

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl für Medizinische Informatik



Diplomarbeit im Studiengang Informatik

Ein Semantic-Web-Ansatz zum Mapping  
klinischer Metadaten am Beispiel eines  
Bioproben-/Projektvermittlungs-Portals für das  
DPKK auf der Basis von i2b2

Sebastian Mate

Betreut durch Prof. Dr. Hans-Ulrich Prokosch,  
Dr. Thomas Ganslandt, PD Dr. Thomas Bürkle



---

## Erklärung der Selbstständigkeit

---

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 14. Januar 2011

Sebastian Mate



---

## Inhaltsverzeichnis

---

<b>1 Einleitung</b>	<b>1</b>
1.1 Problemvorstellung: eine neue Forschungsdatenbank für das Deutsche Prostatakarzinom Konsortium e.V. . . . .	1
1.2 Perspektiven durch Open-Source, Single-Source und das Semantic Web . .	2
1.2.1 i2b2 als generische Forschungsdatenbank . . . . .	2
1.2.2 Single-Source zur Vermeidung von Mehrfachdokumentation . . . .	3
1.2.3 Das Semantic Web zur maschinenlesbaren Abbildung von medizinischem und technischem Wissen . . . . .	3
1.3 Ziele und Fragestellungen dieser Arbeit . . . . .	4
<b>2 Grundlagen</b>	<b>7</b>
2.1 Informatics for Integrating Biology and the Bedside (i2b2) . . . . .	7
2.2 Die klinischen Arbeitsplatzsysteme in Erlangen und Münster . . . . .	9
2.2.1 Siemens Soarian Clinicals und der Data-Warehouse-Export am UK-Erlangen . . . . .	9
2.2.2 Agfa HealthCare ORBIS am UK-Münster . . . . .	13
2.3 Das Semantic Web . . . . .	14
2.3.1 Die Vision des Semantic Web . . . . .	14
2.3.2 Die Extensible Markup Language (XML) . . . . .	15
2.3.3 Das Resource Description Framework (RDF) . . . . .	16
2.3.4 RDF Schema (RDFS) . . . . .	21
2.3.5 Die Web Ontology Language (OWL) . . . . .	23
2.3.6 Die Open-World-Assumption . . . . .	25
2.3.7 Die SPARQL Protocol And RDF Query Language . . . . .	25
2.3.8 Die Semantic Web Rule Language (SWRL) und OWL 2 . . . . .	26
2.4 Werkzeuge . . . . .	27
<b>3 Methoden</b>	<b>29</b>
3.1 Darstellung des Konzepts dieser Arbeit . . . . .	29
3.2 Beschreibung von Zieldatensatz-Ontologien . . . . .	32
3.3 Beschreibung von Quellsystem-Ontologien . . . . .	34

3.3.1	Granularität der Quellsystem-Ontologie . . . . .	34
3.3.2	Strukturierung der Elemente innerhalb der Quellsystem-Ontologie . . . . .	36
3.3.3	Zugriff auf die Nutzdaten über die Ontologie . . . . .	41
3.4	Beschreibung von Mappings zwischen Quellsystem- und Zieldatensatz-Ontologie . . . . .	44
3.4.1	Modellierung von direkten Mappings ohne Transformationen . . . . .	44
3.4.2	Modellierung von indirekten Mappings mit Transformationen . . . . .	45
3.4.3	Einbindung von Datenwerten . . . . .	47
3.5	Daten-Export unter Verwendung der Mapping-Ontologie . . . . .	48
3.5.1	Gewinnung der Abarbeitungsreihenfolge der Knoten . . . . .	48
3.5.2	Berechnung eines einzelnen Zwischenknotens . . . . .	50
3.5.3	Algorithmus zur Abarbeitung von komplexen Mappings . . . . .	54
3.6	Erweiterung der i2b2 Software zum Vermittlungsportal . . . . .	55
4	<b>Ergebnisse</b>	59
4.1	Übersicht über die entwickelten Werkzeuge . . . . .	60
4.2	Erweiterung der Entity im EAV-Datenmodell . . . . .	62
4.3	Erzeugung und Bearbeitung von Zieldatensatz-Ontologien mit OntoEdit .	65
4.4	Bereitstellung von CSV-Daten und Erzeugung von Ontologien mit OntoGen	69
4.5	Erzeugung der Soarian-Ontologie für Erlangen . . . . .	71
4.6	Erzeugung und Bearbeitung von Mapping-Ontologien mit QuickMapp .	74
4.7	Datenexport nach i2b2 . . . . .	78
4.8	Ergebnisse beim Mapping von Münster und Erlangen auf den DPKK-Datensatz . . . . .	85
4.9	Ergebnisse bezüglich des i2b2-DPKK-Vermittlungsportals . . . . .	88
5	<b>Diskussion und Ausblick</b>	91
5.1	Ontologien in Medizin, Genomik und Proteomik . . . . .	92
5.2	Die Methodik des entwickelten Mapping-Systems . . . . .	93
5.2.1	Die Zieldatensatz-Ontologie als gemeinsame Terminologie . . . . .	93
5.2.2	Quellsystem-Ontologien zum Zugriff auf Nutzdaten . . . . .	94
5.2.3	Mapping zwischen Ontologien . . . . .	95
5.2.4	Ansätze zum Zugriff auf heterogene Datenbestände . . . . .	96
5.3	Wiederverwendung der entwickelten Wissensbasis in einer Krankenhaus-Ontologie . . . . .	97
5.4	Vorteile durch die Verwendung der Semantic-Web-Standards . . . . .	100
5.5	Vor- und Nachteile, sowie Verbesserungspotenziale des entwickelten Export-Konzepts . . . . .	101
5.5.1	Übersetzung der Mappings in SQL-Befehle . . . . .	101
5.5.2	Temporale Aspekte beim Zugriff auf getrennte Quellsysteme . . . . .	101
5.5.3	Optimierung der Abarbeitungsreihenfolge . . . . .	103
5.5.4	Optimierungen durch „Superknoten“ . . . . .	104

6 Literaturverzeichnis	105
A Ergänzungen zu RDF, RDFS und OWL	117
A.1 RDF-Serialisierungen . . . . .	117
A.2 Von der Manchester-Syntax zur Tripel-Darstellung . . . . .	118
A.3 Konstrukte in OWL . . . . .	121
B Klassen, Instanzen und Properties der Ontologien	125
B.1 MDR-System.owl . . . . .	125
B.2 OntoMappingSystem.owl . . . . .	127
B.3 Testdatensatz . . . . .	131
C Tabellen und Sonstiges	133
C.1 Soarian-Export: Tabellenbeschreibungen . . . . .	134
C.2 i2b2-Tabellenbeschreibungen . . . . .	138
C.3 DPKK-Datensatz . . . . .	144
C.4 i2b2 QDC-XML-Nachricht . . . . .	148



# KAPITEL 1

---

## Einleitung

---

### 1.1 Problemvorstellung: eine neue Forschungsdatenbank für das Deutsche Prostatakarzinom Konsortium e. V.

Im medizinischen Umfeld wurde zur Durchführung von spezifischen Projekten abseits der Routinebehandlung eine Vielzahl an voneinander getrennten Softwarelösungen entwickelt. Bei den Projekten handelt es sich um unterschiedliche Vorhaben, bei denen eine größere Menge an klinischen Daten in einer Datenbank gesammelt werden, um sie anschließend zentral auszuwerten. Beispiele für derartige Vorhaben sind die Durchführung von klinischen Studien, der Aufbau von speziellen Bio-Banken sowie die Einhaltung von Zertifizierungsvorgaben.

Auch wenn die Zielsetzungen unterschiedlicher Natur sind, so lassen sich häufig folgende Gemeinsamkeiten erkennen:

1. Die Daten müssen händisch in die Datenbank eingepflegt werden, obwohl sie zum Teil bereits in anderen Systemen, z. B. der *Elektronischen Patientenakte* (EPA, zur Begriffsklärung siehe [75]), vorliegen. Diese Mehrfachdokumentation stellt einen zeitlichen Mehraufwand und eine mögliche Fehlerquelle dar.
2. Der zu erfassende Datensatz ist immer auf die Ziele des jeweiligen Projekts zugeschnitten. In manchen Implementierungen ist dieser zusätzlich fest mit der Software verwoben; d. h. es ist schwierig, den Datensatz um weitere Elemente zu erweitern.

Das *Deutsche Prostatakarzinom Konsortium e. V.* (DPKK) ist ein Zusammenschluss von Experten aus der Urologie, Pathologie, Genetik, klinischer Chemie und der Grundlagen-

forschung. Gegründet wurde es 2003; inzwischen zählt es fast 70 Mitglieder, darunter die Urologischen Kliniken der Universitätsklinika Münster und Erlangen. Das DPKK betreibt eine zentrale Datenbank, in der deutschlandweit medizinische Daten zum Prostatakarzinom gespeichert werden, um effizient vernetzt Forschung zu betreiben. [25, 91, 103, 104]

Die beiden oben festgestellten Eigenschaften vieler Softwarelösungen zu medizinischen Forschungszwecken treffen auch auf die vom DPKK eingesetzte Datenbank zu. Durch die erforderliche Mehrfacheingabe und dem damit verbundenen hohen Zeitaufwand wurde das System bisher nicht im vollen Umfang genutzt. Darüber hinaus kam von DPKK-Mitgliedern das Bedürfnis auf, den Datensatz um weitere ergänzende Elemente aus der EPA zu erweitern, was bei der derzeitigen Implementierung nur mit großem Aufwand durchführbar wäre.

Im Rahmen dieser Arbeit soll prototypisch eine neue DPKK-Forschungsdatenbank konzipiert und umgesetzt werden, die im Gegensatz zur derzeitigen Lösung nicht auf einen festen Datensatz beschränkt ist und die keine unnötige Mehrfacheingabe bereits dokumentierter Daten erfordert.

## 1.2 Perspektiven durch Open-Source, Single-Source und das Semantic Web

Die am Beispiel der DPKK-Datenbank gezeigten Probleme sind in der Medizinischen Informatik im Forschungsgebiet der Informationssysteme durchaus bekannt. In den vergangenen Jahren wurde daher verstärkt an Lösungen gearbeitet, die sich auch im Rahmen dieser Arbeit für das DPKK als nützlich erweisen könnten.

### 1.2.1 i2b2 als generische Forschungsdatenbank

Im Rahmen des TMF-Projekts V054-01 IT-Strategie [76, 77] wurden verschiedene Softwarelösungen zur vernetzten Forschung evaluiert. In Erlangen konnte im Rahmen einer Studienarbeit [60] die US-amerikanische Open-Source-Software i2b2 (siehe Abschnitt 2.1 ab Seite 7) in Verbindung mit den Datenschutzmodellen der TMF [81] untersucht und am Universitätsklinikum Erlangen etabliert werden.

i2b2 ist mit einem (im Funktionsumfang reduzierten) Data Warehouse vergleichbar und ermöglicht es, klinische Daten in einer institutionsübergreifenden Rechere-Datenbank zusammenzuführen und bereitzustellen. Das System bietet sich aufgrund seines Funktionsumfangs und des verwendeten generischen Datenmodells als Forschungsdatenbank an.

Dadurch ist eine i2b2-Forschungsdatenbank nicht an einen festen Datensatz gebunden und kann ohne großen Aufwand beliebig angepasst und erweitert werden.

### 1.2.2 Single-Source zur Vermeidung von Mehrfachdokumentation

Die Existenz von geeigneten, flexiblen Datenbankanwendungen wie i2b2 löst jedoch nicht automatisch die Frage, wie bereits existierende Daten aus anderen Systemen geladen werden können, um das oben genannte Problem der Mehrfachdokumentation zu vermeiden.

Die Nutzung von klinischen Daten aus bereits existierenden primären und sekundären Systemen eines Krankenhauses wurde in [78] zu einer von drei Herausforderungen der Medizin-Informatik erkoren und ist aktuell ein Schwerpunkt in der Forschung rund um klinische Informationssysteme [29, 33, 34, 37, 77, 85]. Um die genannte Mehrfacheingabe von Daten zu reduzieren und stattdessen die Nutzung bereits dokumentierten Daten zu forcieren, wurde 2009 innerhalb der GMDS die Projektgruppe „Single Source“ (PG-SS) ins Leben gerufen. Die Bezeichnung „Single Source“ steht für das Ziel, dass Daten nur *einmal* dokumentiert werden und anschließend für andere Projekte genutzt werden, für die sie ursprünglich gar nicht erfasst wurden.

### 1.2.3 Das Semantic Web zur maschinenlesbaren Abbildung von medizinischem und technischem Wissen

Eine Nutzung von heterogenen klinischen Daten im Rahmen von „Single Source“ ist jedoch schwierig, da der ursprüngliche Dokumentationskontext oft nicht dem des neuen Projekts entspricht. Auch wenn es sich augenscheinlich um die gleichen Datenfelder handelt, kann dies nicht einfach angenommen werden, wenn dieser Umstand nirgends dokumentiert wurde.

Das Bedürfnis, Metadaten – also „Daten über Daten“ – von Informationssystemen eines Unternehmens zentral zu verwalten ist nicht neu. Datenbanken dieser Art werden häufig *Metadata Repository* (MDR) genannt. Auch im medizinischen Umfeld wird international seit vielen Jahren an MDR-Lösungen gearbeitet, um klinische Metadaten zentral zu verwalten [16, 70, 88, 90]. Bekannte Beispiele im Umfeld der Krebsforschung sind das CPCTR [71, 72], das ein MDR nach ISO/IEC 11179 verwendet, oder das *Cancer Data Standards Registry and Repository* (caDSR) des *National Cancer Institutes*. Bedingt durch ein feststehendes Modell (in der Regel ist dieses durch ein Tabellschema innerhalb einer relationalen Datenbank vorgegeben) stoßen diese Modelle jedoch an ihre Grenzen

hinsichtlich der abbildbaren Semantik und Erweiterbarkeit [66]. Darüber hinaus befinden sich viele generische Lösungen noch in der Konzeptions- und Entwicklungsphase [48].

Im Gegensatz zu starren Schemata in relationalen Datenbanken verspricht das *Semantic Web* deutlich flexiblere Wege, Informationen und Wissen in maschinenlesbarer Form abzubilden und bereitzustellen. Wissen kann hier „medizinisches Wissen“ bedeuten; ein bekanntes Beispiel ist der *NCI Thesaurus*, der etwa 200.000 Konzepte der Krebsforschung abbildet. Ebenso kann Wissen aber auch „technisches Wissen“ bedeuten, zum Beispiel Wissen über Metadaten im Sinne eines MDRs. Das Semantic Web ermöglicht es darüber hinaus, relativ einfach verschiedene Wissensdomänen miteinander zu verknüpfen.

### 1.3 Ziele und Fragestellungen dieser Arbeit

Bei einer Mitgliederversammlung des DPKK im Dezember 2009 wurde das i2b2-System von Erlangen als mögliche Alternative zur alten DPKK-Datenbank vorgestellt und von den DPKK-Mitgliedern durchweg positiv aufgenommen. Im Rahmen dieser Diplomarbeit soll das i2b2-System für einen möglichen Einsatz als DPKK-Datenbank erweitert werden. Die Urologische Klinik und Medizinische Informatik in Münster haben sich dazu bereiterklärt, dieses Vorhaben zu unterstützen.

Zur semantisch korrekten Zusammenführung der Metadaten zwischen Münster und Erlangen soll ein Semantik-Web-Ansatz erprobt werden.

Die Ziele dieser Arbeit sind somit:

- **Ziel 1:** Die Zusammenführung der klinischen Daten im Sinne eines Single-Source-Ansatzes. Zur semantischen Integration der Metadaten soll ein Semantik-Web-Ansatz verwendet werden.
- **Ziel 2:** Die prototypische Umsetzung einer DPKK-Forschungsdatenbank auf i2b2-Basis zwischen dem UK-Münster und dem UK-Erlangen. Aus den beiden Klinischen Arbeitsplätzen (KAS) Soarian und Orbis soll die Prostatakarzinom-Dokumentation in eine zentrale prototypische i2b2-DPKK-Datenbank abgebildet werden.

Als Fragestellungen und Arbeiten ergeben sich damit für diese Arbeit:

- **Fragestellung 1: Wie können Semantic-Web-Techniken zur Integration klinischer Metadaten und zur Zusammenführung klinischer Fakten-Daten verwendet werden?**

Es soll eine geeignete Lösung zur semantischen Beschreibung des Prostatakarzinom-Datensatzes gefunden werden, mit deren Hilfe eine automatische Zusammenführung der Datensätze aus Erlangen und Münster möglich wird. Hierfür soll auf Semantic-Web-Techniken zurückgegriffen werden.

Die gefundene Technik soll für die DPKK-Datenbank auf i2b2-Basis umgesetzt werden.

- **Fragestellung 2: Wie kann eine DPKK-Forschungsdatenbank auf i2b2-Basis umgesetzt werden?**

Welche Vermittlungs- bzw. Portalfunktionen soll das System unterstützen? Basierend auf diesen Überlegungen soll i2b2 um die nötigen Client- und Server-Funktionen erweitert werden.



# KAPITEL 2

---

## Grundlagen

---

### 2.1 Informatics for Integrating Biology and the Bedside (i2b2)

Bei *Informatics for Integrating Biology and the Bedside*<sup>1</sup> (i2b2) handelt es sich um ein Open-Source-Projekt aus Boston, Massachusetts, USA. Die Software wurde ursprünglich am *Massachusetts General Hospital*<sup>2</sup> entwickelt, das der Non-Profit-Organisation *Partners Healthcare, Inc*<sup>3</sup> angehört. In Zusammenarbeit mit der *Harvard Medical School* wurde i2b2 im Jahr 2004 als Kandidat für eine Fördermittelausschreibung der *National Institutes Of Health* (NIH) angemeldet und gewann den Zuschlag. Seitdem wird i2b2 von den NIH als eines von sieben *National Centers for Biomedical Computing* (NCBC)<sup>4</sup> geführt – Kompetenzzentren, die in den USA die nötige Infrastruktur zur effizienten Verarbeitung klinischer Daten bereitstellen sollen [63]. Im Jahr 2010 hat es das Projekt in die zweite Förderphase geschafft.

Bei i2b2 handelt es sich um eine dreischichtige Client-Server-Architektur [62], die zur Auswertung klinischer Daten verwendet wird. In einer relationalen Datenbank werden klinische Nutzdaten sowie Metadaten vorgehalten. Die eigentliche Serversoftware besteht vollständig aus Webservices, die unterschiedliche Funktionen übernehmen. Durch die „lose Kopplung“<sup>5</sup> der Komponenten soll das System leicht erweiterbar sein. Auf der Client-Seite

---

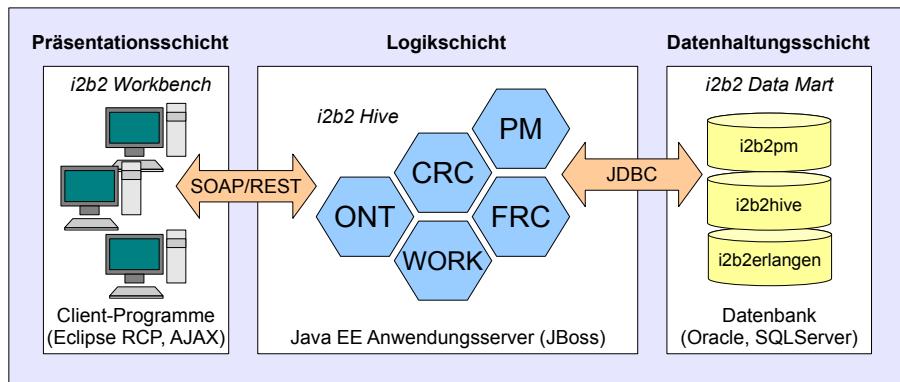
1 <http://www.i2b2.org>, Abruf: 19.12.2010

2 <http://www.mgh.harvard.edu>, Abruf: 19.12.2010

3 <http://www.partners.org>, Abruf: 19.12.2010

4 <http://www.ncbi.nlm.nih.gov>, Abruf: 19.12.2010

5 So die Entwickler, siehe <https://www.i2b2.org/resrcs/pdf/HiveIntroduction.pdf>, Abruf: 19.12.2010



**Abbildung 2.1:** Vereinfachte Darstellung der i2b2-Architektur.

existieren inzwischen zwei Anwendungen: die i2b2-Workbench auf Eclipse-RCP-Basis<sup>1</sup> (Fat Client) sowie eine vollständige JavaScript-Implementierung auf AJAX-Basis (Web Client). Die Architektur ist in Abbildung 2.1 illustriert.

Mit i2b2 können auf besonders benutzerfreundliche Weise Patientenkohorten nach verschiedenen Einschluss- und Ausschlusskriterien selektiert werden [64]. i2b2 bietet sich damit als „abgespecktes“ klinisches Data Warehouse zur Beantwortung von Forschungsfragen oder zur Patientenrekrutierung an – es kann aber auch institutionsübergreifend eingesetzt werden, wie dies im Rahmen dieser Arbeit erfolgen soll. Ein Screenshot der i2b2-Workbench wird in Abbildung 4.18 auf Seite 90 gezeigt.

Da das System bereits ausführlich in [60] analysiert und vorgestellt wurde, soll im Folgenden nur kurz darauf eingegangen werden, welche Arbeiten rund um i2b2 in Deutschland stattgefunden haben.

Innerhalb des Projektes „V054-01 IT-Strategie“ [76, 77] der *Technologie- und Methodenplattform für die vernetzte medizinische Forschung e.V.* (TMF) wurde i2b2 in [60] auf seine Verwendbarkeit in Deutschland – insbesondere unter datenschutzrechtlichen Aspekten – untersucht [61]. Gemeinsam mit der Universität Göttingen konnte im Rahmen des TMF-Projekts im Juni 2009 ein Besuch bei dem i2b2-NCBC in Boston stattfinden, bei dem das System noch besser kennengelernt und ein Memorandum of Understanding formuliert wurde. Zwischen Göttingen und Erlangen konnte so eine enge Kooperation bezüglich i2b2 aufgebaut werden.

Während all dieser Arbeiten konnte i2b2 am Universitätsklinikum Erlangen etabliert werden. Aufgrund des großen Interesses an dem System wurden im Dezember 2009 und im Januar 2010 zwei gut besuchte i2b2-Workshops von der TMF koordiniert.

<sup>1</sup> <http://www.eclipse.org/home/categories/rcp.php>, Abruf: 19.12.2010

## 2.2 Die klinischen Arbeitsplatzsysteme in Erlangen und Münster

### 2.2.1 Siemens Soarian Clinicals und der Data-Warehouse-Export am UK-Erlangen

Bei *Soarian Clinicals* [41] der Firma *Siemens AG* handelt es sich im Wesentlichen um ein webbasiertes *Electronic-Data-Capture-System* (EDC-System)<sup>1</sup>, das am UK-Erlangen als klinisches Arbeitsplatzsystem Verwendung findet.

Ein EDC-System [30, 102] erzeugt Formulare in Form von Bildschirmmasken, in denen die Benutzer verschiedene zu dokumentierende Parameter auswählen oder Werte eingeben können. Die Formulare erinnern vom Aufbau her stark an konventionelle Formulare aus Papier, wobei zur Datenerfassung die bekannten GUI-Widgets [32, S. 452] (Checkboxen, Radio-Buttons, Text-Eingabefelder usw.) dienen. Durch die elektronische Erfassung werden Benutzer-Interaktionen und Plausibilitätsprüfungen der eingegebenen Werte möglich. Ebenso sollen die Daten durch die strukturierte Erfassung leichter ausgewertet werden können. Oft finden EDC-Systeme auch Anwendung bei der effizienten Durchführung klinischer Studien [30].

Am Universitätsklinikum Erlangen wurden im Jahr 2002 erste Arbeiten zur Einführung von Soarian geleistet [51]. In den folgenden Jahren wurde das System im pflegerischen Bereich (z. B. Braden-Skala, Dekubitus- und Sturzdokumentation) und im ärztlichen Bereich flächendeckend ausgerollt. Nach [69, S. 52] wird Soarian seit 2007 auf fast allen Stationen des Klinikums von über 2800 Anwendern benutzt. Im Rahmen der Zertifizierung der Urologischen Klinik als Universitäts-Prostatakrebs-Zentrum (UCC) im Jahr 2009 [98] wurde eine umfangreiche Dokumentation zum Prostatakarzinom in Soarian umgesetzt. Daneben erfolgen inzwischen auch die Dokumentationen der Karzinome an Mamma und Thorax sowie die des kolorektalen Karzinoms elektronisch. Aktuelle Arbeiten an Soarian umfassen den flächendeckenden Roll-Out einer elektronischen Arztbriefschreibung [23] und Auftragskommunikation. Neben Soarian kommt in Erlangen SAP IS-H [38] im Abrechnungswesen und zur Verwaltung von Patientenstammdaten zum Einsatz.

Das System arbeitet patienten- und fallorientiert. Abhängig von den klinischen Behandlungspfaden, die ein Patient während eines Aufenthaltes durchläuft, werden durch die Pflegekräfte und Ärzte unterschiedliche Formulare angelegt und mit den klinischen Daten ausgefüllt. Abbildung 2.2 auf Seite 10 zeigt ein solches Formular.

Technisch betrachtet handelt es sich bei einem EDC-System um eine graphische Benutzerschnittstelle zu einem relationalen Datenbanksystem, bei dem die erfassten Werte in

---

<sup>1</sup> Aspekte wie die mächtige Workflow-Unterstützung sollen hier großzügig vernachlässigt werden, da im Rahmen dieser Arbeit ausschließlich die erfassten klinischen Daten von Interesse sind.

The screenshot shows a web-based application window for SIEMENS Soarian. At the top, the title bar reads "SIEMENS Soarian® [SCCPROD2] - Windows Internet Explorer". The main header includes the Siemens logo, the user name "Felix Kopcke, UKER", and navigation links like "Patientenliste", "Ambulanzsicht", "Leistungsstellen", "Suchen", "Ext. Anwendungen", "Drucken", "Hilfe", and "Abmelden". Below the header, the page title is "Testpatient Prod". The main content area is titled "UR Befund - Organ - TUR - Exzision" and is "Eingegeben von: Sebastian Mate". The form is divided into several sections:

- Tumorhistologie:** Pathologisches Institut: "Pathologisches Institut Universitätsklinikum Erlangen".
- Histologie:** Histologie Datum: [dropdown], Eingangszeit: [dropdown].
- Histo-Patho-Grading (G):** Radio buttons for G1, G2, G3, X, O. Descriptions: G1=Gut differenziert, G2=Mäßig differenziert, G3=Schlecht differenziert, X=Differenzierungsgrad kann nicht bestimmt werden, O=Grading nicht vorgesehen.
- Gleason-Score:** Input fields for Gleason components (e.g., 3 + 4 = 7) and "keine Angabe".
- Gleason-Muster:** Radio buttons for Gi1, Gi2, Gi3, Gi4, Gi5.
- Other sections:** "sonstige Histologie" (checkboxes for Prostatis, Nekrosen, Samenblaseninfiltration, Tumorkalisation, extraprostatische Tumorausbreitung), "Tumorkalisation" (radio buttons for R, B, basal, apikal, mittig).

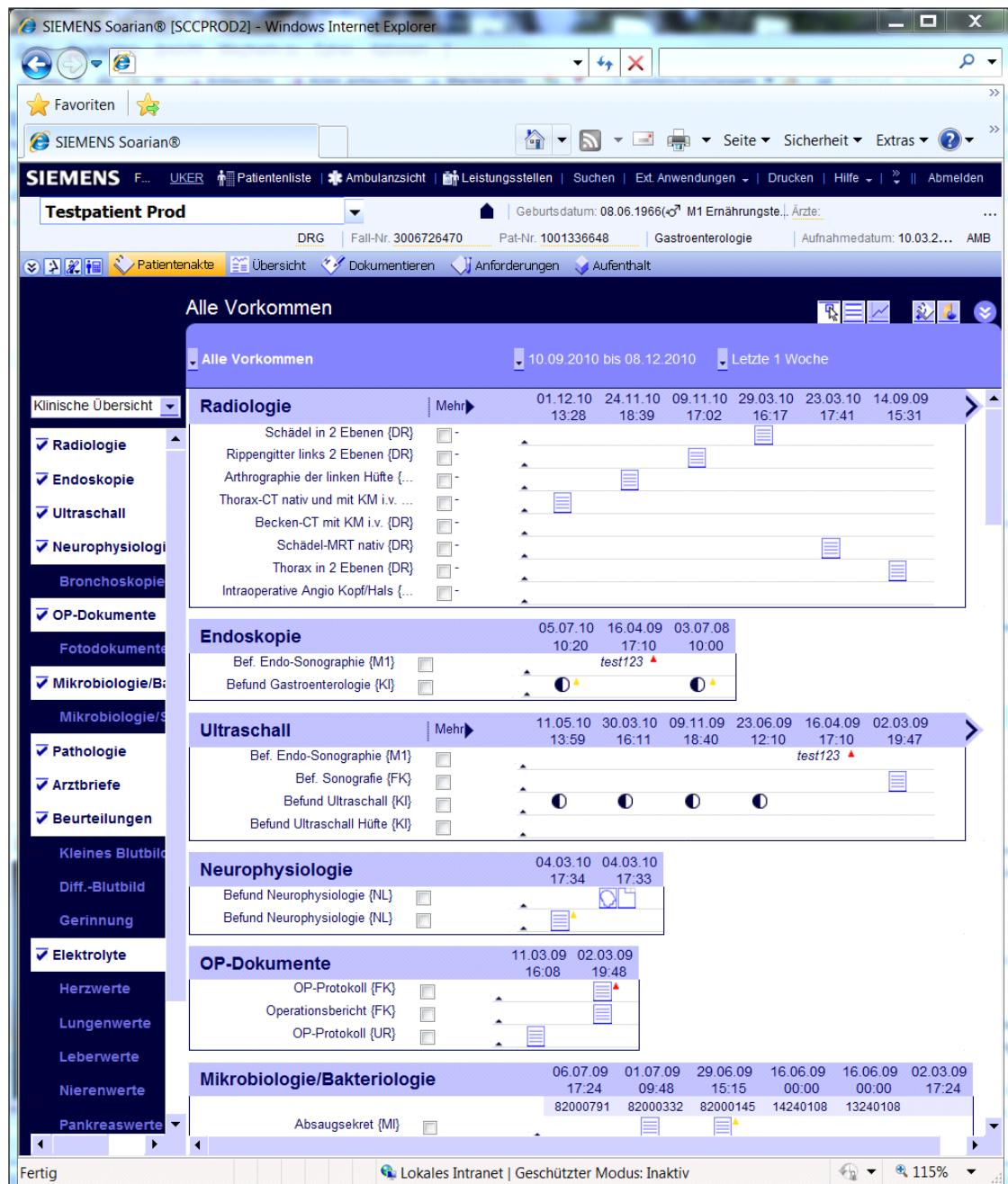
At the bottom, there are status indicators: "Erheben 08.12.2010 14:05", "Dokumentiert für", "Status Vollständig", and "Fertig". A footer note says "Lokales Intranet | Geschützter Modus: Inaktiv".

**Abbildung 2.2:** Soarian, mit der Darstellung eines Dokumentationsformulares. Die einzelnen, zu dokumentierenden Daten werden durch verschiedene GUI-Widgets erfasst. Die einzelnen Dokumentationsformulare werden in der Patientenakten-Ansicht chronologisch gesammelt und angezeigt (siehe Abbildung 2.3 auf Seite 11).

Tabellen gespeichert werden. Soll mit den erfassten Daten gearbeitet werden, stehen in der Regel Export-Funktionen im System zur Verfügung; alternativ kann direkt auf die Datenbank zugegriffen werden.

Zur Entwicklung neuer Formulartypen stellt Soarian die nötigen Werkzeuge bereit; diese Arbeit wird in Erlangen vom *Medizinische Zentrum für Informations- und Kommunikationstechnik* (MIK), der IT-Abteilung des Klinikums, übernommen.

Ein wichtiger Aspekt bei der Konzeption eines neuen Soarian-Formulares ist die Trennung des Formular-Layouts und den einzelnen, darin verwendeten GUI-Widgets, die in Soarian als „Component“ bezeichnet werden. Beide werden getrennt voneinander erstellt; anschließend werden die Components in das Formular eingesetzt. Aufwändige Components (z. B. mit vielen Auswahlmöglichkeiten) können so leicht wiederverwendet werden. Außerdem



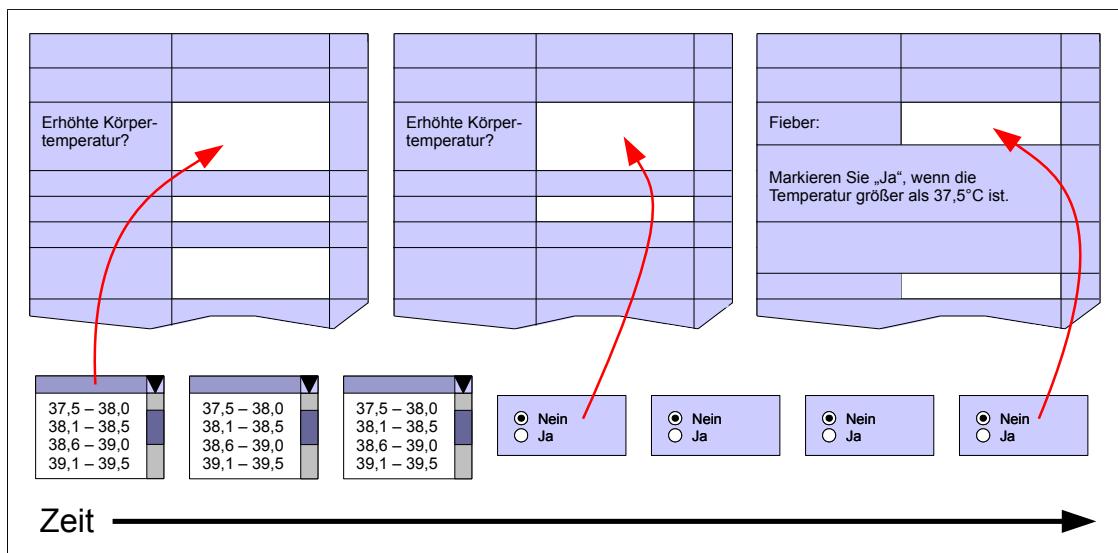
**Abbildung 2.3:** Die Patientenakte in Soarian, die alle im Behandlungszusammenhang anfallenden Informationen (z.B. Befunde in Form von Dokumentationsformularen) sammelt und chronologisch darstellt.

können Werte auch vorausgefüllt werden<sup>1</sup>. Ein Formular stellt somit lediglich ein Gerüst dar, in das die einzelnen Components eingesetzt werden.

Durch die Trennung können Formular und Component getrennt voneinander weiterentwickelt werden. Wird eine Component geändert, steht die neue Version auch in den anderen Formularen zur Verfügung. Beispielsweise können bei einer Component neue Auswahloptionen ergänzt oder entfernt werden; es ist sogar möglich, den Widget-Typ der Component zu ändern.

Abbildung 2.4 illustriert dies anhand eines Beispiels für das Attribut „Fieber“. Sowohl das Formular als auch die Component erfahren Änderungen. Bei der Component wird der Widget-Typ von einer Auswahlliste zu einer Radiobutton-Gruppe geändert. Die ursprünglich enthaltenen Auswahloptionen an Temperaturbereichen werden verworfen und durch ein einfaches „Ja“ und „Nein“ ersetzt.

Im Jahr 2010 wurde durch P. Müller in [65] und F. Köpcke<sup>2</sup> ein Export in das klinische Data Warehouse (DWH) des Uniklinikums entwickelt. Damit stehen bei Bedarf sämtliche Meta-



**Abbildung 2.4:** Metadaten in Soarian: Formulare (oben) und Components (unten) können unabhängig voneinander geändert werden. Die Components werden in die Formulare eingesetzt und können so wiederverwendet werden. Das Formular kann auch Text enthalten, der sich auf die Components bezieht, von diesen aber unabhängig gespeichert wird.

1 Die Vorausfüllung der Component wird über den Parameter „CarryForward“ beschrieben (siehe Tabelle C.1 auf Seite 134).

2 Lehrstuhl für Medizinische Informatik, Universität Erlangen-Nürnberg

und Nutzdaten aus Soarian tagesaktuell in einem einheitlichen, einfachen Datenformat für weitere Projekte zur Verfügung. So wurde in [65] beispielsweise ein vollautomatischer Weitertransport der Daten nach i2b2 sowie in das Studienmanagementsystem *SecuTrial* der Firma *iAS GmbH* umgesetzt. Auch im Rahmen dieser Arbeit wird auf diesen Export zugegriffen. Das genaue Tabellenformat der Meta- und Nutzdaten ist im Anhang C.1 ab Seite 134 beschrieben. In Tabelle C.4 auf Seite 136 wird anhand des Fieber-Beispiels gezeigt, wie sich eine Component im Verlauf der Zeit ändern kann.

## 2.2.2 Agfa HealthCare ORBIS am UK-Münster

In Münster befindet sich seit 2002 das Produkt *ORBIS* der Firma *Agfa HealthCare* in Betrieb, ein in Deutschland, Österreich und in der Schweiz stark verbreitetes klinisches Arbeitsplatzsystem. Nach [31] waren Ende 2003 bereits 2300 Benutzer mit einem ORBIS-Zugang ausgestattet. Da das System den Funktionsumfang betreffend vergleichbar mit Soarian ist, soll es an dieser Stelle nicht weiter vorgestellt werden. Ein guter Überblick über ORBIS findet sich in [52].

The screenshot shows a computer screen displaying a medical record for a prostate biopsy. The window title is "ORBIS® OpenMed - Test 2010 - [Prostatahistologie MS - Ver. 214]". The top status bar shows the date and time: "8.12.2010 16:14:31" and the user "BREILBER/KH@TEST42 <TEST Station 1>". The main content area is titled "Prostata - Histologie: Biopsie". It contains a barcode, patient ID "0201001192", and a telephone number "Tel: 55911". There are fields for "Test, 2010" (S), "Geb. Datum: 21.02.09", "Aufnahmzeit: 10:00", and "Beruf: Arbeit". Below this, there is a note about the test being performed at AOK Schleswig-Holstein. The "Prostatazentrum am UKM" logo is visible on the right. The report lists various parameters: PSA (2.00 ng/ml), free PSA (1.00 ng/ml), Body-Mass-Index (29), and IPSS (22). It also includes a diagram of a prostate with numbered points A through H, each with associated measurements and Gleason scores. The left side of the screen shows a vertical toolbar with icons for "Signieren (abschließen)", "Speichern", "Abbruch", and "Drucken".

**Abbildung 2.5:** Ein Orbis-Dokumentationsformular.

## 2.3 Das Semantic Web

### 2.3.1 Die Vision des Semantic Web

TJ Berners-Lee, Erfinder von HTML und Begründer des World Wide Web [6], stellt in [7] seine Vision des Semantic Web vor. Elektronische Geräte greifen dabei scheinbar „intelligent“ auf die im Internet verfügbaren Informationen zu und erleichtern damit das Leben der Menschen:

In einem fiktiven Szenario sucht ein Gerät für eine Patientin einen Spezialisten. Dieser soll sehr gute Bewertungen durch andere Patienten besitzen, ebenso soll seine Praxis in der Nähe liegen. Das Gerät vergleicht automatisch den Kalender der Patientin mit dem des Spezialisten, stellt unwichtige Termine hinten an und beachtet sogar die Rush-Hour bei der Routen- und Terminplanung.

Das Gerät sammelt und verarbeitet Informationen aus unterschiedlichen Datenquellen des World Wide Web. Damit das aber funktioniert, müssen diese in einer geeigneten Form vorliegen. Er stellt fest: „*Most of the Web's content today is designed for humans to read, not for computer programs to manipulate meaningfully. Computers can adeptly parse Web pages for layout and routine processing—here a header, there a link to another page—but in general, computers have no reliable way to process the semantics [...].*“ Er schlägt mit dem Semantic Web eine Erweiterung des World Wide Web vor, um diesen Umstand zu umgehen: „*The Semantic Web brings structure to the meaningful content to Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users.*“

Das Semantic Web berücksichtigt dabei die heterogene Natur des WWW: Die Daten können z. B. unvollständig oder vollständig, kommerzieller oder akademischer Art sein und unterschiedlichen Kulturen, Sprachen oder Medien entstammen. Ebenso müssen Inhalte unterschieden werden, die für den „Konsum“ durch Menschen oder für Maschinen bestimmt sind. Von Werbespots über Poesie bis hin zu Datenbanken und Sensordaten ist alles vorhanden [7]. Die Ursache hierfür lässt sich gut mit dem AAA-Slogan zusammenfassen: „**A**nyone can say **A**nything about **A**ny topic.“ [1, S. 7]

Mit dem Semantic Web sollen Webseiten durch verschiedene Techniken so erweitert werden, damit das oben beschriebene Szenario irgendwann Realität wird. Berners-Lee geht sogar noch einen Schritt weiter und visioniert: „*In addition, if properly designed, the Semantic Web can assist the evolution of human knowledge as a whole.*“ [7]

Dem Prozess der Standardisierung des Semantic Web hat sich das *World Wide Web Consortium* (W3C, siehe: <http://www.w3.org>) verschrieben, ein Gremium zur Standardisierung

von Techniken, die im World Wide Web eingesetzt werden. Mit seinen „W3C recommendations“ hat es in den vergangenen Jahren verschiedene Informations-Spezifikationssprachen als Standards veröffentlicht, die die Grundbausteine des Semantic Web bilden sollen.

### 2.3.2 Die Extensible Markup Language (XML)

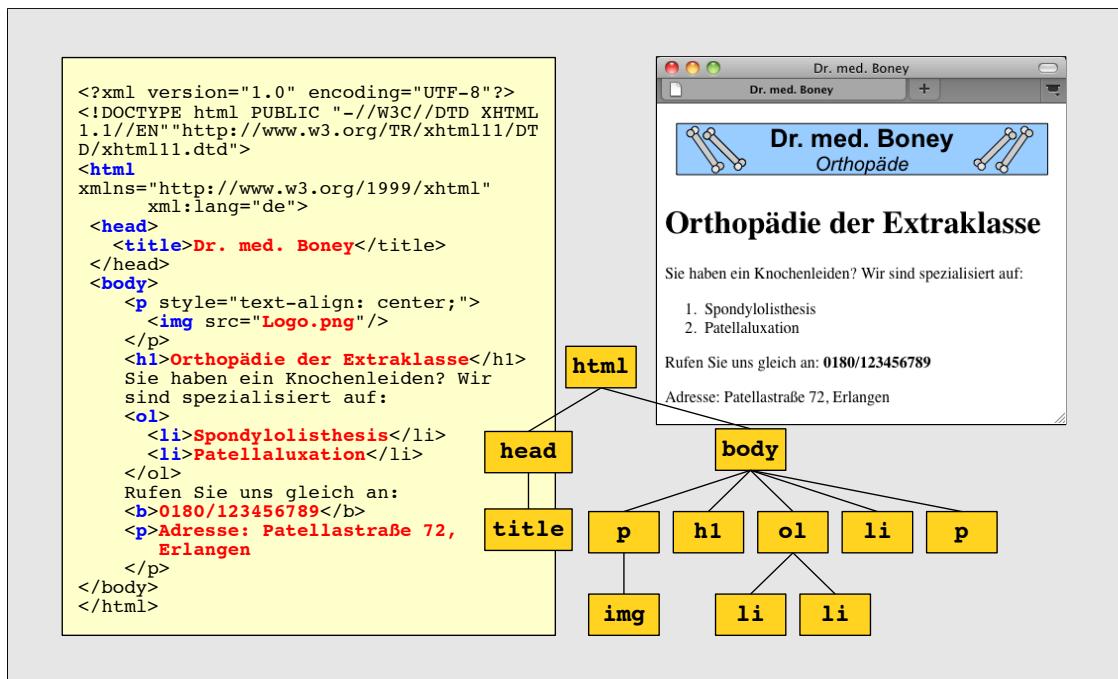
Bei der Extensible Markup Language (XML) [13] handelt es sich genau genommen um keine Semantic-Web-Technik. Sie entstammt dem „herkömmlichen“ Web, wo sie einen Standard zur Speicherung und dem Austausch von strukturierten Daten darstellt. Die später gezeigten Semantic-Web-Techniken verwenden jedoch aus diesem Grund XML zur Serialisierung (Speicherung) ihrer Inhalte.

Bei XML handelt es sich um eine Auszeichnungssprache. Sie dient dazu, einzelne Abschnitte eines Text-Dokumentes auszuzeichnen bzw. zu annotieren. Diese Annotationen stellen zusätzliche Informationen (auch „Metadaten“ genannt) dar, anhand derer die eigentlichen Textdaten durch Softwarekomponenten einheitlich abgelegt, ausgetauscht und verarbeitet werden können. Diese Auszeichnung erfolgt durch das Ablegen der Textdaten in hierarchischen (Baum-)Strukturen. Die Hierarchie wird durch die Verschachtelung einzelner Auszeichner, genannt *Tags*, aufgebaut.

XML selbst definiert jedoch keine einzelnen Tags oder deren Semantik (Bedeutung), sondern beschreibt nur die logische Struktur eines auf XML basierenden Dokuments, also die Syntax. Zum Beispiel wird beschrieben, wie ein XML-Tag in einer XML-Datei formatiert sein muss („spitze Klammer auf, dann der Tag-Name, spitze Klammer zu“). XML ist damit eine Metasprache, mit der eigene Auszeichnungssprachen konstruiert werden können.

**Beispiel:** Der XHTML-Standard [40] ist eine solche auf dem XML-Standard basierende Auszeichnungssprache, die dazu dient, Webseiten in einem einheitlichen Format auszutauschen. Abbildung 2.6 auf Seite 16 zeigt, wie die XML-Syntax dazu verwendet wird, um ein XHTML-Dokument zu strukturieren. Die Semantik der Tags *html*, *head*, *body*, *h1*, usw. ist jedoch Teil des XHTML-Standards. Dort wird beschrieben, wie ein XHTML-kompatibler Browser diese gemäß XHTML-Standard korrekt darzustellen hat. Beispielsweise bedeutet der Tag „*h1*“, dass alles, was zwischen *<h1>* und *</h1>* steht, eine Überschrift ist, die mit einer gewissen Schriftgröße angezeigt werden soll. Nur wenn ein Browser den XHTML-Standard „versteht“, kann er das XML-Dokument korrekt darstellen.

XML selbst soll an dieser Stelle nicht weiter vorgestellt werden. Eine gute Einführung findet sich beispielsweise in [44, S. 17-29].



**Abbildung 2.6:** Beispiel für ein XHTML-Dokument (links) und der Darstellung in einem Browser (rechts oben) sowie der Baumstruktur der wichtigsten Tags.

### 2.3.3 Das Resource Description Framework (RDF)

#### Die Grenzen von XML

XML ist eine syntaktische Auszeichnungssprache, um Daten in strukturierter Form effizient auszutauschen. Jedoch ist es mit XML nicht möglich, die Semantik der Inhalte zu übermitteln.

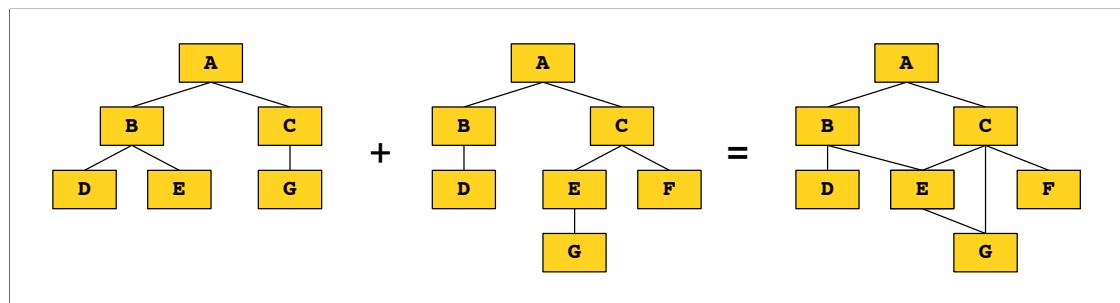
Abbildung 2.7 auf Seite 17 illustriert dies. Durch die geschickte Wahl der XML-Tag-Namen und der Inhalte erkennt man als Mensch sofort, dass die beiden oberen XML-Dokumente dieselbe Information beinhalten: die Kontaktdaten von einem Orthopäden. Beim unteren Dokument erkennt man zwar aufgrund des Inhalts auch noch, dass es sich wohl um die Adresse eines Arztes handelt; dass dieser jedoch ein Orthopäde ist, geht in den willkürlichen Tag-Namen verloren. Solange sich die Tag-Namen und die Baumstruktur des XML-Dokuments nicht ändern, können durch XML Daten in strukturierter Form ausgetauscht werden. Soll aber das rechte XML-Dokument mit einer Software verarbeitet werden, die ursprünglich nur mit dem linken Format umgehen kann, so ist dies nicht ohne



**Abbildung 2.7:** Unterschiedliche XML-Dokumente können mit unterschiedlicher Syntaktik die gleichen Inhalte ausdrücken (oben links und rechts). Die einzelnen Tag-Elemente sind jedoch ohne Semantik (unten links).

