer.headerRowNumber" value="4" /></code>	Die Spaltennamen der virtuellen Tabelle ergeben sich aus den durch den mittels <code>importer.headerRowNumber</code> angegebenen Parameter und dessen „value“ (in diesem Fall 4). Details dazu siehe im folgenden Text.
<code><property name="importer.ExcelFileName" value="projektwarteliste.xlsx"/></code>	Der Parameter „value“ gibt den Dateinamen an.
<code><source name="excelconnector" translator-name="excel" connection-jndi-name="java:/xls-snik"/></code>	Der <code>jndi-name</code> ist eine Referenz auf einen zuvor hinzugefügten Resource Adapter. Dadurch wird im Pfad, der im Resource Adapter angegeben ist, nach der angegebenen Datei gesucht. Die beiden anderen Parameter werden für Excel-Dateien benötigt.
<code></model></code> <code></vdb></code>	Schließ-Tags des Modells (der Datei) und der gesamten VDB

Die Spaltennamen der virtuellen Tabelle ergeben sich aus den durch den `importer.headerRowNumber` angegebenen Parameter und dessen „value“ (in diesem Fall 3). Dadurch wird definiert, dass die in Zeile 3 gefundenen Elemente als die Spaltennamen zu verstehen sind. Abbildung 31 zeigt eine Datei mit Test-Daten. Gelb eingefärbt sind die Spaltennamen, die sich durch den Parameter `name="importer.headerRowNumber" value="3"` ergeben. Die eigentlichen Daten, die am Ende in den Zeilen der Tabelle zu finden sind, blau eingefärbt.

Standardmäßig werden die Zeilen unter der angegebenen Zeile (in diesem Fall Zeile 3) als Daten verwendet.

Beim genauen Betrachten der Konfiguration fallen zwei Punkte auf:

1. Jede Datenquelle (Excel-Datei) muss einzeln eingebunden werden.

2. Manche Datenquellen sind doppelt und mit unterschiedlichen Bezeichnungen eingebunden
(z.B. contrInv1 und contrInv1S)

Zum ersten Punkt:

Es ist in der aktuellen Implementierung von Teiid nicht vorgesehen, automatisch alle gefundenen Datenquellen einzubinden. Die Funktion ist zwar in Arbeit, allerdings noch nicht umgesetzt. Das macht es notwendig, jede Datenquelle einzeln einzubinden.

Zum zweiten Punkt:

Das doppelte Einbinden mancher Dateien ist notwendig, um die singulären Datenfelder in den Excel-Dateien, einzubinden. Anhand von Abbildung 31 lässt sich auch das erkennen. In Zeile 2 sind Daten enthalten, die nur für sich relevant sind. Um diese in Ontop verwenden zu können, muss die Datei in der virtuellen Datenbank-Konfiguration ein weiteres Mal mit den folgenden Parametern eingebunden werden:

```
<property name="importer.headerRowNumber" value="1"/>
<property name="importer.dataRowNumber" value="1"/>
```

Dadurch wird die erste Zeile als Spaltennamen und als Datenzeile verwendet. Abbildung 32 zeigt das anschaulich:

Durch diese Art der Einbindung als Tabelle, ist es später über SQL-Abfragen möglich, genau eine Zeile mit singulären Feldern (in diesem Fall Zeile 2) zu erhalten. Das kann durch eine einfache Einschränkung auf die zweite Zeile in der SQL-Abfrage zum Beziehen der Daten erfolgen.

Für Details kann die Dokumentation⁶⁷ verwendet werden.

⁶⁷ https://teiid.gitbooks.io/documents/content/reference/Microsoft_Excel_Translator.html

Tabelle 21 Testergebnisse

ID	Test-Typ	Name	Erwartetes Ergebnis	Ergebnis des Tests	Im Zusammenhang mit Anforderung
1	Komponenten	MappingTestBinaryFile	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
2	Komponenten	MappingTestEmptyFile	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
3	Komponenten	MappingTestInvalidFile	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
4	Komponenten	MappingTestInvalidOBDA	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
122 von 133	5	MappingTestInvalidOBDADefinition	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	6	MappingTestInvalidOBDADriverClass	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	7	MappingTestInvalidOBDADriverPassword	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	8	MappingTestInvalidOBDADriverURI	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	9	MappingTestInvalidOBDAMapping	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	10	MappingTestInvalidOBDA_SQL	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	11	MappingTestInvalidPath	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten

					(Datei, Daten selbst)
12	Komponenten	MappingTestValidOBDA	Keine Probleme	Keine Probleme	
13	Komponenten	OntologyTestBinaryFile	Exception	NoSuchMethodError	Fehlertoleranz gegenüber fehlerhaften Ontologien
14	Komponenten	OntologyTestEmptyFile	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber fehlerhaften Ontologien
15	Komponenten	OntologyTestInvalidFile	Exception	NoClassDefFoundError	Fehlertoleranz gegenüber fehlerhaften Ontologien
16	Komponenten	OntologyTestInvalidOWL	Exception	NoClassDefFoundError	Fehlertoleranz gegenüber fehlerhaften Ontologien
17	Komponenten	OntologyTestInvalidPath	Exception	OWLOntologyInputSourceException	Fehlertoleranz gegenüber fehlerhaften Ontologien
18	Komponenten	OntologyTestLargeOWL	Keine Probleme	Keine Probleme	
19	Komponenten	OntologyTestValidOWL	Keine Probleme	Keine Probleme	
20	Komponenten	QueryTestBinaryFile	Exception	UncheckedIOException	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
21	Komponenten	QueryTestEmptyFile	Exception	RuntimeException	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
22	Komponenten	QueryTestInvalidFile	Exception	RuntimeException	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.

23	Komponenten	QueryTestInvalidPath	Exception	AccessDeniedException	<p>Beantwortung von Fragestellungen mittels SPARQL-Abfragen.</p> <p>SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.</p>
24	Komponenten	QueryTestInvalidQ	Exception	OntopOWLException	<p>Beantwortung von Fragestellungen mittels SPARQL-Abfragen.</p> <p>SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.</p>
25	Komponenten	QueryTestValidQ	Keine Probleme	Keine Probleme	<p>Beantwortung von Fragestellungen mittels SPARQL-Abfragen.</p> <p>SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.</p> <p>Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)</p>
26	Komponenten	ReasonerTestMappingBinaryFile	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
27	Komponenten	ReasonerTestMappingEmptyFile	Exception	ReasonerInternalException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
28	Komponenten	ReasonerTestMappingInvalidFile	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
29	Komponenten	ReasonerTestMappingInvalidOBDA	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten

124 von 133

					(Datei, Daten selbst)	
30	Komponenten	ReasonerTestMappingInvalidOBDADefinition	Exception	ReasonerInternalException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)	
31	Komponenten	ReasonerTestMappingInvalidOBDADriverClass	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)	
32	Komponenten	ReasonerTestMappingInvalidOBDA_driverPassword	Exception	ReasonerInternalException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)	
33	Komponenten	ReasonerTestMappingInvalidOBDA_driverURI	Exception	ReasonerInternalException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)	
125 von 133	34	Komponenten	ReasonerTestMappingInvalidOBDA_mapping	Exception	IOException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	35	Komponenten	ReasonerTestMappingInvalidOBDA_SQL	Exception	ReasonerInternalException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	36	Komponenten	ReasonerTestMappingInvalidPath	Exception	ReasonerInternalException	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	37	Komponenten	ReasonerTestMappingValidOBDA	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	38	Komponenten	ReasonerTestOntologyBinaryFile	Exception	NoClassDefFoundError	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	39	Komponenten	ReasonerTestOntologyEmptyFile	Keine Probleme	Keine Probleme	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
	40	Komponenten	ReasonerTestOntologyInvalidFile	Exception	NoClassDefFoundError	Fehlertoleranz gegenüber fehlerhaften Ontologien

41	Komponenten	ReasonerTestOntologyInvalidOWL	Exception	NoClassDefFoundError	Fehlertoleranz gegenüber fehlerhaften Ontologien
42	Komponenten	ReasonerTestOntologyInvalidPath	Exception	OWLOntologyInputSourceException	Fehlertoleranz gegenüber fehlerhaften Ontologien
43	Komponenten	ReasonerTestOntologyLargeOWL	Keine Probleme	Keine Probleme	
44	Komponenten	ReasonerTestOntologyValidOWL	Keine Probleme	Keine Probleme	
45	System	SPARQLQueriesTestQ1	Projektnummern	Projektnummern	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
46	System	SPARQLQueriesTestQ2	Projektleiterliste	Projektleiterliste	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
47	System	SPARQLQueriesTestQ3	Budget für angegebenes Projekt	Erfolgreich	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
48	System	SPARQLQueriesTestQ4	Liste Dienstleistungen	Erfolgreich	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.