Änderungen an der Software möglich<sup>1</sup>. Rein technisch gesehen handelt sich bei den oberen XML-Dokumenten – syntaktisch betrachtet – um völlig unterschiedliche Dokumente.

Besondere Schwierigkeiten ergeben sich, wenn Informationen aus unterschiedlich strukturierten XML-Dokumenten miteinander kombiniert werden sollen. Bei einem XML-Dokument handelt es sich um eine monohierarchische Baumstruktur. Werden jedoch zwei solche zusammengeführt, kann sich daraus eine Polyhierarchie ergeben, die mit XML nicht mehr darstellbar ist (siehe Abbildung 2.8).



**Abbildung 2.8:** Werden zwei Monohierarchien (auch Taxonomien genannt, links und mittig) zusammengeführt, kann dies zu einer Polyhierarchie führen (rechts).

<sup>1</sup> Zwar gibt es durchaus Techniken wie XML Stylesheet Transformationen (XSLTs), mit denen ein Syntax in einen anderen übersetzt werden kann; das ändert jedoch nichts daran, dass die Tags eines XML-Dokuments selbst keine Bedeutung haben [42, S. 67].

Anhand des XHTML-Beispiels weiter oben wird die daraus resultierende Problematik für das Semantic Web besonders deutlich: Mit dem XHTML-Dokument kann zwar effizient formatierter Text über das WWW ausgetauscht werden, da er mit vielen HTML-Editoren bearbeitet und mit jedem Browser korrekt dargestellt werden kann; eine weitere maschinelle Verarbeitung der im Text enthaltenen Informationen ist jedoch nicht möglich. Zum einen können die Informationen nicht aus dem XHTML-Dokument extrahiert werden, zum anderen können sie nicht mit den Informationen anderer Webseiten verknüpft werden. Das „intelligente“ Gerät der Patientin in TJ Berners-Lees Vision wird jedenfalls nicht in der Lage dazu sein, unter Verwendung der Webseite in Abbildung 2.6 auf Seite 16 automatisch einen Termin bei Dr. Boney zu vereinbaren.

Man sieht, dass mit XML zwar eine syntaktische Interoperabilität hergestellt werden kann, jedoch keine semantische. Dies soll im Semantic Web durch die Verwendung verschiedener Techniken gelingen, die im Folgenden kurz vorgestellt werden.

### Statements mit RDF

Das *Resource Description Framework (RDF)* [50] bildet das Grundgerüst des Semantic Web. Mit RDF können global eindeutig identifizierbare Inhalte bereitgestellt und im Gegensatz zu XML-Dokumenten leicht miteinander verknüpft werden.

RDF baut auf einem sehr einfachen Datenmodell auf. Jedes RDF-Dokument ist ein gerichteter, benannter Graph. Dies ist eine Menge an Knoten, die mit einer Menge an gerichteten Kanten verbunden ist. Im Fall von RDF sind sowohl die Knoten als auch die Kanten benannt, also mit einer Beschriftung versehen.

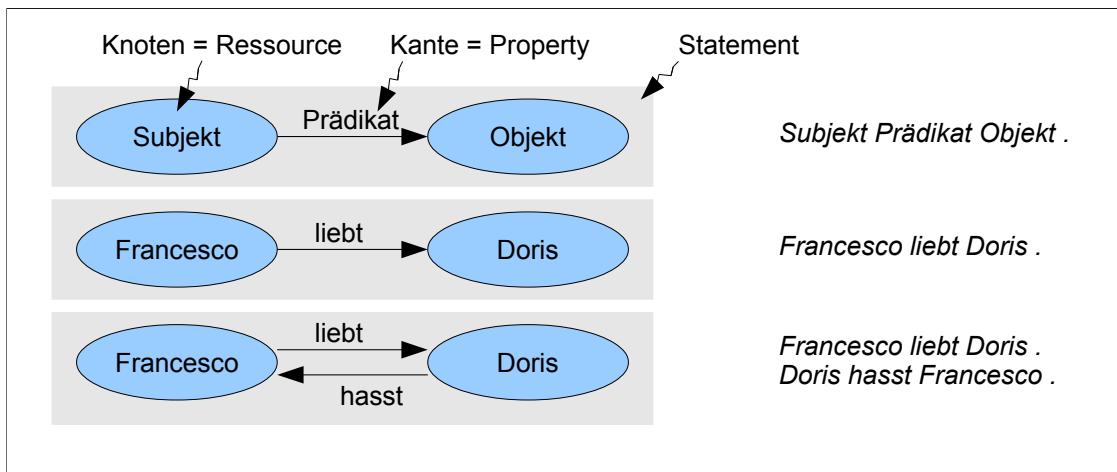
Die Knoten des Graphen werden in RDF als *Ressourcen* bezeichnet, die verknüpfenden Kanten als *Properties*. Zwei Ressourcen, die über eine Property miteinander verbunden sind, werden *Statement* (Aussage) genannt (siehe Abbildung 2.9 auf Seite 19, oben). Ein Statement kann unter Berücksichtigung der Pfeilrichtung als Satz der Form *Subjekt, Prädikat, Objekt* gelesen und niedergeschrieben werden (siehe Abbildung 2.9, Mitte):

**Francesco** **liebt** **Doris** .

Damit ist es möglich, beliebige Aussagen zu machen. Beispielsweise könnte man ergänzen:

**Doris** **hasst** **Francesco** .

Durch Kombination einzelner Statements (siehe Abbildung 2.9, unten) können komplexe Graphen, die beliebige Inhalte beschreiben, konstruiert werden. Man spricht dann von einem *semantischen Netz* oder (im Rahmen des Semantic Web) von einer *Ontologie*.



**Abbildung 2.9:** Oben: einzelne Ressourcen (Knoten = blaue Ellipsen) werden mit Properties (Kanten = „Pfeile“) zu Statements verbunden. Mitte: Statements können als Sätze gelesen und niedergeschrieben werden (rechts). Unten: Statements können miteinander verknüpft werden.

### URIs als eindeutige Bezeichner

Gemäß AAA-Slogan kann im WWW grundsätzlich jeder beliebige Informationen global bereitstellen und Aussagen über irgendetwas treffen. Damit eine Verknüpfung dieser Statements möglich wird, müssen diese eindeutig im WWW identifizierbar sein. Wenn man beispielsweise Aussagen über die Beziehung zwischen Francesco und Doris machen möchte, muss genau bekannt sein, um welchen Francesco und welche Doris es sich handelt. Mit der bisher gezeigten Darstellungsform ist dies noch nicht möglich, da die beiden Personen nur innerhalb der (bisher winzigen) Beispielontologie identifizierbar sind, jedoch nicht global.

Eine global eindeutige Identifizierbarkeit wird durch die Verwendung von *Uniform Resource Identifiers (URIs)* für die Bezeichnung der Ressourcen erreicht. Statt „Doris“ und „liebt“ könnte man die folgenden URIs verwenden:

```
http://www.uk-erlangen.de/Liebesontologie#Doris
http://www.uk-erlangen.de/Liebesontologie#liebt
```

Natürlich betreibt das UK-Erlangen keine Liebesontologie; auf die Adressen kann daher auch nicht zugegriffen werden. Jedoch ermöglicht es die URI, die einzelnen Ressourcen eindeutig global zu definieren: Wird Doris' URI irgendwo auf der Welt in einer anderen

Ontologie verwendet – was gemäß AAA-Slogan möglich ist – dann ist sofort klar, dass es sich um die Doris aus dieser Beispielontologie handelt.

Um eine gewisse Übersicht zu wahren, kann die URI mit einem Namensraum-Prefix abgekürzt werden, beispielsweise: p1:Doris. Das Namensraum-Prefix p1: muss dann separat definiert werden:

```
p1: http://www.uk-erlangen.de/Liebesontologie#
```

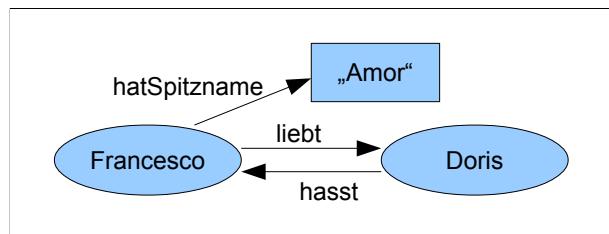
### RDF Literale

Neben den Ressourcen gibt es in RDF-Graphen noch die sog. *Literale*. Mit ihnen können den Ressourcen Datenwerte zugeordnet werden. In Abbildung 2.10 wird beispielhaft Francesco der Spitzname „Amor“ als Literal zugeordnet.

In einem Statement können Literale niemals als Subjekt Verwendung finden, sondern nur als Objekt. Intuitiv ausgedrückt bedeutet dies, dass von einem Literal niemals ein Pfeil ausgehen kann. Für Literale können unter Verwendung der *XML Schema Datatypes (XSD)* [8] verschiedene Datentypen verwendet werden, was hier jedoch nicht weiter erläutert werden soll (siehe hierzu: <http://www.w3.org/TR/rdf-concepts/#section-use-xsd>, Abruf: 10.12.2010).

### Weitere RDF-Konstrukte

Mit RDF können grundsätzlich nur binäre Beziehungen ausgedrückt werden. Sollen n-äre Beziehungen konstruiert werden, funktioniert dies nur über Zwischenknoten. Die sog. *Blank Nodes* sind ein Konstrukt, mit dem Zwischenknoten in einen RDF-Graphen eingefügt werden können, die keine eigentliche Bedeutung besitzen, sondern nur als



**Abbildung 2.10:** Literale stellen den zweiten Knotentyp in RDF dar. Mit ihnen können Ressourcen Datenwerte zugeordnet werden.

„Bindeglied“ dienen sollen. Abbildung 2.11 illustriert dies. Blank Nodes sind auch nur innerhalb der jeweiligen Ontologie gültig und können nicht global über URIs referenziert werden.

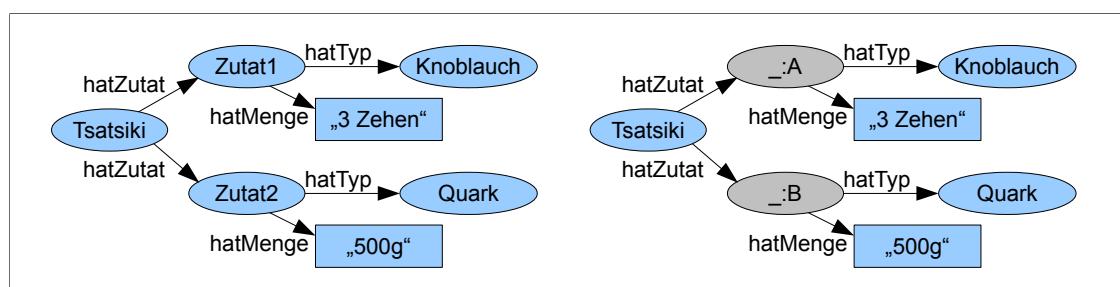
RDF erlaubt es durch *Reifikation*, Aussagen über Aussagen zu konstruieren. Mit speziellen RDF-Konstrukten können Statements wie „Peter behauptet, dass Doris Francesco hasst!“ gebaut werden (siehe [42, S. 88]). Weiterhin existieren in RDF Konstrukte, mit denen Ressourcen gruppiert (als ungeordnete oder geordnete Menge), als Alternativen zu anderen Ressourcen oder als Listen ausgedrückt werden können (siehe [42, S. 88-91]). Ein Beispiel, wie RDF-Listen durch OWL-Konstrukte verwendet werden, ist in Auflistung A.7 auf Seite 120 in den Zeilen 14-18 zu sehen.

### RDF-Serialisierungen

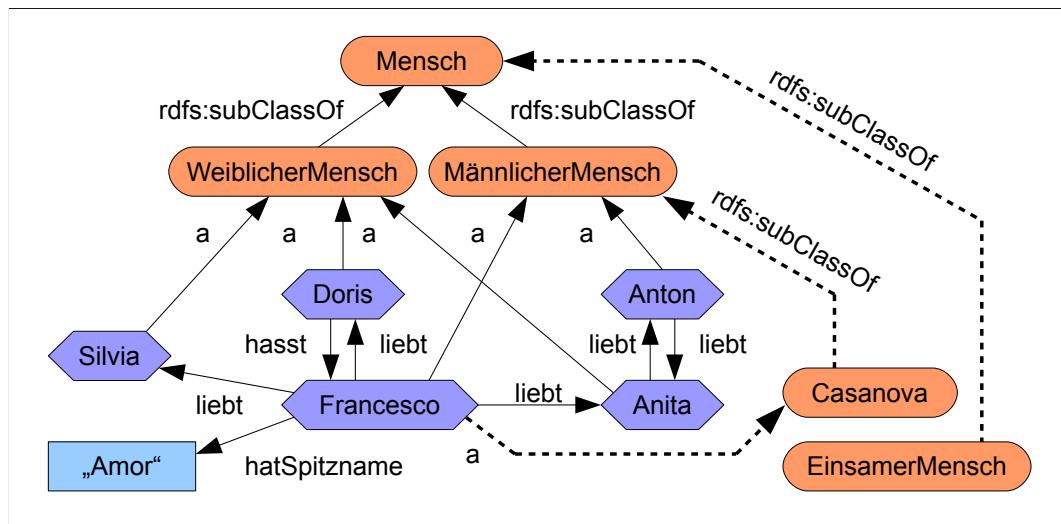
Zum elektronischen Austausch von RDF-Graphen haben sich einige RDF-Serialisierungen durchgesetzt. Eine RDF-Serialisierung beschreibt das Format, in dem der Graph abgespeichert wird. Zu nennen wären hier RDF/XML, die Terse RDF Triple Language (Turtle) und N-Triples. Die einzelnen Formate besitzen unterschiedliche Vor- und Nachteile, die hier jedoch nicht vorgestellt werden können. Anhang A.1 auf Seite 117 illustriert diese Formate anhand mehrerer Beispiele.

#### 2.3.4 RDF Schema (RDFS)

Abbildung 2.12 führt weitere Personen in die Beispiel-Ontologie ein: *Silvia*, *Anita* und *Anton*. (Die anderen Elemente in Abbildung 2.12 werden später erklärt.) Betrachtet man



**Abbildung 2.11:** Mit Blank-Nodes können n-äre Beziehungen besser ausgedrückt werden. Da die Ressourcen „Zutat1“ und „Zutat2“ (links) wenig aussagen, sondern nur dazu dienen, die Zutat mit einer Mengenangabe zu verknüpfen, kann man sie durch Blank-Nodes (rechts) ersetzen. Der Unterstrich als Prefix ist hierfür ein spezieller Platzhalter aus der Turtle-Notation (siehe Text).



**Abbildung 2.12:** Eine Ontologie zur Illustration von Klassen (abgerundet, orange), Instanzen (sechseckig, violett), Literalen (rechteckig, blau) und Properties in RDFS und OWL. Die gestrichelten Elemente sind durch OWL-Inferenz entstanden (siehe Text). Die Property `a` ist eine Abkürzung aus der Turtle-Syntax und steht für `rdf:type`.

als Mensch den RDF-Graph, erkennt man sofort, dass hier Beziehungen zwischen einigen Frauen und Männern modelliert werden. Die Beziehungen werden durch RDF-Properties ausgedrückt, die einzelnen Personen sind durch die RDF-Ressourcen modelliert.

RDF alleine ermöglicht es jedoch nicht, die Bedeutungen dieser Properties oder die Beziehungen zwischen diesen Properties und anderen Ressourcen zu beschreiben. Für einen Computer sind die bisherigen Bezeichner der Ressourcen und Properties lediglich einfache Zeichenketten. Die oben beschriebenen, für Menschen ersichtlichen Hintergrundinformationen sind daher für den Computer nicht verarbeitbar. Dies soll mit der Hilfe von *RDF Schema (RDFS)* [14] möglich werden, das solche Hintergrundinformationen – sog. terminologisches Wissen bzw. Schemawissen – über die verwendeten Bezeichner definiert [44, S. 67]. RDFS definiert Klassen und Properties, mit denen wiederum Klassen, Properties und andere Ressourcen beschrieben werden können.

Mit RDFS können Klassen und Properties durch Generalisierung und Spezialisierung in Hierarchien arrangiert werden. Beispielsweise wurden in Abbildung 2.12 die Klassen `WeiblicherMensch` und `MännlicherMensch` als Spezialisierung der Klasse `Mensch` angelegt, denen dann die einzelnen Personen als Individuen (bzw. Mitglieder, Instanzen) über die Property `rdf:type` (Abgekürzt gemäß Turtle-Syntax mit `a`) zugewiesen wurden<sup>1</sup>. Damit sind alle gezeigten Individuen gleichzeitig auch Menschen.

<sup>1</sup> Zwar existiert mit der Property `rdf:type` eine Typisierung in RDF; allerdings kann mit RDF nicht explizit ausgedrückt werden, dass eine Ressource eine Klasse ist.

RDFS ist ein sog. RDF-Vokabular, das RDF semantisch erweitert. Dieses Vokabular kann innerhalb eines RDF-Dokuments verwendet werden, indem die RDFS-Konstrukte aus dem RDFS-Namespace <http://www.w3.org/2000/01/rdf-schema#> (üblicherweise abgekürzt mit dem Prefix `rdfs:`) als RDF-Ressourcen benutzt werden:

```
WeiblicherMensch rdfs:subClassOf Mensch .
```

Die semantisch korrekte Interpretation und Weiterverarbeitung dieser RDFS-Konstrukte obliegt dann einer RDFS-fähigen *Inferenzmaschine* (auch *Reasoner* genannt). Dies ist eine Software, die aus den explizit in der Ontologie erfassten Statements in Verbindung mit der RDFS-Semantik implizites, „neues“ Wissen ableitet. Beispielsweise kann geschlussfolgert werden, dass **Doris** ein **Mensch** ist, weil sie ein **WeiblicherMensch** ist, und **WeiblicherMensch** eine Unterklasse von **Mensch** ist.

Zusätzlich können einzelnen Properties *Domain* und *Range* zugewiesen werden: Definiert man zum Beispiel für eine Property `hatMädchenName` die Domain **WeiblicherMensch**, dann sind per Definition alle Individuen, die eine Property `hatMädchenName` besitzen, automatisch Mitglieder der Klasse **WeiblicherMensch**. Aus dem Statement **Ingrid** `hatMädchenName "Wolf"` folgt, dass das Individuum **Ingrid** ein Mitglied der Klasse **WeiblicherMensch** ist.

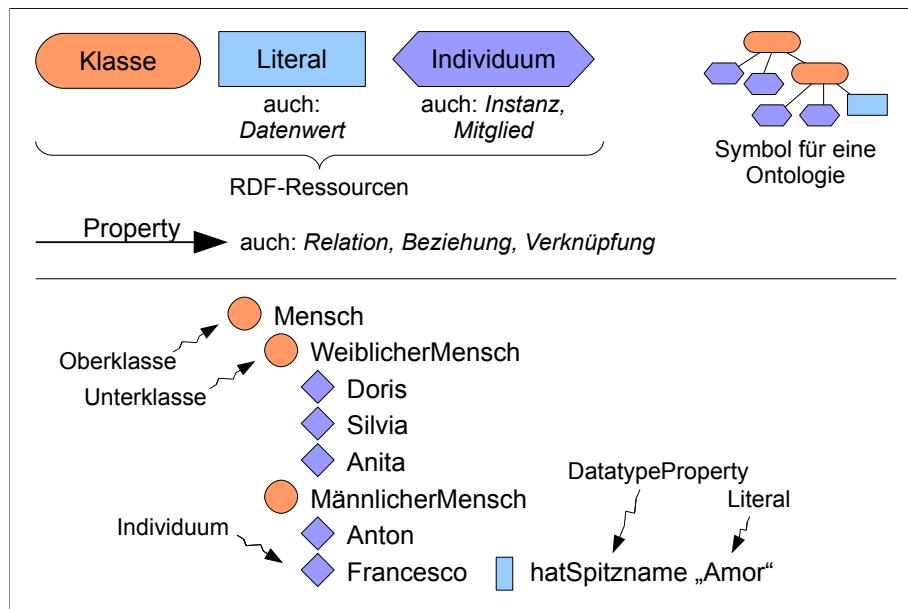
### 2.3.5 Die Web Ontology Language (OWL)

Die *Web Ontology Language (OWL)* [87] ist eine auf der Prädikatenlogik der ersten Stufe [44, S. 163] basierende Ontologie-Sprache, deren Semantik weit über die von RDFS hinausgeht.

Wie bei RDFS besteht eine OWL-Ontologie im Wesentlichen aus Klassen und Individuen. Neben den Ober-/Unterklassen-Beziehungen aus RDFS stehen Konstrukte aus der Mengenlehre zur Verfügung: mit `owl:disjointWith` können Klassen als disjunkt, mit `owl:equivalentClass` als gleich deklariert werden. Gleiches gilt für Individuen, die mit `owl:sameAs` als gleich oder mit `owl:differentFrom` als unterschiedlich ausgezeichnet werden können. Disjunktion (Vereinigung, `owl:unionOf`), Konjunktion (Schnitt, `owl:intersectionOf`), Negation (Verneinung, `owl:complementOf`) und *OWL-Restrictions* ermöglichen die Bildung von neuen Klassen aus bestehenden.

Durch Kombination der OWL-Konstrukte sind komplexe Klassendefinitionen möglich. In der Beispielontologie könnte man beispielsweise eine Klasse **Casanova** wie folgt definieren:

```
MännlicherMensch and (liebt min 2 WeiblicherMensch)
```



**Abbildung 2.13:** Die in dieser Arbeit verwendeten Symbole (oben), ebenso die vereinfachte (und übersichtlichere) Darstellung von Beziehungen zwischen Ober- und Unterklassen sowie Individuen (unten). Die gezeigte Ontologie entspricht vollständig Abbildung 2.12 auf Seite 22 (außer den Klassen **Casanova** und **EinsamerMensch** sowie die durch Inferenz gewonnenen „gestrichelten“ Properties).

Dies bedeutet: Alle Mitglieder der Klasse **Casanova** müssen Mitglieder der Klasse **MännlicherMensch** sein und mindestens zwei **liebt**-Properties zu Mitgliedern der Klasse **WeiblicherMensch** besitzen<sup>1</sup>.

Wie RDFS ist auch OWL lediglich ein RDF-Vokabular, durch dessen Einbindung in ein RDF-Dokument weitere Konstrukte zur Formulierung von semantischen Inhalten bereitgestellt werden. Auch hier ist die korrekte semantische Verarbeitung die Aufgabe einer OWL-fähigen Inferenzmaschine.

Wird die gezeigte Ontologie (einschließlich der Klassendefinition für **Casanova**) einer solchen Inferenzmaschine übergeben, so kann aus dem bisher erfassten, expliziten Wissen implizites Wissen abgeleitet werden, beispielsweise, dass Francesco ein Casanova ist. Nur dieses Individuum erfüllt die oben genannten Eigenschaften der **Casanova**-Klasse. Diese Klassenzugehörigkeit ist in Abbildung 2.12 auf Seite 22 durch den gestrichelten Pfeil

1 Die gezeigte Klassendefinition ist in Manchester-Syntax (Protégé 4) notiert, einer Kurzschreibweise. Anhang A.2 ab Seite 118 zerlegt diese Klassendefinition in RDF-Tripel und zeigt, dass hier die oben genannten OWL-Konstrukte (`owl:equivalentClass`, `owl:intersectionOf` sowie eine OWL-Restriction in der zweiten Bedingung) Verwendung finden.

von **Francesco** zu **Casanova** dargestellt. Außerdem wurde abgeleitet, dass die **Casanova**-Klasse eine Unterklasse von **MännlicherMensch** ist. Die Inferenzmaschine hat bei dem Vorgang neue Statements in die Ontologie eingefügt.

Das gezeigte Beispiel verwendet bereits viele Konstrukte des OWL-Sprachumfangs. Für eine umfassende Einführung in die Modellierung von Ontologien mit RDFS und OWL sei [1] empfohlen. Anhang A.3 ab Seite 121 listet die wichtigsten OWL-Konstrukte auf.

### 2.3.6 Die Open-World-Assumption

Wenn man nun **wirdGeliebtVon** als inverse Property von **liebt** sowie eine Klasse **EinsamerMensch**:

```
Mensch and (wirdGeliebtVon exactly 0 Mensch)
```

definiert, würde man erwarten, dass **Francesco** auch als **EinsamerMensch** erkannt wird (tatsächlich wird Francesco von niemanden in der Ontologie geliebt). Das passiert jedoch aufgrund der sog. *Open-World-Assumption* nicht: Nur, weil es nicht in der Ontologie steht, bedeutet das nicht, dass er nicht doch von jemanden geliebt wird. Gemäß AAA-Slogan könnte diese Aussage schließlich irgendwo anders gemacht werden. Es kann lediglich geschlussfolgert werden, dass die Klasse **EinsamerMensch** eine Unterklasse von **Mensch** ist – dies wurde in der Klassendefinition sogar so definiert.

Die Open-World-Assumption steht damit im Gegensatz zur *Closed-Word-Assumption*, die üblicherweise in herkömmlichen Datenbanken Anwendung findet: steht eine Person nicht in einer Kunden-Kartei, kann man davon ausgehen, dass sie kein Kunde bei dieser Firma ist.

### 2.3.7 Die SPARQL Protocol And RDF Query Language

Mit der *SPARQL Protocol And RDF Query Language* (SPARQL) [79] steht eine SQL-artige Abfragesprache für RDF-Graphen zur Verfügung. Statt SPARQL im Detail vorzustellen (siehe hierzu [42, S. 192ff] oder [44, S. 202ff]), soll die Funktionsweise anhand eines sehr einfachen Beispiels illustriert werden:

```
1   SELECT ?PersonA ?PersonB ?PersonC
2   WHERE {?PersonB liebt ?PersonA .
3           ?PersonC liebt ?PersonA . }
```

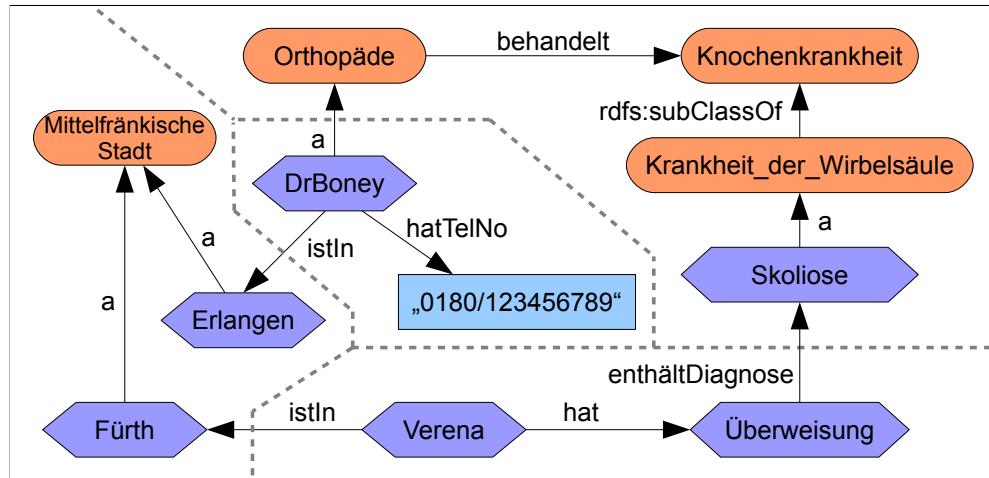
Die Query liefert eine Tabelle mit den drei Spalten für die Variablen `?PersonA`, `?PersonB` und `?PersonC` zurück, die die Statements in den Zeilen 2 und 3 erfüllen müssen. Die Query liefert also alle Personen (`?PersonA`) zurück, die von mindestens zwei anderen Personen (`?PersonB` und `?PersonC`) geliebt werden. Angewendet auf die Beispiel-Ontologie ist dies der folgende Eintrag:

<code>?PersonA</code>	<code>?PersonB</code>	<code>?PersonC</code>
Anita	Francesco	Anton

### 2.3.8 Die Semantic Web Rule Language (SWRL) und OWL 2

Der Vollständigkeit halber sei an dieser Stelle noch die *Semantic Web Rule Language (SWRL)* [45] genannt. Sie wurde entwickelt, um Semantik abzubilden, die über den Sprachumfang von OWL hinausgeht, in der Praxis aber oft benötigt wird. Eine gute Einführung in SWRL findet sich in [42, S. 231-261]. OWL wurde inzwischen weiterentwickelt und liegt in Version 2 vor [99]. Hier wurde versucht, Defizite im Sprachumfang von OWL 1 auszubessern.

Mit den bisher gezeigten Techniken soll es im Semantic Web möglich werden, Informationen global zu verknüpfen und so bereitzustellen, dass sie bei Bedarf durch Maschinen verarbeitet werden können. Abbildung 2.14 auf Seite 26 soll abschließend den Kreis zu TJ Berners-Lees Vision wieder schließen: durch die Bereitstellung und Verknüpfung aller



**Abbildung 2.14:** Informationen aus unterschiedlichen Domänen werden miteinander verknüpft. Links: Ortsinformationen, mittig: Informationen über Dr. Boney, unten: Informationen über die Patientin, rechts oben: medizinisches Wissen.

relevanter Informationen in Form von semantischen Netzen wird es zukünftig vielleicht doch möglich, dass das „schlaue“ Gerät der Patientin automatisch einen Termin bei Dr. Boney vereinbart.

## 2.4 Werkzeuge

Im Zusammenhang mit dem Semantic Web wurden für viele Programmiersprachen Open-Source-Frameworks entwickelt. Das *Jena-Framework*<sup>1</sup>, *Sesame-Framework*<sup>2</sup> und die *OWL-API*<sup>3</sup> sind die bekanntesten Implementierungen zur Manipulation von semantischen Netzen mit Java, aber auch für andere Sprachen wie Python, PHP oder .NET existieren Lösungen (siehe [42, S. 154f]).

Um die Mächtigkeit der OWL-Inferenz in den eigenen Programmen verwenden zu können, können Open-Source-Reasoner in die eigenen Programme eingebunden werden. Es wäre an dieser Stelle sinnlos, alle aufzuzählen (siehe hierfür [42, S. 146f]); genannt werden muss jedoch *Pellet*<sup>4</sup>, der im Rahmen dieser Arbeit an einigen Stellen versuchsweise zum Einsatz kam.

Für diese Arbeit wurde das Jena-Framework ausgewählt, da dieses vollständig OWL unterstützt. Darüber hinaus bietet Jena Schnittstellen für SPARQL-Queries und zur Integration externer Reasoner an. Zusätzlich werden verschiedene Möglichkeiten der Speicherung von Ontologien zur Verfügung gestellt.

Auch zur graphischen Manipulation von OWL-Ontologien existieren kommerzielle und freie Anwendungen. Im Rahmen dieser Arbeit kommt der Open-Source-OWL-Editor *Protégé*<sup>5</sup> in den Versionen 3.4.4 und 4.1 zum Einsatz. Abbildung 2.15 auf Seite 28 zeigt einen Ausschnitt der Beispiel-Ontologie in Protégé 4.1.

Aufgrund der Entscheidung, das Jena-Framework zu verwenden, wurde auch Java als Programmiersprache festgelegt. Als Entwicklungsumgebung für sämtliche Ontologie-Tools dient *Netbeans*<sup>6</sup> 6.9.1, da dieses einen Editor zur schnellen Erstellung von graphischen Benutzeroberflächen auf der Basis von Swing mit sich bringt<sup>7</sup>. Für die Arbeiten am i2b2-Vermittlungsportal kommt dagegen *Eclipse Galileo*<sup>8</sup> zum Einsatz, da die i2b2-Workbench auf der *Eclipse Rich Client Platform*<sup>9</sup> aufbaut.

---

1 <http://jena.sourceforge.net>, Abruf: 19.12.2010

2 <http://www.openrdf.org>, Abruf: 19.12.2010

3 <http://owlapi.sourceforge.net>, Abruf: 19.12.2010

4 <http://pellet.owlowl.com>, Abruf: 19.12.2010

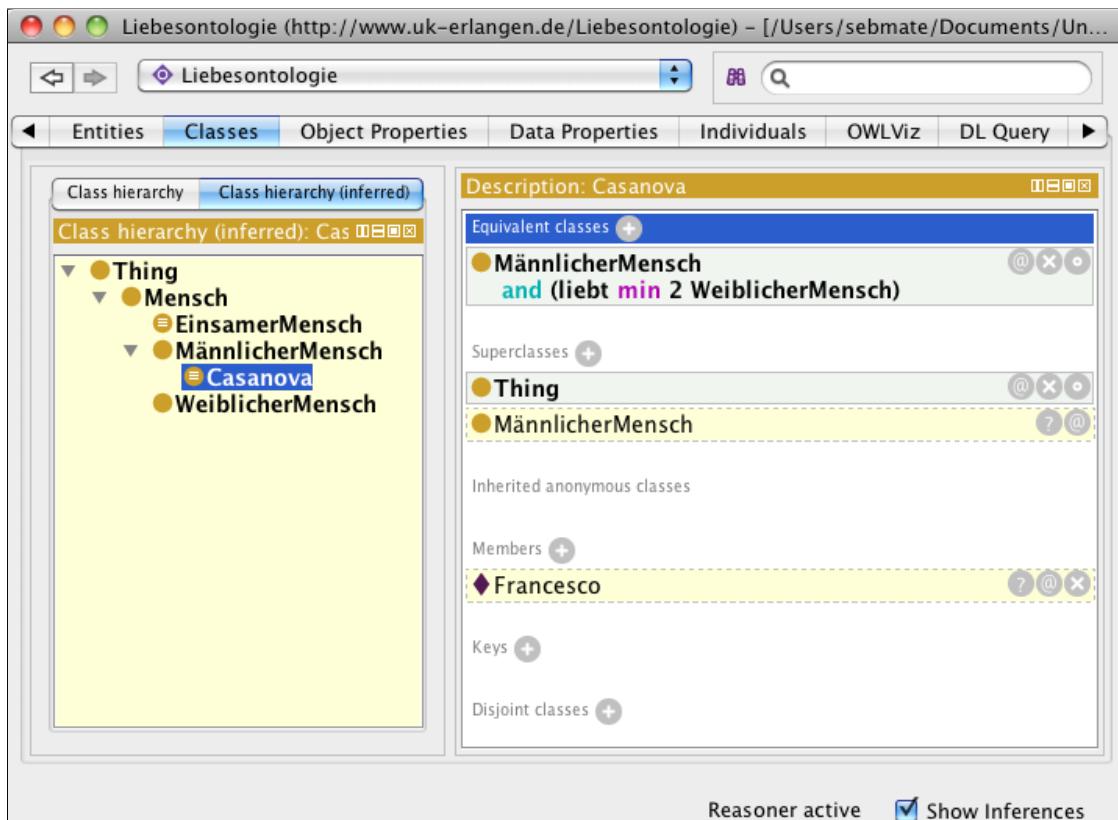
5 <http://protege.stanford.edu>, Abruf: 19.12.2010

6 <http://netbeans.org>, Abruf: 19.12.2010

7 <http://netbeans.org/features/java/swing.html>, Abruf: 19.12.2010

8 <http://www.eclipse.org/galileo>, Abruf: 19.12.2010

9 <http://www.eclipse.org/home/categories/rcp.php>, Abruf: 19.12.2010



**Abbildung 2.15:** Die Liebesontologie in Protégé 4.1 nach der Verarbeitung durch einen Reasoner. Das dadurch gewonnene „neue“ Wissen ist rechts durch die gestrichelt eingerahmten Elemente **MännlicherMensch** und **Casanova** gezeigt.

# KAPITEL 3

---

## Methoden

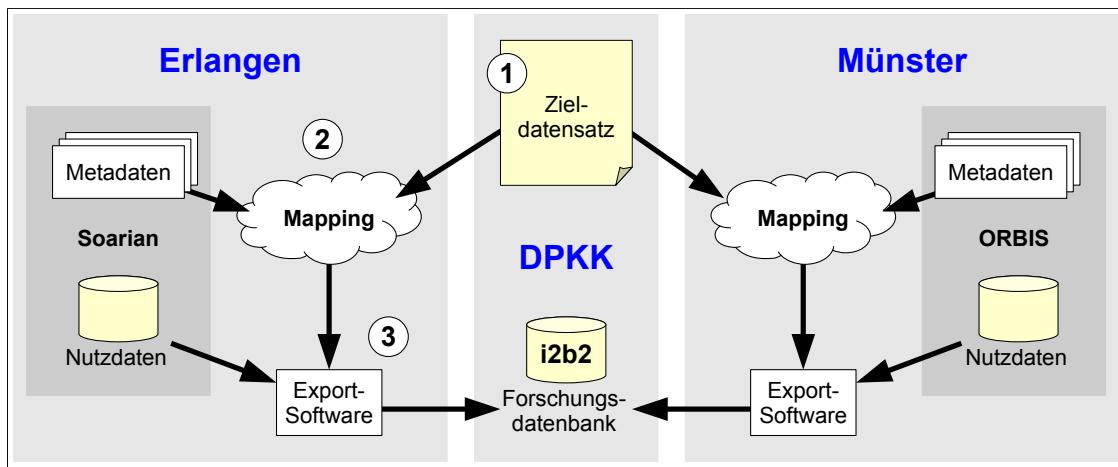
---

### 3.1 Darstellung des Konzepts dieser Arbeit

Im Rahmen dieser Arbeit sollen klinische Routinedaten aus den beiden Quellsystemen Soarian und ORBIS in Erlangen bzw. Münster exportiert und in einer gemeinsamen DPKK-Forschungsdatenbank auf i2b2-Basis zusammengeführt werden. Wenn man die Screenshots der beiden Systeme in den Abbildungen 2.2 und 2.5 auf Seite 10 bzw. 13 betrachtet, erkennt man sofort, dass es sich um völlig unterschiedliche Systeme handelt. Bezuglich der Inhalte kann man zwar gewisse thematische Überschneidungen erwarten – schließlich werden beide zur Prostatakarzinom-Dokumentation verwendet – aufgrund der getrennten Entstehungsgeschichte aber auch große Unterschiede.

Damit eine korrekte Zusammenführung der Daten aus den Quellsystemen möglich wird, müssen diese auf einen gemeinsamen Zieldatensatz gemappt werden, wie es in Abbildung 3.1 auf Seite 30 gezeigt wird. Der Zieldatensatz wird von allen DPKK-Partnern gemeinsam festgelegt und zentral verwaltet (1). Er beschreibt die zwischen den DPKK-Mitgliedern beschlossenen Datenelemente, die in die gemeinsame Forschungsdatenbank aufgenommen werden sollen.

Beim Mapping-Prozess wird der gemeinsame Zieldatensatz mit den vorhandenen Daten in den Quellsystemen verglichen, um Gemeinsamkeiten oder Unterschiede zu identifizieren (2). Datensätze, die der Definition im Zieldatensatz entsprechen, können sofort exportiert werden. Bestehen geringe Abweichungen, können die Daten möglicherweise durch Transformationen in das gewünschte Format gebracht und anschließend auch exportiert werden. Liegen diese Mapping- und Transformationsinformationen in einer maschinell verarbeitbaren Form vor, kann eine Export-Software die Nutzdaten einlesen, sie wie gewünscht transformieren, und in das Zielsystem i2b2 exportieren (3).



**Abbildung 3.1:** Darstellung der konzeptionellen technischen Umsetzung im Rahmen dieser Arbeit.

Die zu exportierenden Daten der klinischen Arbeitsplatzsysteme sind besonders heterogen und entsprechen keinem Kodierungsstandard, mit dessen Hilfe sie sofort exportiert werden könnten. In Kapitel 2.3 ab Seite 14 wurde das Semantic Web vorgestellt und wie man mit verschiedenen Techniken versucht, die nochmals heterogeneren Daten des WWW für eine effiziente maschinelle Verarbeitung semantisch aufzubereiten. Diese Techniken sollen auch im Rahmen dieser Arbeit eingesetzt werden: Statt die einzelnen Datenelemente für das DPKK-Projekt mühselig unter Verwendung generischer ETL-Tools in das gewünschte i2b2-Format zu bringen, soll ein Ansatz entwickelt werden, mit dem die Daten effizient in ein i2b2-System für Auswertungszwecke exportiert werden können. Das aufgebrachte „Mapping-Wissen“ soll dabei nicht in konventionellen SQL- oder Perl-Skripten „verschüttet“ gehen, sondern in Form von semantischen Netzen für weitere Ziele zur Verfügung stehen. Zum Beispiel soll es möglich sein, die gemapten Datensätze in eine größere Krankenhaus-Ontologie einzupflegen. Damit erfüllt der Export nicht nur seinen Selbstzweck – eben den Export der Nutzdaten – er trägt auch im Rahmen der Single-Source-Nutzung zur „semantischen Erschließung“ der immer weiter wachsenden, klinischen Datengräber [36] bei.

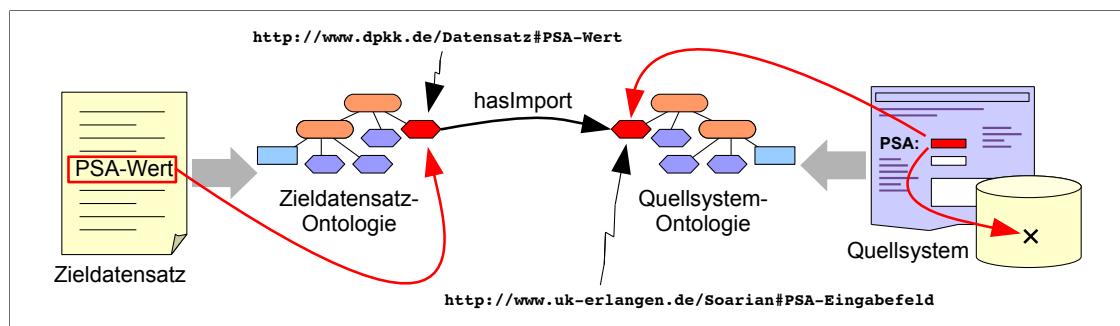
Für den geplanten Semantic-Web-Ansatz müssen alle benötigten Informationen in Form von semantischen Netzen vorliegen, damit man anschließend das oben beschriebene Mapping durch Verknüpfungen mit RDF-Statements ausdrücken kann. Das betrifft insbesondere den gemeinsamen DPKK-Datensatz und die Quellsysteme: Beides wird durch Ontologien dargestellt. Abbildung 3.2 auf Seite 31 symbolisiert die Überführung in Ontologien durch graue Pfeile. Diese „ontologischen Spiegelbilder“ werden im Folgenden mit „Zieldatensatz-Ontologie“ und „Quellsystem-Ontologie“ bezeichnet.

Die Zieldatensatz-Ontologie erfüllt die gleiche Aufgabe wie der Zieldatensatz selbst: sie beschreibt alle Datenelemente, die für die i2b2-Forschungsdatenbank von den DPKK-Mitgliedern beigesteuert werden sollen. Die Quellsystem-Ontologie beschreibt, welche Nutzdaten es im Quellsystem überhaupt gibt und wie man auf diese zugreifen kann. Für jedes Element wird beispielsweise auch erfasst, in welcher Datenbank und in welcher Tabelle die entsprechenden Nutzdatensätze zu finden sind.

Liegt beides als Ontologie vor, können Verknüpfungen zwischen den Datenelementen im Zieldatensatz und den Nutzdaten in den Quellsystemen erstellt werden. Indem man anschließend lediglich die Verbindungen im semantischen Netz verfolgt, kann man von den gedanklichen Konzepten im Zieldatensatz zu den realen Nutzdatensätzen in den Datenbanken gelangen. In Abbildung 3.2 wird beispielsweise das PSA-Datenelement im Zieldatensatz über eine *hasImport*-Beziehung mit den eigentlichen Nutzdaten im Quellsystem, symbolisiert durch das schwarze Kreuzchen, verknüpft.

Abbildung 3.3 auf Seite 32 verbindet diesen Ansatz mit den ursprünglichen Mapping-Überlegungen in Abbildung 3.1 auf Seite 30. Der Zieldatensatz wird weiterhin zentral durch das DPKK verwaltet (1) und gemäß Semantic-Web-Philosophie den teilnehmenden Partnern über das WWW zur Verfügung gestellt. Diese stellen ihre Quellsysteme, wie oben beschrieben, durch Ontologien dar (2). Das eigentliche Mapping wird in Form von Verknüpfungen zwischen der Quellsystem-Ontologie und der Zieldatensatz-Ontologie in Mapping-Ontologien (3) erfasst – schließlich müssen die Verknüpfungen zwischen den beiden Ontologien irgendwo gespeichert werden. Das Erzeugen der Verknüpfungen ist einfach, da man die beiden Ontologien in die Mapping-Ontologie importieren kann und die Verknüpfungen durch einfache RDF-Statements ausdrücken kann. Die weiter oben beschriebene *hasImport*-Beziehung würde man also als solches RDF-Statement in der Mapping-Ontologie anlegen.

Sobald das Mapping erfolgt ist, kann eine Export-Software die Mapping-Ontologie einlesen, die in ihr gespeicherten Informationen abarbeiten und auf die Nutzdaten im Quellsystem



**Abbildung 3.2:** Zur Begriffsklärung von „Zieldatensatz-Ontologie“ und „Quellsystem-Ontologie“.

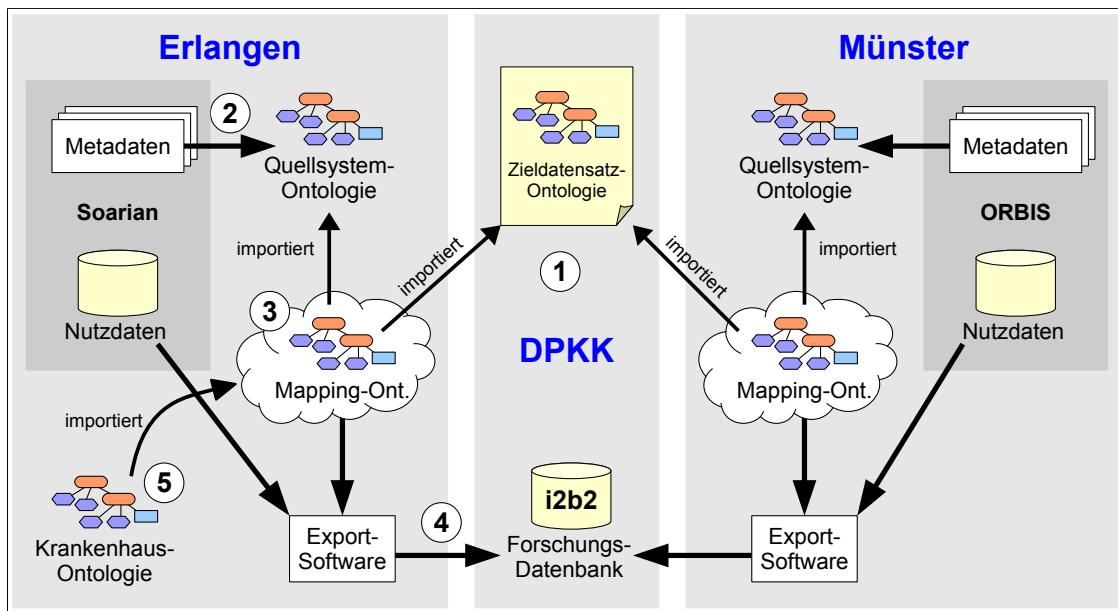


Abbildung 3.3: Umsetzung des Konzepts mittels Ontologien.

anwenden. Dies ist möglich, da man von den Konzepten im Zieldatensatz bis hin zu den Nutzdaten in den Datenbanken gelangt. Beim Export durch die Software werden die Rohdaten aus den Quellsystemen geladen und in das Zielsystem i2b2 geschrieben (4). Da das komplette „Mapping-Wissen“ nun in der Form von RDF-Statements vorliegt, kann es anschließend in eine Krankenhaus-Ontologie eingebbracht werden (5). In zukünftigen Projekten soll auf dieses Wissen wieder zugegriffen werden können.

Die folgenden Abschnitte 3.2 bis 3.5 erklären die in Abbildung 3.3 gezeigten Schritte (1) bis (4) im Detail.