126 von 133

49	System	SPARQLQueriesTestQ5	Positionen Dienstleistungen für gegebenes Projekt.	Erfolgreich	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
50	System	SPARQLQueriesTestQ6	Positionen Investitionen für gegebenes Projekt.	Erfolgreich	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
51	System	SPARQLQueriesTestQ7	Projektliste	Projektliste	Beantwortung von Fragestellungen mittels SPARQL-Abfragen. SPARQL-Abfragen aus dem Kontext Project Management und Project Controlling sollen beantwortet werden können.
127 von 133	Statische Analyse	InspectCode	Report mit allen gefundenen Problemen	Ein paar unnötige Imports konnten entfernt werden. Allerdings viele false-positives: Rechtschreibung prüft nur Englisch, daher ist Deutsch ein Problem; Vorschläge für statische Werte bei manchen Methoden nicht sinnvoll für Modularität und weitere Verwendung des Prototyps	
53	GUI	Reasoner Start Funktionstest	Siehe Beschreibung	Erfolgreich	
54	GUI	Reasoner Start Unterricht	Siehe Beschreibung	Erfolgreich	

55	GUI	"Hole Projektleiter"-Button Funktionstest	Siehe Beschreibung	Erfolgreich	Darstellung der Aggregierten- und Rohdaten (Textuell)
56	GUI	...-Buttons Funktionstest	Siehe Beschreibung	Erfolgreich	
57	GUI	Fortschrittsbalken Funktionstest	Siehe Beschreibung	Erfolgreich	
58	GUI	Auswahl zwischen Datei-Abfrage und Text-Abfrage Funktionstest	Siehe Beschreibung	Erfolgreich	
59	GUI	Ausgabefeld zeigt Abfrage Ergebnis	Siehe Beschreibung	Erfolgreich	Darstellung der Aggregierten- und Rohdaten (Textuell)
60	GUI	Infofeld zeigt Informationen zu Ereignissen	Siehe Beschreibung	Erfolgreich	
128 von 133	61	Debug-Logs werden erstellt	Siehe Beschreibung	Erfolgreich. Durch das Erstellen einer logback.xml-Datei kann das Logging-Verhalten beeinflusst werden. Es wird log4J verwendet.	Kontinuierliches Logging von Abläufen und Fehlern für bessere Wartbarkeit
	62	Alle Exceptions werden dem Nutzer Angezeigt	Siehe Beschreibung	Erfolgreich	Warnung des Nutzers, bzgl. Auftauchender Fehler (z.B. unbehandelte Ausnahmen)
63	System	Teiid Nicht Excel-Dateien	Teiid soll bei Fehlern nicht abstürzen. Sondern Fehler anzeigen	Fehlermeldung zum Beheben	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
64	System	Teiid Sonderzeichen in Excel-Dateien	Probleme mit manchen Zeichen	Sonderzeichen wie ",,:? Sind in Excel-Dateien zu vermeiden	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
65	System	Teiid Sonderzeichen in Pfaden	Probleme mit manchen Zeichen	Umlaute und Systemsonderzeichen (?;,: usw.) sind zu vermeiden	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
66	System	Teiid Nicht Excel-Dateien	Teiid soll bei Fehlern nicht abstürzen. Sondern	Erfolgreich	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)

			Fehler anzeigen		
67	System	Teiid relative Pfade	Teiid akzeptiert relative Pfade	Erfolgreich	Fehlertoleranz gegenüber nicht vorhandenen/fehlerhaften Daten (Datei, Daten selbst)
68	System	Verändern (entfernen, hinzufügen) von Datenquellen	Keine Probleme	Keine Probleme	Hinzufügen/Löschen/Bearbeiten von Datenquellen
69	System	Teiid verarbeitet Daten aus verschiedenen Quellen	Keine Probleme	Word-Dateien werden nicht unterstützt. Dafür aber SQL und Excel.	Aggregation von Daten aus verschiedenen Quellen und Quelltypen Zugriff/Aggregation von Word-, Excel- und Webseiten-Daten (lesend)