### 3.2 Beschreibung von Zieldatensatz-Ontologien

Der Zieldatensatz beschreibt die Datenelemente, die zwischen den DPKK-Mitgliedern vereinbart wurden und die in die gemeinsame Forschungsdatenbank exportiert werden sollen. Die Datenelemente werden in ihm sowohl semantisch (aus medizinischer Sicht), als auch syntaktisch (das Datenformat betreffend) spezifiziert. Beispielsweise muss genau definiert werden, wie ein PSA-Wert zu messen ist, oder wie die Ausprägung „T3“ der T-Achse des pathologischen TNMs vorliegen soll: wird das Datenelement mit „pT3“, „T3“ oder nur mit einer „3“ abgespeichert?<sup>1</sup>

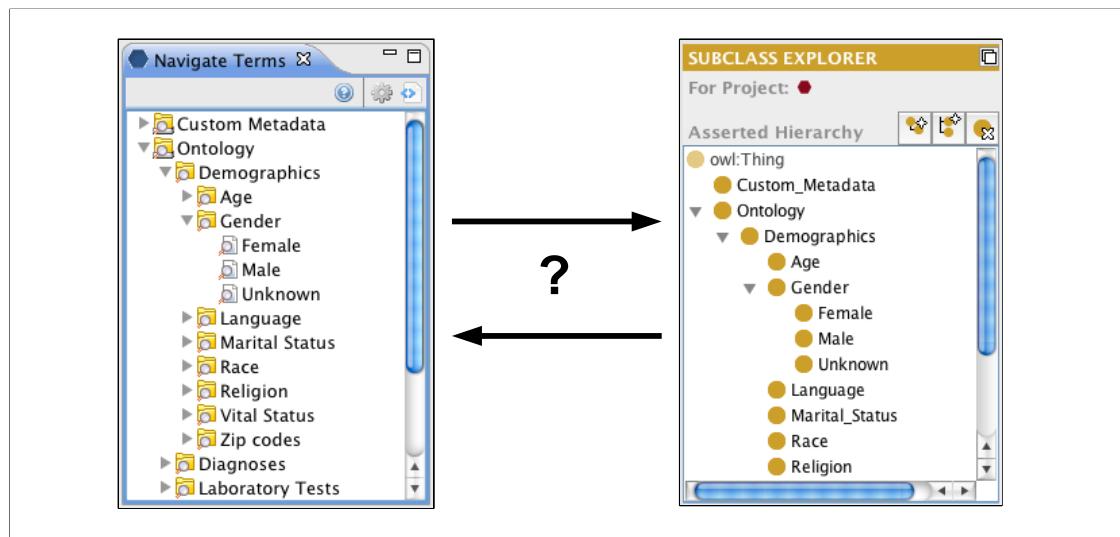
1 Eine einfache „3“ ist vollkommen ausreichend, solange eine Zuordnung dieser Ausprägung zu ihrem Attribut „pTNM-T“ existiert.

Im Rahmen des geplanten Semantic-Web-Ansatzes muss der Zieldatensatz als semantisches Netz vorliegen. Wenn die einzelnen Datenelemente als RDF-Ressourcen dargestellt sind, können Mapping-Statements zu den RDF-Ressourcen in der Quellsystem-Ontologie erzeugt werden (vgl. Schritt (3) in Abbildung 3.3 auf Seite 32).

Da das Hauptziel in einem Daten-Export nach i2b2 besteht, ergeben sich an die Zieldatensatz-Ontologie verschiedene Anforderungen.

Sollen mit i2b2 klinische Daten für Auswertungszecke bereitgestellt werden, müssen in i2b2 Metadaten angelegt werden. i2b2 stellt mit seiner ONT-Zelle eine eigene Metadaten-Verwaltung zur Verfügung, die von den i2b2-Entwicklern auch als i2b2-Ontologie bezeichnet wird. Aus den vorhergehenden Arbeiten in Erlangen an i2b2 ist bekannt, dass in der i2b2-Ontologie alle medizinischen Konzepte in einer einfachen Taxonomie abgelegt werden (siehe [60, S. 24]). Derartige hierarchische Strukturen kann man leicht mit den RDFS- und OWL-Konstrukten erzeugen (siehe Abschnitt 2.3.4 ab Seite 21); darüber hinaus steht mit Protégé ein leistungsfähiger OWL-Editor zur Verfügung. Der Gedanke liegt nahe, die Datenelemente des Zieldatensatzes in einer OWL-Klassenhierarchie anzulegen, die man später automatisch in das i2b2-Metadaten-Format überführt (siehe Abbildung 3.4).

Für die Konzepte in der i2b2-Ontologie können noch weitere Zusatzinformationen, wie z. B. der Datentyp und bei Laborwerten die Einheit angegeben werden. Bei Suchanfragen in i2b2 kann man anschließend mit Vergleichsoperatoren numerische Werte einschränken (z. B. der PSA Wert muss  $> 0,2$  sein) und automatisch Einheiten umrechnen lassen. Erfasst man derartige Zusatzinformationen in der Zieldatensatz-Ontologie, können sie bei der Überführung in das i2b2-Metadaten-Format automatisch mit übernommen werden.



**Abbildung 3.4:** Die i2b2-Metadatenverwaltung (links) und der Protégé-Klassen-Editor (rechts).

Da der vollständige Funktionsumfang der i2b2-Ontologie in den Erlanger i2b2-Projekten bisher noch nicht komplett ausgenutzt wurde, muss dieser zunächst genauer untersucht werden. Erst dann können genauere Überlegungen angestellt werden, wie man die einzelnen i2b2-Funktionen in einer Zieldatensatz-Ontologie abbildet. Auf jeden Fall sollte sie die vollständige i2b2-Semantik abbilden. Das Ergebnis hierzu wird später in Abschnitt 4.3 auf Seite 65 gezeigt.

### 3.3 Beschreibung von Quellsystem-Ontologien

Die Quellsystem-Ontologie beschreibt, welche Nutzdaten in den Quellsystemen vorliegen und wie auf diese zugegriffen werden kann. Sie erfüllt damit grundsätzlich die gleichen Aufgaben wie die Metadaten des jeweiligen Quellsystems; allerdings ermöglicht sie es als semantisches Netz, dass die einzelnen Elemente später mit denen des Zieldatensatzes verknüpft werden können. Im Idealfall lassen sich die Quellsystem-Metadaten direkt in eine Ontologie umwandeln (vgl. Schritt (2) in Abbildung 3.3 auf Seite 32).

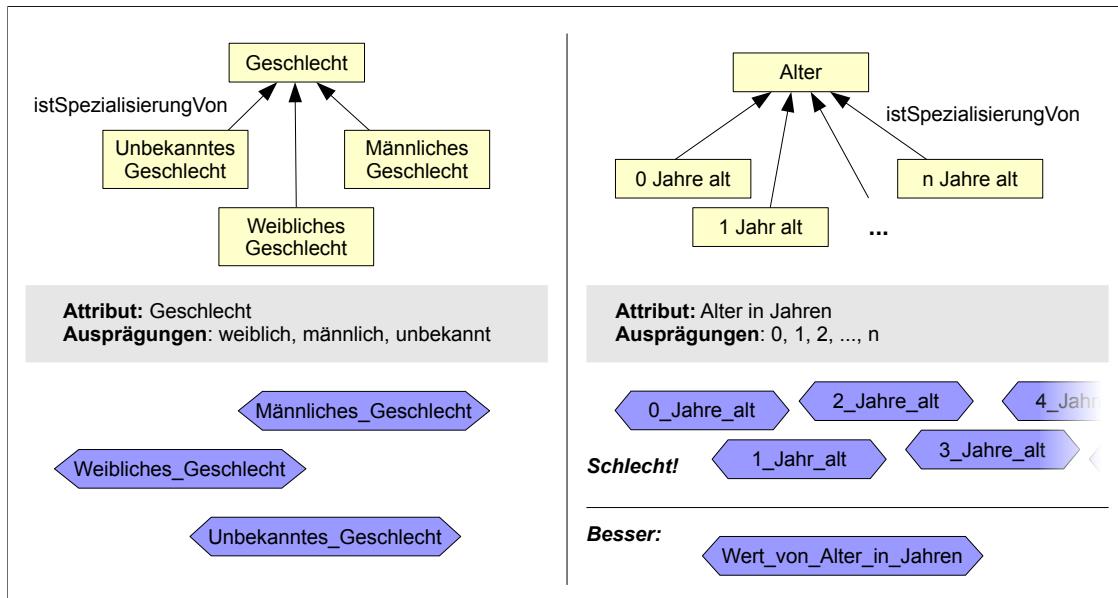
Auch an die Quellsystem-Ontologie ergeben sich verschiedene Anforderungen, zu denen im Folgenden mehrere Überlegungen gemacht werden.

#### 3.3.1 Granularität der Quellsystem-Ontologie

Um mit dem Quell-Datensatz möglichst effizient und differenziert auf die durch ihn beschriebenen Nutzdaten zugreifen zu können, müssen die Konzepte, die er beschreibt, inhaltlich so weit wie möglich zerlegt werden. Die einzelnen Fragmente können dann als RDF-Knoten in der Ontologie angelegt werden.

Die Zerlegbarkeit der medizinischen Konzepte ist jedoch stark an die Art und Weise gekoppelt, mit der die Daten in den Quellsystemen dokumentiert werden. Wenn man klinische EDC-Systeme und die damit erfassten Daten betrachtet, stellt man fest, dass medizinische Konzepte immer durch eine Kombination aus einer Ausprägung mit mehreren Attributen dargestellt werden.

Abbildung 3.5 auf Seite 35 zeigt dies anhand von zwei Beispielen. Das medizinische Konzept „Geschlecht“, das sich durch die Konzepte „Weibliches Geschlecht“, „Männliches Geschlecht“ und „Unbekanntes Geschlecht“ differenzieren lässt, würde man beispielsweise mit einem Attribut „Geschlecht“ und den Ausprägungen „weiblich“, „männlich“ und „unbekannt“ darstellen. Ähnlich kann man das Konzept „Alter“ in die Konzepte „0 Jahre alt“, „1 Jahr alt“, usw. spezialisieren. Das Attribut ist dann das „Alter in Jahren“, die Menge der Ganzzahlen repräsentiert die möglichen Ausprägungen.



**Abbildung 3.5:** Medizinische Konzepte lassen sich in unterschiedlichen Granularitätsstufen spezialisieren (oben). In medizinischen Informationssystemen werden diese durch Attribut-Ausprägungs-Kombinationen dargestellt (mittig). Unten: für die Ontologie muss abgewogen werden, wie granular Ausprägungen dargestellt werden.

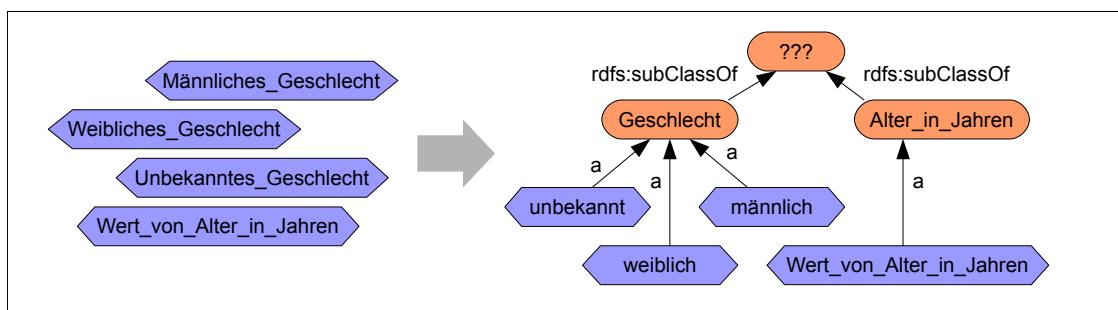
Im ersten Fall kann man die drei Ausprägungen direkt in der Ontologie als RDF-Knoten anlegen (siehe Abbildung 3.5 links unten). Damit ist die optimale Granularität erreicht, da man die einzelnen Konzepte nicht weiter zerlegen kann. Beim zweiten Fall erscheint es jedoch kaum sinnvoll, alle möglichen Ausprägungen in der Ontologie anzulegen: *Wie viele Knoten sollen angelegt werden? Wie alt kann ein Mensch maximal werden?* Hier würde man in der Ontologie nur einen Knoten anlegen, der alle möglichen Alters-Werte repräsentiert. Man könnte diesen Knoten dann „Wert von Alter in Jahren“ nennen (siehe Abbildung 3.5 rechts unten). Wenn man anschließend beim Datenexport nach i2b2 (oder in ein anderes System) auch die Information überträgt, dass es sich um einen numerischen Wert handelt, können die Datensätze trotzdem korrekt ausgewertet werden.

Im ersten Beispiel repräsentieren die Ontologie-Elemente also einzelne Ausprägungen, während sie im zweiten Beispiel eher die Menge aller möglichen Ausprägungen bzw. das Attribut selbst beschreiben. Da hier die Grenze zwischen Attribut und Ausprägung gewissermaßen verschwimmt, werden die nicht weiter zerteilbaren Datenelemente im Folgenden als *Dokumentationselemente* bezeichnet. Von ihnen gibt es zwei Typen:

- **Typ-1-Dokumentationselemente** sind die aufzählbaren Ausprägungen ihrer Attribute. Die Anzahl der Ausprägungen ist damit auf eine kleine Menge beschränkt und nicht weiter zerlegbar. Alle Ausprägungen können als Elemente in der Ontologie angelegt werden. Bei den Ausprägungen „männlich“, „weiblich“ und „unbekannt“ aus dem Beispiel oben handelt es sich um solche Typ-1-Dokumentationselemente.
- **Typ-2-Dokumentationselemente** sind Attribute, deren Ausprägungen eine beliebige Zeichenkette, einen Zahlenwert oder eine Datumsangabe annehmen können. Die Menge der möglichen Ausprägungen ist entweder nicht beschränkt und damit nicht aufzählbar (Zeichenketten ohne Längenbeschränkung, Gleitkommazahlen) oder sehr groß (Zeichenketten mit Längenbeschränkung, Ganzzahlen mit oberer und unterer Schranke). Es ist daher entweder nicht möglich oder nicht sinnvoll, alle möglichen Ausprägungen als Elemente in der Ontologie anzulegen. Das oben genannte Konzept „Alter“ würde man also nicht weiter zerlegen, sondern durch einen einzelnen Knoten anlegen.

### 3.3.2 Strukturierung der Elemente innerhalb der Quellsystem-Ontologie

Bisher wurden nur die Dokumentationselemente ohne weitere Strukturierung in die Ontologie aufgenommen. Um eine gewisse Navigation innerhalb der Ontologie zu ermöglichen, müssen die einzelnen Elemente gruppiert werden (vgl. Abbildung 3.6). Im Folgenden wird eine einfache Methode vorgestellt, mit der eine hierarchische Struktur zur Beschreibung des Quellsystems gewonnen werden kann.



**Abbildung 3.6:** Durch Zuordnung der Dokumentationselemente zu Klassen soll eine Hierarchie aufgebaut werden.

### Strukturgewinnung aus tabellarischen Datenquellen

Die meisten klinischen Informationssysteme ermöglichen einen tabellarischen Export (als CSV- oder Excel-Datei) in „spaltenorientierter“ Form, wie er in Tabelle 3.1 gezeigt wird. Dabei wird für jeden Patienten  $P_1, P_2, \dots, P_m$  eine neue Zeile angelegt. Die Spaltenüberschriften  $A_1, A_2, \dots, A_n$  benennen das Attribut; die möglichen Ausprägungen finden sich in den Zellen darunter. Die Ausprägung für Patient  $P_i$  und Attribut  $A_k$  ist in Zelle  $x_{i,k}$  erfasst.

Das Beispiel in Tabelle 3.2 illustriert dies mit Datenwerten. Man sieht, dass für mehrere Patienten (Spalte „PatNr“ und „FallNr“) zu einem Zeitpunkt (Spalte „Datum“) mehrere Attribute (*Geschlecht*, *Alter*, *Wert 1 bis 3*) erfasst wurden. Die Ausprägungen *Alter* und *Wert 1 bis 3* sind vom Typ 2, da der Wertebereich dieser numerischen Werte unbekannt ist. Das Attribut *Geschlecht* ist dagegen dem Typ 1 zuzuordnen, da es nur vier aufzählbare Ausprägungen zu diesem Attribut gibt.

Die Spalte „Datum“ beschreibt dabei kein Attribut im eigentlichen Sinn; vielmehr beziehen sich die Datumsangaben – zusammen mit der jeweiligen Patienten-Nummer – auf alle anderen Werte in ihrer Zeile. Darüber hinaus kann es Spalten geben, die sich auf einzelne andere Spalten beziehen. Dies wird im Beispiel an Spalte „Datum Wert 3“, die sich auf die Spalte „Wert 3“ bezieht, ersichtlich. Damit wird für „Wert 3“ die Spalte „Datum“ durch „Datum Wert 3“ überschrieben. Diese Überlegungen werden in Kapitel 4.2 auf Seite 62 weiterentwickelt, da das Zusammenspiel einzelner Attribute beim Datenexport relevant ist. Tabelle 3.3 auf Seite 38 fasst die Erkenntnisse zusammen.

Die Ausprägungen der Typ-1-Spalten können nun aggregiert werden, d. h., es werden doppelte und leere Einträge entfernt. Die übrig gebliebenen Einträge sind Ausprägungen ihres Attributs in der jeweiligen Spaltenüberschrift. Damit ist bereits eine einfache hierarchische Beziehung zwischen Attribut und Ausprägung gemäß Abbildung 3.3.2 auf Seite 38 gewonnen. Für alle Typ-2-Spalten werden die Ausprägungen verworfen und ein Ontologie-Knoten mit dem Prefix „Wert\_von\_“ des Attributs angelegt. Abbildung 3.7 auf Seite 39 zeigt die aus Tabelle 3.2 gewonnene Klassenhierarchie mit Instanzen.

<i>Patient</i>	<i>Zeit</i>	$A_1$	$A_2$	$A_3$	...	$A_n$
$P_1$	$z_1$	$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	...	$x_{n,1}$
$P_2$	$z_2$	$x_{1,2}$	$x_{2,2}$	$x_{3,2}$	...	$x_{n,2}$
...	...	...	...	...	...	...
$P_m$	$z_k$	$x_{1,m}$	$x_{2,m}$	$x_{3,m}$	...	$x_{n,m}$

**Tabelle 3.1:** Dokumentation in „spaltenorientierter“ Form. Nach [74].

Datum	PatNr	FallNr	Geschlecht	Alter	Wert 1	Wert 2	Wert 3	Datum Wert 3
02.02.10	123123	135135	M	50				
23.02.10	234234	246246	0	53				
16.03.10	345345	357357	1	56				
06.04.10	456456	468468	W	59	30	2	1,1	03.12.09
27.04.10	567567	579579	W	62	2	3	1,4	07.09.09
18.05.10	678678	680680	W	65	3	4	1,7	09.08.09
08.06.10	789789	513531	M	68	2	5	-0,0002	11.07.09

**Tabelle 3.2:** Beispiel-Tabelle zur Speicherung von Quelldaten in spaltenorientierter Form.

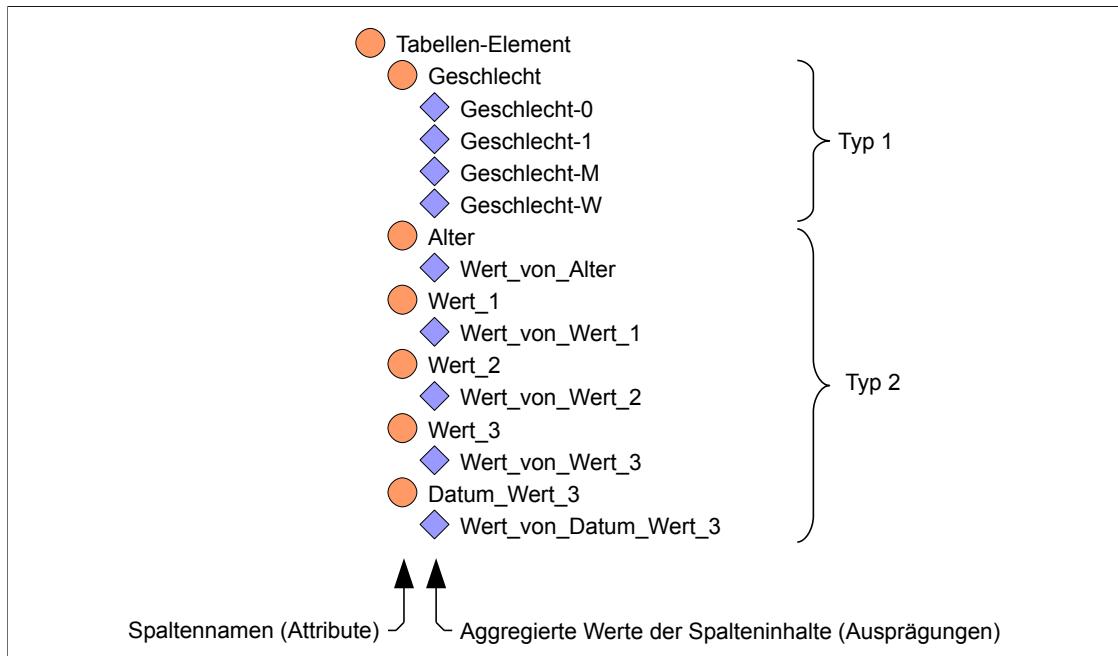
Spalten-Name	Spalten-Typ	Ausprägungen
Datum	allgemeiner Dokumentationszeitpunkt	
PatNr	Patienten-Nummer	
FallNr	Fall-Nummer	
Geschlecht	Typ 1 (Aufzählbar)	0, 1, M, W
Alter	Typ 2 (Ganzzahl)	Wert von „Alter“
Wert 1	Typ 2 (Ganzzahl)	Wert von „Wert 1“
Wert 2	Typ 2 (Ganzzahl)	Wert von „Wert 2“
Wert 3	Typ 2 (Gleitkommazahl)	Wert von „Wert 3“
Datum Wert 3	Datumsangabe für „Wert 3“	Wert von „Datum Wert 3“

**Tabelle 3.3:** Zusammenfassung der Informationen über Tabelle 3.2 auf Seite 38, die für die Erzeugung der Quellsystem-Ontologie benötigt werden.

Beachtet werden muss, dass die gewonnene Quellsystem-Ontologie nicht unbedingt vollständig ist, da nur Ausprägungen aufgenommen werden, die auch in den Nutzdaten vorkommen. Die Erzeugung der Quellsystem-Ontologie aus einer einfachen Tabelle ist eher ein Notbehelf für den Fall, in dem nicht anders als über einen Daten-Export auf die Nutzdaten des Quellsystems zugegriffen werden kann.

### Strukturgewinnung aus Formularbeschreibungen

Im Gegensatz zu einfachen Tabellen, bei denen die nebeneinanderliegenden Spalten nur eine einzige „Hierarchie“-Ebene bilden, lässt sich bei EDC-Systemen aus der Benutzeroberfläche oft eine mehrstufige Hierarchie ableiten. In klinischen Arbeitsplatzsystemen besteht diese zumindest aus Formularen, Attributen und Ausprägungen. Abhängig vom jeweiligen System lassen sich sogar noch weitere Hierarchieebenen gewinnen. Möglicherweise lassen



**Abbildung 3.7:** Darstellung der aus Tabelle 3.2 gewonnenen Metadaten in Form einer Klassen-Hierarchie mit Instanzen. Lediglich für das Attribut „Geschlecht“ konnte eine aufzählbare Menge an Ausprägungen gewonnen werden, da es das einzige Typ-1-Dokumentationselement in diesem Beispiel ist.

sich einzelne Bögen gruppieren, oder es können weitere Hierarchieebenen aus Kapiteln, Überschriften und Attributgruppen erzeugt werden. Ein Beispiel hierfür wurde in [60, S. 47] gezeigt.

Die Vorgehensweise bei der Erzeugung einer Hierarchie soll anhand eines einfachen Beispiels veranschaulicht werden. Abbildung 3.8 auf Seite 40 stellt ein Formular dar, für

Bogen	Attribut	Ausprägung
Therapie-Bogen	Art der Therapie	Regelmäßige Behandlung
Therapie-Bogen	Art der Therapie	Doppelte Strahlendosis
Therapie-Bogen	Ende der Therapie	Reguläres Ende
Therapie-Bogen	Ende der Therapie	Abbruch wegen Komplikationen
Therapie-Bogen	Beendet am	(null)
Therapie-Bogen	Ruhepuls	(null)
Therapie-Bogen	Kommentar	(null)

**Tabelle 3.4:** Tabellarische Darstellung der Benutzeroberfläche in Abbildung 3.8.

das eine äquivalente tabellarische Darstellung der Benutzeroberfläche in 3.4 auf Seite 39 gezeigt ist. An den Tabellenspalten erkennt man, dass die Hierarchie spaltenweise von links nach rechts aufzubauen ist: Jeder Zelle  $z_{i,k}$  ist die rechts neben ihr liegende Zelle  $z_{i+1,k}$  hierarchisch unterzuordnen. Mit dieser Kenntnis lässt sich nun aus allen verschiedenen Paaren  $(z_{i,k}, z_{i+1,k})$  eine Hierarchie aufbauen. Soll das eine RDFS-Klassenhierarchie sein, erfolgt die Erzeugung der Hierarchie über RDF-Statements der Form

$$z_{i+1,k} \text{ rdfs:subClassOf } z_{i,k} .$$

Die Einträge der Spalte „Ausprägung“ werden jedoch gesondert behandelt: Nach Abschnitt 3.3.1 handelt es sich bei den Einträgen dieser Spalte bereits um Dokumentationselemente, über die später ein Zugriff auf die Nutzdaten ermöglicht werden soll. Da diese nicht weiter zerteilt werden können, werden sie als Instanzen ihrer Attribut-RDFS-Klasse angelegt. Abbildung 3.9 auf Seite 41 zeigt die gewonnene Ontologie.

Eine wichtige Voraussetzung zur Erzeugung der Quellsystem-Ontologie besteht natürlich im Zugang zu den Metadaten des Quellsystems. In Erlangen ist diese durch den Soarian-DWH-Export (siehe Abschnitt 2.2.1 auf Seite 9) erfüllt. Aus den Spalten der Tabelle DWH\_METADATEN\_ONTOLOGY (siehe C.1 auf Seite 134) lässt sich nach dem oben beschriebenen Schema leicht eine Hierarchie gewinnen, die aus Bögen, Components und

Therapie-Bogen	
Art der Therapie:	<input checked="" type="checkbox"/> Regelmäßige Behandlung <input checked="" type="checkbox"/> Doppelte Strahlendosis
Ende der Therapie:	<input type="radio"/> Reguläres Ende <input checked="" type="radio"/> Abbruch wegen Komplikationen
Beendet am:	23.03.2010
Ruhepuls:	180
Kommentar:	Der Patient litt unter Angstzuständen. Die Therapie wurde zur Sicherheit abgebrochen.

Attribute      Ausprägungen

Typ 1      Typ 2

**Abbildung 3.8:** Auch bei den Widgets innerhalb der Benutzeroberfläche lassen sich die beiden Typen an Dokumentationselementen identifizieren.

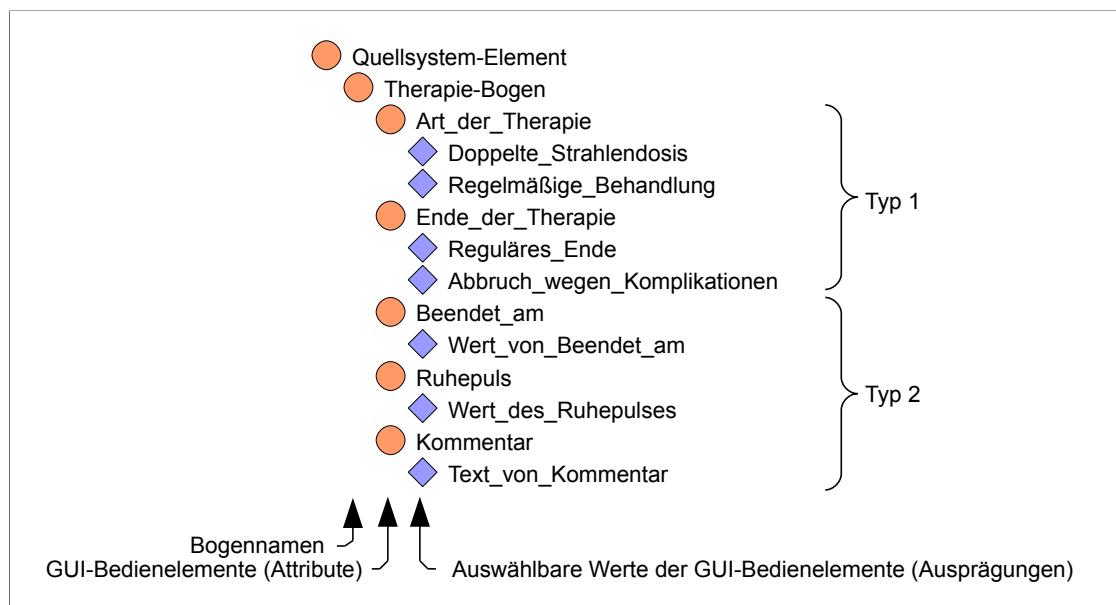
Ausprägungen besteht. In Abschnitt 4.5 auf Seite 71 wird später gezeigt, wie dies durch einfache SQL-Anweisungen erfolgen kann.

### 3.3.3 Zugriff auf die Nutzdaten über die Ontologie

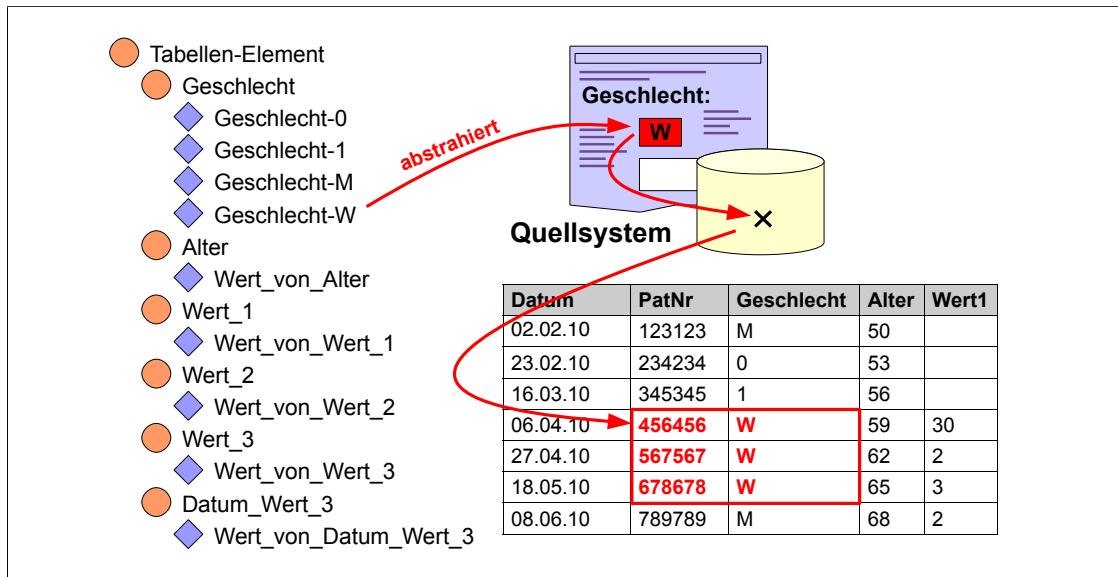
Die oben im Text gewonnenen Hierarchien dienen hauptsächlich dazu, die Dokumentationselemente zu strukturieren, womit eine Navigation innerhalb der Ontologie ermöglicht wird.

Zusätzlich muss die Ontologie weitere Informationen bereitstellen, damit ein Zugriff auf die Nutzdaten in den Quellsystemen möglich wird. Hierzu gehören Angaben über die Datenbankverbindung und die Tabelle, in der die Daten gespeichert sind. Es muss dokumentiert sein, auf welche Spalten in welcher Weise zugegriffen werden muss, um an die Datensätze zu gelangen. Jedes Dokumentationselement ist damit eine Abstraktion seiner Nutzdatensätze in der Datenbank. Beispielsweise symbolisiert das in Abbildung 3.10 auf Seite 42 gezeigte Element „Geschlecht-W“ alle Datensätze, in denen für Patientinnen das Geschlecht „W“ dokumentiert wurde.

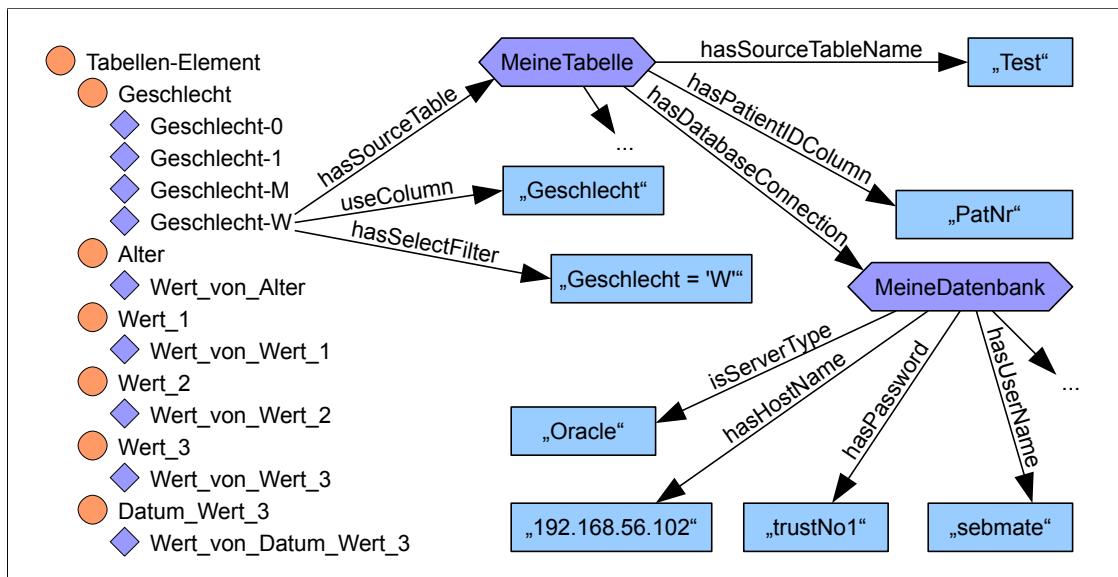
Abbildung 3.11 auf Seite 42 illustriert die hierfür nötige Umsetzung in der Quellsystem-Ontologie anhand eines Beispiels. Jedem Ontologie-Element ist über die Property *has-SourceTable* die Beschreibung einer Datentabelle zugeordnet, die wiederum mit weiteren



**Abbildung 3.9:** Darstellung der aus Abbildung 3.8 gewonnenen Metadaten in Form einer Klassen-Hierarchie mit Instanzen.



**Abbildung 3.10:** Die Dokumentationselemente der Quellsystem-Ontologie sind eine Abstraktion der Nutzdaten in den Quellsystemen.



**Abbildung 3.11:** Informationen in der Quellsystem-Ontologie erlauben einen Zugriff auf die hinter den Dokumentationselementen liegenden Daten.

Informationen über eine Datenbank verknüpft ist, in der diese Tabelle liegt. Der gezeigte Ontologie-Ausschnitt enthält bereits alle nötigen Informationen, um auf die Datensätze hinter dem Ontologie-Element „Geschlecht-W“ zuzugreifen. Mit den Datatype-Properties von „MeineDatenbank“ kann zuerst eine Verbindung zu der beschriebenen Datenbank aufgebaut werden. Anschließend lässt sich aus den Datatype-Properties von „Geschlecht-W“ und „MeineTabelle“ folgendes SQL-Statement konstruieren:

```
1   SELECT DISTINCT PatNr ,  
2           Geschlecht  
3   FROM Test  
4   WHERE Geschlecht = 'W'
```

Wenn die Tabelle „Test“ im SQL-Statement der Tabelle 3.2 auf Seite 38 entspricht, dann liefert die Ausführung des SQL-Statements in der Datenbank die folgende korrekte Ergebnismenge zurück:

PatNr	Geschlecht
456456	W
567567	W
678678	W

Dies ist die Menge aller Datensätze, in denen für Patientinnen das Geschlecht mit der Ausprägung „W“ dokumentiert wurde.

Für den Endanwender, der das Mapping durchführt, sind nur die Dokumentationselemente relevant. Die in Abbildung 3.11 gezeigten Datenzugriffsinformationen können vor ihm vollständig verborgen bleiben, da sie nur maschinell verarbeitet werden. Durch die Abstraktion auf Dokumentationselemente kann besonders leicht mit den Nutzdaten gearbeitet werden: in Abschnitt 3.4.2 auf Seite 45 wird beispielsweise gezeigt, wie einzelne Dokumentationselemente direkt miteinander verrechnet werden können. Der Endanwender muss sich dabei weder mit SQL-Code herumschlagen, noch sich in komplizierte Tabellschemata einarbeiten. Dies wird automatisch durch eine Software erledigt, die die oben gezeigten Informationen „versteht“ und verarbeiten kann.

### 3.4 Beschreibung von Mappings zwischen Quellsystem- und Zieldatensatz-Ontologie

Bisher wurde beschrieben, wie der Zieldatensatz und die Quellsysteme durch Ontologien beschrieben werden können. Diese sollen nun miteinander verknüpft werden, wie es in Abbildung 3.2 auf Seite 31 bereits angedeutet wurde: wenn man die langen URIs in der Grafik mit `dppk:` und `soa:` abkürzt, symbolisiert der mit „`hasImport`“ beschriftete Pfeil das folgende RDF-Statement:

```
dppk:PSA-Wert hasImport soa:PSA-Eingabefeld .
```

Das Statement soll ausdrücken, dass es für das PSA-Datenelement `dppk:PSA-Wert` im DPKK-Datensatz eine entsprechende `soa:PSA-Eingabefeld`-Datenquelle im Quellsystem gibt, aus der Daten in das i2b2-System importiert werden können.

Bei `soa:PSA-Eingabefeld` handelt es sich um ein Dokumentationselement aus der Quellsystem-Ontologie. Dieses ist, wie oben beschrieben, auch mit maschinenverarbeitbaren Informationen zum Zugriff auf die Nutzdaten verknüpft. Über die `hasImport`-Relation wird eine vollständig durch Software interpretierbare Verknüpfung zwischen den Konzepten des Zieldatensatzes und den Nutzdaten der Quellsysteme etabliert. Diese wird anschließend von der Export-Software dazu verwendet, um die Nutzdaten aus den Quellsystemen nach i2b2 zu exportieren.

Leider können nicht immer alle Datenelemente auf die eben beschriebene Weise direkt aufeinander abgebildet werden. Abhängig von der Form, in der die Daten in den Quellsystemen vorliegen, müssen diese erst transformiert werden. Im Folgenden wird beschrieben, wie einzelne Datenelemente mit oder ohne Transformationen gemappt werden können.

#### 3.4.1 Modellierung von direkten Mappings ohne Transformationen

Die Erzeugung von Mappings ohne Transformationen ist nur dann möglich, wenn das Element im Quellsystem in der gewünschten Form bereits dokumentiert ist. Um die Verknüpfung herzustellen, werden die beiden Elemente wie oben beschrieben über die Property `hasImport` miteinander verknüpft:

```
Zieldatensatz-Element hasImport Quelldatensatz-Dokumentationselement .
```

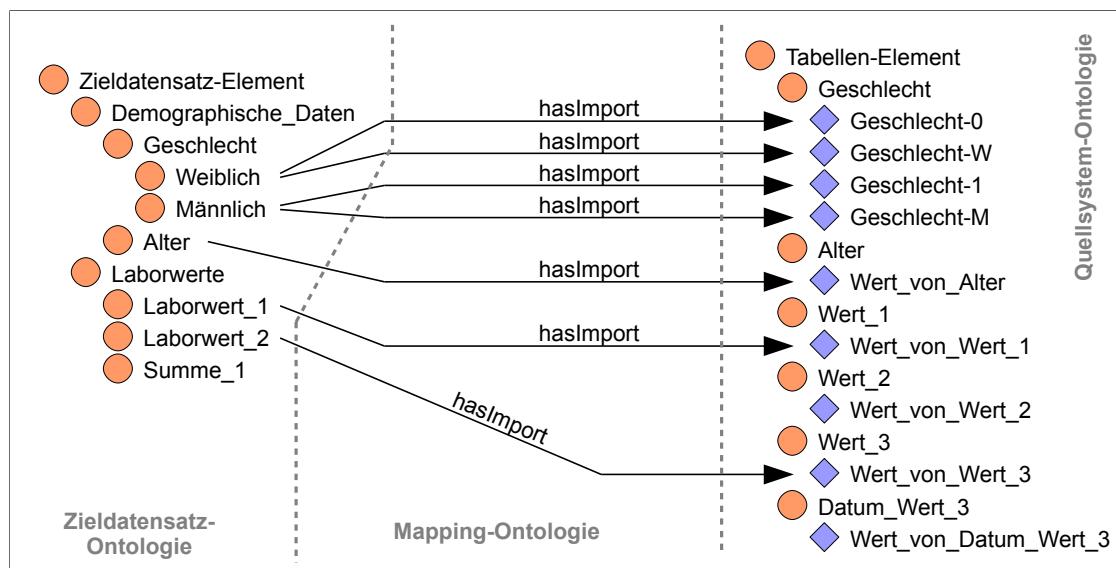
Abbildung 3.12 zeigt ein beispielhaftes Mapping zwischen dem Ziel- und Quellsystem-Datensatz mit derartigen Mappings. Die Grafik zeigt auch, wie 1:n-Mappings zur Zusammenführung von unterschiedlichen Datenquellen verwendet werden können: Alle Elemente mit den Ausprägungen „0“ und „W“ werden auf das Konzept „Weiblich“, alle Datensätze mit den Ausprägungen „1“ und „M“ auf das Konzept „Männlich“ abgebildet.

### 3.4.2 Modellierung von indirekten Mappings mit Transformationen

Kann zwischen zwei Elementen nicht wie in Abschnitt 3.4.1 gemappt werden, so ist dies möglicherweise durch das Verrechnen mit anderen Elementen oder durch eine sonstige Datentransformation möglich.

Abbildung 3.13 auf Seite 46 zeigt ein 1:2-Mapping, bei dem die Summe der Dokumentationselemente „Wert\_von\_Wert\_2“ und „Wert\_von\_Wert\_3“ auf das Konzept „Summe\_1“ abgebildet wird. Dies geschieht durch das Dazwischenschalten von weiteren Knoten, im Beispiel „ADD“ genannt. Ein solcher Zwischenknoten symbolisiert zwei Dinge:

1. Die Operation, die an dieser Stelle mit den beiden Operanden durchgeführt werden muss, beispielsweise eine Addition. Der dem Knoten zugeordnete Operator wird über weitere Properties bestimmt und ergibt sich nicht nur aus dem Knoten-Namen (hier nicht gezeigt).
2. Die Menge aller Ergebnis-Datensätze der Operation, die nach der Verarbeitung des Knotens in einer temporären Tabelle zwischengespeichert wird. Für einen Zwischenknoten werden die gleichen Datenbankzugriffsinformationen wie für Doku-



**Abbildung 3.12:** Einfache 1:1- und 1:n-Mappings zwischen Ziel- und Quellsystem-Datensatz ohne Transformationen.

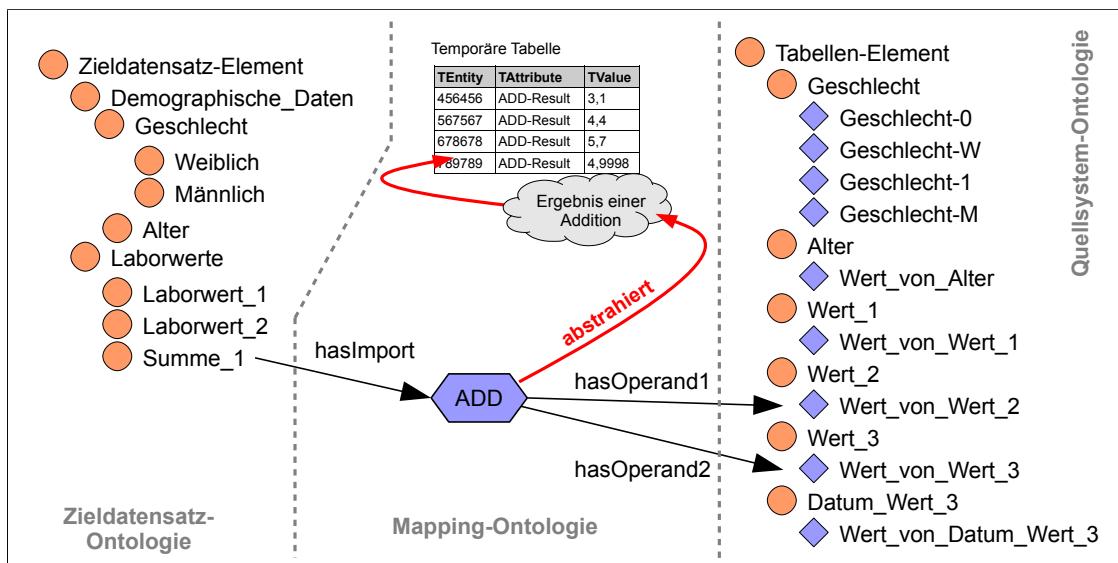
mentationselemente der Quellsystem-Ontologien erfasst (siehe Abschnitt 3.3.3 auf Seite 41).

Man könnte auch sagen, dass ein Zwischenknoten die Menge der Ergebnisdatensätze nach der Durchführung seiner Operation abstrahiert, die in einer Datenbank zwischengespeichert sind (siehe Abbildung 3.13).

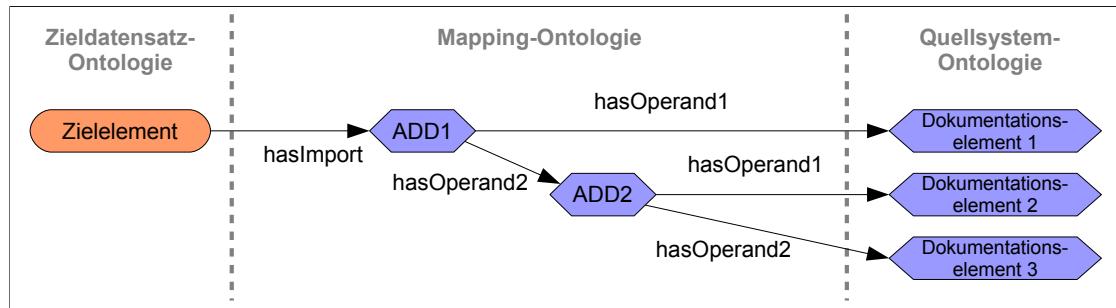
Über die Properties *hasOperand1* und *hasOperand2* wird eine Verbindung zu den Dokumentationselementen hergestellt, die miteinander verrechnet werden sollen. Die Unterscheidung zwischen Operand 1 und Operand 2 ist in Bezug auf die Kommutativität der Operation relevant.

Für jeden Knoten ist die Anzahl an zulässigen Operanden auf zwei beschränkt. Soll ein dritter Operand in die Berechnung aufgenommen werden, so wird dies nicht über einer Property *hasOperand3* abgebildet, sondern es wird, wie es in Abbildung 3.14 gezeigt ist, ein weiterer Zwischenknoten eingefügt. Damit folgt der Ansatz dem bekannten „Teile und herrsche“-Prinzip: das ursprüngliche „schwierige“ Problem wird in mehrere einfache zu lösende Teilprobleme zerlegt. Wenn diese alle gelöst sind, ist auch das ursprüngliche Problem gelöst.

Der Vorteil besteht darin, dass die Anzahl an benötigten Knotentypen auf „eine Hand voll“ beschränkt bleibt. Trotzdem können bei Bedarf durch Kombination dieser Grundope-



**Abbildung 3.13:** 1:2-Mapping zwischen Ziel- und Quellsystem-Datensatz mit Transformation. Auch die Knoten in der Mapping-Ontologie abstrahieren Datensätze in einer relationalen Datenbank. In diesem Fall sind das die Ergebnisse der Operation selbst.



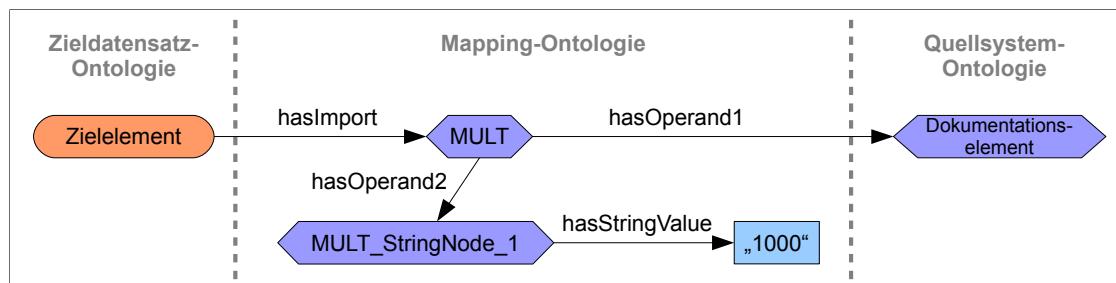
**Abbildung 3.14:** 1:3-Mapping mit Transformationen: die Addition der drei Dokumentationselemente rechts wird durch zwei einzelne Additionen ausgedrückt.

rationen nahezu alle anderen komplexen Transformationen und Regeln konstruiert werden. Außerdem sind nur zwei Properties nötig: *hasOperand1* und *hasOperand2*. Würde man mehrere Operanden zulassen, wäre es nicht mehr möglich, alle Properties direkt in der Ontologie anzulegen, da die Nummerierung Teil der Property-URI ist. Man müsste dann Zwischenknoten anlegen, mit denen eine ternäre Beziehung zwischen Operator-Knoten, Operand-Knoten und der Operand-Nummer aufgebaut wird. Diese Technik wurde im Zusammenhang mit den Blank Nodes in Abbildung 2.11 auf Seite 21 gezeigt.

### 3.4.3 Einbindung von Datenwerten

Bei Berechnungen ist es möglicherweise erforderlich, externe Datenwerte einzubinden. Das Bedürfnis entsteht beispielsweise, wenn ein Laborwert mit einer Multiplikation in eine andere Einheit umgerechnet werden soll. Der gewünschte Faktor liegt dabei nicht als Dokumentationselement in der Quellsystem-Ontologie vor, sondern muss in Form eines Literals als Operand in die Mapping-Ontologie aufgenommen werden.

Da OWL zwischen Object-Properties und Datatype-Properties unterscheidet (siehe Tabelle A.3 auf Seite 122), ist es nicht möglich, diese Literale über die Object-Properties *hasOperand1* und *hasOperand2* einzubinden. Der Lösungsansatz besteht darin, einen Zwischenknoten einzufügen, der das Literal über eine Datatype-Property *hasStringValue* zur Verfügung stellt (siehe Abbildung 3.15).



**Abbildung 3.15:** Die Einbindung von Datenwerten erfolgt über spezielle „String-Zwischenknoten“.

## 3.5 Daten-Export unter Verwendung der Mapping-Ontologie

Bisher wurde beschrieben, wie der Zieldatensatz, die Quellsysteme und das Mapping mit semantischen Netzen beschrieben werden können. In Abschnitt 3.3.3 auf Seite 41 wurde erklärt, dass man die Dokumentationselemente in der Quellsystem-Ontologie als eine Abstraktion der Nutzdaten in den Quellsystemen auffassen kann, wenn man sie mit besonderen Datenbankzugriffsinformationen anreichert. In einem Beispiel konnte damit ein einfaches SQL-Statement erzeugt werden, das die relevanten Datensätze zurückliefert. In Abschnitt 3.4.2 auf Seite 45 wurde beschrieben, dass die gleichen Datenbankzugriffsinformationen auch für die Zwischenknoten in der Mapping-Ontologie hinterlegt werden, um auf die temporären Zwischenergebnisse der Berechnungen zugreifen zu können.

In den folgenden Abschnitten wird gezeigt, wie man die Datenbankzugriffsinformationen von mehreren Knoten miteinander kombinieren kann, um daraus komplexere SQL-Statements zu erzeugen, und wie man dadurch die Daten wunschgemäß in eine i2b2-Forschungsdatenbank laden kann.

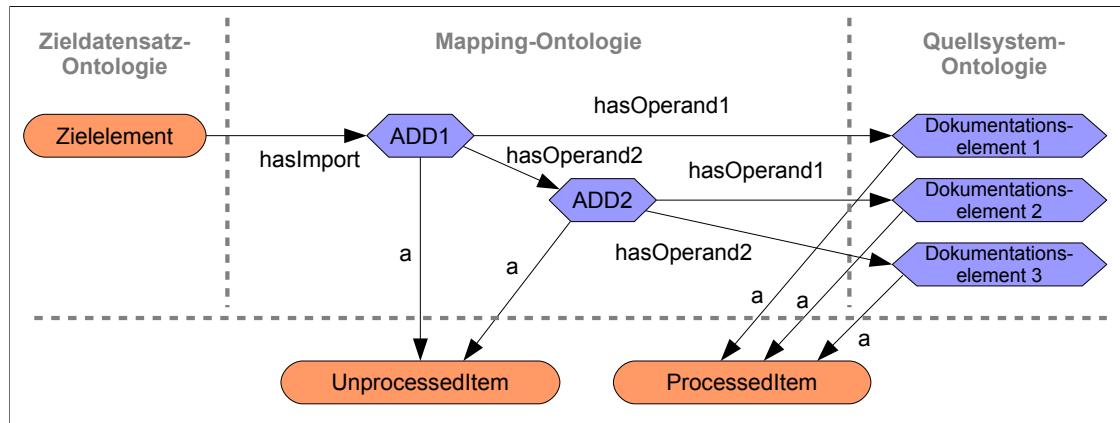
### 3.5.1 Gewinnung der Abarbeitungsreihenfolge der Knoten

Um diesen Prozess durchführen zu können, muss zunächst eine gültige Abarbeitungsreihenfolge der Zwischenknoten ermittelt werden. Es ist offensichtlich, dass ein Knoten des Mapping-Netzes erst bearbeitet werden kann, wenn die anderen Knoten, auf die er über die *hasOperand1*- und *hasOperand2*-Relationen lesend zugreift, schon bearbeitet sind.

Ein Knoten erhält den Status „bearbeitet“, sobald die Ergebnismenge an Datensätzen vorliegt, die durch ihn symbolisiert wird. Alle Dokumentationselemente der Quellsystem-Ontologie gelten somit als „bearbeitet“, da für sie bereits Nutzdaten in den Quellsystemen vorliegen. Die Knoten der Mapping-Ontologie sind zu Beginn des Exports dagegen alle noch „unbearbeitet“. Erst nach der Berechnung ihres Ergebnisses werden sie als „bearbeitet“ markiert.

Einen Knoten zu markieren ist denkbar leicht: da es sich bei den Knoten um Instanzen handelt, können sie einfach als Mitglieder einer Klasse definiert werden, mit der dieser Status symbolisiert wird. In Abbildung 3.16 auf Seite 49 ist dies durch Klassen **ProcessedItem** und **UnprocessedItem** illustriert. Man sieht, dass zum dargestellten Startzeitpunkt  $t = 0$  alle Dokumentationselemente als „bearbeitet“ und die Zwischenknoten in der Mapping-Ontologie als „unbearbeitet“ gelten.

Wird mit der Abarbeitung der Knoten in Abbildung 3.16 begonnen, kann im ersten Verarbeitungsschritt ( $t = 1$ ) nur der Knoten **ADD2** abgearbeitet werden, da er der einzige



**Abbildung 3.16:** Klassifizierung der Knoten als „unbearbeitet“ und „bearbeitet“ zum Zeitpunkt  $t = 0$ .

Knoten ist, der auf „bearbeitete“ Knoten lesend zugreift. Ist ADD2 verarbeitet, wird dies durch eine Zuordnung zur Klasse ProcessedItem festgehalten, wobei die Klassenzugehörigkeit zu UnprocessedItem entfernt wird. Anschließend ( $t = 2$ ) erfüllt auch ADD1 die Voraussetzung zur Verarbeitung.

Die Abarbeitung des Mapping-Netzes liefert immer ein korrektes Ergebnis, solange die oben festgestellte Bedingung bei der Auswahl des nächsten Knotens eingehalten wird. Es ist dabei irrelevant, welcher der zur Abarbeitung bereiten Knoten als nächster bearbeitet wird.

Mit einer einfachen SPARQL-Anfrage über der Mapping-Ontologie (siehe Abschnitt 2.3.7 auf Seite 25) lässt sich ein bearbeitbarer Knoten ermitteln:

```

1   SELECT ?operator
2   WHERE {?operator a UnprocessedItem .
3           ?operator hasOperand1 ?operand1 .
4           ?operator hasOperand2 ?operand2 .
5           ?operand1 a ProcessedItem .
6           ?operand2 a ProcessedItem . }
7   LIMIT 1

```

Die Anfrage liefert in der Variable `?operator` (Zeile 1) höchstens einen Knoten zurück, der eine Instanz der Klasse `UnprocessedItem` ist (2) und der die Relationen `hasOperand1` und `hasOperand2` zu zwei weiteren Knoten (Zeilen 3 und 4) besitzt. Diese müssen beide Instanzen der Klasse `ProcessedItem` sein (Zeilen 5 und 6).

### 3.5.2 Berechnung eines einzelnen Zwischenknotens

Wie oben beschrieben, kann über zusätzliche Informationen in der Ontologie auf die eigentlichen Nutzdaten zugegriffen werden. Dies kann beispielsweise durch SQL-Operationen erfolgen. Im Folgenden wird beschrieben, wie diese und andere Informationen kombiniert werden können, um die Berechnung eines Zwischenknotens durchzuführen.

Im Gegensatz zu dem einfachen SQL-Beispiel in Abschnitt 3.3.3 auf Seite 41, bei dem auf eine Tabelle im spaltenorientierten Format zugegriffen wurde, soll von nun an das EAV-Format für alle Quellsystem-Tabellen und temporären Tabellen angenommen werden. Zwar würde ein Zugriff auf das spaltenorientierte Format bei den Quellsystem-Tabellen auch funktionieren (das SQL-Beispiel in Abschnitt 3.3.3 hat dies gezeigt), jedoch eignet sich nur das EAV-Format zur Zwischenspeicherung der Ergebnisse der Mapping-Knoten in einer einzigen Tabelle.

Um das Beispiel aus Abbildung 3.13 auf Seite 46 fortzuführen, soll Tabelle 3.5 als (unvollständige) EAV-Darstellung [67, 74] der Tabelle 3.2 auf Seite 38 dienen. In dem Beispiel sollen die Ausprägungen der Attribute „Wert2“ und „Wert 3“ addiert werden.

PatNr	Attribut	Wert
...	...	...
456456	Wert 2	2
567567	Wert 2	3
678678	Wert 2	4
789789	Wert 2	5
...	...	...
456456	Wert 3	1,1
567567	Wert 3	1,4
678678	Wert 3	1,7
789789	Wert 3	-0,0002
...	...	...

**Tabelle 3.5:** EAV-Darstellung der Tabelle 3.2 auf Seite 38. Gezeigt sind nur Einträge der Attribute „Wert 2“ und „Wert 3“

Um die Werte korrekt zu addieren, sind folgende Schritte nötig:

1. Zuerst müssen die beiden Datensätze, die miteinander verrechnet werden, aus der EAV-Tabelle geladen werden. Dies kann durch zwei einfache SELECT-Statements erfolgen, bei denen die Attribut-Spalte nach „Wert 2“ bzw. „Wert 3“ gefiltert wird.
2. Um die Addition in der Datenbank durchführen zu können, müssen die beiden Datensätze nebeneinander stehen. Dabei dürfen nur „Wert 2“-Einträge mit „Wert3“-Einträgen des selben Patienten addiert werden. Dies kann über einen JOIN erreicht werden, bei dem die beiden Datensätze an der Entity-Spalte ausgerichtet werden.
3. Stehen die Datensätze nebeneinander, können die Einträge der beiden Value-Spalten durch eine Datenbankoperation addiert werden.
4. Die Summen der Werte können anschließend in die Tabelle für temporäre Zwischenergebnisse gespeichert werden.

Das SQL-Statement in Auflistung 3.1 führt genau diese Schritte für das Beispiel oben durch. In Zeile 6 und 8 werden die relevanten Datensätze mit SELECT-Statements in den Speicher des Datenbanksystems geladen, in den Zeilen 7 und 9 findet der JOIN statt (dessen Inhalt ist in Abbildung 3.6 auf Seite 52 gezeigt). Die Addition der einzelnen Werte erfolgt in Zeile 4. Das äußere SELECT-Statement (Zeilen 2-4) produziert die neuen Datensätze gemäß EAV-Format, die durch Zeile 1 in die temporäre Tabelle geschrieben werden. Das Ergebnis dieser SQL-Operation ist in Tabelle 3.7 auf Seite 3.7 zu sehen.

```
1  INSERT INTO TempTable(TEntity, TAttribute, TValue)
2  SELECT DISTINCT OP1.Entity,
3      'ADD-Result',
4      OP1.Value + OP2.Value
5  FROM (
6      (SELECT PatNr Entity, Wert Value FROM Test WHERE Attribut = 'Wert 2') OP1
7      FULL OUTER JOIN
8      (SELECT PatNr Entity, Wert Value FROM Test WHERE Attribut = 'Wert 3') OP2
9      ON OP1.Entity = OP2.Entity
10 );
```

**Auflistung 3.1:** SQL-Statement, das die Berechnung des Knotens in Abbildung 3.13 auf Seite 46 durchführt.

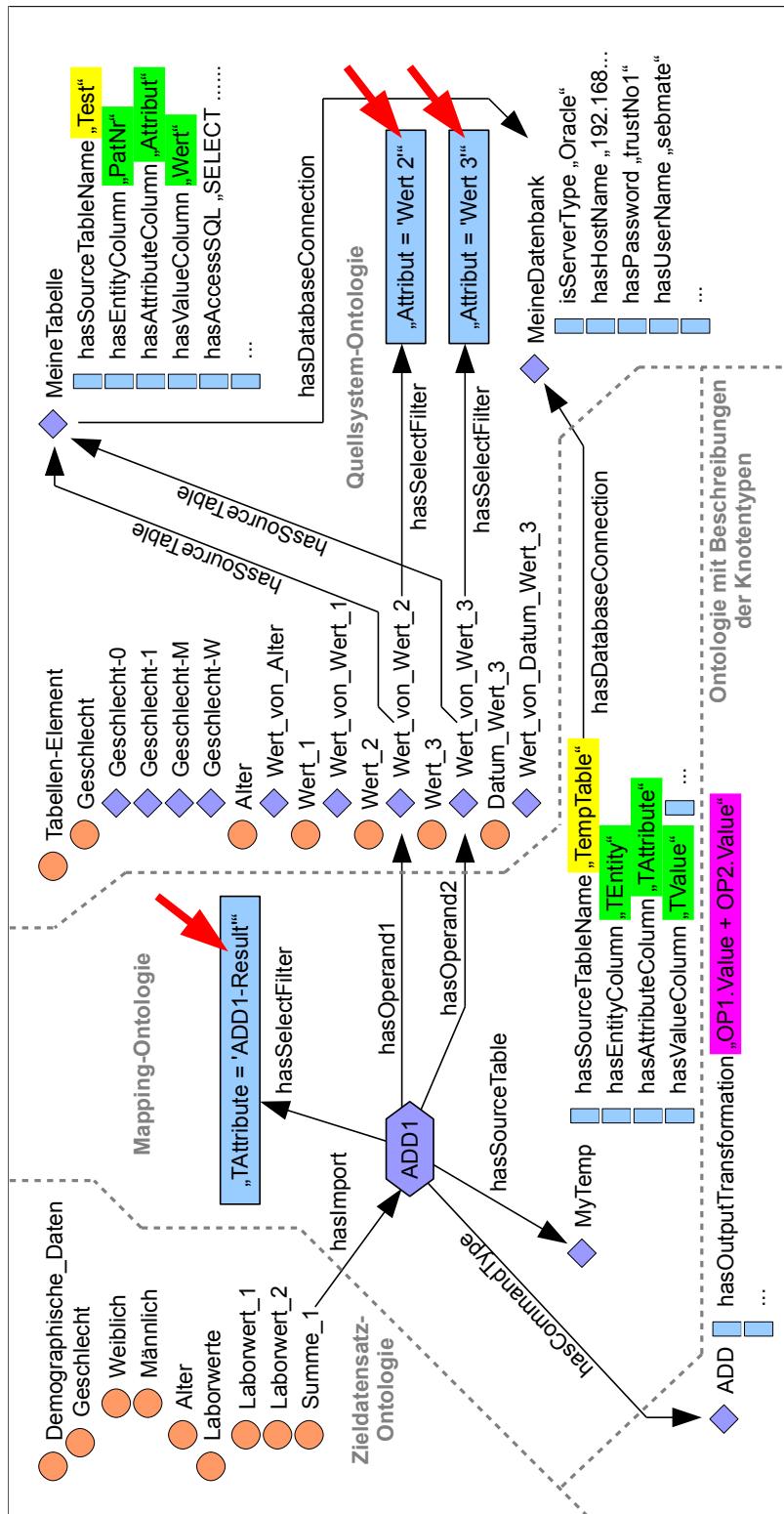
<b>OP1.Entity</b>	<b>OP1.Value</b>	<b>OP2.Entity</b>	<b>OP2.Value</b>
456456	2	456456	1,1
567567	3	567567	1,4
678678	4	678678	1,7
789789	5	789789	-0,0002

**Tabelle 3.6:** Die relevanten Daten nach dem Join.

<b>TEntity</b>	<b>TAttribute</b>	<b>TValue</b>
456456	ADD-Result	3,1
567567	ADD-Result	4,4
678678	ADD-Result	5,7
789789	ADD-Result	4,9998

**Tabelle 3.7:** Ergebnis des SQL-Statements in Auflistung 3.1, falls dieses auf Tabelle 3.5 angewendet wird. Siehe auch Abbildung 3.13 auf Seite 46.

Die Erzeugung derartiger SQL-Statements kann aufgrund der in den Ontologien gespeicherten Datenbankzugriffsinformationen leicht durch eine Software erfolgen. Die Illustration in Abbildung 3.17 auf Seite 53 fasst die Lage aller relevanten Informationen zur Erzeugung der SQL-Statements nochmals zusammen. Die Angaben, welche Tabellennamen für die SELECT-Statements in den Zeilen 6 und 7 und für das INSERT-Statement in Zeile 1 zu nehmen sind, sind durch Verfolgen der Relationen *hasSourceTable* und *hasSourceTableName* ermittelbar (gelb hinterlegt). Die Namen der EAV-Spalten der jeweiligen Tabelle sind ebenfalls so hinterlegt (grün). Über die Property *hasSelectFilter* wird ein SQL-Codefragment angegeben, mit dem die jeweilige Attribut-Spalte in den Zeilen 6 und 7 nach den gewünschten Nutzdatensätzen gefiltert werden kann (markiert durch rote Pfeile). Über eine Beziehung *hasCommandType* wird für die einzelnen Zwischenknoten außerdem angegeben, welche Datenbankoperation in Zeile 4 durchzuführen ist (violett hinterlegt).



**Abbildung 3.17:** Übersicht über die Lage der Informationen, die zur Konstruktion der SQL-Statements für die Abarbeitung der Zwischenknoten dienen. Diese sind gelb, grün, magenta und mit roten Pfeilen markiert.

### 3.5.3 Algorithmus zur Abarbeitung von komplexen Mappings

Wenn alle Informationen zum Zugriff auf die Datenbank in den Ontologien vorliegen und eine Regel zur Ermittlung einer gültigen Abarbeitungsreihenfolge ausfindig gemacht wurde, kann ein Nutzdaten-Export auf die oben beschriebene Weise umgesetzt werden. Der folgende Algorithmus ist eine Zusammenfassung des vollständigen Importprozesses:

1. Erzeuge die i2b2-Ontologie aus der Zieldatensatz-Ontologie.
2. Solange die Klasse `UnprocessedItem` Mitglieder besitzt (das sind alle unbearbeiteten Knoten):
  - a) Finde irgendeinen (unbearbeiteten) Knoten, der zur Klasse `UnprocessedItem` gehört und dessen Operanden zur Klasse `ProcessedItem` gehören. Dies kann mit einer SPARQL-Query erfolgen, wie es in Abschnitt 3.5.1 auf Seite 48 gezeigt wurde.
  - b) Lade alle nötigen Informationen aus der Ontologie, die nötig sind, um ein SQL-Statement zur Berechnung des Knoten-Ergebnisses zu erzeugen. Benötigt werden insbesondere alle Angaben zum Zugriff auf die Nutzdatensätze der beiden Operanden und über die Datebankoperation, mit der die Nutzdatensätze verrechnet werden sollen.
  - c) Erzeuge das SQL-Statement und führe es im Datenbanksystem aus.
  - d) Ordne dem Knoten über die `hasSourceTable` die temporäre Tabelle zu.
  - e) Ändere die Klassenzugehörigkeit des Knotens von `UnprocessedItem` zu `ProcessedItem`.
3. Exportiere die Daten aller Knoten in das Zielsystem i2b2, die eine eingehende `hasImport`-Beziehung besitzen. Das sind alle Zwischenergebnisse aus Schritt 2 sowie die Daten der ohne Transformation gemappten Dokumentationselemente aus der Quellsystem-Ontologie (siehe Abschnitt 3.4.1 auf Seite 44).

Die Nutzdatensätze werden auf diese Weise von Knoten zu Knoten „weitergereicht“ und bearbeitet, bis die endgültigen Ergebnisse schließlich im Wurzel-Zwischenknoten vorliegen und in das Zielsystem geladen werden können.

## 3.6 Erweiterung der i2b2 Software zum Vermittlungsportal

Die Idee, i2b2 in einem institutionsübergreifenden Szenario einzusetzen, ist nicht neu. Mit SHRINE [101] wurde ein auf i2b2 basierendes System vorgestellt, das Client-Anfragen an die i2b2-Server von mehreren Einrichtungen weiterleitet. Diese durchsuchen dann ihre eigenen Datenbestände und übermitteln das Ergebnis zurück an den Client.

Auch in Erlangen wurden bereits Ende 2009 erste Überlegungen dazu angestellt, wie man i2b2 im DPKK-Netzwerk als neue Forschungsdatenbank verwenden könnte. Die Motivation hierfür lag insbesondere in der benutzerfreundlichen Oberfläche zur Konstruktion von Anfragen, dem generischen Datenmodell und dem Open-Source-Charakter, der i2b2 dazu prädestiniert, als Grundlage für derartige Eigenentwicklungen zu dienen. Statt jedoch wie bei SHRINE mehrere verteilte i2b2-Datenbanken zu durchsuchen, sollten im DPKK-Szenario die Daten aller teilnehmenden DPKK-Mitglieder einem gemeinsamen Recherche-i2b2 beigesteuert werden.

Abbildung 3.18 auf Seite 56 auf Seite illustriert den vollständigen Ansatz. Auf der untersten Ebene befinden sich alle DPKK-Mitglieder, von denen jedes eine eigene i2b2-Installation betreibt. In diese importieren die Mitglieder ihre eigene Prostatakarzinom-Dokumentation unter Verwendung der in dieser Arbeit vorgestellten Mapping-Techniken. Diese lokalen i2b2s können dann bereits für In-House-Recherchen über der eigenen Dokumentation verwendet werden.

In regelmäßigen Intervallen werden die Daten anonymisiert und an ein zentrales, DPKK-internes i2b2 übermittelt (1). Bei diesem Schritt wird jedoch jeder Datensatz mit einem Kennzeichen versehen, anhand dessen nachvollzogen werden kann, aus welchem Institut er stammt. Von hier aus werden die Daten in ein weiteres, von außen zugängliches i2b2 geladen (2), wobei dann auch das Kennzeichen entfernt wird. Diese beiden i2b2-Server werden zentral vom DPKK verwaltet, allerdings kann nur auf das vollständig anonymisierte i2b2 – in der Abbildung ganz oben – über das Web zugegriffen werden.

Dort können dann DPKK-Mitglieder beliebige Recherchen nach Bioproben durchführen (3). Als Antwort auf eine solche Datenbankanfrage wird die Anzahl an Patienten genannt, von denen entsprechende Bio-Proben im DPKK-Verbund vorliegen. Aufgrund der vollständigen Anonymisierung kann ein Mitglied jedoch nicht feststellen, in welchem Institut sich das Biomaterial befindet. Besteht daran Interesse, so muss die Query-Definition – das ist die Beschreibung, nach welchen Kriterien in der Datenbank gesucht wurde – von dem i2b2-Server (4) entgegengenommen und an das zentrale i2b2 übermittelt werden (5). Die Zentrale startet die Query dann auf ihrer eigenen i2b2-Installation (6), um zu ermitteln, bei welchen DPKK-Mitgliedern sich die Proben tatsächlich befinden. Zum Schluss wird die Query-Definition von der Zentrale an diese relevanten Partner weitergeleitet (7), die dann die Proben der Patienten ermitteln (8) und sich nach eigenem Ermessen mit dem suchenden DPKK-Mitglied zwecks Materialaustausch in Verbindung setzen.

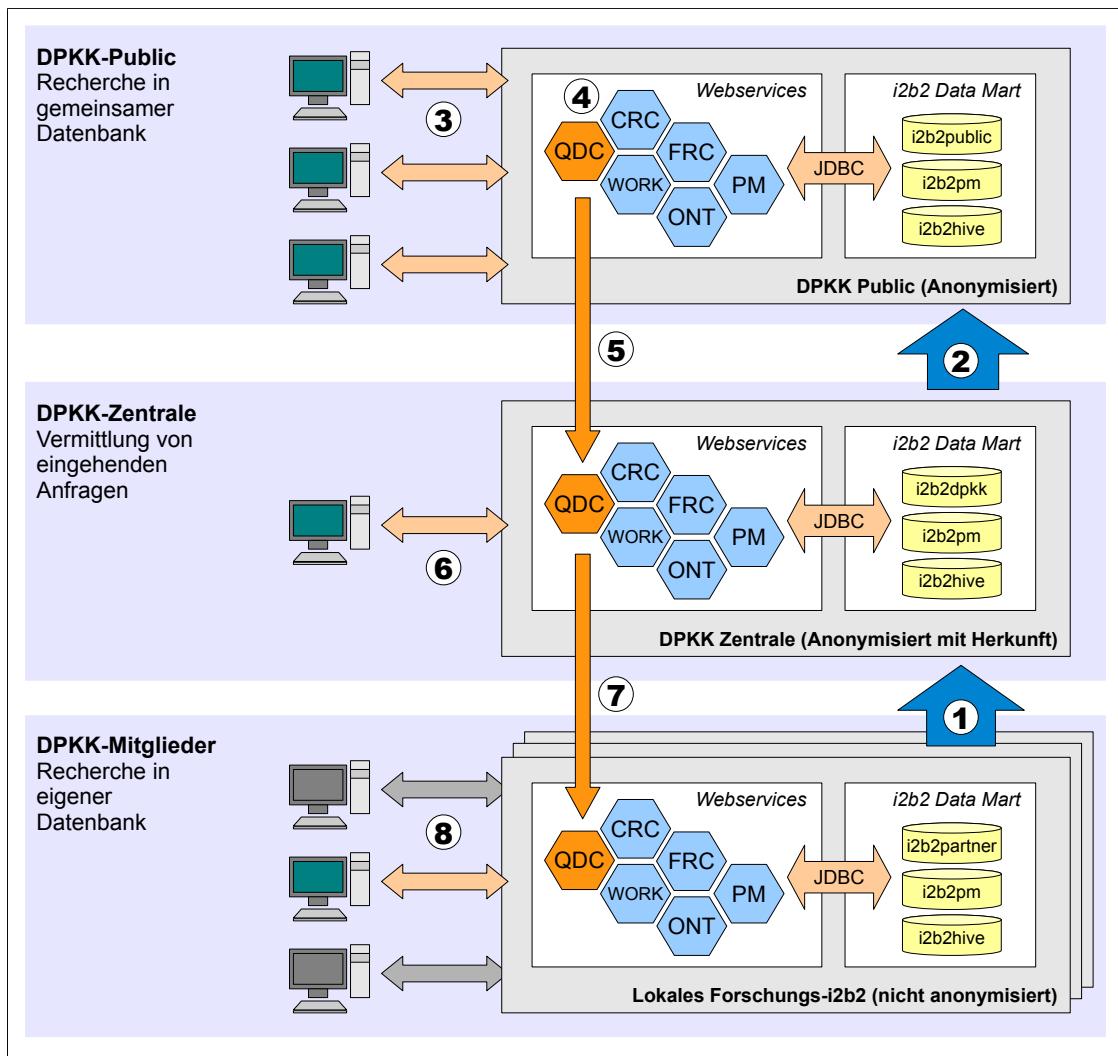


Abbildung 3.18: Das dreischichtige DPKK-Vermittlungsportal auf i2b2-Basis.

Erste, für dieses i2b2-Vermittlungsportal relevante Arbeiten, wurden bereits Ende 2009 von K. Starke<sup>1</sup> prototypisch umgesetzt und sollen in dieser Arbeit nicht erneut aufgegriffen werden. Das betrifft in den Schritten (1) und (2) die Anonymisierung und die Nutzdatentransfers zwischen den einzelnen i2b2s. Ebenso sollen die damit zusammenhängenden datenschutzrechtlichen Aspekte keine weitere Beachtung finden. In [60] konnte jedoch gezeigt werden, dass sich der TMF-PID-Generator [81] zur Pseudonymisierung leicht in den i2b2-Importprozess einbinden lässt; in [80] wurden erste Ergebnisse in Bezug auf eine zweistufige Pseudonymisierung durch den TMF-Pseudonymisierungsdienst [81] geliefert.

<sup>1</sup> Medizinisches Zentrum für Informations- und Kommunikationstechnik, Universitätsklinikum Erlangen

Darüber hinaus bietet i2b2 seit Version 1.4 verschiedene Funktionen an, mit denen man die Anzeige eines zu kleinen Patientenkollektivs serverseitig unterdrücken kann.

Im Rahmen dieser Arbeit soll lediglich die Übermittlung einer i2b2-Query-Definition von einem i2b2-Server auf einen anderen adressiert werden. Die Komponente soll „Query Definition Communicator“ (QDC) heißen. Hierfür sind sowohl Arbeiten auf der Serverseite als auch auf der Clientseite von i2b2 erforderlich. Serverseitig soll eine neue i2b2-Zelle, die QDC-Zelle, implementiert und zur Verfügung gestellt werden. Für die Interaktion mit dem Benutzer ist zusätzlich ein neues Workbench-View zu erstellen.

An das Workbench-View ergeben sich aufgrund des oben beschriebenen Szenarios folgende Anforderungen:

1. Das View muss per Drag&Drop eine Query-Definition aus den anderen Views „Previous Queries“, „Query Tool“ oder „Workplace“ entgegen nehmen<sup>1</sup> und die XML-Darstellung der Query-Definition ermitteln.
2. Bevor die Query-Definition verschickt wird, sollte der Benutzer eine Nachricht an die Anfrage anheften können.
3. Außerdem sollte gezielt ausgewählt werden können, an welche anderen i2b2s die Anfrage weitergeleitet wird. Dies ist jedoch nur relevant für die DPKK-Zentrale. Im öffentlichen i2b2 können die Query-Definitionen nur an die DPKK-Zentrale weitergeleitet werden; die i2b2-Server der DPKK-Mitglieder selbst leiten auch keine Queries weiter.
4. Die Query Definition muss dann zusammen mit der Nachricht an den QDC-Webservice übermittelt werden.

---

<sup>1</sup> Die Funktionen der einzelnen Views sind in [60, S. 17] erklärt.

Die QDC-Zelle muss damit als serverseitiges Pendant folgende Aufgaben erfüllen:

1. Sie muss die XML-Nachricht von der Workbench entgegennehmen und an die Ziel-i2b2s weiterleiten.
2. Sie muss Query-Definitionen, die sie von anderen QDC-Zellen zugesendet bekommt, entgegen nehmen und dem Benutzer im Hive bereitstellen. Die eingehenden Queries sollen im „Workplace“-View bereitgestellt werden.

Die Umsetzung soll auf der Basis des bereits existierenden Quellcodes der PFT-Zelle (Pulmonary Function Test Processing Cell) und des dazugehörigen Workbench-Views erfolgen. Da es bisher noch keine Leitfäden für die Entwicklung von i2b2-Erweiterungen gibt, wird diese Vorgehensweise sogar von den i2b2-Entwicklern empfohlen<sup>1</sup>. Die PFT-Zelle dient der Extraktion von einzelnen Messwerten aus textuellen Reporten eines Lungenfunktionsgerätes. Dazu nimmt das PFT-View den Report entgegen und übermittelt ihn an die PFT-Zelle. Diese parst den Report und schickt das Ergebnis anschließend an das PFT-View zurück<sup>2</sup>. Aufgrund der geringen Komplexität lässt sich leicht ein neues Eclipse-View sowie der dazugehörende Webservice erstellen.

---

1 Siehe Kommentar von M. Mendis am 20.04.2010, 10:34 Uhr: <https://community.i2b2.org/wiki/display/community/AUG+E-mail+Repository>, Abruf: 19.12.2010

2 Screenshots und eine ausführlichere Beschreibung der PFT-Zelle befinden sich in der Installationsanleitung, siehe: <https://www.i2b2.org/software/repository.html?t=doc&p=12>, Abruf: 19.12.2010

# KAPITEL 4

---

## Ergebnisse

---

Im Rahmen dieser Arbeit konnten alle der in Kapitel 3 vorgestellten Methoden zum Mapping der klinischen Daten in der Form von kleinen Computerprogrammen prototypisch umgesetzt werden.

Die Stellen, an denen diese Werkzeuge im Projekt zwischen Münster und Erlangen Anwendung finden, sind in Abbildung 4.1 auf Seite 60 gezeigt. Wie in der Graphik ersichtlich, musste in einigen Details jedoch vom ursprünglichen Konzept abgewichen werden. Beispielsweise war es aus Zeitgründen nicht mehr möglich, eine Technik zu entwickeln, mit der man automatisiert eine vollständige Quellsystem-Ontologie für ORBIS hätte generieren können. Um die Nutzdaten aus ORBIS dennoch verwenden zu können, musste der Datenimport über einen Umweg mit einem in Rahmen dieser Arbeit entstandenen Tool erfolgen.

Die Quelldaten aus Erlangen und Münster konnten – soweit das die Datensätze erlauben – erfolgreich auf den DPKK-Datensatz gemappt werden.

Auch die in Abschnitt 3.6 ab Seite 55 angedachte Erweiterung der i2b2-Software zum DPKK-Vermittlungsportal konnte ebenfalls erfolgreich umgesetzt werden. Bei einer Präsentation des i2b2-Vermittlungsportals auf einer DPKK-Sitzung am 28.10.2010 in Erlangen wurde das i2b2-System von den anwesenden DPKK-Mitgliedern grundsätzlich als eine für die Ziele des DPKKs geeignete Lösung angesehen. Die Entscheidung, i2b2 zu benutzen, fiel jedoch noch nicht.

Die folgenden Abschnitte stellen die Ergebnisse des Methoden-Teils in Verbindung mit diversen Implementierungsdetails vor.

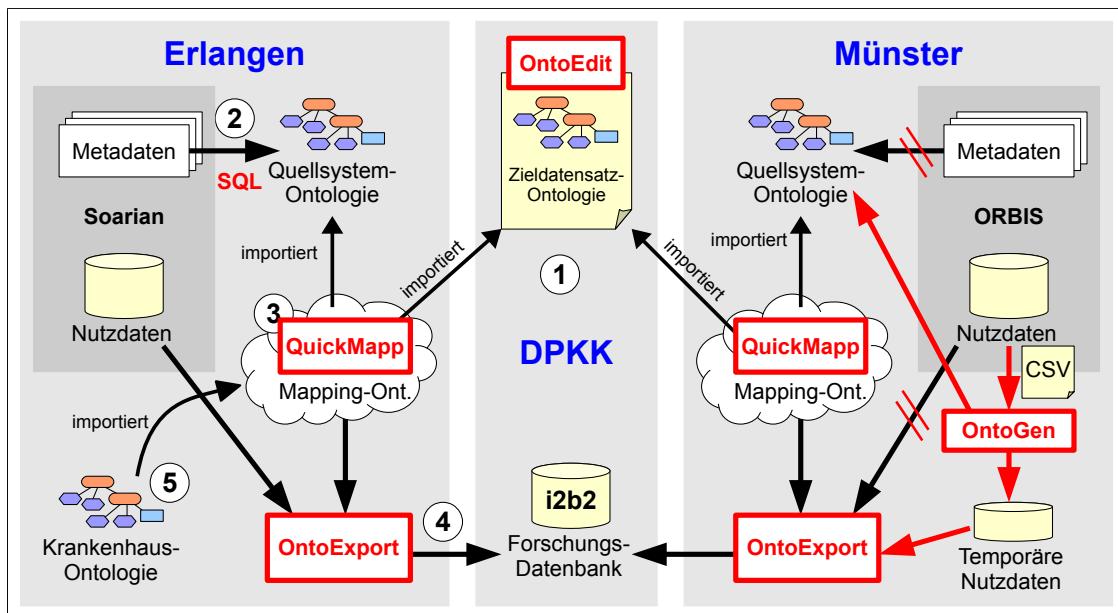


Abbildung 4.1: Einordnung der entwickelten Tools in das Rahmenkonzept dieser Arbeit.

## 4.1 Übersicht über die entwickelten Werkzeuge

Im Rahmen dieser Arbeit konnte ein Demonstrations-Paket zusammengestellt werden, mit dem der erdachte Ansatz unabhängig von jeglicher Klinikums-EDV auf einem beliebigen PC nachvollzogen werden kann. Statt beispielsweise auf die Datenbank eines klinischen Arbeitsplatzsystems zugreifen zu müssen, kann eine einfache mitgelieferte Excel-Tabelle mit Testdaten (siehe Tabelle B.1 auf Seite 132) oder ein realer CSV-Datenexport als Datenquelle dienen.

Abbildung 4.2 zeigt Interaktionen und Datenströme zwischen den Tools und den anderen, erforderlichen Systemen.

Falls ein direkter Zugriff auf die Datenbank des Quellsystems nicht möglich ist, kann der vollständige Workflow mit einem CSV-Export (1) gestartet werden. Das Tool **OntoGen** „kippt“ dann die CSV-Datei (2) in das EAV-Format und stellt die darin enthaltenen Daten in einer temporären Datenbank den anderen Programmen zur Verfügung (3). Aus den Spaltenüberschriften und den aggregierten Ausprägungen wird gemäß Abschnitt 3.3.2 auf Seite 37 eine einfache Quellsystem-Ontologie erzeugt, ebenso eine entsprechende Zieldatensatz- und Mapping-Ontologie (4).

Mit **OntoEdit** kann anschließend die Zieldatensatz-Ontologie bearbeitet werden (5). Im Gegensatz zu generischen OWL-Editoren wie Protégé ist OntoEdit speziell auf die

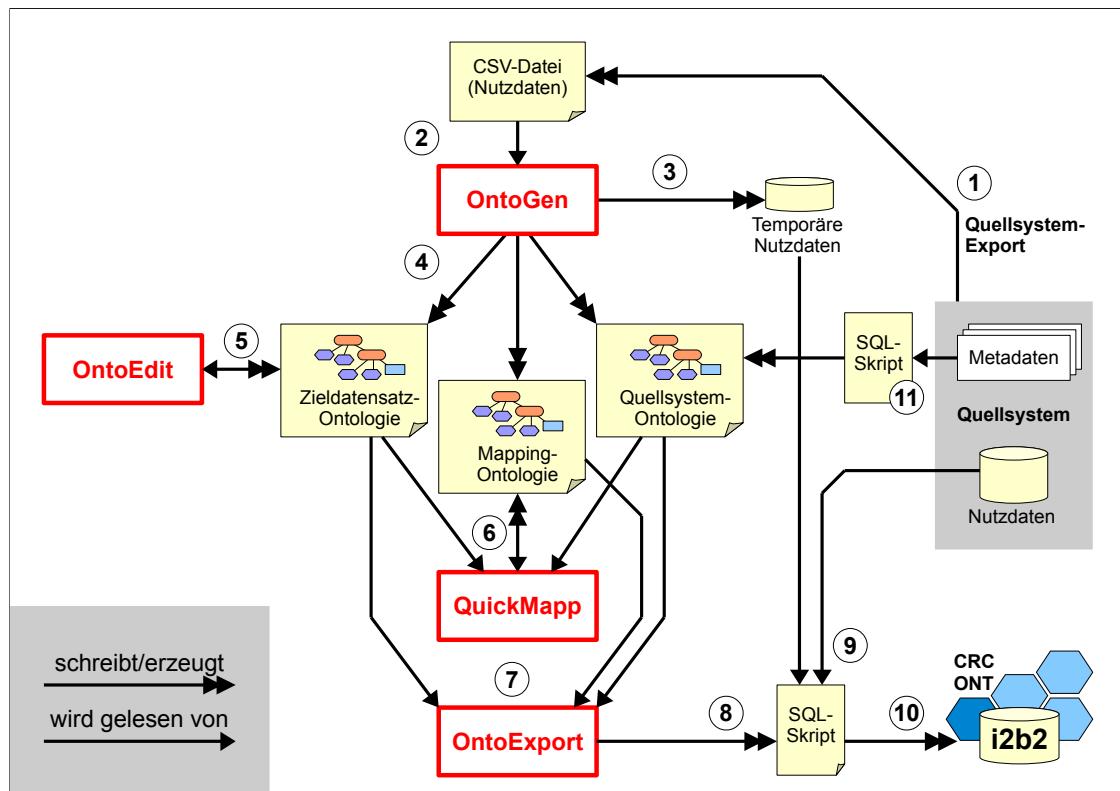


Abbildung 4.2: Zusammenspiel der im Rahmen dieser Arbeit entwickelten Tools.

Anforderungen der i2b2-Metadatenverwaltung zugeschnitten, die in Abschnitt 3.2 auf Seite 32 bereits angekündigt wurden und später zusammen mit OntoEdit in Abschnitt 4.3 auf Seite 65 genauer erklärt werden.

**QuickMapp** erlaubt das effiziente Erstellen und Editieren von Mappings zwischen Quellsystem- und Zieldatensatz-Ontologie. Da die zu erstellenden Mapping-Netze beachtliche Dimensionen annehmen können, die nur schwierig mit Standard-OWL-Editoren bearbeitbar sind (siehe beispielsweise Abbildung 4.16 auf Seite 87), wurde mit QuickMapp eine Lösung entwickelt, mit der die Netze durch eine einfache Klammersprache in Textform ausgedrückt werden können. Die Ausdrücke werden automatisch in RDF-Statements übersetzt und anschließend in die Mapping-Ontologie eingebracht (6).

Das Tool **OntoExport** liest schließlich alle Ontologien ein (7) und erzeugt gemäß Abschnitt 3.5 ab Seite 48 aus den darin enthaltenen Informationen SQL-Statements (8), die die Daten aus den Nutzdatentabellen der Quellsysteme einlesen und in die i2b2-Datenbank schreiben (10). Mit OntoExport wird außerdem die i2b2-Ontologie aus der Zieldatensatz-Ontologie erzeugt.

Wenn wie in Erlangen ein Direktzugriff auf die Datenbanken der Quellsysteme möglich ist, können die Schritte (1) bis (4) entfallen. In diesem Fall sollte die Quellsystem-Ontologie durch eine Speziallösung erzeugt und den Tools zur Verfügung gestellt werden. Entsprechend der Umsetzung in Erlangen ist dies in Abbildung 4.2 durch ein SQL-Skript (11) dargestellt. Die Generierung der Soarian-Ontologie wird in Abschnitt 4.5 auf Seite 71 vorgestellt.

## 4.2 Erweiterung der Entity im EAV-Datenmodell

Bevor auf die Umsetzung der in Kapitel 3 entwickelten Methoden im Detail eingegangen werden kann, müssen zunächst einige Erkenntnisse, die in Bezug auf die Verarbeitung der Datensätze gemacht wurden, erläutert werden. In Abschnitt 3.5.2 auf Seite 50 wurde bereits darauf hingewiesen, dass diese Verarbeitung unter Verwendung des Entity-Attribute-Value-Datenmodells (EAV) erfolgen soll. Dieses ermöglicht es, klinische Datensätze mit einer großen Anzahl an Attributen in einer sehr einfachen Tabelle mit – theoretisch – drei Spalten zu speichern.

Statt die Ausprägungen spaltenweise aufzuführen,

<b>PatNr</b>	<b>Geschlecht</b>	<b>Wert 1</b>	<b>Wert 2</b>
654654	W	12	24
313321	M		3

würde man sie im EAV-Format fortlaufend untereinander auflisten:

<b>Entity</b>	<b>Attribute</b>	<b>Value</b>
654654	Geschlecht	W
654654	Wert 1	12
654654	Wert 2	24
313321	Geschlecht	M
313321	Wert 2	3

Auf diese Weise können leere Tabelleneinträge weggelassen werden (vgl. „Wert 1“ von Patient 313321), was bei besonders großen Tabellen, die nur spärlich ausgefüllt sind, Speicherplatz einspart (siehe [74]).

In diesem Beispiel wurde die Patienten-Nummer als „Entity“-Spalte verwendet. In der Praxis reicht dies jedoch noch nicht aus, da man einem Patienten noch eine Fall-Nummer und den Attributen ein Datum zuordnen muss. Dies sieht man beispielsweise in Tabelle 3.2 auf Seite 38 anhand der Spalten „FallNr“ und „Datum“. Hinsichtlich der späteren Datenverarbeitung ist es sinnvoll, diese Parameter als zusätzliche Spalten in die EAV-Tabelle mit aufzunehmen:

<b>Entity-PatNr</b>	<b>Entity-FallNr</b>	<b>Entity-Datum</b>	<b>Attribute</b>	<b>Value</b>
654654	555887	02.03.2008	Geschlecht	W
654654	555887	02.03.2008	Wert 1	12
654654	555887	02.03.2008	Wert 2	24

Die Spalten, die zusammen der Beschreibung der Entity dienen, werden im Folgenden als *Entity-Attribute* bezeichnet.

Im Rahmen dieser Arbeit hat es sich jedoch gezeigt, dass die Aufnahme der Fall-Nummer und des Dokumentations-Zeitpunktes noch nicht für eine eindeutige Zuordnung zwischen Entität, Attribut und Wert ausreichen, sobald mit Daten aus EDC-Systemen gearbeitet wird. Beispielsweise können für einen Patienten im gleichen Fall und am gleichen Tag mehrere Formulare des gleichen Typs angelegt werden. Mit den bisherigen Entity-Attributten ist es noch nicht möglich, die einzelnen Attribute-Value-Paare diesen Bögen zuzuordnen. Dies funktioniert erst, wenn man für die Formulare noch eine weitere Spalte anlegt:

<b>Entity-DocumentID</b>	<b>Entity-PatNr</b>	<b>Entity-FallNr</b>	<b>Entity-Datum</b>	<b>Attribute</b>	<b>Value</b>
41	654654	555887	02.03.2008	Geschlecht	W
41	654654	555887	02.03.2008	Wert 1	12
41	654654	555887	02.03.2008	Wert 2	24
42	654654	555887	05.03.2008	Geschlecht	W
42	654654	555887	05.03.2008	Wert 1	14
42	654654	555887	05.03.2008	Wert 1	20

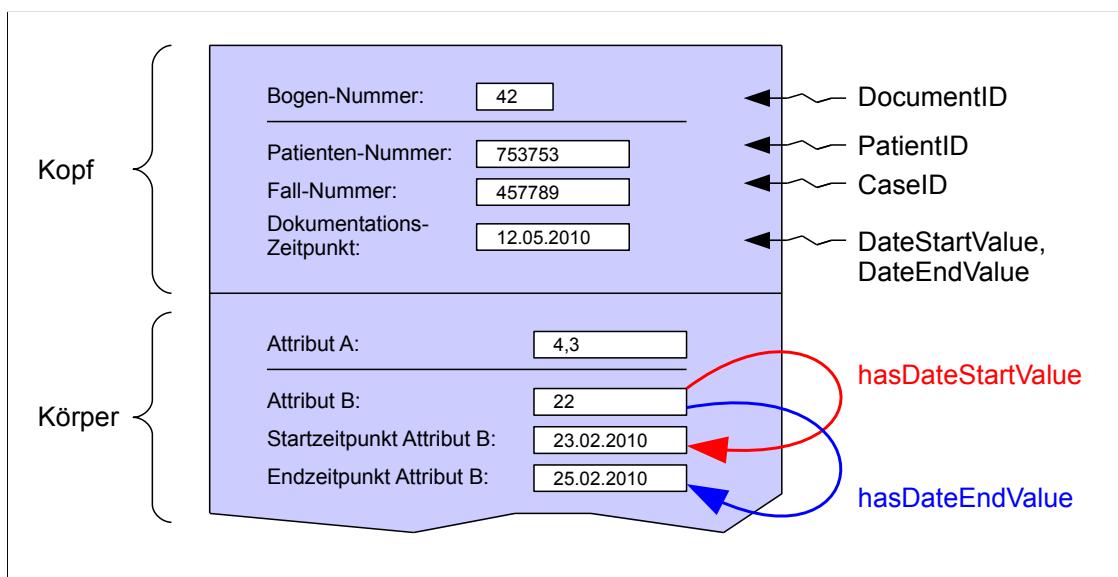
Durch die Spalte „Entity-DocumentID“ kann nun für jeden der Werte festgestellt werden, in welchem Formular (41 oder 42) er dokumentiert wurde. Dies ermöglicht es erst, die Datenelemente innerhalb eines Bogens miteinander zu verrechnen. Sollen beispielsweise Wert 1 und Wert 2 addiert werden, so ergibt dies 36 für den Bogen 41 und 34 für den Bogen 42. Ohne die Spalte „Entity-DocumentID“ wäre diese Zuordnung nicht möglich.