9 Literaturverzeichnis

- [1] Guarino, N., Oberle, D., Staab, S.: What Is an Ontology? Handbook on Ontologies. In: Handbook on Ontologies SE - International Handbooks on Information Systems. pp. 1–17 (2009).
- [2] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 4900 LNCS, 133–173 (2008).
- [3] Ian Sommerville: Software Engineering (Ninth Edition). In: Software Engineering (Ninth Edition). pp. 147 – 175 (2010).
- [4] West, M.: Developing High Quality Data Models. Eur. Process Ind. STEP Tech. Liaison Exec. 62 (1999).
- [5] Guarino, N., Giaretta, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. Towar. Very Large Knowl. Bases Knowl. Build. Knowl. Shar. 1, 25–32 (1995).
- [6] Gagnon, M.: Ontology-based integration of data sources. 2007 10th Int. Conf. Inf. Fusion. 1–8 (2007).
- [7] Rodríguez-Muro, M., Kontchakov, R., Zakharyashev, M.: Ontology-based data access: Ontop of databases. Iswc. 8218 LNCS, 558–573 (2013).
- [8] Winter, A., Haux, R., Ammenwerth, E., Brigl, B., Hellrung, N., Jahn, F.: Health Information Systems. Springer London, London (2011).
- [9] Haas, L.: Beauty and the Beast : The Theory and Practice of Information Integration. Database Theory - ICDT 2007 11th Int. Conf. Barcelona, Spain, January 10-12, 2007. Proc. 28 – 43 (2007).
- [10] Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. 22, 183–236 (1990).
- [11] Schaaf, M., Jahn, F., Tahar, K., Kücherer, C., Winter, A., Paech, B.: Entwicklung und Einsatz einer Domänenontologie des Informationsmanagements im Krankenhaus. Inform. 2015 Inform. Energ. und Umwelt, 45. Jahrestagung der Gesellschaft für Inform. LNI, 753–765 (2015).
- [12] Arbeitsgruppe Software Engineering der Universität Heidelberg: Richtlinien zur Literaturrecherche. Internes Dokument, 22.10.2015.
- [13] Calvanese, D., Cogrel, B., Komla-ebri, S., Kontchakov, R., Lanti, D.: Ontop : Answering SPARQL Queries over Relational Databases. Semant. Web J. 0, (2015).
- [14] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semant. Web. 2, 43–53 (2011).
- [15] Giacomo, G. De, Lembo, D., Lenzerini, M.: Mastro: A reasoner for effective ontology-based data access. Proc. ORE. (2012).
- [16] Park, J., Lee, E.K., Wang, Q., Li, J., Lin, Q., Pu, C.: Health-connect: An ontology-based model-driven information integration framework and its application to integrating clinical databases. Proc. 2012 IEEE 13th Int. Conf. Inf. Reuse Integr. IRI 2012. 393–400 (2012).
- [17] Rodriguez-Muro, M., Calvanese, D.: Quest, an OWL 2 QL reasoner for ontology-based data

access. Owled 2012. (2012).

- [18] Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information-a survey of existing approaches. *IJCAI Work. Ontol. Inf. Shar.* 108–117 (2001).
- [19] Erling, O., Mikhailov, I.: RDF support in the virtuoso DBMS. *Stud. Comput. Intell.* 221, 7–24 (2009).
- [20] IEEE: IEEE Recommended Practice for Software Requirements Specification. (1998).