Darüber hinaus sind auch Beziehungen zwischen den einzelnen Datenelementen möglich. In Tabelle 3.2 auf Seite 38 ist dies anhand der Spalte „Datum Wert 3“ ersichtlich, die zusätzliche Datumsangaben für Spalte „Wert 3“ bereitstellt. Wenn mit den Datensätzen aus „Wert 3“ gearbeitet werden soll, ist diesen das Datum aus „Datum Wert 3“ zuzuordnen, nicht das aus Spalte „Datum“.

Abbildung 4.3 auf Seite 4.3 fasst diese Erkenntnisse nochmals zusammen. Um die dokumentierten Werte aus einem Bogen eindeutig einem Patienten zuordnen zu können, sind folgenden Entity-Attribute notwendig: „DocumentID“, „PatientID“ und „CaseID“, sowie „DateStartValue“ und „DateEndValue“.

„DateStartValue“ und „DateEndValue“ entstehen durch eine Auftrennung der Angabe „Dokumentations-Zeitpunkt“ in Start- und End-Zeitpunkt. Damit wird es bei Bedarf möglich, den Dokumentations-Zeitpunkt mit einem viel genaueren Entity-Attribute-Paar, bestehend aus „DateStartValue“ und „DateEndValue“, zu überschreiben. Dies wird in der Abbildung für das Attribut B durch die farbigen Pfeile illustriert, die die Datums-Angabe „12.05.2010“ mit den Werten „23.02.2010“ und „25.02.2010“ überschreiben. Für Attribut A gilt dagegen der 12.05.2010 als Start- und End-Zeitpunkt, da für dieses Attribut keine genaueren Zeitangaben erfasst wurden.

In der Umsetzung dieser Arbeit wird die ursprüngliche „Entity“-Spalte durch mehrere Spalten ersetzt, die den oben identifizierten fünf Entity-Attributen entsprechen.



**Abbildung 4.3:** Das Quintupel (DocumentID, PatientID, CaseID, DateStartValue, DateEndValue) steht für die Entity, für die alle Werte der im Bogen aufgelisteten Attribute erfasst wurden. Es sind jedoch auch Ausnahmen möglich, da einzelne Parameter der Entity durch andere Attribute „überschrieben“ werden können (rot, blau).

### 4.3 Erzeugung und Bearbeitung von Zieldatensatz-Ontologien mit OntoEdit

Wie bereits in Abschnitt 3.2 auf Seite 32 im Zusammenhang mit der Zieldatensatz-Ontologie hervorgehoben wurde, bestand das Hauptziel dieser Arbeit in einem Export von klinischen Daten nach i2b2. Die Auflistung und die Definition der hierfür erforderlichen Datenelemente erfolgt dabei in einem gemeinsamen Zieldatensatz, dargestellt durch die Zieldatensatz-Ontologie.

Wie in Kapitel 3 beschrieben wurde, werden beim Mapping-Prozess *hasImport*-Beziehungen zwischen den Dokumentationselementen der Zieldatensatz-Ontologie und den Zwischenergebnissen der Mapping-Ontologie aufgebaut, anhand derer anschließend die Nutzdaten in die i2b2-Datenbank geladen werden. Neben diesem eigentlichen Import müssen jedoch auch Metadaten in i2b2 bereitgestellt werden, damit strukturiert auf die Daten mit i2b2 zugegriffen werden kann. Diese Metadaten werden in i2b2 in einer Taxonomie verwaltet, die in der i2b2-Workbench als aufklappbare Baumstruktur bereitgestellt wird – vergleichbar mit einem herkömmlichen Dateisystem-Browser eines Betriebssystems (vgl. Abbildung 3.4 auf Seite 33, links). Aus den Elementen können dann mit benutzerfreundlichen Drag&Drop-Operationen i2b2-Datenbank-Queries konstruiert werden.

Es hat sich also angeboten, die Zieldatensatz-Ontologie so zu gestalten, dass sie automatisch in das i2b2-Metadatenformat überführt werden kann. Dabei ist es offensichtlich, dass in der Zieldatensatz-Ontologie so viele Informationen enthalten sein müssen, wie von i2b2 anschließend verarbeitet werden sollen.

Die Darstellung der Zieldatensatz-Datenelemente musste also in Form einer Taxonomie erfolgen. i2b2 speichert diese Taxonomie in einer Datenbanktabelle, wobei es alle Pfade von der Wurzel zu den Blättern durch Zeichenketten darstellt, die durch Backslashes getrennt sind. Dies wird exemplarisch in Tabelle 4.1 auf Seite 66 für die Taxonomie der Quellsystem-Ontologie in Abbildung 3.12 auf Seite 45 gezeigt.

Die Generierung derartiger Einträge aus einer OWL-Klassenhierarchie ist einfach: Für die Wurzelklasse der Zieldatensatz-Ontologie wird überprüft, ob sie Unterklassen hat. Wenn ja, wird für jede dieser Unterklassen eine Funktion aufgerufen, die sich selbst so lange rekursiv aufruft, bis keine Unterklassen mehr gefunden werden. Bei jedem Rekursionsschritt wird der jeweils aktuelle Klassename an eine Zeichenkettenvariable angehängt, die bei der Rekursion weitergegeben wird. Ist die Abbruchbedingung erreicht, kann der Inhalt der Zeichenkette in die „c\_fullname“-Spalte der i2b2-Metadatentabelle geschrieben werden.

Über diese Hierarchie werden die Konzepte in i2b2 lediglich strukturiert abgelegt. Zur weiteren Beschreibung müssen in den anderen Spalten und in einem XML-Feld weitere

c_level	c_fullname	c_basecode
0	\i2b2\	
1	\i2b2\Zieldatensatz-Element\	
2	\i2b2\Zieldatensatz-Element\Demographische_Daten\	
3	\i2b2\Zieldatensatz-Element\Demographische_Daten\Geschlecht\	
4	\i2b2\Zieldatensatz-Element\Demographische_Daten\Geschlecht\Weiblich\	SEX:F
4	\i2b2\Zieldatensatz-Element\Demographische_Daten\Geschlecht\Männlich\	SEX:M
3	\i2b2\Zieldatensatz-Element\Demographische_Daten\Alter\	
2	\i2b2\Zieldatensatz-Element\Laborwerte\	
3	\i2b2\Zieldatensatz-Element\Laborwerte\Laborwert_1\	LAB:Val1
4	\i2b2\Zieldatensatz-Element\Laborwerte\Laborwert_2\	LAB:Val2
4	\i2b2\Zieldatensatz-Element\Laborwerte\Summe_1\	CALC:Sum1

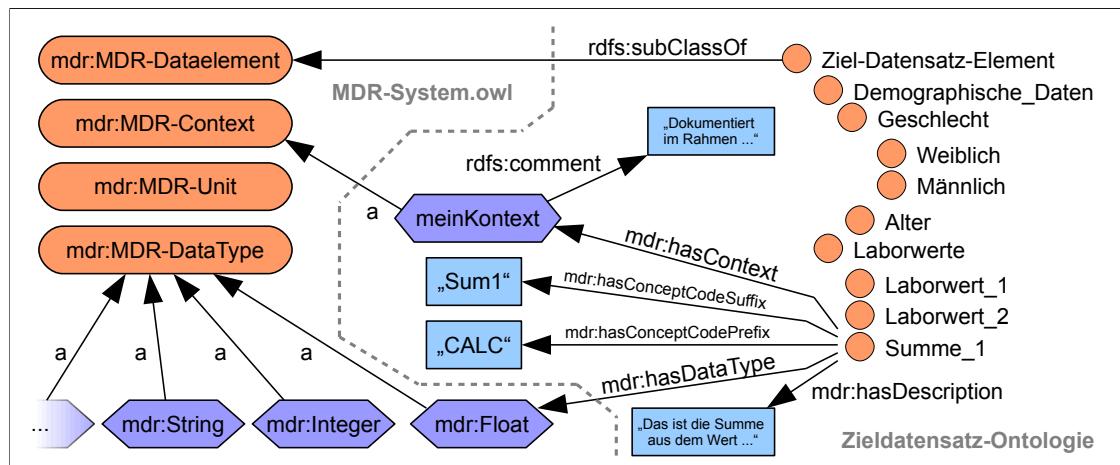
**Tabelle 4.1:** Beispielhafte Taxonomie im i2b2-Format, erzeugt aus der Quellsystem-Ontologie in Abbildung 3.12 auf Seite 45. Die restlichen Spalten werden in Tabelle C.8 auf Seite 140 gezeigt und erläutert.

Informationen hinterlegt werden (siehe Tabellen C.8 und C.9 ab Seite 140). Im Rahmen der Arbeit wurden die restlichen Spalten und das XML-Feld genauer untersucht. Seit i2b2 Version 1.5 liegt der Workbench ein Ontologie-Editor einschließlich einer kurzen Dokumentation bei, mit dem i2b2-Ontologie-Einträge bearbeitet werden können. Beides hat sich bei der Analyse der i2b2-Ontologie als nützlich erwiesen.

Es wurden mehrere Attribute ermittelt, die den einzelnen Elementen der Zielsystem-Ontologie über Datatype- und Object-Properties zugewiesen werden können. Die Definitionen dieser Properties, sowie die verschiedener Klassendefinitionen, wurden in eine Ontologie mit der Bezeichnung **MDR-System** ausgelagert. Anschließend kann diese in die Zieldatensatz-Ontologie importiert und deren Konstrukte zur Definition des Zieldatensatzes verwendet werden.

Abbildung 4.4 auf Seite 67 illustriert dies an einem Beispiel. Dem Element **Summe\_1** ist über die Datatype-Properties **mdr:hasConceptCodePrefix** und **mdr:hasConceptCodeSuffix** ein i2b2-Concept-Code „SUM:Calc“ zugewiesen (vgl. auch Tabelle 4.1 auf Seite 66). Durch die Beziehung **mdr:hasDatatype** **mdr:Float** wird festgehalten, dass es sich um eine Gleitkommazahl handelt. Diese Information wird beim Erzeugen der i2b2-Ontologie in das XML-Feld eingetragen, damit die numerischen Datensätze von **Summe\_1** über die i2b2-Workbench auch ausgewertet werden können.

Über die Property **mdr:hasDescription** kann dem Element eine textuelle Beschreibung zugewiesen werden und über **mdr:hasContext** der Kontext, in dem das Datenelement verwendet wird. In weiteren, hier nicht gezeigten Properties können noch Wertebereiche



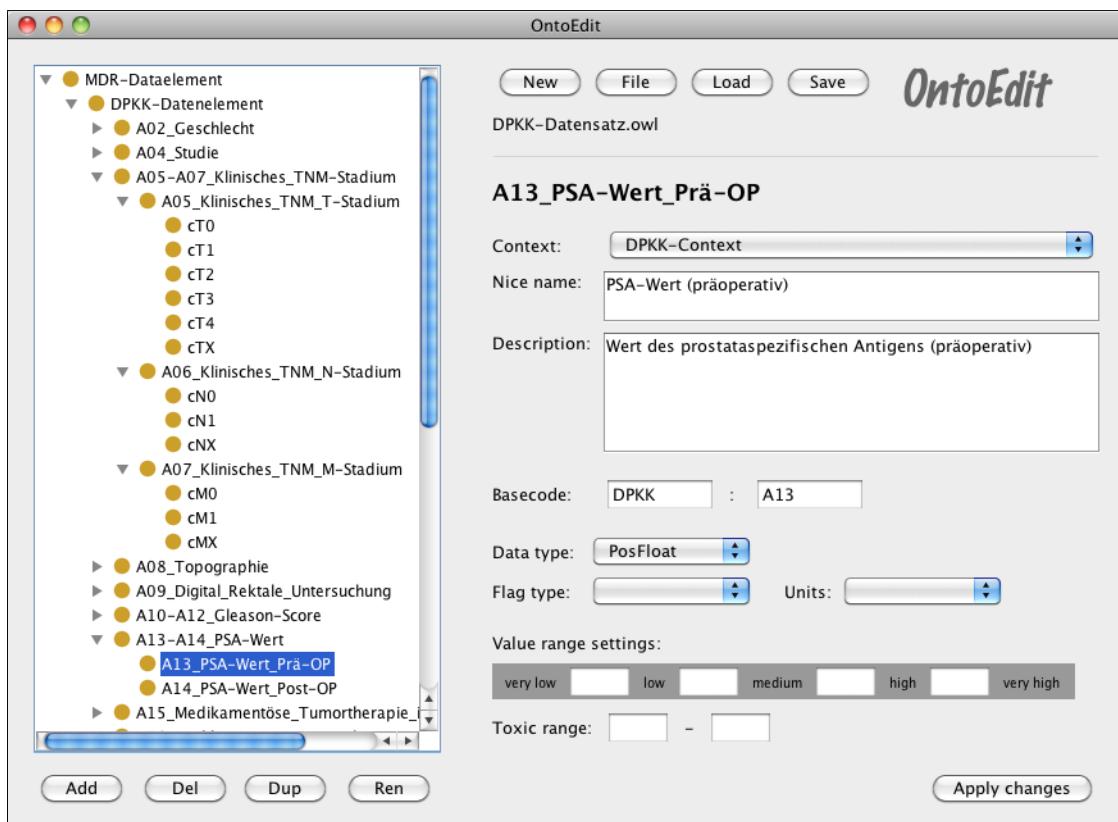
**Abbildung 4.4:** Die Ontologie „MDR-System“ wird in die Zieldatensatz-Ontologie importiert und stellt Properties und Klassen zur Beschreibung einer i2b2-kompatiblen Datensatzbeschreibung zur Verfügung.

für Laborwerte oder der toxische Bereich einer Medikation angegeben werden (siehe hierfür Tabelle C.11 auf Seite 144 und Tabelle B.3 auf Seite 126).

Neben den Properties gibt es vier Klassen. `mdr:MDR-Dataelement` ist die Wurzelklasse, in der die i2b2-Taxonomie anzulegen ist, `mdr:MDR-Context` ist ein Container für alle Kontexte und in `mdr:MDR-DataType` werden die von i2b2 verarbeitbaren Datentypen bereitgestellt.

Die Elemente der Klasse `mdr:MDR-Unit` sollen Informationen über Einheiten und deren Umrechnungsfaktoren enthalten, beispielsweise, dass eine mit 1000 multiplizierte Literangabe einer Milliliterangabe entspricht. i2b2 kann derartige Umrechnungen durchführen, wenn man solche Informationen in der i2b2-Ontologie bereitstellt (siehe XML-Tag `<UnitValues>` in Tabelle C.9 auf Seite 141). Statt diese Informationen jedoch einzeln über Datenwerte in der Zieldatensatz-Ontologie abzubilden, erscheint es sinnvoller, eine extra Einheiten-Ontologie aufzubauen, aus der diese Informationen „automatisch“ abgeleitet werden können. Dieser Schritt konnte im Rahmen dieser Arbeit jedoch nicht mehr erfolgen. Ähnliches gilt für die Verwaltung von synonymen Begriffen und für den i2b2-Datentyp „Enum“.

Die Herleitung der Properties ist in den Tabellen C.10 und C.11 ab Seite 143 aufgelistet, die Inhalte der vollständigen MDR-System-Ontologie in Abschnitt B.1 ab Seite 125. Es wurde außerdem festgestellt, dass sich viele der zu hinterlegenden Einträge aus anderen Informationen ableiten lassen und nicht über extra Properties in die Quellsystem-Ontologie mit aufgenommen werden müssen. Beispielsweise kann das XML-Attribut `<Oktousevalues>` immer auf „Y“ gesetzt werden, wenn in der Spalte „valtype\_cd“ festgehalten wurde,



**Abbildung 4.5:** OntoEdit: soll eine effiziente Erstellung oder Nachbearbeitung der Zieldatensatz-Ontologien ermöglichen. Diese Ontologien werden anschließend durch OntoExport in das i2b2-Metadatenformat übersetzt.

dass es sich bei dem Datenelement um einen numerischen Wert handelt – Schließlich möchte man es dem Anwender auch ermöglichen, nach numerischen Werten zu suchen, wenn man solche in die i2b2-Datenbank lädt.

Die Zieldatensatz-Ontologie sollte ursprünglich direkt mit Protégé bearbeitet werden. Es hat sich jedoch schnell gezeigt, dass dies damit nur recht umständlich zu bewerkstelligen ist. Da der DPKK-Datensatz bereits aus 172 Elementen (Attribute und Ausprägungen) bestand, sollte die Bearbeitung mit einer kleinen Eigenentwicklung beschleunigt werden. Das Ergebnis, *OntoEdit*, ist in Abbildung 4.5 zu sehen.

Das Tool lädt die Unterklassen von `mdr:MDR-Dataelement` in einen JTree auf der linken Seite, in dem die Klassenhierarchie über Steuerbuttons oder Drag&Drop-Operationen bearbeitet werden kann. Auf der rechten Seite können die Datenwerte für die oben beschriebenen Datatype-Properties in Texteingabe-Felder eingegeben werden; für Object-Properties stehen Drop-Down-Listen zur Verfügung, in die die Bezeichner der einzelnen

Instanzen geladen und durch den Benutzer ausgewählt werden können. Beispielsweise werden alle Instanzen der Klasse `mdr:MDR-DatType` in die Drop-Down-Liste rechts neben „Data type“ geladen. Dass mit OntoEdit dabei eigentlich OWL-Ontologien bearbeitet werden, bleibt dem Benutzer völlig verborgen.

#### 4.4 Bereitstellung von CSV-Daten und Erzeugung von Ontologien mit OntoGen

Für den Fall, dass kein Direktzugriff auf die Datenbanken der Quellsysteme möglich ist, wurde das Tool *OntoGen* implementiert. In der Regel erlauben klinische Informationssysteme einen Datenexport im CSV-Format, bei dem die einzelnen Attribute in den Spalten nebeneinander stehen. Dieses Formats wurde in Abschnitt 3.3.2 auf Seite 37 ausführlich vorgestellt.

OntoGen verwendet die dort ebenfalls gezeigte Methodik, um aus den Spaltenüberschriften und den aggregierten Ausprägungen der Tabelle eine Quellsystem-Ontologie zu generieren. Ebenso wird eine Ziel-System-Ontologie erstellt, die im Wesentlichen der Quellsystem-Ontologie entspricht. Die Elemente der Quellsystem- und Ziel-System-Ontologie werden durch die Software mittels einfacher 1:1-Mappings verknüpft. Abbildung 4.6 auf Seite 70 zeigt die erzeugten Ontologien, die aus der CSV-Datei in Abbildung B.1 auf Seite 132 gewonnen wurden.

Die Zielsystem-Ontologie kann anschließend mit dem in Abschnitt 4.3 auf Seite 65 gezeigten Tool OntoEdit bearbeitet werden; die einfachen 1:1-Mappings können mit QuickMapp (siehe nächster Abschnitt) geändert werden. Beispielsweise könnte man dann die Einträge `Sex-F` und `Sex-M` auf der linken Seite mit OntoEdit löschen, und anschließend die `Sex-F` und `Sex-M` (rechts) auf `Sex-Female` und `Sex-Male` (links) mappen.

Vor der Verarbeitung der CSV-Datei durch OntoGen ist es jedoch erforderlich, eine zweite Zeile in die Datei einzufügen, aus denen OntoGen die Datentypen der einzelnen Spalten ableiten kann. Zur Verfügung stehen die i2b2-Datentypen (Float, PosFloat, Integer, PosInteger, String). Wenn eine Zelle leer bleibt, wird angenommen, dass es sich um eine Typ-1-Spalte mit aufzählbaren Ausprägungen handelt. In diesem Fall werden alle Einträge der Spalte aggregiert und als Konzepte mit in der Zieldatensatz-Ontologie angelegt (ersichtlich in Abbildung 4.6 am Attribut „Sex“).

Darüber hinaus müssen die Entity-Attribute (siehe Abschnitt 4.2 auf Seite 62) mit angegeben werden (DocumentID, GlobalDate, PatientID)<sup>1</sup>. Liegen diese Informationen

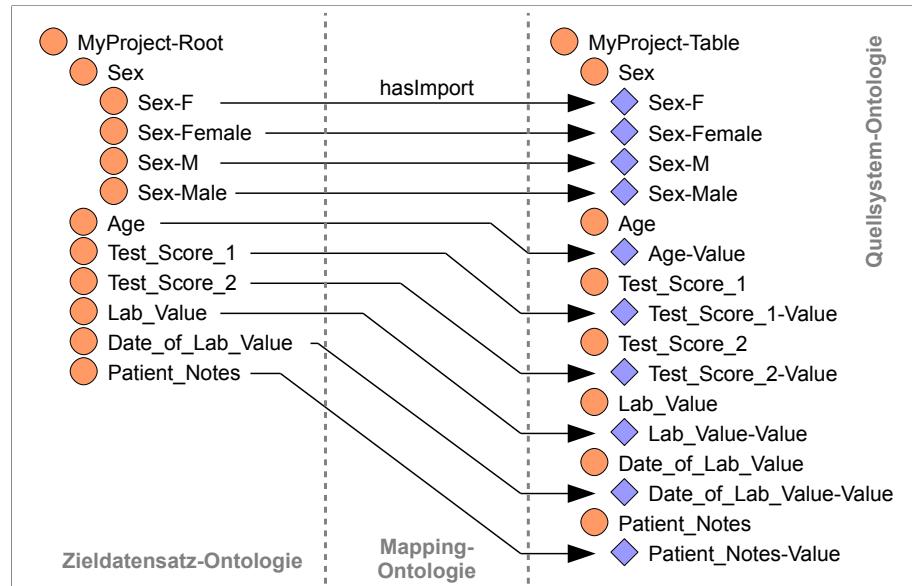
---

<sup>1</sup> Die Implementierung unterscheidet noch nicht zwischen „DateStartValue“ und „DateEndValue“, sondern stellt nur den Dokumentations-Zeitpunkt über „GlobalDate“ bereit. Ebenso arbeitet das System noch ohne Unterstützung für Fallnummern.

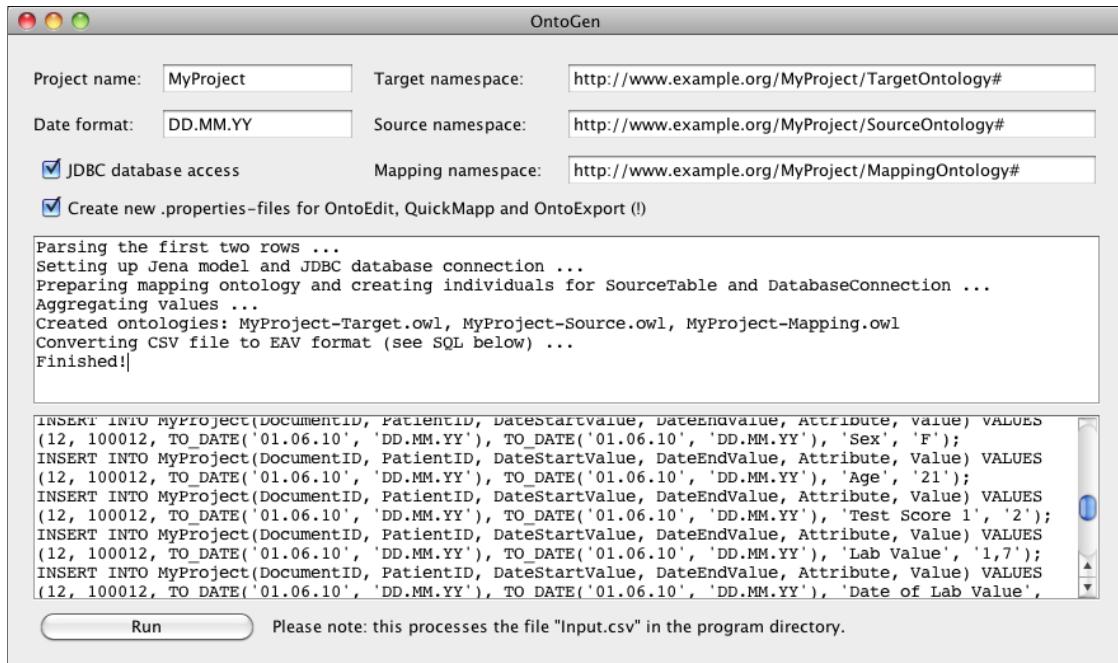
nicht vor, muss eine zusätzliche Spalte mit Dummy-Werten eingefügt werden. Gibt es beispielsweise keine ID, anhand derer das Formular identifiziert wird (weil möglicherweise nur ein Formular pro Patient angenommen wird), so kann einfach eine fortlaufende Nummer für die „DocumentID“-Spalte dienen (siehe Abbildung B.1 auf Seite 132).

Wie bereits in Abschnitt 3.3.2 auf Seite 37 festgestellt wurde, werden nur Ausprägungen in beide Ontologien aufgenommen, die auch in der Tabelle vorkommen. Ist jedoch bekannt, welche Ausprägungen fehlen, können Zeilen mit „Dummy-Patienten“ angelegt werden, in denen die restlichen Ausprägungen dokumentiert werden.

Abbildung 4.7 auf Seite 71 zeigt einen Screenshot von OntoGen. Rechts oben können die Namespaces der drei Ontologien angegeben werden. Im unteren Textfeld werden die SQL-Statements ausgegeben, welche die Daten im EAV-Format in eine temporäre Datenbank schreiben.



**Abbildung 4.6:** OntoGen erzeugt aus den Spaltenüberschriften einer CSV-Datei eine Zieldatensatz-, Mapping- und Quellsystem-Ontologie, nachdem es die Daten ins EAV-Format „gekippt“ und in eine temporäre Datenbank geladen hat. Der Inhalt der CSV-Datei ist in Abbildung B.1 auf Seite 132 zu sehen.



**Abbildung 4.7:** OntoGen kippt eine CSV-Datei im spaltenweisen Form in das EAV-Format und erzeugt die Quellsystem-, Zieldatensatz- und Mapping-Ontologie.

## 4.5 Erzeugung der Soarian-Ontologie für Erlangen

Bevor auf die Implementierung der Tools QuickMapp und OntoExport eingegangen werden kann, wird gezeigt, wie die Quellsystem-Ontologie für Soarian erzeugt wurde, da dieser Arbeitsschritt die Entwicklung dieser Tools maßgeblich beeinflusst hat.

In Abschnitt 2.2.1 wurden Eigenheiten von Soarian vorgestellt, insbesondere die Versionierung von Formularen und Components, sowie die Wiederverwendbarkeit von Components in anderen Bögen. Die Soarian-Ontologie muss diese Eigenschaften mit abbilden. Beispielsweise müssen alle unterschiedlichen Component-Versionen vorhanden sein, damit man anschließend beim Mapping überprüfen kann, ob zwei Ausprägungen das gleiche medizinische Konzept beschreiben. Ebenso muss es möglich sein zu unterscheiden, in welchem Formular die Component verwendet wurde, wenn man nur Datensätze von einem bestimmten Formular exportieren möchte.

In Abschnitt 3.3.2 auf Seite 38 wurde eine einfache Methodik vorgestellt, mit der man aus der hierarchischen Struktur eines Formulars eine Klassenhierarchie gewinnen kann. In Erlangen ist der dafür erforderliche Zugriff auf die Metadaten des Quellsystems durch den Soarian-DWH-Export (siehe Abschnitt 2.2.1 auf Seite 9) bereits gegeben.

Die Erzeugung der Soarian-Ontologie konnte im Rahmen dieser Arbeit leicht durch ein einfaches SQL-Skript erfolgen, das mit SELECT-Statements auf die DWH-Export-Tabelle DWH\_METADATEN\_ONTOLOGY zugreift (siehe C.1 auf Seite 134) und die RDF-Tripel durch Konkatenation der einzelnen Spalten konstruiert. Diese werden durch das SQL-Skript in eine einspaltige Tabelle eingefügt. Anschließend werden sie aus der Datenbank geladen, durch ein Bash-Skript geringfügig umformatiert und um den OWL-Ontologie-Kopf mit den Namespace-Angaben ergänzt.

Die Soarian-Ontologie besteht aus einer Klassenhierarchie, die sich aus Bögen, Components und Ausprägungen zusammensetzt. Da die Bezeichner dieser Elemente aufgrund des Zeichensatzes nicht ohne weiteres in RDF abbildbar sind, werden die Ontologie-Elemente teilweise durch Kombination der numerischen IDs zusammengesetzt. Zur Erzeugung der Soarian-Ontologie sind innerhalb des Skripts folgende Schritte nötig:

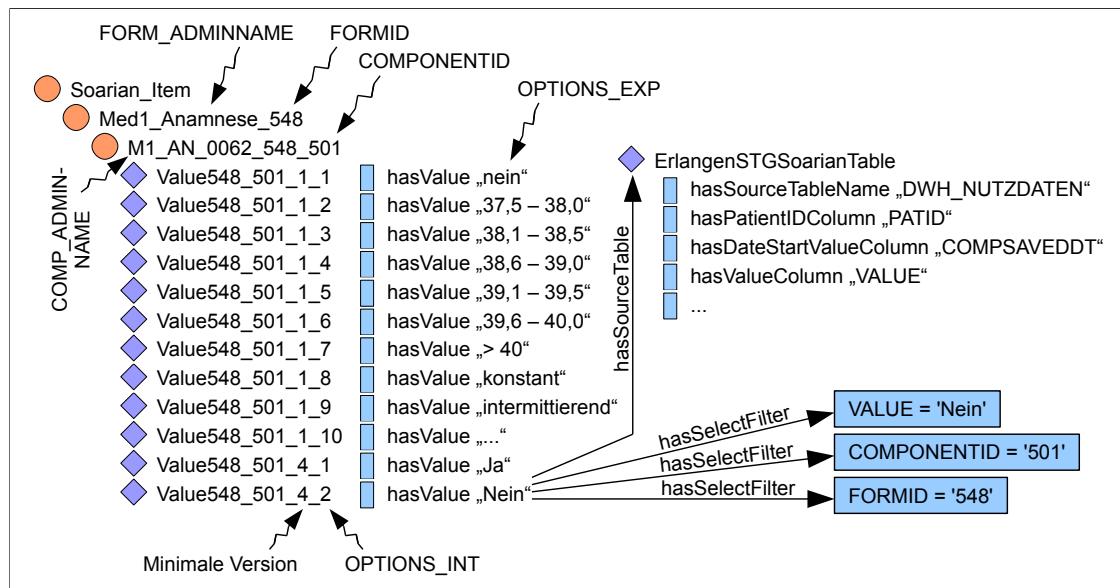
1. Benenne alle Einträge in der Tabelle so um, dass daraus RDF-Resourcen (RDFS-Klassen, Instanzen) erzeugt werden können. Hierzu werden alle Zeichen, außer A-Z, a-z und 0-9, durch einen Unterstrich ersetzt. Oracle-SQL stellt für diese Transformation die Funktion REGEXP\_REPLACE zur Verfügung.
2. Erzeuge durch SQL-Statements folgende Tripel, die in eine temporäre Tabelle ONTOLOGY\_N3 geschrieben werden:
  - a) Lege eine Wurzel-Klasse **Soarian\_Item** an.
  - b) Lege alle Soarian-Bögen als Unterklassen der Wurzel-Klasse an. Ihr Name setzt sich aus den Spalten „Form\_AdminName“ und „FormID“ zusammen.
  - c) Lege alle Soarian-Components als Unterklassen der jeweiligen Bogen-Klassen an. Ihr Name setzt sich aus den Spalten „Comp\_AdminName“, „FORMID“ und „ComponentID“ zusammen. Durch die Aufnahme der FormID kann die Verwendung der Component im jeweiligen Bogen zugeordnet werden.
  - d) Lege alle Ausprägungen der Typ-1-Components als Instanzen der jeweiligen Component-Klasse an. Ihr Name setzt sich aus dem String „Value“, den Spalten „FormID“ und „ComponentID“, der minimalen Version, in der die Ausprägung zum ersten mal aufgetreten ist und der Spalte „Options\_Int“ zusammen (siehe Beispiel unten).
  - e) Lege für die Typ-2-Components eine einzige Instanz in der jeweiligen Component-Klasse an, die sich wie bei den Typ-1-Components zusammensetzt – nur ohne „Options\_Int“.

Abbildung 4.8 zeigt die im Kasten beschriebene Namensgebung nochmals anhand der Dokumentationselemente.

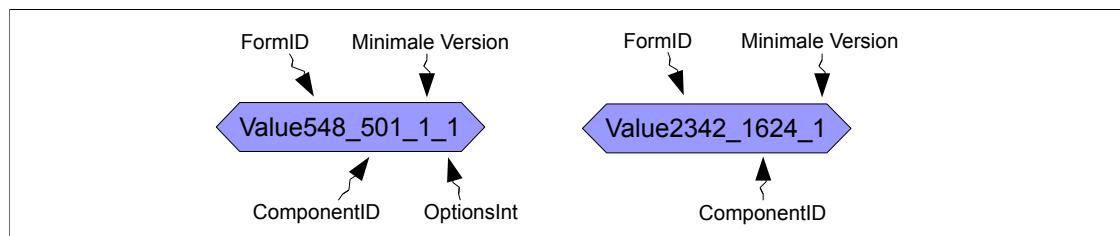
In Abbildung 4.9 wird der komplette Ausschnitt für die Fieber-Component gezeigt, die bereits in Abschnitt 2.2.1 auf Seite 12 vorgestellt wurde. Die Metadaten, aus denen dieser Ontologieteil erzeugt wurde, sind in Tabelle C.4 auf Seite 136 zu sehen. Da bei Version 1 und 2 alle Elemente gleich sind, werden nur Version 1 und Version 4 in die Ontologie aufgenommen.

Die Soarian-Ontologie stellt darüber hinaus alle Informationen zur Verfügung, aus denen SQL-Statements zum Zugriff auf die Nutzdaten (siehe Abschnitt 3.5.2 auf Seite 50) erzeugt werden können. Beispielsweise werden die gezeigten Datenwerte der *hasSelectFilter*-Properties zu einer WHERE-Klausel kombiniert.

Mit der gezeigten Technik kann täglich eine neue Soarian-Ontologie erzeugt und den anderen Tools zur Verfügung gestellt werden. Tabelle 4.2 gibt abschließend noch einen Überblick über die Anfang Dezember 2010 enthaltene Menge an Klassen und Instanzen.



**Abbildung 4.9:** Repräsentation der Fieber-Component innerhalb der Soarian-Ontologie. Für jedes Element sind Informationen hinterlegt, mit denen eine Software auf die Nutzdaten zugreifen kann.



**Abbildung 4.8:** Namensgebung der Soarian-Dokumentationselemente vom Typ 1 (links) und Typ 2 (rechts). Die Namen werden durch Zusammenfügen mehrerer Spalten erzeugt.

	Bögen	Attribute (Typ 1)	Ausprägungen (Typ 1)	Attribute (Typ 2)	Ausprägungen (Typ 2)
<b>Urologie</b>	15	454	1786	359	360
<b>Soarian komplett</b>	532	7440	30875	10729	11243

**Tabelle 4.2:** Statistik über die Anzahl der Elemente in der Soarian-Quellsystem-Ontologie (Stand: Anfang Dezember 2010). Hervorgehoben sind die Dokumentationselemente, die restlichen Einträge bilden die RDFS-Klassenhierarchie.

## 4.6 Erzeugung und Bearbeitung von Mapping-Ontologien mit QuickMapp

In Abschnitt 3.4 auf Seite 44 wurden umfangreiche Überlegungen dazu angestellt, wie Mappings ohne oder mit Transformationsregeln zwischen der Quellsystem- und Zieldatensatz-Ontologie aufgebaut werden können. Die erste Variante sollte durch einfache *hasImport*-Verknüpfungen erfolgen, die zweite über Zwischenknoten in der Mapping-Ontologie (siehe Kapitel 3.4.2 ab Seite 45).

Auch hier hat es sich schnell herauskristallisiert, dass ein Erstellen von komplexeren Mappings mit Standard-Editoren wie Protégé kaum möglich ist. Zwar können mit Protégé beliebige RDF-Ressourcen miteinander verknüpft werden; jedoch ist dies sehr zeitaufwändig.

Problematisch ist auch die Suche nach Ontologie-Elementen, wenn man deren Namen nicht kennt. Oben wurde beschrieben, dass die Soarian-Ontologie über 42.000 Dokumentationselementen enthält, deren Bezeichner nur aus numerischen IDs bestehen. Obwohl den einzelnen Elementen der Soarian-Ontologie über *hasNiceName*-Relationen die entsprechenden Bezeichner zugeordnet sind, können diese nicht so leicht mit Protégé gesucht werden. Auch die strikte Trennung von Klassen und Instanzen erschwert die Arbeit.

Als Lösung wurde das Programm QuickMapp entwickelt, dass den Benutzer beim Mapping-Vorgang so gut wie möglich unterstützen soll. Das Programmfenster ist in Abbildung 4.10 gezeigt.

Das Tool lädt die Mapping-, Zieldatensatz- und Quellsystem-Ontologie. Die Zieldatensatz-Ontologie wird auf der linken Seite in einem kombinierten Klassen-/Instanzenbrowser dargestellt, die Quellsystem-Ontologie auf der rechten. Unter jedem der Browser befindet sich eine Statement-Tabelle, in der alle Properties mit den dazugehörigen Ressourcen des je-

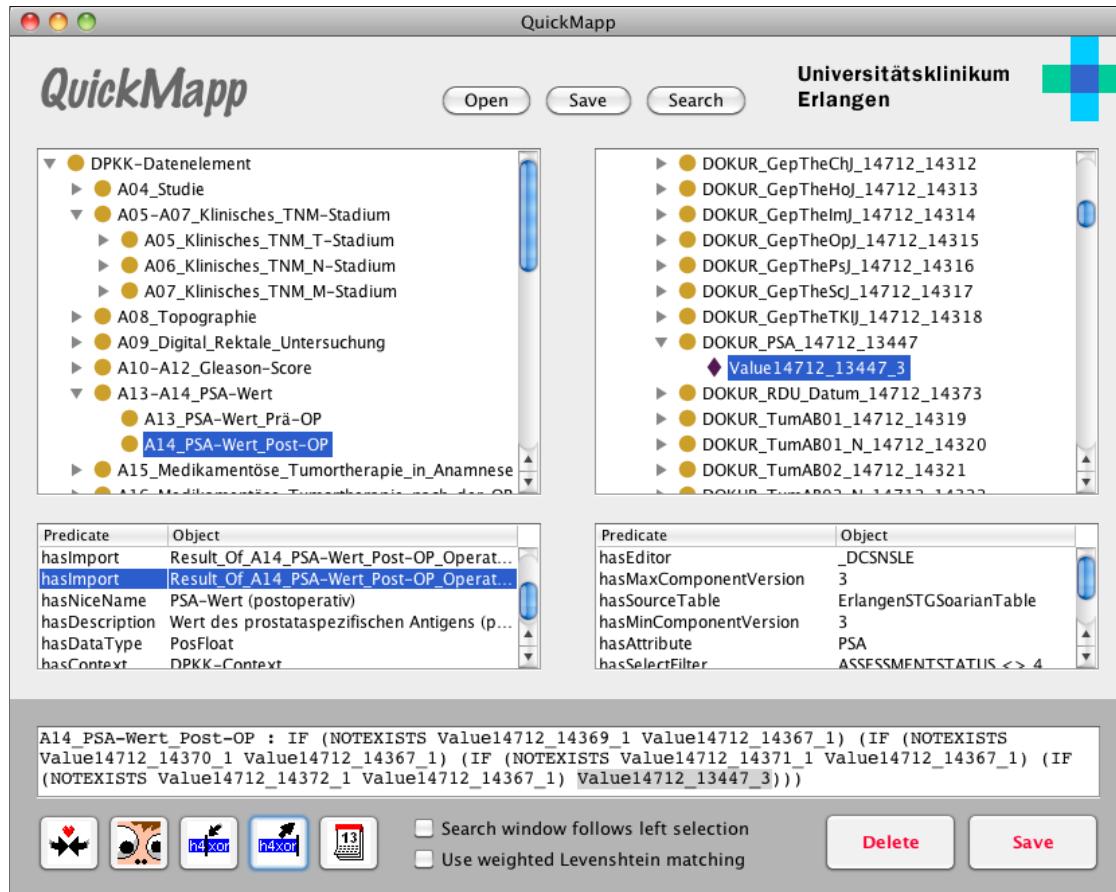


Abbildung 4.10: QuickMapp: erzeugt Mappings zwischen dem Zieldatensatz (links) und dem Quellsystem-Datensatz (rechts).

weils darüber ausgewählten Elementen angezeigt werden. Die Properties `rdfs:subClassOf` und `rdf:type` werden dabei herausgefiltert, um die Tabellen möglichst übersichtlich zu halten.

Im unteren Bereich befindet sich ein Text-Editor, in den man einen Mapping-Code eingeben kann. Dieser wird beim Klick auf den Button „Save“ von QuickMapp in ein Mapping-Netz (siehe Abschnitt 3.4 ab Seite 44) übersetzt und in die Mapping-Ontologie mit aufgenommen. Die erstellten Mappings erkennt man in der linken Statement-Tabelle anhand der `hasImport`-Beziehungen zu Dokumentationselementen der Quellsystem-Ontologie oder zu Zwischenknoten der Mapping-Ontologie.

Über die Buttons unten links können die in den Klassen-/Instanzenbrowsern ausgewählten Elementen in den Text-Editor eingefügt werden, um das Erstellen des Mapping-Codes zu vereinfachen. Umgekehrt können auch Dokumentationselemente im Text markiert

und in den Browsern angezeigt werden. Über den Kalender-Button können Relationen zwischen zwei Dokumentationselementen modelliert werden. Beispielsweise kann damit erfasst werden, dass ein Element das Datum eines anderen darstellt (vgl. Abschnitt 4.2 auf Seite 62).

Die Erstellung von Mappings ist einfach. Direkte Mappings ohne Transformationen können durch einfache Ausdrücke der Form

```
Ziel-Element : Dokumentationselement
```

angelegt werden. Sollen Mappings mit Transformationen angegeben werden, so muss dies in einer Prefix-Notation erfolgen:

```
Ziel-Element : ADD Element1 Element2
```

Dieser Ausdruck addiert „Element1“ und „Element2“. Ein äquivalentes Mapping-Netz ist in Abbildung 3.13 auf Seite 46 zu sehen. Durch Klammerung können Ausdrücke beliebig verschachtelt werden:

```
Ziel-Element : ADD Element1 (ADD Element2 Element3)
```

Ein vergleichbares Netz für diesen Ausdruck wurde in Abbildung 3.14 auf Seite 3.14 gezeigt. Datenwerte können eingebunden werden, indem man sie in Anführungszeichen setzt:

```
Ziel-Element : MULT Element1 "1000"
```

Dieser Ausdruck multipliziert das Element1 mit 1000. Dabei wird nicht zwischen numerischen und textuellen Werten unterschieden. Die Einbindung in das semantische Netz wurde in Abschnitt 3.4.3 auf Seite 47 gezeigt. Die Ausdrücke können nahezu beliebig komplex werden:

```
Greetings : TAIL (HEAD Patient_Notes-Value  
SUBTR (STRPOS Patient_Notes-Value "]") "1"))  
(ADD (STRPOS Patient_Notes-Value "[" ) "1")
```

Dieser Ausdruck ist auf den Test-Datensatz in Abbildung B.1 auf Seite 132 anzuwenden. Er extrahiert aus der Spalte „Patient\_Notes-Value“ die rot hervorgehobenen, zwischen eckigen Klammern stehenden Grußbotschaften. Die Abarbeitung funktioniert folgendermaßen: Zuerst wird mit dem **STRPOS** in der zweiten Zeile die Position der schließenden eckigen Klammer ermittelt. Da die eckigen Klammern nicht mit exportiert werden sollen, wird von dieser Position mit dem Operator **SUBTR** der Wert 1 abgezogen. Der Befehl **HEAD** liefert dann die Teilzeichenkette von „Patient\_Notes-Value“ bis zu der berechneten Position zurück und verwirft den Rest. Die neue Zeichenkette wird anschließend an den Operator **HEAD** übergeben, der sie hinter der öffnenden eckigen Klammer „abschneidet“ und den vorderen Teil verwirft. Diese Position wird über den anderen Ausdruck in der dritten Zeile durch **STRPOS** und **ADD** ermittelt. Als Ergebnis werden dann folgende Einträge zurückgeliefert:

PatientID	Value
100010	Hello World
100011	Hallo DPKK
100012	Huhu IMI

Auch wenn der Nutzen in diesem Beispiel zweifelhaft erscheinen mag, demonstriert er doch, welch komplexe Operationen mit QuickMapp modelliert werden können. Auf diese Weise kann auch effizient auf fehlerhafte Datensätze zugegriffen werden.

Derartige Zeichenketten-Operationen können auch dazu verwendet werden, um eine Art primitives *Natural Language Processing* (NLP) zu implementieren. Damit beispielsweise die rauchenden Patienten des Test-Datensatzes ermittelt werden, können mehrere Mapping-Ausdrücke mit Schlüsselwörtern nach dem folgenden Schema angelegt werden:

```
Raucher : INSTR Patient_Notes-Value "Morley"
Raucher : INSTR Patient_Notes-Value "lung cancer"
Raucher : INSTR Patient_Notes-Value "smoking"
```

Diese Ausdrücke liefern ein „TRUE“ zurück, wenn das Schlüsselwort in Operand 2 eine Teilzeichenkette von Operand 1 ist. Beim Datenexport werden die gefundenen Datensätze mit nach i2b2 exportiert; in i2b2 steht dann das Konzept „Raucher“ zur Verfügung. Natürlich ergibt das Raucher-Beispiel auch erst Sinn, wenn die Schlüsselworte in mehreren Datensätzen als in einem einzigen vorkommen.

Sobald der Benutzer auf den Save-Button klickt, wird der Ausdruck von QuickMapp geparsst und in das Mapping-Netz umgewandelt. Aufgrund der einfachen Klammersprache gestaltete sich die Implementierung einfach: Die Parser-Routine versucht, zuerst den

Operator einzulesen, dann die Operanden. Sobald sie jedoch auf eine öffnende Klammer trifft, wird der eingeklammerte Ausdruck eingelesen und rekursiv an die Parser-Routine selbst übergeben. Ist eine innere Klammer erreicht, wird die Rekursion wieder verlassen. Während der Abarbeitung legt die Routine für jeden Operator einen Zwischenknoten an und verbindet sie durch die Properties *hasOperand1* und *hasOperand2*. Die Namen der Zwischenknoten werden dabei so gebildet, dass sie eindeutig sind.

Zum Schluss sei noch angemerkt, dass QuickMapp die einzelnen Befehle selbst nicht interpretieren kann. Werden unbekannte Befehle verwendet, erzeugt QuickMapp ein syntaktisch korrektes Mapping-Netz, das allerdings semantisch ins „Leere“ läuft. Die Semantik sämtlicher Befehle wird in einer externen Ontologie mit der Bezeichnung **OntoMappingSystem** definiert, die im folgenden Abschnitt näher vorgestellt wird.

## 4.7 Datenexport nach i2b2

Wenn das Mapping fertig aufgebaut ist, kann es mit OntoExport verarbeitet werden. Das Tool liest sämtliche Ontologien ein und übersetzt die darin enthaltenen Information in SQL-Statements, mit denen dann die Daten aus den Quellsystemen nach i2b2 exportiert werden. OntoExport folgt dabei dem Algorithmus aus Abschnitt 3.5.3 auf Seite 54.

Im ersten Schritt wird aus der Zieldatensatz-Ontologie die i2b2-Ontologie erzeugt. Die Vorgehensweise hierfür wurde bereits in Abschnitt 4.3 auf Seite 65 im Zusammenhang mit den Anforderungen an die Zieldatensatz-Ontologie beschrieben. Dabei werden aus der Klassenhierarchie die Einträge der Spalte „c\_fullname“ erzeugt, aus den Properties die Einträge der restlichen Spalten sowie das XML-Feld.

Entsprechend Schritt 2 des Algorithmus können als nächstes die Zwischenknoten in der Mapping-Ontologie abgearbeitet werden, wie es in Abschnitt 3.5 auf Seite 48 beschrieben wurde. Dabei wird für jeden Zwischenknoten ein SQL-Statement erzeugt, das die Daten von seinen beiden Operand-Knoten einliest und entsprechend der gewünschten Transformation miteinander verrechnet. Da nur EAV-Tabellen verwendet werden, konnte ein generischer SQL-Codeblock konstruiert werden, in den die relevanten Informationen einfach eingesetzt werden. Wie gleich gezeigt wird, können damit beliebige Datentransformationen umgesetzt werden. Der Codeblock ist in Auflistung 4.1 zu sehen; er entspricht im Wesentlichen der einfachen Variante in Auflistung 3.1 auf Seite 51, ist jedoch um die in Abschnitt 4.2 auf Seite 62 gewonnenen Ergebnisse erweitert. Die in Dollarzeichen eingefassten Bezeichner stellen Variablen dar, die durch OntoExport ersetzt werden.

Bevor jedoch mit der Erzeugung des SQL-Statements begonnen werden kann, muss erst ein zur Bearbeitung bereiter Zwischenknoten ermittelt werden (Schritt 2a im Algorithmus). OntoExport führt hierzu gemäß Abschnitt 3.5.1 auf Seite 48 eine SPARQL-Query durch.

Aufgrund zu der nun folgenden Beschreibung zur Abarbeitung eines Knotens sei auf die Abbildungen in den Seiten 82 und 83 hingewiesen. Diese stellen einen Ontologie-Ausschnitt und einen fertiggestellten SQL-Codeblock dar und illustrieren damit das Verfahren. Über die farblichen Markierungen kann man leicht erkennen, welche Informationen aus der Ontologie in den generischen SQL-Codeblock eingesetzt werden.

```

1  INSERT INTO OntoExportTemp (NodeName, DocumentID, PatientID, CaseID, StartDate,
2                               EndDate, Value)
3
4  SELECT DISTINCT
5
6      $NODENAME$ NodeName,
7
8      (CASE WHEN OP1.DocumentID IS NULL AND OP2.DocumentID IS NOT NULL THEN OP2.
9            DocumentID ELSE OP1.DocumentID END) DocumentID,
10     (CASE WHEN OP1.PatientID IS NULL AND OP2.PatientID IS NOT NULL THEN OP2.
11           PatientID ELSE OP1.PatientID END) PatientID,
12     (CASE WHEN OP1.CaseID IS NULL AND OP2.CaseID IS NOT NULL THEN OP2.CaseID ELSE
13           OP1.CaseID END) CaseID,
14
15     $DATESTARTVALUE$ DateStartValue,
16     $DATEENDVALUE$ DateEndValue,
17     $OUTPUT$ Value
18
19 FROM
20
21     ($OPERANDFETCHSQL1$) OP1
22
23     FULL OUTER JOIN
24
25     ($OPERANDFETCHSQL2$) OP2
26
27     ON OP1.DocumentID = OP2.DocumentID
28
29 $RESULTFILTER$
```

**Auflistung 4.1:** Generischer Oracle-SQL-Codeblock zur Abarbeitung eines Mapping-Knotens.

Ist der zu bearbeitende Knoten ermittelt, werden in Schritt 2b zunächst die nötigen Tabellenzugriffs-Informationen aus den Ontologien geladen, die für den Zugriff auf die Datensätze der beiden Operand-Knoten erforderlich sind. Wie in Abschnitt 3.3.3 auf Seite 41 beschrieben wurde, sind diese Informationen für jede Quelltabelle über eine *hasSourceTable*-Beziehung zu einem „Tabellen-Individuum“ in der Ontologie hinterlegt. In Abbildung 4.12 auf Seite 82 repräsentieren die beiden Individuen *OntoExportTemp* (oben links) und *ErlagenSTGSoarianTable* (oben rechts) die beiden Datentabellen.

In Abschnitt 4.2 auf Seite 62 wurden fünf Entity-Attribute ermittelt, die im Rahmen dieser Arbeit zusammen mit den Spalten „Attribute“ und „Value“ das Standard-Tabellenformat bilden. Da jedoch beliebige EAV-Tabellen als Datenquelle zum Einsatz kommen sollen, müssen diese Entity-Attribute den jeweils gegebenen Spalten der Quelltabelle zugeordnet werden können. Dies erfolgt in den Ontologien über die Properties *hasPatientIDColumn*, *hasCaseIDColumn*, *hasDocumentIDColumn*, *hasDateStartValueColumn*, *hasDateEndValueColumn*, und *hasValueColumn*. Die Attribut-Spalte kann dabei außer Acht gelassen werden, da die in ihr gespeicherten Einträge nicht exportiert werden. In Abbildung 4.12 sind die Zuordnungen für die beiden Beispieltabellen gelb und grün markiert.

Daneben müssen noch die Filterkriterien aus der Ontologie geladen werden, die über die Property *hasSelectFilter* abgebildet werden. In Abbildung 4.12 sind diese rot hinterlegt bzw. mit einem roten Pfeil markiert.

Sind alle Spalten-Namen und Filterkriterien bekannt, können die SELECT-Statements für jeden der beiden Operanden konstruiert werden. Die Werte aus der Ontologie werden dabei einfach in das SQL-Fragment in Auflistung 4.2 eingesetzt. Beispielsweise wird der Platzhalter `$DOCUMENTID$` durch den Spalten-Namen ersetzt, der in der Ontologie über die Property *hasDocumentID* hinterlegt wurde.

```
1   SELECT DISTINCT $DOCUMENTID$ DocumentID, $PATIENTID$ PatientID, $CASEID$ CaseID,
    $DATESTARTVALUE$ DateStartValue, $DATEENDVALUE$ DateEndValue, $VALUE$ Value
    FROM $SOURCETABLE$ WHERE $SOURCEFILTER$
```

**Auflistung 4.2:** Inhalt von `$OPERANDFETCHSQL1$` und `$OPERANDFETCHSQL2$`, wie sie in den SQL-Code in Auflistung 4.1 auf Seite 79 eingesetzt werden.

Da die Tabellenbeschreibungen für jede Tabelle separat vorliegen, kann auf die Nutzdaten aus unterschiedlichen Tabellen gleichzeitig zugegriffen werden. In Abbildung 4.12 auf Seite 82 ist dies für den IF-Knoten<sup>1</sup> daran zu erkennen, dass die Nutzdaten für Operand 1 in der temporären Tabelle liegen, die für Operand 2 dagegen in der Soarian-DWH-Export-Tabelle. Die beiden erzeugten SELECT-Statements werden anschließend in die Zeilen 17 und 21 des generischen SQL-Codeblocks eingesetzt (siehe Abbildung 4.13 auf Seite 83).

Auf die gleiche werden alle Informationen bezüglich der durchzuführenden Datenbankkoperation in das SQL-Statement eingefügt. Hierfür besitzen alle Zwischenknoten der Mapping-Ontologie eine *hasCommandType*-Verknüpfung zu Individuum einer externen Ontologie `OntoMappingSystem`. Jedes dieser Individuen stellt für einen bestimmten Befehlstyp alle nötigen Informationen bereit, um einen Zwischenknoten dieses Befehlstyps korrekt abzuarbeiten. Die Werte befinden sich hinter den Properties *hasDateStartValue*, *hasDateEndValue*, *hasOutputTransformation* und *hasSelectFilter*, die von `OntoExport` in die Zeilen 11–13 und 25 des generischen SQL-Codeblocks eingesetzt werden.

Nachdem alle Platzhalter im SQL-Statement ersetzt wurden, kann dieses durch das Datenbanksystem ausgeführt werden (Schritt 2c). Anschließend wird dem Knoten über die *hasSourceTable*-Beziehung noch die temporäre Tabelle zugeordnet (Schritt 2d), damit andere Zwischenknoten, die den eben bearbeiteten als Operand verwenden, auf die berechneten Ergebnisse zugreifen können. Zum Abschluss (Schritt 2e) wird noch die Klassenzugehörigkeit von `UnprocessedItem` zu `ProcessedItem` geändert – der Knoten ist damit vollständig bearbeitet.

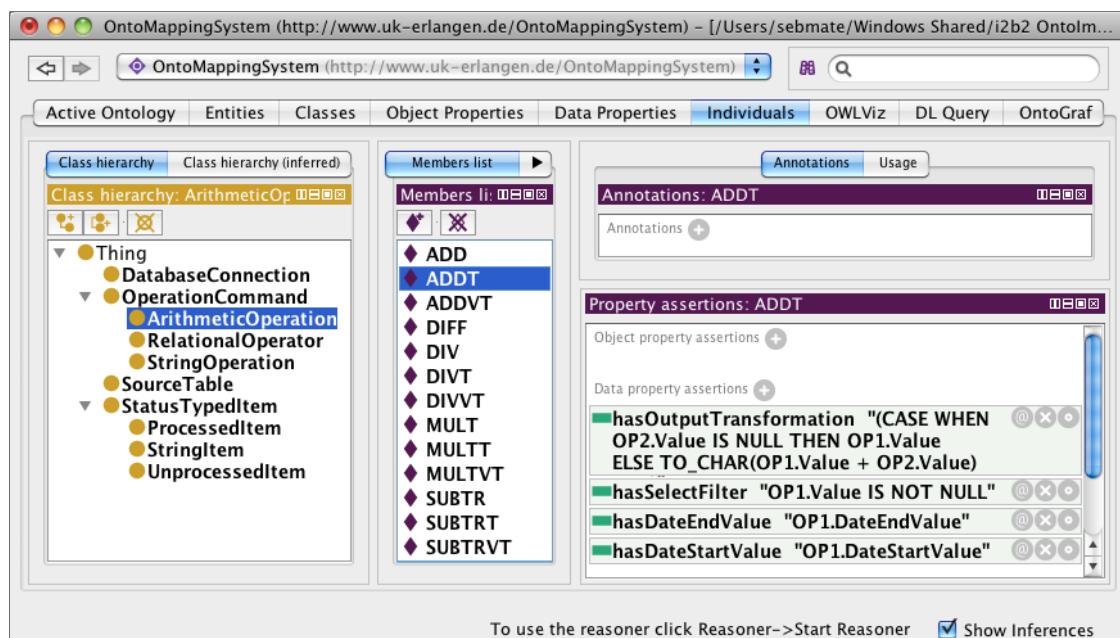
---

<sup>1</sup> Die tatsächliche, eindeutige Bezeichnung des Knotens ist „Result\_Of\_A14\_PSA-Wert-Post-OP\_Operation\_IF3“.

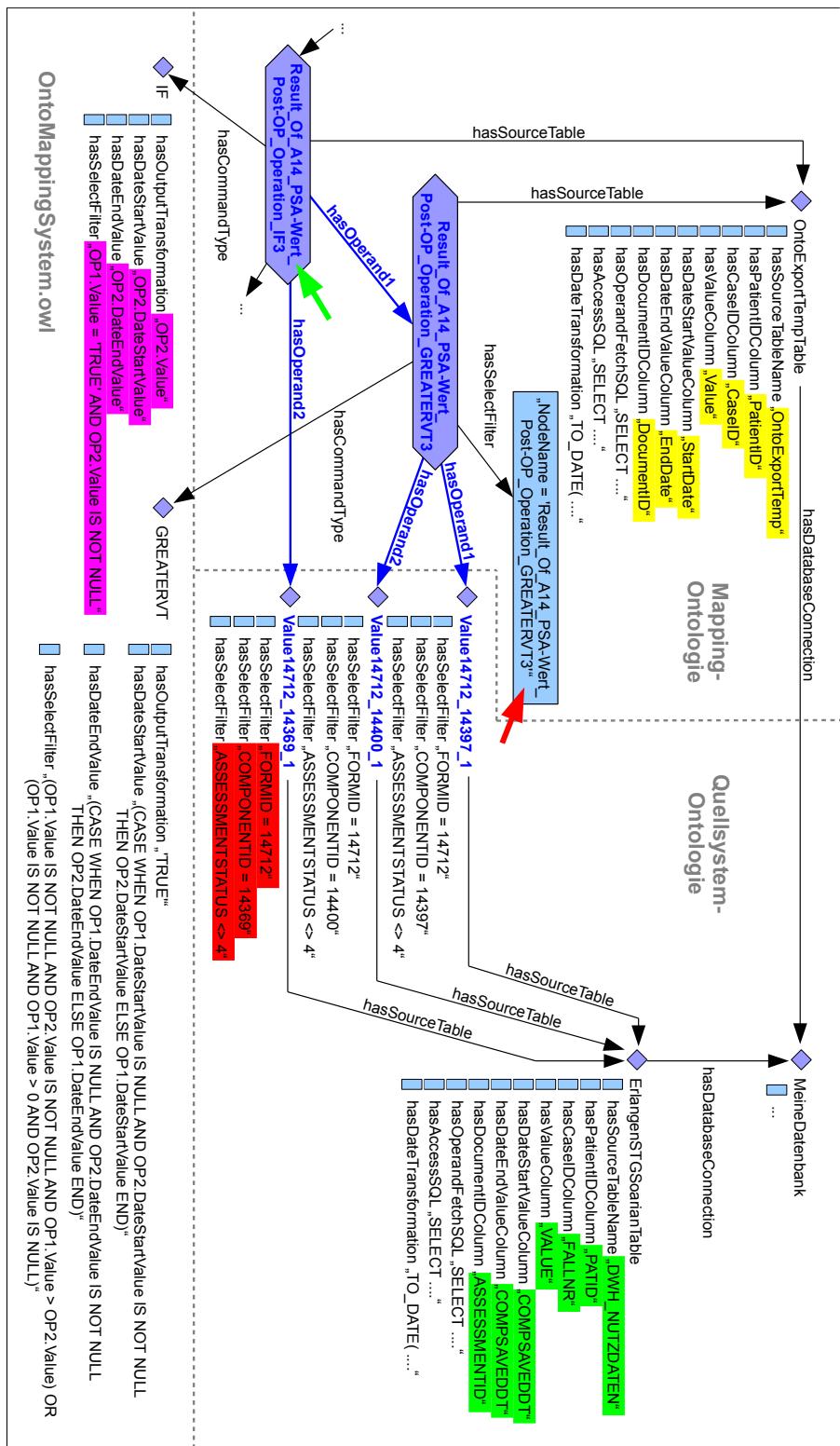
Sind alle Zwischenknoten bearbeitet, können in Schritt 3 alle Datensätze in die i2b2-Datenbank geladen werden. Dafür wird in der Zieldatensatz-Ontologie nach allen Klassen gesucht, die eine *hasImport*-Beziehung zu einem Zwischenknoten der Mapping-Ontologie oder zu einem Dokumentationselement der Quellsystem-Ontologie besitzen. Der Zugriff auf die Nutzdaten erfolgt bei diesem Schritt ebenfalls auf die oben beschriebene Weise.

Durch die oben beschriebene Umsetzung sind alle Informationen, die zur Abarbeitung eines Befehls nötig sind, in eine externe Ontologie ausgelagert. Soll ein neuer Befehl zur Verfügung gestellt werden, muss in der Ontologie lediglich ein neues Individuum angelegt werden, für das dann die vier Properties *hasDateStartValue*, *hasDateEndValue*, *hasOutputTransformation* und *hasSelectFilter* auszufüllen sind. Die kann schnell mit Protégé erfolgen (siehe Abbildung 4.11).

Über die Properties *hasDateStartValue* und *hasDateEndValue* kann gesteuert werden, von welchem Operand der Start- bzw. Endzeitpunkt einer Diagnose übernommen werden soll. Üblicherweise wird die Datumsangabe von Operand 1 übernommen. Eine Ausnahme bildet z. B. der IF-Befehl: da der Wert von Operand 1 durch den IF-Befehl nur auf „TRUE“ geprüft wird und ggf. der Wert von Operand 2 zurückgeliefert wird, müssen bei IF auch die Datumsangaben von Operand 2 übermittelt werden.



**Abbildung 4.11:** Protégé 4.1 mit der Ontologie OntoMappingSystem. Gezeigt ist der „tolerante“ ADD-Befehl ADDT (das Suffix T steht für „tolerant“), bei dem der zweite Operand NULL sein darf.



```

1  INSERT INTO OntoExportTemp (NodeName, DocumentID, PatientID, CaseID, StartDate, EndDate, Value)
2  SELECT DISTINCT
3
4  'Result_of_A14_FSA-Wert_Operation_IF3' NodeName,
5
6  (CASE WHEN OP1.DocumentID IS NULL AND OP2.DocumentID IS NOT NULL THEN OP1.DocumentID ELSE OP2.DocumentID END) DocumentID,
7  (CASE WHEN OP1.PatientID IS NULL AND OP2.PatientID IS NOT NULL THEN OP2.PatientID ELSE OP1.PatientID END) PatientID,
8  (CASE WHEN OP1.CaseID IS NULL AND OP2.CaseID IS NOT NULL THEN OP2.CaseID ELSE OP1.CaseID END) CaseID,
9
10 OP2.DateStartValue DateStartValue,
11 OP2.DateEndValue DateEndValue,
12 OP2.Value Value
13
14
15 FROM
16 (SELECT DISTINCT DocumentID, PatientID, CaseID, StartDate, DateStartValue, EndDate, DateEndValue,
17 Value FROM OntoExportTemp WHERE NodeName = 'Result_of_A14_FSA-Wert_Post-OP_Operation_GREATERTHAN3') OP1
18 FULL OUTER JOIN
19
20 (SELECT DISTINCT ASSESSMENTID DocumentID, PATID PatientID, FALN CaseID, COMPSAVEDDT DateStartValue, COMPSAVEDDT DateEndValue,
21 DateEndValue, VALUE Value FROM DWH_NUTZDATEN WHERE FORMID = 14712 AND COMPONENTID = 14369 AND ASSESSMENTSTATUS <> 4) OP2
22 ON OP1.DocumentID = OP2.DocumentID
23 WHERE OP1.Value = 'TRUE' AND OP2.Value IS NOT NULL
24
25

```

Abbildung 4.13: SQL-Code zum Arbeiten des GREATERVT-Knotens aus Abbildung 4.12.

Universitätsklinikum Erlangen



**Select Input File** UKER-DPKK-Mapping.owl

**Root class:** http://www.dpkk.de/Datensatz#DPKK-Datene

**Output format:** Inferencing level: Direct i2b2 Import None

**Export targets:**

Metadata  Facts  JDBC database access

**Start**

```

processing node: Result_of_A14_PSA-Wert_Post-OP_Operation_IF7
processing node: Result_of_A14_PSA-Wert_Post-OP_Operation_GREATERWT12
processing node: Result_of_A14_PSA-Wert_Post-OP_Operation_IF12
processing node: Result_of_A14_PSA-Wert_Post-OP_Operation_IF11
processing node: Result_of_A14_PSA-Wert_Post-OP_Operation_IF10
No more intermediate nodes found!
Loading data into i2b2 ...
Importing Data: A10_Gleason-Score_1 : value14070_13926_2
Importing Data: OPS_5-600.0-operation : Result_of_A14_PSA-Wert_Post-OP_Operation_IF7
Importing Data: A14_PSA-Wert_Post-OP : Result_of_A14_PSA-Wert_Post-OP_Operation_IF7
Importing Data: G3-Histologie-Grading : value13514_13436_1_26
Importing Data: G3-Histologie-Grading : value14714_13932_2_3
Importing Data: ct2 : Value33515_13417_1_10
Importing Data: ct1 : Value13515_13417_1_9

```

```

INSERT INTO OntoExportTemp(NodeName, DocumentID, PatientID, StartDate, EndDate, Value) SELECT DISTINCT 'Result_of_A14_PS
A-Wert_Post-OP_Operation_GREATERWT6' NodeName, (CASE WHEN OPI.DocumentID IS NULL AND OP2.DocumentID IS NOT NULL THEN OP2
.DocumentID ELSE OPI.DocumentID END) DocumentID, (CASE WHEN OPI.PatientID IS NULL AND OP2.PatientID IS NOT NULL THEN OP2
.PatientID ELSE OPI.PatientID END) DocumentID, (CASE WHEN OPI.DatestartValue IS NULL AND OP2.DatestartValue IS NOT NULL
THEN OP2.DatestartValue ELSE OPI.DatestartValue) DatestartValue, (CASE WHEN OPI.DateendValue IS NULL AND OP2.Dateend
Value IS NOT NULL THEN OP2.DateendValue ELSE OPI.DateendValue) DateendValue, 'TRUE' Value FROM (SELECT DISTINCT PA
TID PatientID, ASSESSMENTID DocumentID, COMPSAVEDDT DatestartValue, COMPSAVEDDT DateendValue, VALUE Value FROM SIG_SOARI
AN_DWH_NUTZDATEN WHERE FORMID = 14712 AND ASSESSMENTSTATUS <> 4 AND COMPONENTID = 14398) OPI FULL OUTER JOIN (SELECT D
ISTINCT PATID PatientID, ASSESSMENTID DocumentID, COMPSAVEDDT DatestartValue, COMPSAVEDDT DateendValue, VALUE Value FROM
SIG_SOARIAN_DWH_NUTZDATEN WHERE FORMID = 14712 AND ASSESSMENTSTATUS <> 4 AND COMPONENTID = 14400) OP2 ON OPI.DocumentID
= OP2.DocumentID WHERE (OPI.Value IS NOT NULL AND OP2.Value > OPI.Value) OR (OPI.Value IS NO
T NULL AND OPI.Value > 0 AND OP2.Value IS NULL) OR (OP2.Value IS NOT NULL AND OP2.Value < 0 AND OPI.Value IS NULL);

INSERT INTO OntoExportTemp(NodeName, DocumentID, PatientID, StartDate, EndDate, Value) SELECT DISTINCT 'Result_of_A14_PS

```

**Abbildung 4.14:** OntoExport: übersetzt die Mapping-Ontologie in SQL-Statements, die die Rohdaten aus den Quellsystemen laden, gemäß Mapping-Regeln transformieren und nach i2b2 exportieren. Darüber hinaus wird aus dem Zieldatensatz die i2b2-Ontologie erzeugt.

## 4.8 Ergebnisse beim Mapping von Münster und Erlangen auf den DPKK-Datensatz

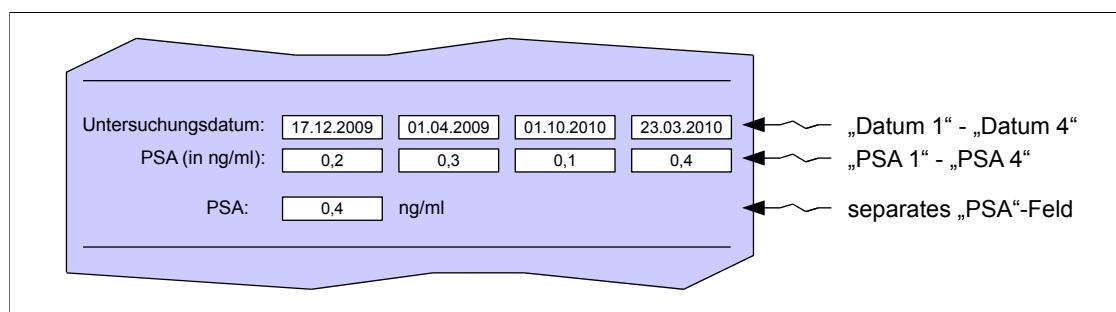
Eine Vergleich des DPKK-Datensatzes mit den Daten der Quellsystemen in Erlangen und Münster hat gezeigt, dass sich die in dieser Arbeit entwickelte Technik dazu eignet, die Datenelemente effizient zu mappen und in eine i2b2-Datenbank zu exportieren.

Da der neue DPKK-Datensatz vor allem unter der Federführung der Urologie in Erlangen entstand, ist es nicht verwunderlich, dass nahezu alle für den DPKK-Datensatz ausgesuchten Datenelemente bereits in Soarian dokumentiert werden und größtenteils ohne Transformationen direkt gemappt werden konnten.

Nur an wenigen Stellen musste auf komplexe Operationen mit Zwischenknoten zurückgegriffen werden. Beispielsweise soll in die DPKK-Datenbank nur der aktuellste postoperative PSA-Wert exportiert werden. In Soarian werden bis zu fünf PSA-Werte dokumentiert, wovon vier ein Datum zugeordnet werden kann (siehe Abbildung 4.15). Beim Export muss dann von den vier Datumsfeldern das mit dem spätesten Zeitpunkt ermittelt werden, anschließend kann der Wert des entsprechenden PSA-Feldes exportiert werden. Wenn die oberen vier PSA-Felder keinen Wert enthalten, soll das fünfte Feld exportiert werden. Bei dem Beispiel Abbildung 4.15 wäre dies dann der Wert 0,1 von „Datum 3“.

In QuickMapp lässt sich dieser Export leicht umsetzen, wenn man die ursprüngliche Aufgabenstellung etwas umformuliert:

1. Exportiere das Feld „PSA 1“ nur, wenn „Datum 1“ das letzte ist.
2. Exportiere das Feld „PSA 2“ nur, wenn „Datum 2“ das letzte ist.
3. Exportiere das Feld „PSA 3“ nur, wenn „Datum 3“ das letzte ist.
4. Exportiere das Feld „PSA 4“ nur, wenn „Datum 4“ das letzte ist.
5. Exportiere das Feld „PSA“ nur, wenn die Felder „PSA 1“ bis „PSA 4“ leer sind.



**Abbildung 4.15:** Dokumentation des PSA-Wertes in Erlangen.

Die ersten vier Teilexporte können jeweils umgesetzt werden, indem man das jeweilige Datumsfeld schrittweise mit den anderen Datumsfeldern vergleicht. Solange der Wert im zu untersuchenden Feld größer ist als der des Vergleichsfeldes, wird der Wert aus dem entsprechenden PSA-Feld weitergereicht. Der folgende Mapping-Ausdruck bearbeitet beispielsweise „Datum 1“:

```

1  PSA-Wert : IF (GREATERTVT Datum1 Datum2)
2      (IF (GREATERTVT Datum1 Datum3)
3          (IF (GREATERTVT Datum1 Datum4) PSA1))

```

Die GREATERTVT-Befehle führen dabei den Größervergleich durch<sup>1</sup>. Wenn diese „TRUE“ zurückliefern, wird dies von den IF-Befehlen erkannt und Wert von Operand 2 des IF-Befehls weitergereicht. Auf die gleiche Weise kann überprüft werden, ob die Felder „PSA 1“ bis „PSA 4“ leer sind:

```

1  PSA-Wert: IF (NOTEXISTS PSA1 PSA)
2      (IF (NOTEXISTS PSA2 PSA)
3          (IF (NOTEXISTS PSA3 PSA)
4              (IF (NOTEXISTS PSA4 PSA) PSA)))

```

Die NOTEXISTS-Befehle überprüfen, ob die Felder in Operand 1 nicht ausgefüllt wurden. Operand 2 stellt dabei ein Vergleichsfeld dar, von dem man annimmt, dass es ausgefüllt wurde<sup>2</sup>. Abbildung 4.16 auf Seite 87 zeigt das vollständige Mapping-Netz.

Für Münster gelang es zum Ende der Arbeit nicht mehr, eine vollständige ORBIS-Ontologie zu erzeugen. Wie bereits in Abbildung 4.1 auf Seite 60 gezeigt wurde, musste hier der vollständige Importprozess über das Tool OntoGen erfolgen, wobei auf einen CSV-Export des ORBIS-Report-Generators zurückgegriffen wurde.

Ein Update der bereitgestellten Daten aus Münster gestaltet sich daher wie folgt:

1. Die Nutzdaten der CSV-Datei werden durch OntoGen in das EAV-Format konvertiert und in der temporären Datenbank bereitgestellt. OntoGen erzeugt dabei eine aktualisierte Quellsystem-Ontologie, die alle Dokumentationselemente der CSV-Datei aufzählt.
2. Die ebenfalls durch OntoGen automatisch erzeugten Zieldatensatz- und Mapping-Ontologien mit den 1:1-Mappings werden durch die gemeinsame DPKK-Datensatz-Ontologie und eine eigene ORBIS-DPKK-Mapping-Ontologie ersetzt.

---

1 Das Suffix „VT“ ist eine Abkürzung für „very tolerant“ und bedeutet, dass auch leere Felder miteinander verglichen werden dürfen.  
 2 Dieser Schritt ist nötig, da es im EAV-Format keine leeren Felder gibt: nur über den FULL OUTER JOIN (der bei der Abarbeitung eines jeden Mapping-Knoten gebildet wird) mit einem ausgefüllten Feld kann überprüft werden, ob ein anderes Feld leer ist.

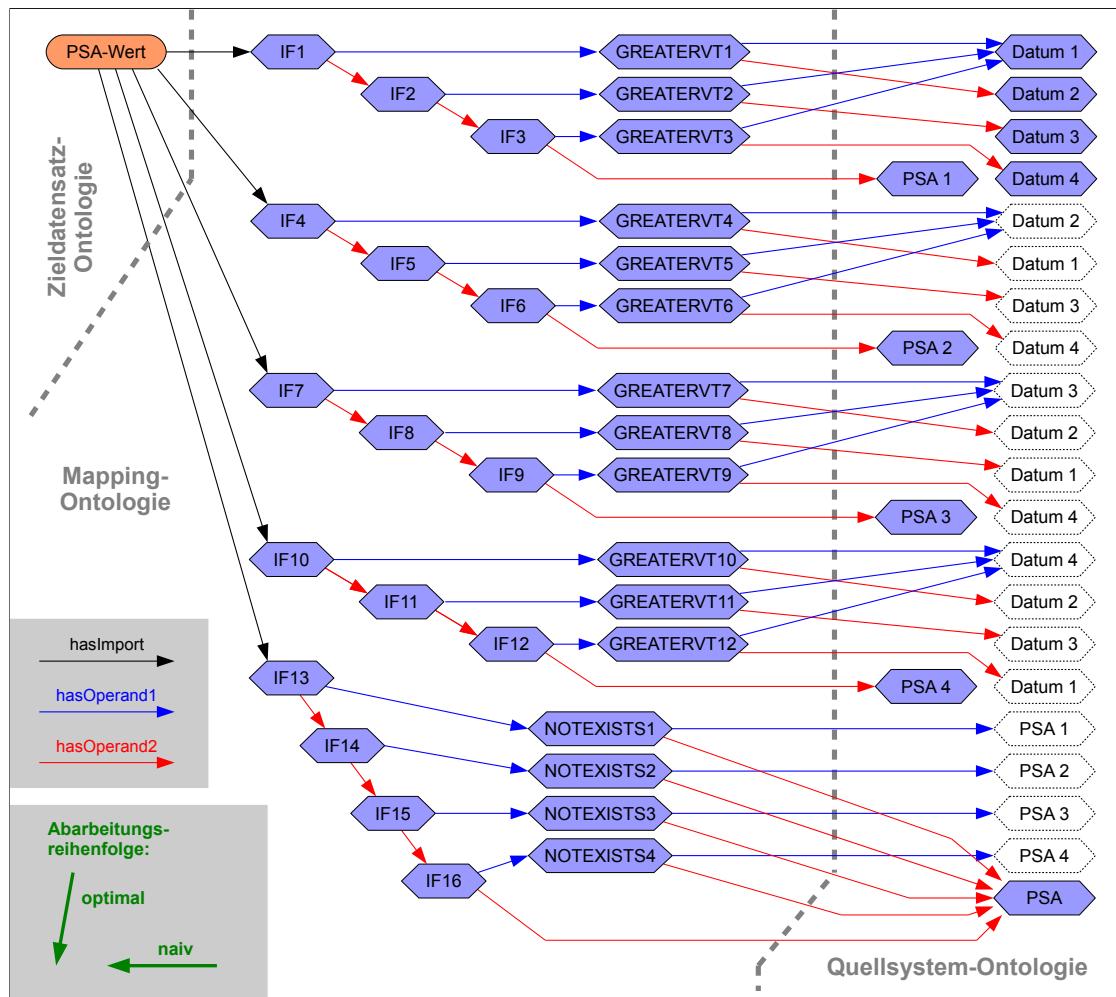


Abbildung 4.16: Das Export-Mapping des PSA-Werts für Erlangen. Bei den gestrichelten Elementen handelt es sich um „Kopien“ der ausgefüllten Elemente, um eine Übersichtlichkeit der Grafik zu bewahren.

3. Falls der ORBIS-Export neue Spalten oder Ausprägungen enthält, finden sich diese auch in der neuen Quellsystem-Ontologie in der Form von Dokumentationselementen wieder. Bei Bedarf können diese dann mit QuickMapp auf den DPKK-Datensatz gemappt werden.
4. Der eigentliche Datenexport in die i2b2-Datenbank kann dann mit OntoExport erfolgen.

Bezüglich des Mappings konnte gezeigt werden, dass sich alle in Münster dokumentierten Inhalte mit der vorgestellten Technik aus den ORBIS CSV-Exportdaten extrahieren lassen. Dort, wo dies nicht möglich ist, liegt eher ein inhaltliches als technisches Problem vor.

Die Tabellen im Anhang C.3 ab Seite 144 veranschaulichen dies. Viele Datenelemente, die in den Tabellen mit „N/A“ markiert sind, werden derzeit in ORBIS noch nicht dokumentiert. Bei anderen erfolgt die Dokumentation nicht ausführlich genug, um die Elemente auf die des DPKK-Datensatzes abzubilden. Dies sieht man beispielsweise an den Elementen A28 bis A32: Münster erfasst hier nur die Summe der rechten und linken Lymphknotenanzahl. Eine Unterscheidung zwischen „rechts“ und „links“ ist dann natürlich nicht mehr möglich.

## 4.9 Ergebnisse bezüglich des i2b2-DPKK-Vermittlungsportals

Die in Abschnitt 3.6 auf Seite 55 erdachten Änderungen an der i2b2-Software konnten in einem kurzen Zeitrahmen von drei Wochen erfolgreich prototypisch umgesetzt und auf einem DPKK-Treffen am 28.10.2010 in Erlangen vorgeführt werden.

Entgegen der geplanten Umsetzung hat es sich jedoch gezeigt, dass eine aktive Weiterleitung der Query-Definitionen einige Nachteile mit sich bringen würde. Zum einen befinden sich die empfangenden i2b2-Server in der Regel hinter einer Firewall, was einen Mehraufwand bei der Netzwerkkonfiguration an den Standorten bedeuten würde. Zum anderen müssten die Server-Adressen der Empfänger-i2b2s dem Sender bekannt sein. Selbst wenn man den i2b2s der DPKK-Mitglieder ausschließlich statische Adressen vergeben könnte, würde dies den Administrationsaufwand in der DPKK-Zentrale erhöhen.

Was jedoch besonders gegen eine aktive Weiterleitung der Queries spricht, ist die Datensicherheit: Würde es einem Angreifer gelingen, das öffentliche DPKK-i2b2 zu kompromittieren, könnte er über die darin enthaltene Serverliste auch an die Daten der Mitglieder-i2b2s gelangen. Es erschien in der Umsetzung daher sinnvoll, eine passive Übermittlung der Query-Definitionen umzusetzen. Sobald bei einer QDC-Zelle eine Anfrage eingeht, wird diese nicht aktiv an die Empfänger weitergeleitet, sondern in virtuellen „Briefkästen“ bereitgestellt. Die im Workflow nachgeschalteten i2b2s fragen dann in festen Zeitabständen bei dem ihnen übergeordneten i2b2 an, ob für sie neue Nachrichten vorliegen.

Der Screenshot in Abbildung 4.18 auf Seite 90 zeigt die Implementierung des QDC-Views (rechts unten). Auf einem Drop-Feld können Elemente der Views „Previous Queries“, „Workplace“ und „Query Tool“ abgelegt werden. Im Screenshot wurde hier bereits ein Element mit der Bezeichnung „Test-Anfrage“ abgelegt. Das View führt dann eine der Query-Definition entsprechende Datenbank-Query durch, um zu ermitteln, wo sich die gefundenen Patienten befinden. Für die gefundenen Institute wird dann eine Checkbox-Liste erstellt, bei der markiert werden kann, für welche anderen i2b2s die Anfrage in den oben genannten „Briefkästen“ zur Verfügung gestellt wird. Neben dem Namen des Instituts wird in einer Klammer außerdem die Anzahl der gefundenen Patienten angezeigt. Im Fall des öffentlich zugänglichen i2b2s wird hier jedoch nur „DPKK“ angezeigt, da im öffentlichen DPKK-i2b2 keinerlei Informationen über den Ursprung der einzelnen Datensätze vorliegen (siehe Abschnitt 3.6 auf Seite 55) – der Screenshot in Abbildung 4.17 entspricht bereits der Weiterverarbeitung der Query in der DPKK-Zentrale.

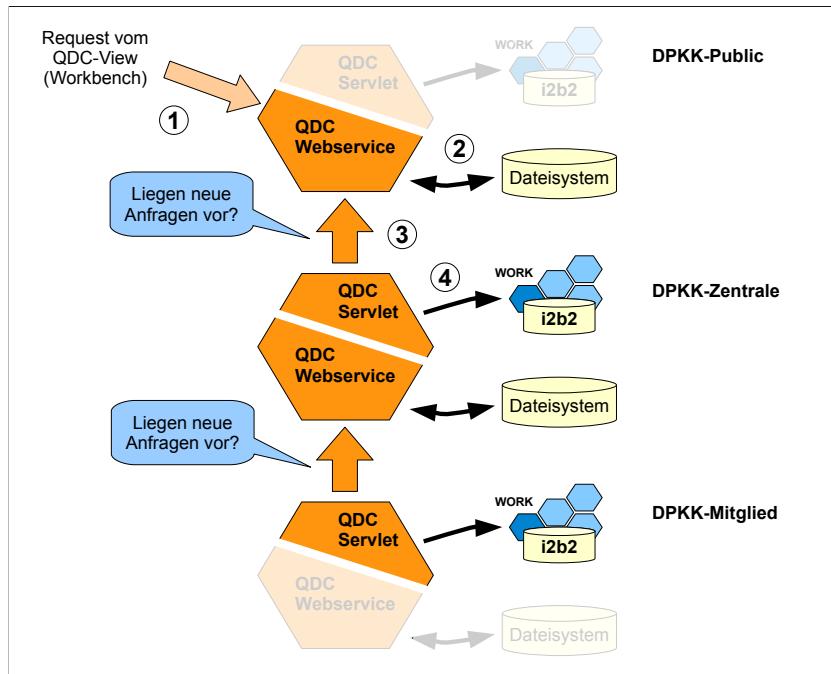


Abbildung 4.17: Serverseitige Umsetzung der QDC-Zelle.

Auflistung C.3 auf Seite 148 zeigt eine vom QDC-View generierte XML-Nachricht, die beim Klick auf den „Submit Request“-Button an die QDC-Zelle übermittelt wird.

Abbildung 4.17 zeigt die serverseitige Implementierung. Wie man in der Abbildung erkennen kann, musste die QDC-Zelle in zwei Komponenten aufgeteilt werden. Der QDC-Webservice, der innerhalb der SOAP-Engine Apache Axis2<sup>1</sup> ausgeführt wird, nimmt die eingehenden Queries der i2b2-Workbenches entgegen (1) und speichert sie in den gewünschten Empfänger-„Briefkästen“ zwischen (2). Im Rahmen dieser Arbeit werden hier einzelne Dateien im Dateisystem angelegt. Außerdem beantwortet der Webservice eingehende Anfragen (3) von anderen QDC-Zellen, die jedoch, wie in der Grafik ersichtlich, nicht von dem Webservice gestellt werden, sondern von einem Servlet.

Diese Trennung war erforderlich, da der QDC-Webservice nach dem Start des Anwendungsservers JBoss<sup>2</sup> nicht automatisch aktiv werden kann um in festen Zeitintervallen nach neuen Nachrichten zu suchen (3) – bei einem Servlet ist dies dagegen über die `init()`-Methode leicht umzusetzen (siehe [46, S. 43]). Liegt eine neue Anfrage vor, wird diese nach einem Benutzer/Passwort-Handshake von dem Webservice an das Servlet übermittelt. Das Servlet legt dann die Query-Definition im i2b2-Workplace innerhalb eines INBOX-Ordners an (4), um sie dem Benutzer des empfangenden i2b2s zugänglich zu machen (siehe Abbildung 4.18 rechts oben). Dieser Schritt erfolgt durch einen JDBC-Direktzugriff auf die Tabelle „WORKPLACE“ (siehe [60, S. 100]).

1 <http://axis.apache.org/axis2/java/core>, Abruf: 19.12.2010

2 Siehe Abbildung 2.1 auf Seite 8 sowie <http://www.jboss.org>, Abruf: 19.12.2010

i2b2 Workbench

Sebastian Mate Status: (282)

**i2b2 Workbench for DPKK Zentrale**

● Navigate Terms X

Query Name: Test-Anfrage

Analysis Types

Patient list  2010-10-27 21.17.26  
Number of...  
Gender pat...  
Vital Status...  
Race patient...  
Age patient...  
 Test-Anfrage

Workplace X

separate  INBOX  
 2010-10-28 08.31.43  
 2010-10-28 08.38.51  
 2010-10-28 16.01.09

Timeline View X

Timeline View X

Create model for Timeline  Render a Timeline

Previous Queries X

The terms of this group are joined then intersected with other groups

The terms of this group are joined then intersected with other groups

Drag items from Navi then intersected with other groups

Find and Workplace into timeline

Run Query Above

Patient(s) returned: 37

Test-Anfrage

Your name: Sebastian Mate  
Institution: Uni Erlangen  
Phone: 26785  
e-Mail: sebastian.mate@uk-erlangen.de

Query Definition Commu X

Research interest:  
Notes: Das ist nur ein Test, ob alles funktioniert ...

Submit Request

Patient Set: 37 Patients start 11 increment 10

2 / 9

▼ Person\_#100000001\_Male\_7Dyrolde\_(Muenster) 2/4 5 6 7 8 9 10

▼ pN0

▼ Adenokarzinom

▼ Person\_#100000008\_Male\_7Dyrolde\_(Muenster)

▼ pN0

▼ Test-Anfrage [12-20-2010] [sebmate]

▼ pN0-Adenokar [02-04-42] [12-20-2010] [sebmate]

▼ pN0 [02-04-42] [12-20-2010] [sebmate]

▼ pN0 [02-04-42] [12-20-2010] [sebmate]

▼ PSA-Wert@02:03:42 [12-20-2010] [sebmate]

▼ Anzahl der Bef [02-02:44] [12-20-2010] [sebmate]

▼ Links@09:59:58 [10-28-2010] [sebmate]

▼ Einschlussstudie@02:38:27 [10-28-2010] [sebmate]

▼ 4711 [10-28-2010] [sebmate]

Abbildung 4.18: Die i2b2-Workbench mit dem QDC-View (rechts unten).

# KAPITEL 5

---

## Diskussion und Ausblick

---

In dieser Arbeit wurde ein prototypisches Bioproben-/Projektvermittlungs-Portal auf i2b2-Basis umgesetzt. Die Motivation, den eigentlichen Fokus auf das Mapping klinischer Datenelemente zu legen, entstand insbesondere in Ermangelung einer Strategie, mit der heterogene KAS-Daten effizient in eine i2b2-Umgebung geladen werden können.

Diese Problematik wurde bereits während der initialen Evaluation von i2b2 in Erlangen im Jahre 2009 erkannt, wobei auch erste Ansätze zum Mappen von Datenelementen entwickelt wurden [60]. Obwohl sich die damals entwickelten Techniken gut dafür eignen, Daten in den i2b2-Hive zu laden und an die gewünschte Stelle in der i2b2-Ontologie „einzuhängen“, ist es mit ihnen nicht möglich, heterogene Daten zu harmonisieren und auf einen gemeinsamen Datensatz abzubilden. Für eine Forschergemeinde wie das DPKK würde dies bedeuten, dass jedes Mitglied eine eigene, aufwändige Lösung finden muss, mit der es seine Quelldaten aus den eigenen Quellsystemen extrahieren, transformieren und in die i2b2-Datenbank laden kann. Die Zeitersparnis durch den geplanten Single-Source-Ansatz, bei dem die Mehrfachdokumentation entfällt, wäre dann vermutlich verloren.

Dieses Problem ist jedoch nicht nur auf das DPKK-Projekt in Verbindung mit i2b2 beschränkt, sondern tritt immer auf, wenn heterogene klinische Daten aus Quellsystemen für Forschungsprojekte oder für andere Auswertungen exportiert werden sollen. Zwar liegt die komplette klinische Dokumentation in elektronischer Form vor; der Aufwand, auf sie zuzugreifen, ist jedoch groß. Es muss zunächst ermittelt werden, welche Datenelemente in den Quellsystemen überhaupt vorhanden sind. Ohne geeignete Werkzeuge kann man nur die Eingabemasken der Quellsysteme begutachten und seine Erkenntnisse dann auf dem Papier oder in Office-Dateien notieren – beim nächsten Bogen-Update sind die gesammelten Informationen möglicherweise schon wieder veraltet. Da die Bögen der Quellsysteme oft in getrennten Abteilungen erstellt sowie anschließend für Exportzwecke

wiederverwendet werden, bedingt eine vollständige Erschließung der Daten einen nicht zu unterschätzenden Kommunikationsaufwand. Im Idealfall betreibt das Klinikum bereits ein Data Warehouse, in dem ein Großteil der Dokumentation zentral abgelegt wird. Aber auch dort werden nur häufig benötigte Daten gesammelt und strukturiert, wie zum Beispiel administrative und abrechnungsrelevante Daten. Für wissenschaftliche Fragestellungen müssen dagegen die einzelnen Datenelemente genau untersucht werden.

Aufgrund dieser Umstände entstand am UK-Erlangen der Wunsch nach einer Lösung, mit der leicht auf solche heterogenen Daten zugegriffen werden kann, und es fielen die üblichen Begriffe wie *Krankenhaus-Ontologie*, *Metadata-Repository (MDR)*, *ISO/IEC 11179*, usw. Wie das System konzeptionell und technisch umzusetzen sei, war zunächst völlig offen. Da mit dem DPKK-Datensatz begonnen werden sollte, war klar, dass dieser zunächst im Sinne eines MDRs beschrieben werden muss. Hinsichtlich der technischen Realisierung wurde zu Beginn noch eine relationale Datenbank mit Web-Oberfläche in Betracht gezogen, durch Hinweise auf Ontologie-Tools wie Protégé fiel der Fokus jedoch schnell auf die Semantic-Web-Standards RDF, RDFS und OWL. Diese wurden, nachdem sich der Autor intensiv mit diesen Techniken befasst hat, als für geeignet befunden und im Rahmen dieser Arbeit verwendet.

## 5.1 Ontologien in Medizin, Genomik und Proteomik

In dieser Arbeit wurde ein System konzipiert und implementiert, mit dem ein gemeinsamer Datensatz durch eine Ontologie beschrieben und anschließend weiterverarbeitet werden kann. Das Bedürfnis für derartige Ontologien ist nicht neu: Bereits im 17. Jahrhundert hat man mit den *Bills of Mortality* etwa 200 Todesursachen aufgelistet, die dann für eine konsistente Dokumentation im Rahmen statistischer Auswertungen verwendet wurden [10]. Basierend auf dieser Auflistung wurde später die *International Classification of Diseases*<sup>1</sup> (ICD) entwickelt, die heute aus dem klinischen Alltag für eine einheitliche Kodierung von Krankheiten nicht mehr wegzudenken ist.

Mit dem Fortschritt in der Medizin – insbesondere in der Genomik und Proteomik [15] – wurde eine Vielzahl an ähnlichen *Nomenklaturen*, *Klassifikationen*, *Vokabularen*, *Thesauren* und *Terminologien*<sup>2</sup> entwickelt. O. Bodenreider gibt in [10] einen umfassenden Überblick über derartige Kataloge – die dort einfach als „Ontologien“ bezeichnet werden – und zeigt, wie diese zum Knowledge-Management, zur Integration und zum Austausch von Daten sowie zur Entscheidungsunterstützung eingesetzt werden; weitere Einführungen befinden sich in [4, 12, 15, 21, 83, 105]. Viele dieser Ontologien sind frei erhältlich [10]

---

1 <http://www.who.int/classifications/icd/en>, Abruf: 08.01.2011

2 Für eine genaue Differenzierung dieser Begriffe siehe [16].

und liegen inzwischen sogar im OWL-Format vor – das bekannteste Beispiel hierfür dürfte der NCI-Thesaurus sein [24, 68, 86]. Es werden sogar Online-Repositories gepflegt, um die Vielzahl an Ontologien zentral zur Verfügung zu stellen [15, 82].

Auch wenn die gute Verfügbarkeit solcher Ontologien eine große Rolle bei der Festlegung auf die Semantic-Web-Techniken in dieser Arbeit gespielt hat, würde es den Rahmen der Arbeit bei weitem sprengen, einen umfassenden Überblick über medizinische Ontologien und deren Einsatzgebiete zu geben. Darüber hinaus wurden bisher noch keine externen Ontologien in das entwickelte System eingebunden – der interessierte Leser sei deshalb auf die oben genannten Quellen verwiesen. Stattdessen sollen im Folgenden die in dieser Arbeit verwendeten Methoden und die erreichten Ergebnisse mit anderen, themennahen Arbeiten in Bezug gesetzt werden.

## 5.2 Die Methodik des entwickelten Mapping-Systems

### 5.2.1 Die Zieldatensatz-Ontologie als gemeinsame Terminologie

Nach [10, S. 69] ist die einfache Auflistung von Konzepten eine wichtige Funktion von Ontologien; sie übernehmen dann eigentlich die Funktion einer Terminologie. Im Rahmen des DPKK-Projekts übernimmt die Zieldatensatz-Ontologie diese Rolle, da sie die einzelnen Datenelemente auflistet, die das DPKK in seiner Forschungsdatenbank bereitstellen möchte.