10 Abbildungs- und Tabellenverzeichnis

Abbildung 1 Beispiel Vereinfachtes Datenmodell.....	15
Abbildung 2 Datenmodellierungsprozess. Quelle: [4].....	16
Abbildung 3 Ausschnitt "Projektmanagement" der CIOx Ontologie des SNIK-Projekts (OntoGraf Export)	18
Abbildung 4 Ausschnitt Projektmanagement der CIOx-Ontologie mit Fokus auf "Projektpriorität" (OntoGraf Export).....	19
Abbildung 5 Beispiel Zuordnung zwischen Ontologie-Element "Projekt" und dessen Eigenschaften mit der Datenquelle "Projekt(warte)liste	21
Abbildung 6 Zuordnung zwischen CIOx-Ontologie und SQL-Datenquelle in OBDA-Syntax	22
Abbildung 7 Schema Beispiel-Ontologie	23
Abbildung 8 Schematischer Aufbau OBDA.....	25
Abbildung 9 Schema Query Rewriting.....	27
Abbildung 10 Schematische Darstellung eines Multibank Systems.....	31
Abbildung 11 Schematisches Beispiel eines Data Warehouse.....	33
Abbildung 12 Schema des Metamodells der CIOx-Ontologie. Quelle: SNIK Projekt, IMISE Leipzig.....	34
Abbildung 13 Projektleiter Details (Screenshot Protégé)	35
Abbildung 14 Projekt Details (Screenshot Protégé)	35
Abbildung 15 Personell_Planning Details (Screenshot Protégé).....	36
Abbildung 16 Ausschnitt Rollen der Ontologie (OWLWiz export).....	36
Abbildung 17 Ansätze der Integration von Ontologien. Quelle: [6].....	48
Abbildung 18 Schematische Darstellung des modellgetriebenen Ansatzes zur Datenintegration mittels Ontologien. Quelle: [16]	50
Abbildung 19 OBDA über Reasoner mit Query Rewriting	51
Abbildung 20 Schema Virtuoso Funktionen und Einsatzszenario. Quelle: http://virtuoso.openlinksw.com/	56
Abbildung 21 Aufbau der Schichten des Ontop Frameworks. Quelle: [13]	60
Abbildung 22 Schematische Darstellung einer auf Teiid basierten virtuellen Datenbank (VDB) Quelle: http://teiid.jboss.org/basics/virtualdatabases/	61
Abbildung 23 Sequenzdiagramm der Nutzung der manuellen Lösung.....	78
Abbildung 24 Diagramm-Übersicht der SNIK-Ontologie Ausschnitt „Projektmanagement“	79
Abbildung 25 Ausschnitt Darstellung Ontologie und Datenquellen Zuordnung.....	80
Abbildung 26 Komponentendiagramm der Umsetzung	82
Abbildung 27 SPARQL-Abfrage "showProjectsOf.q"	84
Abbildung 28 Nutzeroberfläche des Prototyps nach dem ersten Starten	85
Abbildung 29 Nutzeroberfläche des Prototyps nach Nutzung	87
Abbildung 30 Ausschnitt der Konfiguration für SNIK-VDB	88
Abbildung 31 Excel-Datenquelle Controlling-Liste Sachkonto mit Test-Daten.....	88
Abbildung 32 Excel-Datenquelle Controlling-Liste Sachkonto mit Test-Daten (zweite Konfiguration)	89
Abbildung 33 Screenshot Protégé Reiter Ontop Mappings → Datasource Manager.....	90
Abbildung 34 Screenshot Protégé Reiter Ontop Mappings → Mapping Manager.....	91
Abbildung 35 Screenshot Protégé Reiter Ontop SPARQL	92
Abbildung 36 Ordner-Struktur der Lösung.....	108
Abbildung 37 Teiid Übersicht	109
Abbildung 38 Teiid "Deployments" Reiter	110
Abbildung 39 Teiid "Configuration" Reiter mit Fokus auf "Resource Adapters"	111

Abbildung 40 Teiid "Runtime" Reiter mit Fokus auf "Virtual Databases"	111
Abbildung 41 Übersicht der in der VDB "snik" bereitgestellten Datenquellen.....	112
Abbildung 42 Teiid Tabellen-Schema Details	113
Abbildung 43 Teiid Details zu einzelner Datenquelle.....	113
Abbildung 44 Teiid Reiter "Administration"	114
Abbildung 45 Protégé Datenquellen Test	114
Abbildung 46 Fenster für eine neue Zuordnung	115
Abbildung 47 Fenster mit Zuordnung und SQL-Daten (nach dem Klick auf "Test SQL...")	116
Abbildung 48 Start des Reasoners über Menü.....	117
 Tabelle 1 Tabellen-Beispiel.....	17
Tabelle 2 Beispiele für valide Tripel:	20
Tabelle 3 CIOx-Ontologie Metriken.....	34
Tabelle 4 Suchergebnisse Literaturrecherche	38
Tabelle 5 Übersicht gefundener Publikationen.....	40
Tabelle 6 Bewertungskriterien für die Bewertung der Frameworks.....	52
Tabelle 7 Vom Prototyp zu beantwortende Fragestellungen und deren Formulierung als SPARQL Request (falls möglich)	66
Tabelle 8 Repository Vergleich der Frameworks.....	68
Tabelle 9 Vergleichstabelle verschiedener Frameworks (Teil 1).....	71
Tabelle 10 Vergleichstabelle verschiedener Frameworks (Teil 2).....	72
Tabelle 11 Übersicht über die Bewertungen der Frameworks	73
Tabelle 12 Übersicht der Ontologie-Elemente.....	74
Tabelle 13 Erklärung der Zuordnungs-Syntax	75
Tabelle 14 Zuordnungen für Controllingliste invest	76
Tabelle 15 Zuordnungen für Controllingliste Budget	76
Tabelle 16 Zuordnungen für Projektwarteliste	77
Tabelle 17 Zuordnungen für Projektbudgetliste Sachkonto	77
Tabelle 18 Anforderung und deren Status (mit Begründung ob erfüllt/nicht erfüllt)	94
Tabelle 19 Status der geforderten SPARQL-Abfragen.....	95
Tabelle 20 Bedeutung der VDB-Konfiguration	120
Tabelle 21 Testergebnisse	122