Eng mit den Zielen des DPKKs verwandt sind die des *Cooperative Prostate Cancer Tissue Resource* Konsortiums<sup>1</sup> (CPCTR) [71, 72]. In diesem von den *National Cancer Institutes*<sup>2</sup> (NCI) gegründeten Verbund werden ebenfalls Bioproben des Prostatakarzinoms für Forschungszwecken gesammelt und bereitgestellt. Bei der Indexierung der einzelnen Proben liegt beim CPCTR der Fokus ebenfalls auf klinischen Parametern, die im Behandlungszusammenhang erfasst wurden und damit weit über die eigentlichen Daten des Biomaterials (z. B. Konservierungsart) hinausgehen. Das CPCTR hat, ebenso wie das DPKK, einen gemeinsamen Datensatz entwickelt (siehe Abbildung 3 in [72]), dessen 145 Datenelemente<sup>3</sup> als *Common Data Elements* (CDEs) bezeichnet und in einer relationalen Datenbank zur Verfügung gestellt werden. Auf die CPCTR-Datenbank kann sogar öffentlich zugegriffen werden.

---

1 <http://www.cpctr.info>, Abruf: 08.01.2011

2 <http://www.cancer.gov>, Abruf: 08.01.2011

3 Der DPKK-Datensatz besteht aus 172 Elementen, wenn man sich auf die Ausprägungs-Ebene begibt.

### 5.2.2 Quellsystem-Ontologien zum Zugriff auf Nutzdaten

Die Bereitstellung der Metadaten der Quellsysteme in Form von semantischen Netzen ist eine notwendige Bedingung für den Ansatz, der in dieser Arbeit umgesetzt wurde.

In Erlangen konnte die Erzeugung der Quellsystem-Ontologie (vgl. Abschnitt 4.5 auf Seite 71) aufgrund der umfangreichen Vorarbeiten in [65] leicht umgesetzt werden. Das SQL-Skript, das die Soarian-Ontologie baut, ist lediglich 160 Zeilen lang; hinzu kommt noch ein 10-zeiliges Bash-Skript, das den Ontologie-Kopf ergänzt, die Datei geringfügig umformatiert und anschließend durch das Jena-Framework in die RDF/XML-Serialisierung konvertiert (siehe Abschnitt A.1 auf Seite 117). Die Abarbeitung dieses Vorgangs erfordert in der aktuellen Implementierung noch ein wenig Handarbeit; soll sie zukünftig vollständig automatisiert werden, dürfte sich der dafür notwendige Aufwand jedoch in Grenzen halten.

Wie es auf Seite 86 beschrieben wurde, gelang es im Rahmen dieser Arbeit zum Schluss nicht mehr, eine vollständige ORBIS-Ontologie für Münster zu erzeugen. Zwar konnte eine hierarchische Metadaten-Tabelle erzeugt werden (vgl. Abschnitt 3.3.2 ab Seite 38); es war jedoch nicht möglich, alle Ausprägungen in die Ontologie zu integrieren. An dieser Stelle hat sich OntoGen (siehe Abschnitt 4.4 auf Seite 69) als nützlich erwiesen, obwohl dieses Tool ursprünglich nur dazu implementiert wurde, um den vollständigen Importprozess für Demonstrationszwecke abzudecken. Auch wenn die mit OntoGen erzeugte Quellsystem-Ontologie keine richtige Hierarchie bereitstellt (vgl. Abschnitt 3.3.2 auf Seite 37), konnten die einzelnen Dokumentationselemente mithilfe der in QuickMapp integrierten Suchfunktionen schnell aufgefunden und – soweit es die Testdaten aus Münster ermöglichten – gemappt werden.

Eine derartige Darstellung von Quellsystemen durch Ontologien ist immer nötig, wenn Daten in Kombination mit den Semantic-Web-Techniken verwendet werden sollen, diese aber nicht in RDF, RDFS oder OWL vorliegen. Aus diesem Grund wurden bereits einige Werkzeuge und Techniken entwickelt, über die [42, S. 301-360] einen guten Überblick gibt. Besondere erwähnenswert ist die D2RQ-Software<sup>1</sup>, mit der sich relationale Datenbanken als Ontologien bereitstellen lassen. Das System wurde im medizinischen Umfeld bereits des Öfteren eingesetzt (siehe z. B. [26, 53, 84]). Da die mit D2RQ erzeugten Ontologien nach [42, S. 345] allerdings nur ein einfaches „Spiegelbild“ der Datenbank-Schemata darstellen und die Erzeugung der Quellsystem-Ontologien einfach umzusetzen erschien (siehe oben), fand D2RQ im Rahmen dieser Arbeit keine weitere Beachtung.

---

<sup>1</sup> <http://www4.wiwiss.fu-berlin.de/bizer/d2rq>, Abruf: 08.01.2011

### 5.2.3 Mapping zwischen Ontologien

Das Mapping zwischen Ontologien ist ebenfalls ein häufig anzutreffendes Thema in der Literatur. Die inzwischen große Anzahl an verfügbaren Ontologien ist sowohl ein Segen als auch ein Fluch [10], denn die meisten Ontologien sind oft nur auf eine einzige Domäne beschränkt. Eine „Universal-Ontologie“ wird es möglicherweise nie geben [19].

Häufig existieren jedoch Überschneidungen zwischen zwei Ontologien, womit diese dann miteinander verknüpft werden können [15, S. 69]. Bei Projekten, wie dem *Unified Medical Language System* (UMLS), versucht man, verschiedene Terminologien zu integrieren [9, 10, 47, 56] und für die Forschergemeinde bereitzustellen.

Da das Mapping bei diesen umfangreichen Ontologien (mit teilweise mehreren Millionen Konzepten) jedoch kaum manuell zu bewerkstelligen ist [89], wurden – und werden – große Bemühungen unternommen, diesen Vorgang so weit wie möglich zu automatisieren. In diesem Forschungsumfeld finden sogar alljährliche Wettbewerbe statt, bei denen Entwickler ihre Mapping-Algorithmen gegeneinander antreten lassen können [39]. Meistens basieren diese Verfahren [28, 96] auf lexikalischen Vergleichen der Konzeptnamen [39, 95] oder auf strukturellen Vergleichen, bei denen z. B. Ober- und Unterklassen-Beziehungen und andere Relationen in den beiden Ontologien miteinander verglichen werden [19, 59]. Neben Mischformen [28] wurden sogar Verfahren vorgestellt, die sich des Machine-Learnings bedienen [27].

Oft werden zusätzliche Ontologien dafür eingesetzt, um das automatische Mapping zwischen zwei anderen Ontologien zu vereinfachen. Beispielsweise wird LOINC<sup>1</sup> gerne zum automatischen Mapping von Laborwerten aus unterschiedlichen Quellsystemen verwendet [5, 49, 54]; auch UMLS findet in Mapping-Projekten des öfteren Anwendung [11, 20, 35]. In [35] wurde zum Beispiel von SNOMED-CT<sup>2</sup> auf ICD-9-CM<sup>3</sup> unter Verwendung von LOINC gemappt, in [20] von ICD-9-CM auf MeSH<sup>4</sup> unter Verwendung des UMLS.

Im Rahmen des DPKK-Projekts muss untersucht werden, ob eine Integration derartiger Automatismen für den mit weniger als 200 Elementen verhältnismäßig kleinen Datensatz zu rechtfertigen ist. Eine Verwendung einfacher lexikalischer Techniken könnte den Mapping-Prozess jedoch deutlich beschleunigen. QuickMapp könnte versuchen, relevante Dokumentationselemente des Quellsystems automatisch zu identifizieren und anschließend mögliche Übereinstimmungen dem Benutzer in Form einer Rangliste vorschlagen. Jeder mögliche Treffer wird dabei mit einem Score bewertet und in eine Auswahlliste

1 <http://loinc.org>, Abruf: 08.01.2011

2 <http://www.ihtsdo.org/snomed-ct>, Abruf: 08.01.2011

3 ICD in der 9. Version, „CM“ steht für „Clinical Modification“

4 <http://www.nlm.nih.gov/mesh>, Abruf: 08.01.2011

eingefügt, die dann nach dem Score sortiert wird. Tatsächlich wurden in der QuickMapp-Benutzeroberfläche die hierfür nötigen Checkboxen bereits angelegt (siehe Abbildung 4.10 auf Seite 75), die angedachte Integration des bekannten Levenshtein-Algorithmus [55] für unscharfe Zeichenketten-Vergleiche konnte aus Zeitgründen jedoch nicht mehr umgesetzt werden. Einige Datenelemente des DPKK-Datensatzes, wie z. B. „Einschlussstudie“, „TNM“ und „Durchgeführte Operation“, sollten sich damit effizient automatisch mappen lassen. Sie enthalten entweder besonders lange Wörter oder Großbuchstaben-Zahlen-Kombinationen, die man bei der Score-Berechnung stärker gewichten könnte, um sie in der Rangliste aufsteigen zu lassen. In [39] wurde festgestellt, dass solche einfachen Algorithmen durchaus sehr gute Mapping-Ergebnisse erzielen. Trotzdem müssen die einzelnen Mappings anschließend von Menschen überprüft werden, um fehlerhafte Datensätze in der DPKK-Forschungsdatenbank zu vermeiden.

#### 5.2.4 Ansätze zum Zugriff auf heterogene Datenbestände

Das im Rahmen dieser Arbeit entwickelte System erlaubt aufgrund seiner Konzeption den Zugriff und die Integration heterogener Datenbestände. Derartige Systeme werden nach W. Sujansky [93] als *Heterogeneous Database Systems* (HDBS) bezeichnet. Hier muss unterschieden werden, ob die Datenzusammenführung auf der Basis von prozeduralen oder deklarativen Mappings durchgeführt wird: Während im ersten Fall beliebige Funktionen zum Einsatz kommen können (möglicherweise steht für die Durchführung sogar eine Turing-vollständige [97] Programmiersprache wie T-SQL oder PL/SQL zur Verfügung), werden die Mappings im zweiten Fall in einer formalen Sprache spezifiziert und unabhängig von der Software gespeichert, die für die Datenbankzugriffe erforderlich ist. Nach W. Sujansky sind solche Mappings weniger an die Quellsysteme gekoppelt. Außerdem können Datenbankabfragen durch eine Software, die die Mappings „versteht“, optimiert werden.

Das im Rahmen dieser Arbeit entwickelte System wäre somit der zweiten Kategorie zuzuordnen. Tatsächlich sind die Mappings vom Quellsystem weitestgehend entkoppelt: Ändert sich die Tabellenstruktur der Quellsysteme<sup>1</sup>, können die Quellsystem-Ontologien entsprechend angepasst werden, um einen Zugriff auf die neuen Datenstrukturen zu ermöglichen. Solange sich die URIs der Dokumentationselemente nicht ändern, bleiben die bestehenden Mappings in den Mapping-Ontologien weiterhin gültig. Eine mögliche Optimierung durch „Superknoten“ wird in Abschnitt 5.5.4 auf Seite 104 gezeigt.

Der in dieser Arbeit entwickelte Ansatz stellt außerdem ein ETL-Verfahren (Extract, Transform, Load) dar, bei dem die Nutzdaten aus den heterogenen Quellsystemen extrahiert, transformiert und in die i2b2-Datenbank geladen werden. Andere Projekte verfolgen

---

<sup>1</sup> Beispielsweise könnten sich Tabellen- oder Spaltennamen ändern, Tabellen könnten aufgeteilt oder auf einen anderen Server verlegt werden.

dagegen zumeist einen *Query-Mediator*-Ansatz. Hier werden die Daten zur Auswertung nicht zuerst zusammengeführt, sondern die Datenbank-Query umformuliert, aufgeteilt und an die einzelnen Datenbanken der Quellsysteme übergeben. Damit entfallen turnusmäßige Datenbankaktualisierungen; bei der Abarbeitung der verteilten Queries sind jedoch längere Laufzeiten hinzunehmen. Beispiele für derartige Systeme sind TAMBIS [9, 43, 83], BioMediator [17, 57, 100], OntoFusion [2, 3, 58, 73], das von W. Sujansky entwickelte TransFER [92–94] sowie Y. Suns MEDIATE [96]. Einige dieser Systeme verwenden auch Semantic-Web-Standards zur Speicherung der Mappings.

Da mit i2b2 ein besonders benutzerfreundliches und leistungsfähiges Query-Werkzeug zur Verfügung steht, erscheint der umgesetzte ETL-Ansatz sinnvoll. Im Zusammenhang mit dem DPKK-Projekt ist er aus datenschutzrechtlichen Gründen sogar zwingend erforderlich; ein System wie SHRINE für verteilte Queries wurde von vornherein ausgeschlossen (siehe Abschnitt 3.6 auf Seite 55). Im Rahmen einer Weiterentwicklung der bisher erfolgten Arbeiten sollte ein Query-Mediator-Ansatz jedoch untersucht werden, da er insbesondere für Inhouse-Anwendungsfälle interessant ist und relativ leicht umzusetzen sein sollte. Bei dem hier entwickelten System müsste lediglich der letzte Schritt entfallen, der darin besteht, die Daten nach i2b2 zu laden. Die übrigen SQL-Statements stellen genau die Datenbankanfragen für den Zugriff auf die Nutzdaten der Quellsysteme dar, sowie deren Transformationen. Die Software könnte direkt in den i2b2-Hive integriert werden oder ein eigenständiges, von i2b2 unabhängiges System bilden.

### 5.3 Wiederverwendung der entwickelten Wissensbasis in einer Krankenhaus-Ontologie

Die konventionelle Vorgehensweise würde darin bestehen, die Daten mittels SQL-Skripten, Datenmodellierungs-Tools oder durch eine andere geeignete Software mit einem Export-Job aus den Quellsystemen zu extrahieren, zu transformieren und anschließend in die Zieldatenbank zu schreiben. Bei genauerer Betrachtung dieses Prozesses stellt man fest, dass hier Wissen aus unterschiedlichen Domänen in elektronischer Form erfasst wird:

1. Durch die Entscheidung, welche Datenfelder in den Export-Datensatz aufgenommen werden sollen, wird ein medizinischer Forschungskontext definiert, in dem die Daten anschließend wiederverwendet werden. Typischerweise sind in diesen Vorgang mehrere Personen involviert. In diesem aufwändigen Prozess wird untersucht, welche Datenelemente für klinische Fragestellungen in den Quellsystemen überhaupt vorhanden sind. Bei diesem Prozess wird medizinisches Wissen elektronisch abgebildet.
2. Beim Erstellen der Transformationsregeln wird untersucht, wie die Datenelemente gefiltert und transformiert werden müssen, um dem Export-Datensatz zu entsprechen. Hierbei wird technisches Wissen in elektronischer Form erfasst.

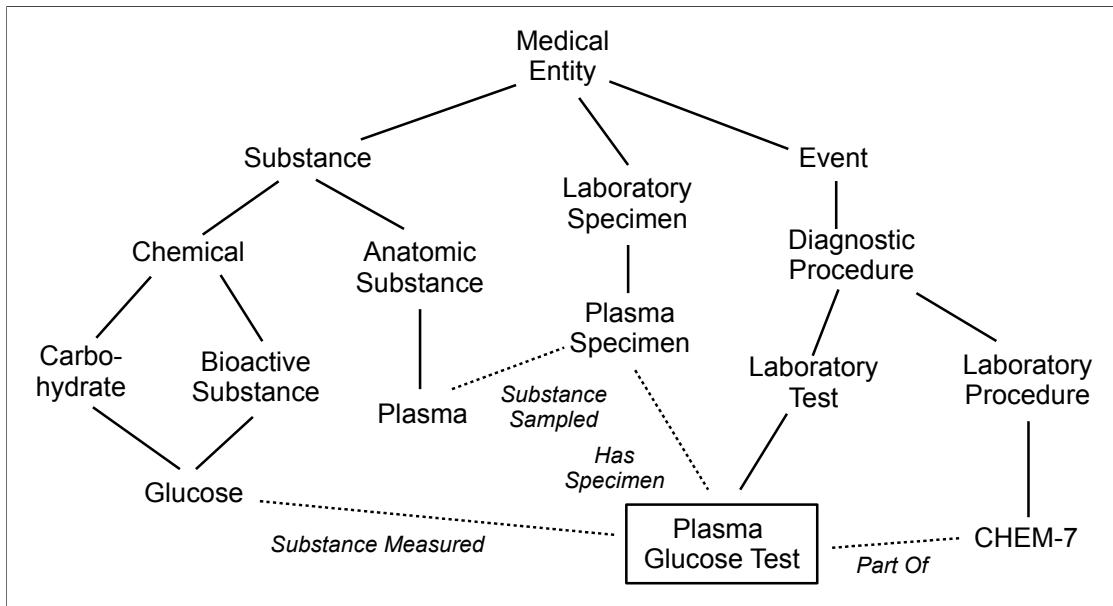
3. Darüber hinaus findet ein Review der Quellsysteme statt. Zum Beispiel wird ermittelt, welche Datenelemente möglicherweise redundant in den Quellsystemen dokumentiert werden. Beim Ausführen des Export-Jobs werden diese verteilten Datenelemente dann zusammengeführt.

Wenn die Daten auf eine konventionelle Weise exportiert werden, ergibt sich daraus der Nachteil, dass das eingebrachte Wissen anschließend lediglich durch die für den Export-Job geschaffene Software maschinell verarbeitbar ist. Ein SQL-Skript kann zum Beispiel nur durch ein geeignetes Datenbanksystem verarbeitet werden. Im günstigsten Fall halten sich die Mitarbeiter noch an SQL-Standards, damit das Skript auf eine andere SQL-Datenbank portiert werden kann; das zur Erstellung des Skriptes aufgebrachte Fachwissen bleibt jedoch auf das Skript selbst beschränkt.

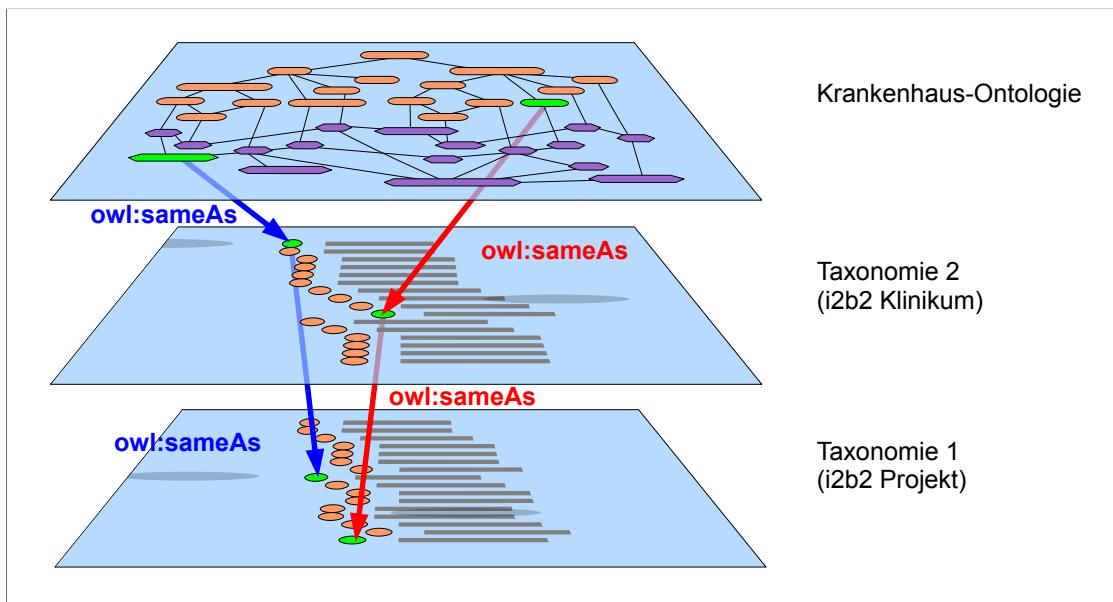
Im Gegensatz zu einer solchen Methodik wird in dieser Arbeit zuerst eine Wissensbasis in der Form von deklarativen Mappings aufgebaut, die dann für unterschiedliche Ziele wiederverwendet werden kann. Das primäre Ziel, das in der Zusammenführung und der Transformation der Nutzdaten sowie im Export dieser nach i2b2 bestand, konnte unter Verwendung der Wissensbasis erfolgreich umgesetzt werden.

Ein weitläufiges Ziel könnte in der Einbringung der erzeugten Mappings in eine größere Krankenhaus-Ontologie bestehen. Einzelne Projekte, wie zum Beispiel das DPKK-Mapping auf das KAS, könnten zentral zusammengeführt werden. Einerseits wird der oben unter Punkt 1 beschriebene, erarbeitete Forschungskontext in einer nachvollziehbaren Form erfasst, andererseits geht das unter Punkt 2 beschriebene technische Mapping-Wissen nicht verloren: Wird in einem anderen Projekt beispielsweise wieder der PSA-Wert benötigt, so muss lediglich die URI des PSA-Wertes referenziert werden – das Mapping auf die Quellsysteme wird automatisch übernommen.

In Abschnitt 4.3 auf Seite 65 wurde beschrieben, dass i2b2 seine medizinischen Konzepte lediglich in einer einfachen Taxonomie verwaltet. Für i2b2 ist diese Darstellungsform zweckmäßig, da in einer Taxonomie gut navigiert werden kann und viele medizinische Klassifikationen ebenfalls in taxonomischer Form (vgl. Abbildung 2.8 auf Seite 17) vorliegen. Jedoch stößt eine derartige monohierarchische Darstellung schnell an ihre Grenzen, sobald eine größere Menge an Konzepten verwaltet oder wenn neben hierarchischen Beziehungen weitere semantische Verknüpfungen zwischen den einzelnen Konzepten modelliert werden sollen – denn damit könnte man auch komplexes medizinisches Wissen in der Ontologie erfassen (siehe Abbildung 5.1 auf Seite 99). Die einzelnen i2b2-Taxonomien könnten im Rahmen ihres Projekts getrennt voneinander erstellt und über Verbindungen mit Konzepten der Krankenhaus-Ontologie verknüpft werden (siehe Abbildung 5.2 auf Seite 99).



**Abbildung 5.1:** Eine Krankenhaus-Ontologie sollte mehr abbilden können, als die monohierarchischen Zieldatensatz-Ontologien in dieser Arbeit: Diese Abbildung aus [18] zeigt eine Polyhierarchie (durchgängige Linien, man beachte das Konzept „Glucose“) mit semantischen Verknüpfungen (gestrichelte Linien).



**Abbildung 5.2:** In einer Krankenhaus-Ontologie (oben) könnten medizinische Konzepte mit deutlich ausdrucksstärkeren Mitteln modelliert werden. Einzelne Konzepte könnten dann in mehreren „einfachen“ i2b2-Taxonomien wiederverwendet werden, um so die eigentlichen Nutzdaten einem i2b2-Nutzerkreis komfortabel zur Verfügung zu stellen.

Im Rahmen dieser Arbeit konnte kein Konzept für den Aufbau einer vollständigen Krankenhaus-Ontologie bzw. eines Vokabulars entwickelt werden; vielmehr wurde mit dem hier entwickelten Ansatz eine Methodik entwickelt, mit der die Konzepte einer beliebigen Krankenhaus-Ontologie effizient mit den eigentlichen Nutzdaten der Quellsysteme verknüpft werden können.

Ein oft angestrebtes Ziel ist die Kompatibilität mit der ISO/IEC-11179-Norm<sup>1</sup> [72, 90], die für eine vollständige Krankenhaus-Ontologie jedoch auch unzureichend ist. Beispielsweise definiert sie keinerlei Konstrukte zur Erzeugung von Hierarchien [90], was auch an der Umsetzung des *Cancer Data Standards Registry and Repositorys*<sup>2</sup> (caDSR) durch P. M. Nadkarni [66] kritisiert wurde. Weitere, in der Literatur allgemein anerkannte Anforderungen hat J. J. Cimino in [22] gestellt.

## 5.4 Vorteile durch die Verwendung der Semantic-Web-Standards

Eine Modellierung von Hierarchien und anderen semantischen Beziehungen zwischen medizinischen Konzepten lässt sich auch ohne spezielle Ontologiesprache z. B. mit relationalen Datenbanken umsetzen; einige Beispiele hierfür werden ausführlich in [16] beschrieben. Da Aussagen in semantischen Netzen nur aus den drei Bausteinen Subjekt, Prädikat und Objekt bestehen, können diese in einer Tabelle mit nur drei Spalten gespeichert werden; man betrachte hierfür die N-Tripel-Serialisierung für RDF in den Auflistungen A.2 und A.3 auf Seite 118. Tatsächlich verwenden auch die in Abschnitt 2.4 auf Seite 27 genannten Frameworks nur einfache Tabellen in relationen Datenbanken, wenn die effiziente Speicherung und Abfrage von großen semantischen Netzen im Vordergrund steht. Ein anschauliches Beispiel ist in [42] auf Seite 145 zu sehen.

Allerdings bieten die im Rahmen dieser Arbeit verwendeten Standards RDF, RDFS und OWL einige Vorteile, die auch von A. Ruttenberg in [84] hervorgehoben werden. Er stellt fest: *“We have come to believe the judicious application of Semantic Web technologies can lead to faster movement of innovation from research laboratory to clinic or hospital. The Semantic Web approach offers an expanding mix of standards, technologies, and social practices layered on top of the most successful information dissemination and sharing apparatus in existence – the World Wide Web.“*

RDF-Ressourcen sind aufgrund der eindeutigen URIs **global gültig** (siehe Seite 19). Für das DPKK bedeutet dies beispielsweise, dass der Zieldatensatz an einer Stelle verwaltet und durch die Mitglieder benutzt werden kann; diese müssen sich lediglich auf die URI beziehen, mit der das Konzept modelliert wird. Mit RDF, RDFS und OWL konstruierte Ontologien sind **flexibel, erweiterbar** und können auch **dezentral**

---

1 <http://metadata-stds.org/11179>, Abruf: 08.01.2011

2 <https://cabig.nci.nih.gov/concepts/caDSR>, Abruf: 08.01.2011

vorliegen. In einer Krankenhaus-Ontologie könnten Informationen aus unterschiedlichen, frei über das Internet verfügbaren Ontologien zusammengeführt werden.

RDFS- und OWL-Ontologien **beschreiben sich selbst**, falls sie vernünftig gestaltet wurden – man betrachte zum Beispiel Abbildung 2.14 auf Seite 26. Ressourcen können durch Annotationen über die Property `rdfs:comment` direkt in der Ontologie erklärt werden. Außerdem verwenden Ontologien, die auf diesen Standards basieren, immer die gleichen RDFS- und OWL-Konstrukte zur Definition von Hierarchien (z. B. `rdfs:subClassOf`) oder komplexen Klassen (z. B. `owl:intersectionOf`). Die Ontologien sind damit auf eine einheitliche Weise durch Computersoftware **maschinell verarbeitbar**, wie es in Abschnitt 2.3.5 ab Seite 23 mit der Casanova-Klasse gezeigt wurde. Durch Inferenzmaschinen stehen außerdem Möglichkeiten zur **Konsistenzprüfung** zur Verfügung. Da Fehler in Datenquellen leider häufig anzutreffen sind, stellt dies einen nützlichen Aspekt dar. Ein Beispiel hierfür wird in [84] beschrieben.

## 5.5 Vor- und Nachteile, sowie Verbesserungspotenziale des entwickelten Export-Konzepts

### 5.5.1 Übersetzung der Mappings in SQL-Befehle

Mittels Übersetzung der Mapping-Ontologien in SQL-Syntax lassen sich aus der aufgebauten Wissensbasis Datenbankinstruktionen zur Extraktion und zur Transformation der Nutzdaten aus den einzelnen Quellsystemen ableiten. Ein Vorteil dieses Ansatzes besteht darin, dass die gewünschten Datentransformationen durch die Datenbank selbst abgearbeitet werden; es ist also nicht nötig, die einzelnen Datensätze durch ein extra zu implementierendes Programm zu verarbeiten. Erzeugte SQL-Skripte können beispielsweise in den ETL-Prozess eines Data Warehouses integriert und turnusmäßig ausgeführt werden.

### 5.5.2 Temporale Aspekte beim Zugriff auf getrennte Quellsysteme

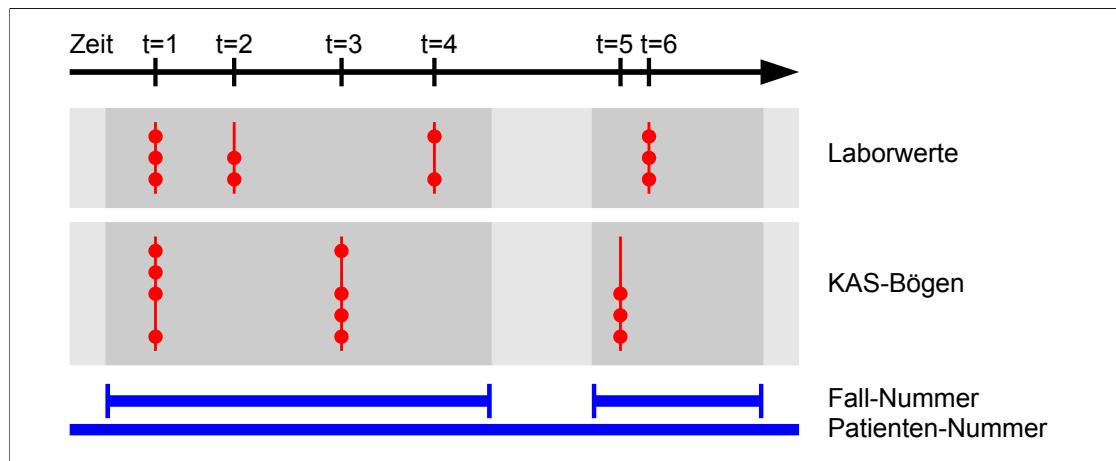
Ein Zugriff auf verteilte Datenbanksysteme wurde im Rahmen dieser Arbeit nur konzeptiell umgesetzt. Zwar kann für jedes Dokumentationselement einer Quellsystem-Ontologie eine Tabelle sowie eine Datenbankverbindung angegeben werden; das erzeugte SQL-Skript kann jedoch immer nur auf die Datenbestände einer einzelnen Datenbank lesend und schreibend zugreifen. Soll diese Barriere überwunden werden, so wird die Implementierung einer speziellen Software nötig. Mapping-Netze, die auf Dokumentationselemente

in verschiedenen Datenbanken zugreifen, müssten auf die einzelnen Datenbanken aufgeteilt werden. Wenn dabei Elemente aus zwei unterschiedlichen Datenbanken miteinander verrechnet werden sollen, müssen diese von der Software zuerst in einer temporären Datenbank zusammengeführt werden.

Soll das System für einen Zugriff auf voneinander getrennte Quellsysteme erweitert werden, muss zunächst genau untersucht werden, ob eine Verrechnung von einzelnen Datenelementen aus unterschiedlichen Quellsystemen überhaupt möglich ist oder sinnvoll erscheint. Da klinische Daten wiederholt oder zu unterschiedlichen Zeitpunkten anfallen können, ist es nicht immer klar, welche zwei Einträge miteinander verrechnet werden können.

Dies wird in Abbildung 5.3 auf Seite 103 illustriert: Zum Beispiel können die KAS-Bögen mehrfach hintereinander angelegt und ausgefüllt werden ( $t = 1$ ,  $t = 3$  und  $t = 5$ ). In der bisherigen Implementierung können nur die Datenelemente aus einem einzelnen Bogen (in der Abbildung durch die senkrechten, roten Striche symbolisiert) miteinander verrechnet werden. Dabei werden die Datensätze über das in der Ontologie festgelegte Entity-Attribut *DocumentID*, das die Datensätze eines Bogens eindeutig identifiziert (siehe Abschnitt 4.2 auf Seite 62), miteinander gejoint. Bei Soarian in Erlangen ist dies zum Beispiel die Tabellenspalte „AssessmentID“ (siehe Tabelle C.2 auf Seite 135). Sollen jedoch die Einträge aus unterschiedlichen Quellsystemen miteinander verrechnet werden, so kann der Join natürlich nicht über die unterschiedlichen *DocumentIDs* der Quellsysteme erfolgen. Man müsste dann auf einen systemübergreifenden Identifier, wie z. B. die Fallnummer, ausweichen. Aber auch mit diesem Ansatz stößt man schnell auf Probleme: Während in dem Beispiel in Abbildung 5.3 die Daten zu den Zeitpunkten  $t = 5$  und  $t = 6$  über die Fallnummer noch eindeutig zugeordnet werden können, ist dies bei den älteren Datenelementen nicht möglich. Soll beispielsweise ein Datenelement aus dem KAS-Bogen von Zeitpunkt  $t = 3$  mit einem Laborwert verrechnet werden, so kann nicht automatisch entschieden werden, ob hier der Wert aus dem Laborbefund vom Zeitpunkt  $t = 2$  oder  $t = 4$  genommen wird.

Durch die Verwendung von i2b2 als Analysewerkzeug der Daten ist eine derartige Verrechnung von Datenelementen möglicherweise gar nicht nötig: Der i2b2-erfahrene Leser hat sich beim Betrachten von Abbildung 5.3 möglicherweise an das Timeline-View in der i2b2-Workbench erinnert, das die Query-Ergebnisse in einer sehr ähnlichen Form auf einer Zeitleiste darstellt (vgl. Abbildung 4.18 auf Seite 90). Anhand dieser Visualisierung kann der i2b2-Benutzer die Datenwerte selbst beurteilen.



**Abbildung 5.3:** Die Verrechnung von Datenelementen aus unterschiedlichen Quellsystemen ist aufgrund von temporalen Aspekten nicht immer möglich (siehe Text).

### 5.5.3 Optimierung der Abarbeitungsreihenfolge

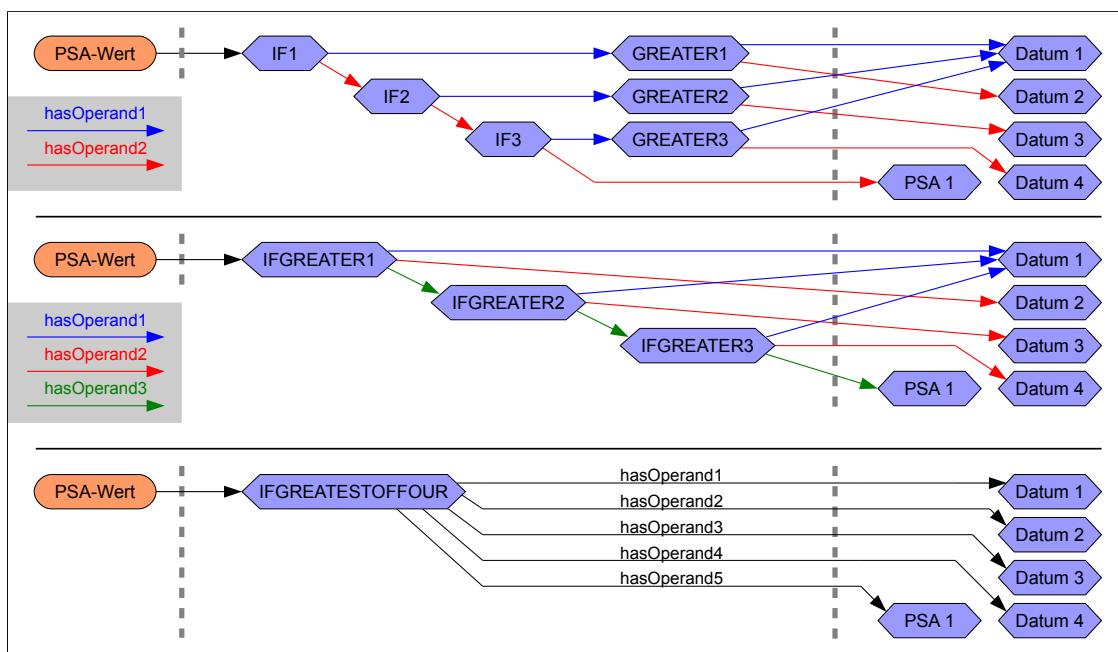
Völlig unbeachtet blieben im Rahmen dieser Arbeit Performance-Optimierungen. Beispielsweise werden die Mapping-Zwischenknoten nur in einer naiven Reihenfolge abgearbeitet. Zur Ermittlung des nächsten Knotens wird nur die Bedingung gestellt, dass die beiden Operanden-Knoten, auf die ein Knoten lesend zugreift, beide schon bearbeitet sein müssen. Die daraus resultierende Abarbeitungsreihenfolge ist in Abbildung 4.16 auf Seite 87 links unten gezeigt. Im statistischen Mittel wird das Mapping-Netz von rechts nach links abgearbeitet, wobei sich die temporäre Tabelle immer weiter füllt. Dies führt jedoch dazu, dass die Datenbankzugriffe mit der Zeit immer langsamer werden.

Es erscheint sinnvoll, das Verfahren so zu ändern, dass erst ein vollständiger Mapping-Ausdruck (in Abbildung 4.16 wären dies die vier Teilbäume mit den Datumsvergleichen sowie der Teilbaum mit den Überprüfungen auf Nicht-Existenz der PSA-Felder 1 bis 4) abgearbeitet wird, bevor mit dem nächsten begonnen wird. Danach kann das Ergebnis des Ausdrucks in die i2b2-Datenbank exportiert, die temporäre Tabelle geleert und mit dem nächsten Ausdruck fortgefahrene werden. Dies könnte durch folgende Änderung des Abarbeitungsalgorithmus erreicht werden: Statt den nächsten Knoten mit der oben genannten Regel zu ermitteln, wird die zusätzliche Bedingung gestellt, dass es sich bei einem der beiden Operand-Knoten um einen Knoten handeln muss, der während dieses Exports bearbeitet wurde (damit werden alle Dokumentationselemente, die auch als „bearbeitet“ gelten, ausgeschlossen). Erst wenn auf diese Weise kein weiterer Knoten mehr gefunden wird, wird wieder die ursprüngliche Suchstrategie angewandt.

### 5.5.4 Optimierungen durch „Superknoten“

Die Beschränkung auf lediglich zwei Operanden für jeden Zwischenknoten ermöglichte einige Vereinfachungen bei der Implementierung des Systems, ohne dass diese die Funktionalität eingeschränkt hätten. Beispielsweise müssen bei den meisten Operationen ohnehin immer zwei Datenelemente miteinander verrechnet werden. Wenn für eine Transformation jedoch mehr als nur zwei Operanden benötigt werden, so kann diese Transformation in einzelne Operationen zerlegt werden, die dann ineinander verschachtelt werden. In der Praxis hat es sich jedoch gezeigt, dass die häufige Verschachtelung von IF-Operationen mit Vergleichsoperatoren wie GREATER umständlich ist. Insbesondere bei der Formulierung von Mapping-Ausdrücken in QuickMapp wären bereits kombinierte Operationen wünschenswert, wie z. B. IFGREATER. Hierfür benötigt man jedoch drei Operanden.

Betrachtet man semantisch äquivalente Mapping-Ausdrücke, bei denen zwei, drei oder  $n$  Operanden zugelassen sind, so stellt man fest, dass diese automatisch von der einen Darstellung in die andere überführt werden können (siehe Abbildung 5.4 auf Seite 104). Beispielsweise können die IF-Operationen mit den GREATER-Operationen zu IFGREATER-Operationen zusammengefasst und umgekehrt wieder zerlegt werden. Das Verfahren ließe sich in Form eines Suche-und-Ersetze-Algorithmus umsetzen, dem die Verschmelzungs- bzw. Auftrennungsregeln bekannt sind und der direkt über dem semantischen Netz arbeitet. In QuickMapp könnte man dann die „High-Level“-Operationen mit drei Operanden verwenden, die dann automatisch in die „Low-Level“-Operationen mit nur zwei Operanden übersetzt und gespeichert werden. Auf der Seite von OntoExport könnte dann der umgekehrte Weg dazu benutzt werden, um effizientere SQL-Statements zu erzeugen.



**Abbildung 5.4:** Abbildung von 2-Operand-Knoten auf „Superknoten“ mit mehreren Operanden. Die drei gezeigten Mappings sind semantisch äquivalent und können automatisch von einer Darstellung in die anderen beiden überführt werden.

# KAPITEL 6

---

## Literaturverzeichnis

---

- [1] ALLEMANG, D. ; HENDLER, J. A.: *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Burlington, MA : Morgan Kaufmann, 2008 (Zitiert auf Seiten 14 und 25)
- [2] ALONSO-CALVO, R. ; MAJO, V. ; BILLHARDT, H. ; MARTIN-SANCHEZ, F. ; GARCIA-REMESAL, M. ; PEREZ-REY, D. : An agent- and ontology-based system for integrating public gene, protein, and disease databases. In: *J Biomed Inform* 40 (2007), Februar, S. 17–29 (Zitiert auf Seite 97)
- [3] ANGUITA, A. ; MARTIN, L. ; CRESPO, J. ; TSIKNAKIS, M. : An ontology based method to solve query identifier heterogeneity in post-genomic clinical trials. In: *Stud Health Technol Inform* 136 (2008), S. 3–8 (Zitiert auf Seite 97)
- [4] BACLAWSKI, K. ; NIU, T. : *Ontologies for Bioinformatics (Computational Molecular Biology)*. The MIT Press, 2005 (Zitiert auf Seite 92)
- [5] BAORTO, D. M. ; CIMINO, J. J. ; PARVIN, C. A. ; KAHN, M. G.: Combining laboratory data sets from multiple institutions using the logical observation identifier names and codes (LOINC). In: *Int J Med Inform* 51 (1998), Juli, S. 29–37 (Zitiert auf Seite 95)
- [6] BERNERS-LEE, T. ; FISCHETTI, M. : *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, 1999 (Zitiert auf Seite 14)
- [7] BERNERS-LEE, T. ; HENDLER, J. ; LASSILA, O. : The Semantic Web. In: *Scienc-*

- tific American* 284 (2001), Nr. 5, 34–43. <http://www.scientificamerican.com/article.cfm?id=the-semantic-web>, Abruf: 19.12.2010 (Zitiert auf Seite 14)
- [8] BIRON, P. V. ; MALHOTRA, A. : *XML Schema Part 2: Datatypes*. W3C Recommendation (Mai 2001). <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>, Abruf: 19.12.2010 (Zitiert auf Seite 20)
- [9] BODENREIDER, O. : The Unified Medical Language System (UMLS): integrating biomedical terminology. In: *Nucleic Acids Res.* 32 (2004), Januar, S. 267–270 (Zitiert auf Seiten 95 und 97)
- [10] BODENREIDER, O. : Biomedical ontologies in action: role in knowledge management, data integration and decision support. In: *Yearb Med Inform* (2008), S. 67–79 (Zitiert auf Seiten 92, 93 und 95)
- [11] BODENREIDER, O. ; NELSON, S. J. ; HOLE, W. T. ; CHANG, H. F.: Beyond synonymy: exploiting the UMLS semantics in mapping vocabularies. In: *Proc AMIA Symp* (1998), S. 815–819 (Zitiert auf Seite 95)
- [12] BODENREIDER, O. ; STEVENS, R. : Bio-ontologies: current trends and future directions. In: *Brief. Bioinformatics* 7 (2006), September, S. 256–274 (Zitiert auf Seite 92)
- [13] BRAY, T. ; PAOLI, J. ; SPERBERG-MCQUEEN, C. M. ; MALER, E. ; YERGEAU, F. ; COWAN, J. : *Extensible Markup Language (XML) 1.1 (Second Edition)*. W3C Recommendation (August 2006). <http://www.w3.org/TR/2006/REC-xml11-20060816>, Abruf: 19.12.2010 (Zitiert auf Seite 15)
- [14] BRICKLEY, D. ; GUHA, R. ; McBRIDE, B. : *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation (Februar 2004). <http://www.w3.org/TR/2004/REC-rdf-schema-20040210>, Abruf: 19.12.2010 (Zitiert auf Seite 22)
- [15] BURGUN, A. ; BODENREIDER, O. : Accessing and integrating data and knowledge for biomedical research. In: *Yearb Med Inform* (2008), S. 91–101 (Zitiert auf Seiten 92, 93 und 95)
- [16] BÜRKLE, T. : *Klassifikation, Konzeption und Anwendung medizinischer Data Dictionaries*, Medizinisches Zentrum für Ökologie, Institut für Medizinische Informatik des Klinikums der Justus-Liebig-Universität Gießen, Habilitationsschrift, 2000 (Zitiert auf Seiten 3, 92 und 100)
- [17] CADAG, E. ; LOUIE, B. ; MYLER, P. J. ; TARCY-HORNOCH, P. : Biomediator data integration and inference for functional annotation of anonymous sequences. In: *Pac Symp Biocomput* (2007), S. 343–354 (Zitiert auf Seite 97)

- [18] CIMINO, J. J.: Terminology tools: state of the art and practical lessons. In: *Methods Inf Med* 40 (2001), S. 298–306 (Zitiert auf Seite 99)
- [19] CIMINO, J. J. ; BARNETT, G. O.: Automated translation between medical terminologies using semantic definitions. In: *MD Comput* 7 (1990), S. 104–109 (Zitiert auf Seite 95)
- [20] CIMINO, J. J. ; JOHNSON, S. B. ; PENG, P. ; AGUIRRE, A. : From ICD9-CM to MeSH using the UMLS: a how-to guide. In: *Proc Annu Symp Comput Appl Med Care* (1993), S. 730–734 (Zitiert auf Seite 95)
- [21] CIMINO, J. J. ; ZHU, X. : The practical impact of ontologies on biomedical informatics. In: *Yearb Med Inform* (2006), S. 124–135 (Zitiert auf Seite 92)
- [22] CIMINO, J. J.: Desiderata for Controlled Medical Vocabularies in the Twenty-First Century. In: *Methods of Information in Medicine*, S. 394–403 (Zitiert auf Seite 100)
- [23] COLSMAN, A. ; KUNZMANN, U. ; SEGGEWIES, C. ; MAHLER, V. : Arztbriefschreibung in der Dermatologie im Rahmen eines elektronischen Patientendatenmanagements. In: *Der Hautarzt* 60 (2009), S. 821–825 (Zitiert auf Seite 9)
- [24] CORONADO, S. de ; HABER, M. W. ; SIOUTOS, N. ; TUTTLE, M. S. ; WRIGHT, L. W.: NCI Thesaurus: using science-based terminology to integrate cancer research results. In: *Stud Health Technol Inform* 107 (2004), S. 33–37 (Zitiert auf Seite 93)
- [25] DEUTSCHES PROSTATAKARZINOM KONSORTIUM E. V. (DPKK): *Webseite des DPKK e. V.* <http://www.dpkk.de>, Abruf: 19.12.2010 (Zitiert auf Seite 2)
- [26] DHANAPALAN, L. ; CHEN, J. Y.: A case study of integrating protein interaction data using semantic web technology. In: *Int J Bioinform Res Appl* 3 (2007), S. 286–302 (Zitiert auf Seite 94)
- [27] DOAN, A. ; DOMINGOS, P. ; HALEVY, A. : Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In: *In SIGMOD Conference*, 2001, S. 509–520 (Zitiert auf Seite 95)
- [28] DOLIN, R. H. ; HUFF, S. M. ; ROCHA, R. A. ; SPACKMAN, K. A. ; CAMPBELL, K. E.: Evaluation of a „lexically assign, logically refine“ strategy for semi-automated integration of overlapping terminologies. In: *J Am Med Inform Assoc* 5 (1998), S. 203–213 (Zitiert auf Seite 95)
- [29] DUGAS, M. ; LANGE, M. ; MULLER-TIDOW, C. ; KIRCHHOF, P. ; PROKOSCH, H.-U. :

- Routine data from hospital information systems can support patient recruitment for clinical studies. In: *Clin Trials* 7 (2010), April, Nr. 2, S. 183–189 (Zitiert auf Seite 3)
- [30] EL EMAM, K. ; JONKER, E. ; SAMPSON, M. ; KRLEZA-JERIĆ, K. ; NEISA, A. : The Use of Electronic Data Capture Tools in Clinical Trials: Web-Survey of 259 Canadian Trials. In: *Journal of Medical Internet Research* 11 (2009), März, Nr. 1 (Zitiert auf Seite 9)
- [31] ENGEL, P. A.: *EDV im Krankenhaus: Evaluation der Einführung der elektronischen Arztbriefschreibung am UKM*, Westfälischen Wilhelms-Universität Münster, Inaugural-Dissertation, 2004 (Zitiert auf Seite 13)
- [32] FOLEY, J. D. ; DAM, A. van ; FEINER, S. K. ; HUGHES, J. F.: *Computer Graphics: Principles and Practice (Second Edition)*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1990 (Zitiert auf Seite 9)
- [33] FRITZ, F. ; STANDER, S. ; BREIL, B. ; DUGAS, M. : Steps towards single source-collecting data about quality of life within clinical information systems. In: *Stud Health Technol Inform* 160 (2010), S. 188–192 (Zitiert auf Seite 3)
- [34] FRITZ, F. ; DZIUBALLE, P. ; HERZBERG, S. ; BREIL, B. ; DUGAS, M. : Single-Source-Ansätze am Universitätsklinikum Münster. In: *mdi Forum der Medizin-Dokumentation und Medizin-Informatik* 2 (2010), S. 57–59 (Zitiert auf Seite 3)
- [35] FUNG, K. W. ; BODENREIDER, O. ; ARONSON, A. R. ; HOLE, W. T. ; SRINIVASAN, S. : Combining lexical and semantic methods of inter-terminology mapping using the UMLS. In: *Stud Health Technol Inform* 129 (2007), S. 605–609 (Zitiert auf Seite 95)
- [36] GANSLANDT, T. ; JANTSCH, S. ; MASCHER, K. ; PROKOSCH, H.-U. : Digging for hidden gold: timeline-based visualization of heterogeneous clinical data. In: *Journal for quality of life research* 3 (2005), S. 82–84 (Zitiert auf Seite 30)
- [37] GANSLANDT, T. ; PROKOSCH, H.-U. ; BEYER, A. ; SCHWENK, M. ; STARKE, K. ; BÄRTHLEIN, B. ; KÖPCKE, F. ; MATE, S. ; MARTIN, M. ; SCHÖCH, W. ; SEGGEWIES, C. ; BÜRKLE, T. : Single-Source-Projekte am Universitätsklinikum Erlangen. In: *mdi Forum der Medizin-Dokumentation und Medizin-Informatik* 2 (2010), S. 62–66 (Zitiert auf Seite 3)
- [38] GELL, G. ; SCHMUCKER, P. ; PEDEVILLA, M. ; LEITNER, H. ; NAUMANN, J. ; FUCHS, H. ; PITZ, H. ; KOLE, W. : SAP and partners: IS-H and IS-H\* MED. In: *Methods Inf Med* 42 (2003), S. 16–24 (Zitiert auf Seite 9)
- [39] GHAZVINIAN, A. ; NOY, N. F. ; MUSEN, M. A.: Creating mappings for ontologies

- in biomedicine: simple methods work. In: *AMIA Annu Symp Proc* 2009 (2009), S. 198–202 (Zitiert auf Seiten 95 und 96)
- [40] GROUP, W. H. W.: *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation (Januar 2000, überarbeitet August 2002). <http://www.w3.org/TR/2002/REC-xhtml1-20020801>, Abruf: 19.12.2010 (Zitiert auf Seite 15)
- [41] HAUX, R. ; SEGGEWIES, C. ; SOBEZ, B. W. ; KULLMANN, P. ; REICHERT, H. ; LUEDECKE, L. ; SEIBOLD, H. : Soarian–workflow management applied for health care. In: *Methods Inf Med* 42 (2003), Nr. 1, S. 25–36 (Zitiert auf Seite 9)
- [42] HEBELER, J. ; FISHER, M. ; BLACE, R. : *Semantic web programming*. Indianapolis, Ind. : Wiley, 2009 (Zitiert auf Seiten 17, 21, 25, 26, 27, 94, 100, 121 und 123)
- [43] HERNANDEZ, T. ; KAMBHAMPATI, S. : Integration of biological sources: current systems and challenges ahead. In: *SIGMOD Rec.* 33 (2004), September, S. 51–60 (Zitiert auf Seite 97)
- [44] HITZLER, P. ; KRÖTZSCH, M. ; RUDOLPH, S. ; SURE, Y. : *Semantic Web: Grundlagen*. Springer, 2008 (Zitiert auf Seiten 15, 22, 23 und 25)
- [45] HORROCKS, I. ; PATEL-SCHNEIDER, P. F. ; BOLEY, H. ; TABET, S. ; GROSOFAND, B. ; DEAN, M. : *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission (Mai 2004). <http://www.w3.org/Submission/SWRL>, Abruf: 19.12.2010 (Zitiert auf Seite 26)
- [46] HUNTER, J. ; CRAWFORD, W. : *Java Servlet Programming*. O'Reilly & Associates, Inc., 2001 (Zitiert auf Seite 89)
- [47] INGENERF, J. ; REINER, J. ; SEIK, B. : Standardized terminological services enabling semantic interoperability between distributed and heterogeneous systems. In: *Int J Med Inform* 64 (2001), Dezember, S. 223–240 (Zitiert auf Seite 95)
- [48] INSTITUT FÜR MEDIZINISCHE INFORMATIK, STATISTIK UND EPIDEMIOLOGIE (IMISE), MEDIZINISCHE FAKULTÄT, UNIVERSITÄT LEIPZIG: Workshop zum BMBF-Projekt „Metadata Repository“ am 16.06.2010 (Zitiert auf Seite 4)
- [49] KHAN, A. N. ; GRIFFITH, S. P. ; MOORE, C. ; RUSSELL, D. ; ROSARIO, A. C. ; BERTOLLI, J. : Standardizing laboratory data by mapping to LOINC. In: *J Am Med Inform Assoc* 13 (2006), S. 353–355 (Zitiert auf Seite 95)
- [50] KLYNE, G. ; CARROLL, J. J. ; McBRIDE, B. : *Resource Description Framework*

- (RDF): *Concepts and Abstract Syntax*. W3C Recommendation (Februar 2004). <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210>, Abruf: 19.12.2010 (Zitiert auf Seite 18)
- [51] KRANKENHAUS IT-JOURNAL: Pilotprojekt: Soarian an der Universitätsklinik Erlangen. Pressegespräch mit Prof. Dr. Hans-Ulrich Prokosch und Martin Skerra. In: *Krankenhaus IT-Journal* 1 (2005), S. 22–24 (Zitiert auf Seite 9)
- [52] KUHN, K. A. ; LENZ, R. ; ELSTNER, T. ; SIEGELE, H. ; MOLL, R. : Experiences with a generator tool for building clinical application modules. In: *Methods Inf Med* 42 (2003), S. 37–44 (Zitiert auf Seite 13)
- [53] LAM, H. Y. ; MARENCO, L. ; SHEPHERD, G. M. ; MILLER, P. L. ; CHEUNG, K. H.: Using web ontology language to integrate heterogeneous databases in the neurosciences. In: *AMIA Annu Symp Proc* (2006), S. 464–468 (Zitiert auf Seite 94)
- [54] LAU, L. M. ; JOHNSON, K. ; MONSON, K. ; LAM, S. H. ; HUFF, S. M.: A method for the automated mapping of laboratory results to LOINC. In: *Proc AMIA Symp* (2000), S. 472–476 (Zitiert auf Seite 95)
- [55] LEVENSHTEIN, V. : Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In: *Soviet Physics Doklady* 10 (1966), S. 707 (Zitiert auf Seite 96)
- [56] LINDBERG, D. A. ; HUMPHREYS, B. L. ; MCCRAY, A. T.: The Unified Medical Language System. In: *Methods Inf Med* 32 (1993), August, S. 281–291 (Zitiert auf Seite 95)
- [57] LOUIE, B. ; MORK, P. ; SHAKER, R. ; KOLKER, N. ; KOLKER, E. ; TARCY-HORNOCH, P. : Integration of data for gene annotation using the BioMediator system. In: *AMIA Annu Symp Proc* (2005), S. 1036 (Zitiert auf Seite 97)
- [58] MAOJO, V. ; CRESPO, J. ; CALLE, G. de l. ; BARREIRO, J. ; GARCIA-REMESAL, M. : Using web services for linking genomic data to medical information systems. In: *Methods Inf Med* 46 (2007), S. 484–492 (Zitiert auf Seite 97)
- [59] MASARIE, F. E. ; MILLER, R. A. ; BOUHADDOU, O. ; GIUSE, N. B. ; WARNER, H. R.: An interlingua for electronic interchange of medical information: using frames to map between clinical vocabularies. In: *Comput. Biomed. Res.* 24 (1991), August, S. 379–400 (Zitiert auf Seite 95)
- [60] MATE, S. : *Evaluation von i2b2 am Universitätsklinikum Erlangen*, Friedrich-Alexander-Universität Erlangen-Nürnberg, Studienarbeit, 2009 (Zitiert auf Seiten 2, 8, 33, 39, 56, 57, 89 und 91)

- [61] MATE, S. ; BECKER, A. ; PROKOSCH, H.-U. ; GANSLANDT, T. : Evaluation der Einsetzbarkeit von i2b2 im deutschen Umfeld. In: *Abstractband der 54. GMDS-Jahrestagung*, 2009, S. 299–300 (Zitiert auf Seite 8)
- [62] MURPHY, S. N. ; MENDIS, M. ; HACKETT, K. ; KUTTAN, R. ; PAN, W. ; PHILLIPS, L. C. ; GAINER, V. ; BERKOWICZ, D. ; GLASER, J. P. ; KOHANE, I. ; CHUEH, H. C.: Architecture of the open-source clinical research chart from Informatics for Integrating Biology and the Bedside. In: *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium* (2007), S. 548–552 (Zitiert auf Seite 7)
- [63] MURPHY, S. N. ; MENDIS, M. E. ; BERKOWITZ, D. A. ; KOHANE, I. ; CHUEH, H. C.: Integration of clinical and genetic data in the i2b2 architecture. In: *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium* (2006), S. 1040 (Zitiert auf Seite 7)
- [64] MURPHY, S. N. ; WEBER, G. ; MENDIS, M. ; GAINER, V. ; CHUEH, H. C. ; CHURCHILL, S. ; KOHANE, I. : Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). In: *Journal of the American Medical Informatics Association : JAMIA* 17 (2010), März, Nr. 2, S. 124–130. – ISSN 1527-974X (Zitiert auf Seite 8)
- [65] MÜLLER, P. : *Mehrfachnutzung von Daten der Elektronischen Krankenakte am Beispiel der Dokumentation für das kolorektale Karzinom*, Friedrich-Alexander-Universität Erlangen-Nürnberg, Studienarbeit, 2010 (Zitiert auf Seiten 12, 13 und 94)
- [66] NADKARNI, P. M. ; BRANDT, C. A.: The Common Data Elements for Cancer Research: Remarks on Functions and Structure. In: *Methods of Information in Medicine* 45 (2006), S. 594–601 (Zitiert auf Seiten 4 und 100)
- [67] NADKARNI, P. M. ; MARENCO, L. ; CHEN, R. ; SKOUFOS, E. ; SHEPHERD, G. ; MILLER, P. : Organization of heterogeneous scientific data using the EAV/CR representation. In: *Journal of the American Medical Informatics Association* 6 (1999), Nr. 6, S. 478–493 (Zitiert auf Seite 50)
- [68] NOY, N. F. ; CORONADO, S. de ; SOLBRIG, H. ; FRAGOSO, G. ; HARTEL, F. W. ; MUSSEN, M. A.: Representing the NCI Thesaurus in OWL DL: Modeling tools help modeling languages. In: *Appl Ontol* 3 (2008), Januar, S. 173–190 (Zitiert auf Seite 93)
- [69] OSCHEM, M. : *Evaluation und Schaffung eines objektivierbaren Vergleichs von Methoden zur Optimierung der Arztbriefschreibung in der Hautklinik des Universitätsklinikums Erlangen*, Friedrich-Alexander-Universität Erlangen-Nürnberg, Inaugural-Dissertation, 2010 (Zitiert auf Seite 9)

- [70] PARK, Y. R. ; KIM, H. J.: Metadata registry and management system based on ISO 11179 for cancer clinical trials information system. In: *AMIA 2006 Symposium Proceedings*, 2006, S. 1056 (Zitiert auf Seite 3)
- [71] PATEL, A. A. ; GILBERTSON, J. R. ; PARWANI, A. V. ; DHIR, R. ; DATTA, M. W. ; GUPTA, R. ; BERMAN, J. J. ; MELAMED, J. ; KAJDACSY-BALLA, A. ; ORENSTEIN, J. ; BECICH, M. J.: An informatics model for tissue banks—lessons learned from the Cooperative Prostate Cancer Tissue Resource. In: *BMC Cancer* 6 (2006) (Zitiert auf Seiten 3 und 93)
- [72] PATEL, A. A. ; KAJDACSY-BALLA, A. ; BERMAN, J. J. ; BOSLAND, M. ; DATTA, M. W. ; DHIR, R. ; GILBERTSON, J. R. ; MELAMED, J. ; ORENSTEIN, J. ; TEI, K.-F. ; BECICH, M. J.: The development of common data elements for a multi-institute prostate cancer tissue bank: The Cooperative Prostate Cancer Tissue Resource (CPCTR) experience. In: *BMC Cancer* 5 (2005) (Zitiert auf Seiten 3, 93 und 100)
- [73] PEREZ-REY, D. ; MAOJO, V. ; GARCIA-REMESAL, M. ; ALONSO-CALVO, R. ; BILLHARDT, H. ; MARTIN-SANCHEZ, F. ; SOUSA, A. : ONTOFUSION: ontology-based integration of genomic and clinical databases. In: *Comput. Biol. Med.* 36 (2006), S. 712–730 (Zitiert auf Seite 97)
- [74] PROKOSCH, H.-U. : Datenmodellierung und Datenbankdesign für relationale Datenbanken. In: *Software Kurier für Mediziner und Psychologen* 4 (1991), S. 39–45 (Zitiert auf Seiten 37, 50 und 62)
- [75] PROKOSCH, H.-U. : KAS, KIS, EKA, EPA, EGA, E-Health - ein Plädoyer gegen die babylonische Begriffsverwirrung in der Medizinischen Informatik. In: *Informatik, Biometrie und Epidemiologie in Medizin und Biologie* 32,4 (2001), S. 371–382 (Zitiert auf Seite 1)
- [76] PROKOSCH, H.-U. : *Abschlussbericht des TMF-Projekts „IT-Strategie“*. Februar 2010 (Zitiert auf Seiten 2 und 8)
- [77] PROKOSCH, H.-U. : Single-Source-Aktivitäten in Deutschland. In: *mdi Forum der Medizin-Dokumentation und Medizin-Informatik* 2 (2010), S. 56–57 (Zitiert auf Seiten 2, 3 und 8)
- [78] PROKOSCH, H.-U. ; GANSLANDT, T. : Perspectives for Medical Informatics. Reusing the Electronic Medical Record for Clinical Research. In: *Methods of Information in Medicine* 48 (2009), S. 38–44 (Zitiert auf Seite 3)
- [79] PRUD'HOMMEAUX, E. ; SEABORNE, A. : *SPARQL Query Language for RDF*. W3C

- Recommendation (Januar 2008). <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115>, Abruf: 19.12.2010 (Zitiert auf Seite 25)
- [80] REIMANN, L. : *Concept for the integration of pseudonymization services in a translational research database*, Georg-August-Universität Göttingen, Masterarbeit, 2010 (Zitiert auf Seite 56)
- [81] RENG, C.-M. ; DEBOLD, P. ; SPECKER, C. ; POMMERENING, K. : *Generische Lösungen zum Datenschutz für die Forschungsnetze in der Medizin*. Medizinische Wissenschaftliche Verlagsgesellschaft, 2006 (Zitiert auf Seiten 2 und 56)
- [82] RUBIN, D. L. ; LEWIS, S. E. ; MUNGALL, C. J. ; MISRA, S. ; WESTERFIELD, M. ; ASHBURNER, M. ; SIM, I. ; CHUTE, C. G. ; SOLBRIG, H. ; STOREY, M. A. ; SMITH, B. ; DAY-RICHTER, J. ; NOY, N. F. ; MUSEN, M. A.: National Center for Biomedical Ontology: advancing biomedicine through structured organization of scientific knowledge. In: *OMICS* 10 (2006), S. 185–198 (Zitiert auf Seite 93)
- [83] RUBIN, D. L. ; SHAH, N. H. ; NOY, N. F.: Biomedical ontologies: a functional perspective. In: *Brief. Bioinformatics* 9 (2008), Januar, S. 75–90 (Zitiert auf Seiten 92 und 97)
- [84] RUTTENBERG, A. ; CLARK, T. ; BUG, W. ; SAMWALD, M. ; BODENREIDER, O. ; CHEN, H. ; DOHERTY, D. ; FORSBERG, K. ; GAO, Y. ; KASHYAP, V. ; KINOSHITA, J. ; LUCIANO, J. ; MARSHALL, M. S. ; OGBUJI, C. ; REES, J. ; STEPHENS, S. ; WONG, G. T. ; WU, E. ; ZACCAGNINI, D. ; HONGSERMEIER, T. ; NEUMANN, E. ; HERMAN, I. ; CHEUNG, K. H.: Advancing translational research with the Semantic Web. In: *BMC Bioinformatics* 8 Suppl 3 (2007), S. S2 (Zitiert auf Seiten 94, 100 und 101)
- [85] SCHNEIDER, H. : Der Single-Source-Ansatz am Universitätsklinikum Essen. In: *mdi Forum der Medizin-Dokumentation und Medizin-Informatik* 2 (2010), S. 60–61 (Zitiert auf Seite 3)
- [86] SIOUTOS, N. ; CORONADO, S. de ; HABER, M. W. ; HARTEL, F. W. ; SHAIU, W. L. ; WRIGHT, L. W.: NCI Thesaurus: a semantic model integrating cancer-related clinical and molecular information. In: *J Biomed Inform* 40 (2007), Februar, S. 30–43 (Zitiert auf Seite 93)
- [87] SMITH, M. K. ; WELTY, C. ; MCGUINNESS, D. L.: *OWL Web Ontology Language Guide*. W3C Recommendation (Februar 2004). <http://www.w3.org/TR/2004/REC-owl-guide-20040210>, Abruf: 19.12.2010 (Zitiert auf Seite 23)
- [88] SOLBRIG, H. R.: Metadata and the Reintegration of Clinical Information: ISO

11179. In: *MD computing: computers in medical practice* (2000), S. 25–28 (Zitiert auf Seite 3)
- [89] SPERZEL, D. ; ERLBAUM, M. ; FULLER, L. ; SHERERTZ, D. ; OLSON, N. ; SCHUYLER, P. ; HOLE, W. ; SAVAGE, A. ; PASSARELLI, P. ; TUTTLE, M. : Editing the UMLS Metathesaurus: Review and Enhancement of a Computed Knowledge Source. (1990). <http://www.ncbi.nlm.nih.gov/article/fcgi?artid=2245573> (Zitiert auf Seite 95)
- [90] STAUSBERG, J. ; LÖBE, M. ; VERPLANCKE, P. ; DREPPER, J. ; HERRE, H. ; LÖFFLER, M. : Foundations of a Metadata Repository for Databases of Registers and Trials. In: ADLASSNIG, K.-P. (Hrsg.) ; BLOBEL, B. (Hrsg.) ; MANTAS, J. (Hrsg.) ; MASIC, I. (Hrsg.): *Medical Informatics in a United and Healthy Europe, Proceedings of MIE 2009*, 2009, S. 409–413 (Zitiert auf Seiten 3 und 100)
- [91] STÖCKLE, M. ; UNTEREGGER, G. ; WULLICH, B. ; ZWERGEL, T. : Fachnachricht: Netzwerk zur Erforschung des Prostatakarzinoms. In: *Der Urologe* 42 (2003), S. 484 (Zitiert auf Seite 2)
- [92] SUJANSKY, W. : *A formal model for bridging heterogenous databases in clinical medicine*, Stanford University, Ph. D. thesis, 1996 (Zitiert auf Seite 97)
- [93] SUJANSKY, W. : Heterogeneous database integration in biomedicine. In: *J Biomed Inform* 34 (2001), August, S. 285–298 (Zitiert auf Seite 96)
- [94] SUJANSKY, W. ; ALTMAN, R. : An evaluation of the TransFER model for sharing clinical decision-support applications. In: *Proc AMIA Annu Fall Symp* (1996), S. 468–472 (Zitiert auf Seite 97)
- [95] SUN, J. Y. ; SUN, Y. : A system for automated lexical mapping. In: *J Am Med Inform Assoc* 13 (2006), S. 334–343 (Zitiert auf Seite 95)
- [96] SUN, Y. : Methods for automated concept mapping between medical databases. In: *J Biomed Inform* 37 (2004), Juni, S. 162–178 (Zitiert auf Seiten 95 und 97)
- [97] TURING, A. M.: On Computable Numbers, with an application to the Entscheidungsproblem. In: *Proc. London Math. Soc.* 2 (1936), Nr. 42, S. 230–265 (Zitiert auf Seite 96)
- [98] UNIVERSITÄTSKLINIKUM ERLANGEN: *Erstes Universitäts-Prostatakrebs-Zentrum in Bayern gegründet*. [http://www.uk-erlangen.de/e467/e583/e11287/e13374/index\\_ger.html](http://www.uk-erlangen.de/e467/e583/e11287/e13374/index_ger.html), Abruf: 19.12.2010 (Zitiert auf Seite 9)

- [99] W3C OWL WORKING GROUP: *OWL 2 Web Ontology Language Document Overview*. W3C Recommendation (Oktober 2009). <http://www.w3.org/TR/owl2-overview>, Abruf: 19.12.2010 (Zitiert auf Seite 26)
- [100] WANG, K. ; TARCZY-HORNOCH, P. ; SHAKER, R. ; MORK, P. ; BRINKLEY, J. F.: BioMediator data integration: beyond genomics to neuroscience data. In: *AMIA Annu Symp Proc* (2005), S. 779–783 (Zitiert auf Seite 97)
- [101] WEBER, G. M. ; MURPHY, S. N. ; McMURRY, A. J. ; MACFADDEN, D. ; NIGRIN, D. J. ; CHURCHILL, S. ; KOHANE, I. S.: The Shared Health Research Information Network (SHRINE): A Prototype Federated Query Tool for Clinical Data Repositories. In: *Journal of the American Medical Informatics Association* 16 (2009), Juni, Nr. 5, 624–630. <http://dx.doi.org/10.1197/jamia.M3191> (Zitiert auf Seite 55)
- [102] WELKER, J. A.: Implementation of electronic data capture systems: Barriers and solutions. In: *Contemporary Clinical Trials* 28 (2007), Nr. 3, S. 329–336 (Zitiert auf Seite 9)
- [103] WERNERT, N. ; WULLICH, B. ; UNTEREGGER, G. : 7. Interdisziplinärer Workshop des Deutschen Prostatakarzinom- Konsortiums (DPKK) e.V. In: *Der Urologe* 49 (2010), S. 415 (Zitiert auf Seite 2)
- [104] WULLICH, B. ; WERNERT, N. ; UNTEREGGER, G. : 6. Interdisziplinärer Workshop des Deutschen Prostatakarzinom- Konsortiums (DPKK) e.V. In: *Der Urologe* 48 (2009), S. 299 (Zitiert auf Seite 2)
- [105] YU, A. C.: Methods in biomedical ontology. In: *J Biomed Inform* 39 (2006), Juni, S. 252–266 (Zitiert auf Seite 92)



# ANHANG A

---

## Ergänzungen zu RDF, RDFS und OWL

---

### A.1 RDF-Serialisierungen

Die folgenden Auflistungen zeigen einige Serialisierungen der kleinen Beispiel-Ontologie aus Abschnitt 2.3.3. Die Ontologie wurde in Abbildung 2.10 auf Seite 20 gezeigt.

```
1 <rdf:RDF
2   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:p1="http://www.uk-erlangen.de/Liebesontologie#"
4   xmlns:owl="http://www.w3.org/2002/07/owl#"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
6   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
7   <rdf:Description rdf:about="http://www.uk-erlangen.de/Liebesontologie#Francesco
8     "做人
9       <p1:liebt rdf:resource="http://www.uk-erlangen.de/Liebesontologie#Doris"/>
10      <p1:hatSpitzname>Amor</p1:hatSpitzname>
11    </rdf:Description>
12    <rdf:Description rdf:about="http://www.uk-erlangen.de/Liebesontologie#Doris">
13      <p1:hasst rdf:resource="http://www.uk-erlangen.de/Liebesontologie#Francesco"/>
14    </rdf:Description>
</rdf:RDF>
```

**Auflistung A.1:** RDF/XML-Serialisierung, erzeugt mit dem Jena-Framework. Dieses Format ist der De-Facto-Standard für RDF. Für das menschliche Auge ist dieser Syntax jedoch schlecht zu lesen.

```

1 <http://www.uk-erlangen.de/Liebesontologie#Francesco> <http://www.uk-erlangen.de/
2   Liebesontologie#liebt> <http://www.uk-erlangen.de/Liebesontologie#Doris> .
3 <http://www.uk-erlangen.de/Liebesontologie#Francesco> <http://www.uk-erlangen.de/
4   Liebesontologie#hatSpitzname> "Amor" .
5 <http://www.uk-erlangen.de/Liebesontologie#Doris> <http://www.uk-erlangen.de/
6   Liebesontologie#hasst> <http://www.uk-erlangen.de/Liebesontologie#Francesco> .

```

**Auflistung A.2:** N-Tripel-Darstellung von Auflistung A.1. Wegen der langen URIs und den daraus resultierenden Zeilenumbrüchen ist auch diese Darstellung schwierig zu lesen. Hier werden keine Namespaces deklariert, da die Ressourcen bereits aus vollständigen URIs bestehen.

```

1 <Francesco> <liebt> <Doris> .
2 <Francesco> <hatSpitzname> "Amor" .
3 <Doris> <hasst> <Francesco> .

```

**Auflistung A.3:** Der Inhalt von Auflistung A.2, bei dem aus jeder URI das `http://www.uk-erlangen.de/Liebesontologie#` entfernt wurde. Jetzt erkennt man leicht die einzelnen Tripel und damit das einfache Datenmodell.

```

1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2 @prefix p1: <http://www.uk-erlangen.de/Liebesontologie#> .
3 @prefix owl: <http://www.w3.org/2002/07/owl#> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

6
7 p1:Francesco
8   p1:hatSpitzname "Amor" ;
9   p1:liebt p1:Doris .
10
11 p1:Doris
12   p1:hasst p1:Francesco .

```

**Auflistung A.4:** Turtle-Serialisierung der Beispiel-Ontologie, erzeugt durch das Jena-Framework. Zeilen 7-9 zeigen eine Turtle-Abkürzung: das doppelt verwendete Subjekt `Francesco` wird hier nur einmal notiert, die dazugehörenden Prädikate und Objekte werden mit einem Strichpunkt getrennt.

## A.2 Von der Manchester-Syntax zur Tripel-Darstellung

Die folgenden Auflistungen sollen illustrieren, dass auch bei komplexen Klassendefinitionen in Manchester-Syntax einfache RDF-Tripel die Grundlage bilden. Ausgangspunkt ist die Klassendefinition von `Casanova`, wie sie in Protégé 4 eingeben wird:

MännlicherMensch and (liebt min 2 WeiblicherMensch)

```

1 Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
2 Prefix: swrlb: <http://www.w3.org/2003/11/swrlb#>
3 Prefix: owl: <http://www.w3.org/2002/07/owl#>
4 Prefix: protege: <http://protege.stanford.edu/plugins/owl/protege#>
5 Prefix: : <http://www.uk-erlangen.de/Liebesontologie#>
6 Prefix: xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#>
7 Prefix: xml: <http://www.w3.org/XML/1998/namespace>
8 Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
9 Prefix: swrl: <http://www.w3.org/2003/11/swrl#>
10 Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
11 Prefix: skos: <http://www.w3.org/2004/02/skos/core#>
12
13 Ontology: <http://www.uk-erlangen.de/Liebesontologie>
14
15 Object-Property: liebt
16 Class: owl:Thing
17 Class: WeiblicherMensch
18 Class: MaennlicherMensch
19
20 Class: Casanova
21     EquivalentTo:
22         MaennlicherMensch
23         and (liebt min 2 WeiblicherMensch)
24     SubClassOf:
25         owl:Thing

```

**Auflistung A.5:** Serialisierung der Klasse Casanova in Manchester-Syntax, erzeugt mit Hilfe von Protégé 4 (unnötige Leerzeilen wurden entfernt).

```

1 @prefix p1: <http://www.uk-erlangen.de/Liebesontologie#> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix owl: <http://www.w3.org/2002/07/owl#> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6
7 p1:Casanova
8     a owl:Class ;
9     rdfs:subClassOf owl:Thing ;
10    owl:equivalentClass
11        [ a owl:Class ;
12            owl:intersectionOf (p1:MaennlicherMensch [ a owl:Restriction ;
13                owl:minQualifiedCardinality "2"^^xsd:
14                    nonNegativeInteger ;
15                    owl:onClass p1:WeiblicherMensch ;
16                    owl:onProperty p1:liebt
17                ])
18        ] .

```

**Auflistung A.6:** Serialisierung der Klasse Casanova (OWL 2) in Turtle-Syntax, erzeugt durch das Jena-Framework.

```

1 @prefix p1: <http://www.uk-erlangen.de/Liebesontologie#> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix owl: <http://www.w3.org/2002/07/owl#> .
4 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6
7 p1:Casanova rdf:type owl:Class .
8 p1:Casanova owl:equivalentClass _:A .
9 p1:Casanova rdfs:subClassOf owl:Thing .
10
11 _:A rdf:type owl:Class .
12 _:A owl:intersectionOf _:B .
13
14 _:B rdf:first p1:MaennlicherMensch .
15 _:B rdf:rest _:C .
16
17 _:C rdf:first _:D .
18 _:C rdf:rest rdf:nil .
19
20 _:D rdf:type owl:Restriction .
21 _:D owl:onProperty p1:liebt .
22 _:D owl:onClass p1:WeiblicherMensch .
23 _:D owl:minQualifiedCardinality "2"^^xsd:nonNegativeInteger .

```

**Auflistung A.7:** Serialisierung der Klasse Casanova (OWL 2) in Turtle-Syntax, bei der alle Turtle-Abkürzungen (vgl. Auflistung A.6) entfernt wurden, um die einzelnen Tripel sichtbar zu machen. Dies ist händisch über einen Zwischenschritt in N-Tripel-Serialisierung erfolgt.

Die Statements in Auflistung A.7 sind so zu lesen: die Klasse Casanova (Zeile 7) ist äquivalent zu einer Klasse \_:A (8), die aus der Schnittmenge (12) der Instanzen der Klasse MaennlicherMensch (14) und der Klasse \_:D besteht. Die Klasse \_:D (20-23) beschreibt alle Instanzen, die mindestens zwei (23) liebt-Properties (21) zu Instanzen der Klasse WeiblicherMensch besitzen.

Anmerkung: Das Beispiel entstand mit Protégé 4.1, das bereits OWL2 verwendet. Wird das Beispiel mit Protégé 3.4.4 durchgespielt, ändern sich die Zeilen 13-15 in Auflistung A.6 von

```

13   owl:minQualifiedCardinality "2"^^xsd:nonNegativeInteger ;
14   owl:onProperty p1:liebt ;
15   owl:onClass p1:WeiblicherMensch

```

gemäß OWL1 zu:

```

13   owl:minCardinality "2"^^xsd:int ;
14   owl:onProperty p1:liebt ;
15   owl:valuesFrom p1:WeiblicherMensch

```

### A.3 Konstrukte in OWL

Die folgende Zusammenstellung basiert auf [42, S. 525-532], verzichtet aber auf die OWL2-Konstrukte und versucht, die Zusammenhänge formaler zu beschreiben. Für alle Tabellen gilt (wenn nicht anders angegeben): (A p B) sind Statements mit Subjekt A, Prädikat p und Objekt B. A, B, und C können Klassen, Individuen oder Properties sein. Das p steht für die jeweilige Property in der linken Spalte. (B, C, ...) steht für eine RDF-Collection, wobei B, C, ... eine Aufzählung von RDF-Ressourcen ist (siehe <http://www.w3.org/TR/REC-rdf-syntax/#section-Syntax-parsetype-Collection>).

Property p	Definition
rdf:type	(A p B) gibt an, dass A vom Typ B ist. Wenn B eine Klasse (RDFS oder OWL) ist, ist A ein Mitglied von Klasse B.
owl:sameAs	(A p B) gibt an, dass die beiden Individuen A und B das selbe sind.
owl:differentFrom	(A p B) gibt an, dass die Individuen A und B nicht das selbe Individuum sind.
owl:AllDifferent	Eine Klasse, mit der zusammen mit owl:distinctMembers eine Menge an Individuen angegeben werden kann, die paarweise unterschiedlich sind. Siehe Auflistung A.8.

Tabelle A.1: RDFS- und OWL-Properties zur Beschreibung von Individuen.

```

1 []      a          owl:AllDifferent ;
2      owl:distinctMembers (p1:Anita p1:Anton p1:Doris p1:Francesco p1:Silvia) .

```

**Auflistung A.8:** Beispiel zu owl:AllDifferentFrom in Turtle-Syntax. Es besagt, dass die aufgezählten Individuen alle unterschiedliche sind. [] ist eine RDF Blank Node.

Property p	Definition
rdfs:subClassOf	(A p B) gibt an, dass A eine Unterklasse (Spezialisierung) von B ist.
owl:equivalentClass	(A p B) gibt an, dass die Klassen A und B äquivalent sind.
owl:Thing	Die Klasse aller Individuen. Werden in OWL Klassen definiert, sind diese automatisch Unterklassen von owl:Thing.
owl:Nothing	Das Komplement von owl:Thing. Das ist die Klasse, die nichts enthält.
owl:oneOf	(A p (B, C, ...)) gibt an, dass die möglichen Mitglieder der Klasse A nur Individuen der Individuen in (B, C, ...) sein können.
owl:intersectionOf	(A p (B, C, ...)) gibt an, dass die Mitglieder der Klasse A alle Mitglieder in den angegebenen Klassen (B, C, ...) sein müssen (Schnittmenge).
owl:unionOf	(A p (B, C, ...)) gibt an, dass die Mitglieder der Klasse A die Mitglieder aus der angegebenen Klassen (B, C, ...) sind (Vereinigung).
owl:complementOf	(A p B) bedeutet, dass A das Komplement von B ist. Beispiel: Fleisch owl:complementOf NichtFleisch.
owl:disjointWith	(A p B) Die Klasse A ist disjunkt von Klasse B. D.h., dass ein Individuum, das Mitglied von Klasse A ist, keines von Klasse B sein kann (und umgekehrt).

**Tabelle A.2:** RDFS- und OWL-Properties zur Beschreibung von Klassen.

Klasse bzw. Property p	Definition
owl:Object-Property	Die Klasse aller Properties, die ein Individuum oder eine Klasse mit einem anderen Individuum oder einer anderen Klasse verbinden.
owl:Datatype-Property	Die Klasse aller Properties, die einem Individuum oder einer Klasse einen Datenwert (Literal) zuordnen.
rdfs:domain	(A p B) gibt an, dass die Subjekte eines Statements mit Property A aus der angegebenen Klasse B sein müssen.
rdfs:range	(A p B) gibt an, dass die Objekte eines Statements mit Property A aus der angegebenen Klasse B sein müssen.
rdfs:subPropertyOf	(A p B) gibt an, dass die Property A eine Unter-Property (Spezialisierung) der Property B ist.
owl:equivalentProperty	(A p B) gibt an, dass die Properties A und B äquivalent sind.
owl:inverseOf	(A p B) gibt an, dass die Property A die umgekehrte Property von Property B ist.

**Tabelle A.3:** RDFS- und OWL-Konstrukte zur Beschreibung von Properties.

Property-Klasse	Definition
owl:SymmetricProperty	(A p B) impliziert (B p A).
owl:TransitiveProperty	(A p B) und (B p C) impliziert (A p C).
owl:FunctionalProperty	(A p B) und (A p C) impliziert B = C.
owl:InverseFunctionalProperty	(A p B) und (C p B) impliziert A = C.
owl:AsymmetricProperty (OWL 2)	(A p B) impliziert, dass es kein Statement (B p A) gibt.
owl:ReflexiveProperty (OWL 2)	Impliziert (A p A) für alle A.
owl:IrreflexiveProperty (OWL 2)	Impliziert, dass es kein Statement (A p A) für alle A gibt.

**Tabelle A.4:** OWL-Property-Klassen, nach [42, S. 120]. **Achtung:** diese Tabelle ist anders zu lesen. Die Statements in der rechten Spalte (z. B. (A p B)) sind beliebige Statements, bei denen eine Property vom Typ p (linke Spalte) verwendet wird. Beispiel: ist die Property `liebt` eine `owl:SymmetricProperty` und es gilt `Francesco liebt Silvia`, dann würde auch Silvia Francesco lieben. `liebt` würde dann – im Gegensatz zur Verwendung in der Beispielontologie in Kapitel 2.3.3 auf Seite 16 – in beide „Richtungen“ gelten.

Property p	Definition
owl:onProperty	(A p B) gibt die Property B an, auf die sich die Restriction bezieht.
owl:allValuesFrom	(A p B) gibt an, dass die Individuen der Restriction A ausschließlich Properties (wie mit <code>owl:onProperty</code> angegeben) zu Individuen der Klasse B haben.
owl:someValuesFrom	(A p B) gibt an, dass die Individuen mindestens eine Property (wie mit <code>owl:onProperty</code> angegeben) zu Individuen der Klasse B haben.
owl:hasValue	(A p B) gibt an, dass alle Individuen der Restriction A den über die Property (wie mit <code>owl:onProperty</code> angegeben) verknüpften Wert B haben müssen.
owl:minCardinality	(A p N) gibt an, dass die Individuen mindestens N Properties (wie mit <code>owl:onProperty</code> angegeben) zu Individuen der mit <code>owl:allValuesFrom</code> angegebenen Klasse haben.
owl:maxCardinality	(A p N) gibt an, dass die Individuen maximal N Properties (wie mit <code>owl:onProperty</code> angegeben) zu Individuen der mit <code>owl:allValuesFrom</code> angegebenen Klasse haben.
owl:cardinality	(A p N) gibt an, dass die Individuen genau N Properties (wie mit <code>owl:onProperty</code> angegeben) zu Individuen der mit <code>owl:allValuesFrom</code> angegebenen Klasse haben.

**Tabelle A.5:** Properties, die zur Definition von Restrictions A (A = Variable in der rechten Spalte), die Mitglieder der Klasse `owl:Restriction` sind, benötigt werden. Eine Definition einer Restriction in Tripel-Form ist in den Zeilen 20-23 von Auflistung A.7 auf Seite 120 zu sehen.



# ANHANG B

---

## Klassen, Instanzen und Properties der Ontologien

---

### B.1 MDR-System.owl

Diese Ontologie enthält Klassen und Properties, die zur Beschreibung von Zieldatensatz-Ontologien verwendet werden sollen. Aus den Informationen wird beim Datenexport die i2b2-Ontologie generiert.

Namespace dieser Ontologie: `mdr: http://www.uk-erlangen.de/MDR-System#`

Klasse	Beschreibung
MDR-Context	Klasse aller Kontexte, die den Unterklassen von MDR-Dataelement über die Property <code>hasContext</code> zugewiesen werden können
MDR-Dataelement	Die Unterklassenhierarchie hiervon wird zur Konstruktion der i2b2-Taxonomie verwendet.
MDR-Unit	Klasse zur Beschreibung aller Einheiten und deren Umrechnungsbeziehungen
MDR-DataType	Klasse aller von i2b2-unterstützten Datentypen (siehe Tabelle B.2)

**Tabelle B.1:** Klassen der Ontologie MDR-System.owl.

Instanz	Beschreibung
Enum	Aufzählung
Float	Gleitkommazahl
Integer	Ganzzahl
PosFloat	Positive Gleitkommazahl
PosInteger	Positive Ganzzahl
String	Zeichenkette

**Tabelle B.2:** Instanzen der Klasse MDR-Datatype.

Property	Beschreibung / Mapping
i2b2OntoDatatype-Property	Oberproperty aller Datatype-Properties, aus denen die i2b2-Ontologie erzeugt wird.
hasDescription	Inhalt für die Spalte „c_tooltip“
hasFlagsToUse	Inhalt für das XML-Attribut <Flagstouse>
hasNiceName	Inhalt für die Spalte „c_name“
i2b2ConceptCodeProperty	Oberproperty aller Datatype-Properties, aus denen der i2b2-Concept-Code erzeugt wird.
hasConceptCodePrefix	Prefix des i2b2-Concept-Codes in Spalte „c_basecode“
hasConceptCodeSuffix	Suffix des i2b2-Concept-Codes in Spalte „c_basecode“
i2b2LabValueProperty	Oberproperty aller Datatype-Properties, die Laborwerte beschreiben.
hasLowOfLowValue	Inhalt für das XML-Attribut <LowofLowValue>
hasHighOfLowValue	Inhalt für das XML-Attribut <HighofLowValue>
hasLowOfHighValue	Inhalt für das XML-Attribut <LowofHighValue>
hasHighOfHighValue	Inhalt für das XML-Attribut <HighofHighValue>
i2b2MedicationProperty	Oberproperty aller Datatype-Properties, die Medikationen beschreiben.
hasLowOfToxicValue	Inhalt für das XML-Attribut <LowofToxicValue>
hasHighOfToxicValue	Inhalt für das XML-Attribut <HighofToxicValue>

**Tabelle B.3:** Datatype-Properties der Ontologie MDR-System.owl, aus denen der Inhalt der i2b2-Ontologie generiert wird. Die Verwendung der Einträge in i2b2 ist in den Tabellen C.8 und C.9 ab Seite 140 ersichtlich.

Property	Beschreibung
hasContext	Weist einem Element der Klasse MDR-Dataelement eine Instanz aus MDR-Context zu.
hasDataType	Weist einem Element der Klasse MDR-Dataelement eine Instanz aus MDR-DataType zu und dient als Inhalt für das XML-Attribut <DataType>.
hasUnits	Weist einem Element der Klasse MDR-Dataelement eine Instanz aus MDR-Unit zu und dient als Inhalt für das XML-Attribut <NomalUnits>.

**Tabelle B.4:** Object-Properties der Ontologie MDR-System.owl, aus denen der Inhalt der i2b2-Ontologie generiert wird. Die Verwendung der Einträge in i2b2 ist in den Tabellen C.8 und C.9 ab Seite 140 ersichtlich.

## B.2 OntoMappingSystem.owl

Diese Ontologie enthält Klassen und Properties, die zur Beschreibung der Datenzugriffsinformationen und zur Konstruktion der Zwischenknoten der Mapping-Ontologie dienen. Aus den Informationen werden beim Datenexport die SQL-Statements generiert.

Namespace dieser Ontologie: omsys: <http://www.uk-erlangen.de/OntoMappingSystem#>

Klasse	Beschreibung
DatabaseConnection	Klasse aller Datenbankverbindungen
SourceTable	Klasse aller Quelltabellen
OperationCommand	Klasse aller Knotentypen
ArithmeticOperation	Klasse aller Knotentypen zur Beschreibung arithmetischer Operationen
RelationalOperator	Klasse aller Knotentypen zur Beschreibung von Vergleichsoperatoren
StringOperation	Klasse aller Knotentypen zur Beschreibung von String-Operationen
StatusTypedItem	Klasse alle Dokumentationselemente und Zwischenknoten
UnprocessedItem	Klasse der unbearbeiteten Dokumentationselemente
ProcessedItem	Klasse aller Dokumentationselemente und alle bearbeiteten Zwischenknoten
StringItem	Klasse der Zwischenknoten für String-Datenwerte (siehe Abschnitt 3.4.3 auf Seite 47)

**Tabelle B.5:** Klassenhierarchie der Ontologie OntoMappingSystem.owl

Die folgenden Tabellen sind eine Übersicht über die in der Arbeit umgesetzten Knotentypen, die in Form von Instanzen der Klasse OperationCommand angelegt wurden. Die Kreuzchen in den Spalten „T“ bzw. „VT“ geben an, ob eine tolerante (Operand 2 kann

NULL sein) bzw. sehr tolerante Variante (Operand 1 oder Operand 2 kann NULL sein) des Befehls zur Verfügung steht. Soll eine solche verwendet werden, ist „T“ bzw. „VT“ an den Befehl anzuhängen. Die einzelnen Knoten reichen die Entity-Attribute von Operand 1 weiter, wenn nicht anders angegeben. Operator 1 und Operator 2 sind mit OP1 bzw. OP2 abgekürzt.

Operator	T	VT	Beschreibung
ADD	X	X	Addiert OP1 und OP2.
SUBTR	X	X	Subtrahiert OP2 von OP1.
MULT	X	X	Multipliziert OP1 mit OP2.
DIV	X	X	Dividiert Operand 1 durch OP2.
DIFF			Berechnet die Differenz zwischen OP1 und OP2.

**Tabelle B.6:** Instanzen der Klasse ArithmeticOperation.

Operator	T	VT	Beschreibung
EQUALS	X	X	Liefert 'TRUE' zurück, wenn OP1 = OP2.
GREATER	X	X	Liefert 'TRUE' zurück, wenn OP1 > OP2.
GREATEREQUAL	X	X	Liefert 'TRUE' zurück, wenn OP1 >= OP2.
LESSER	X	X	Liefert 'TRUE' zurück, wenn OP1 < OP2.
LESSEREQUAL	X	X	Liefert 'TRUE' zurück, wenn OP1 <= OP2.
NOTEXISTS			Liefert 'TRUE' zurück, wenn OP1 existiert und OP2 nicht existiert.
IF			Wenn OP1 den Wert 'True' hat, wird OP2 einschließlich der Entity-Attribute von OP2 weitergereicht.

**Tabelle B.7:** Instanzen der Klasse RelationalOperator.

Operator	Beschreibung
INSTR	Liefert 'TRUE' zurück, wenn OP1 eine Teilzeichenkette von OP2 ist.
STRPOS	Liefert die Position von OP2 innerhalb von OP1 zurück. Wenn OP2 keine Teilzeichenkette von OP1 wird, wird nichts zurückgeliefert.
STRPOS2	Liefert die Position von OP2 innerhalb von OP1 zurück. Wenn OP2 keine Teilzeichenkette von OP1 wird, wird „0“ zurückgeliefert.
LCASE	Liefert OP1 in Kleinbuchstaben zurück. Für OP2 muss ein Dummy-Wert übergeben werden.
UCASE	Liefert OP1 in Großbuchstaben zurück. Für OP2 muss ein Dummy-Wert übergeben werden.
TAIL	Liefert einen Teilstring von OP1 zurück, der an Position OP2 beginnt und am Ende des Strings in OP1 endet.
HEAD	Liefert einen Teilstring von OP1 zurück, der an Position 1 beginnt und an Position OP1 endet.

**Tabelle B.8:** Instanzen der Klasse StringOperation.

Property	Beschreibung
hasDatabaseConnection	Weist einer Nutzdatentabelle der Klasse SourceTable eine Datenbankverbindung der Klasse DatabaseConnection zu.
hasSourceTable	Weist einem Dokumentation-Element aus der Quellsystem-Ontologie oder einem Zwischenknoten aus der Mapping-Ontologie eine Nutzdatentabelle der Klasse SourceTable zu.
hasCommandType	Weist einem Zwischenknoten aus der Mapping-Ontologie den Befehlstyp durch eine Instanz der Klasse OperationCommand zu.
hasImport	Weist einem Element der Zieldatensatz-Ontologie ein Dokumentationselement aus der Quellsystem-Ontologie oder einen Zwischenknoten aus der Mapping-Ontologie zu.
hasOperand	Property, die ausdrückt, dass ein Zwischenknoten der Mapping-Ontologie einen anderen Knoten als Operand hat.
hasOperand1	Weiβt einem Zwischenknoten der Mapping-Ontologie den ersten Operand zu.
hasOperand2	Weiβt einem Zwischenknoten der Mapping-Ontologie den zweiten Operand zu.

**Tabelle B.9:** Object-Properties der Ontologie OntoMappingSystem.owl.

Property	Beschreibung
CommandProperty	Properties, die zur Konstruktion des neuen Tabelleneintrags verwendet werden, der bei der Knotenbearbeitung in die temporäre Tabelle geschrieben wird
hasDateEndValue	Gibt an, von welchem Operand der Endzeitpunkt der Diagnose übernommen werden soll.
hasDateEndValue	Gibt an, von welchem Operand der Startzeitpunkt der Diagnose übernommen werden soll.
hasOutputTransformation	Gibt den Datenbank-Befehl an, mit dem Operand 1 und 2 verrechnet werden sollen.
hasSelectFilter	Gibt den Filter an, nachdem die Ergebnisdatensätze gefiltert werden sollen.
DatabaseConnectionProperty	Definiert eine Datenbankverbindung.
hasHostName	Definiert den Hostnamen des Datenbankservers.
hasUserName	Gibt den Benutzernamen an.
hasPassword	Gibt das Benutzerpasswort an.
hasPort	Gibt den Port an.
hasSID	Gibt die SID an.
TableDescriptionProperty	Beschreibt Informationen zum Zugriff auf die Nutzdatentabelle.
hasSourceTableName	Gibt den Tabellennamen an.
hasAccessSQL	Enthält einen generischen SQL-Block, mit dem zwei Knoten verrechnet werden können. In dem SQL-Block sind Variablen enthalten, die durch OntoExport ersetzt werden.
hasOperandFetchSQL	Enthält einen generischen SQL-Block, mit dem ein SELECT-Statement zum Laden eines der beiden Operanden erfolgen kann.
hasDocumentIDColumn	Gibt die Spalte an, in der der Datensatz eindeutig identifiziert werden kann.
hasPatientIDColumn	Gibt die Spalte an, in der die Patientennummer steht.
hasCaseIDColumn	Gibt die Spalte an, in der die Fallnummer steht.
hasDateStartValueColumn	Gibt die Spalte an, in der der Startzeitpunkt der Diagnose steht.
hasDateEndValueColumn	Gibt die Spalte an, in der der Endzeitpunkt der Diagnose steht.
hasDateTransformation	Gibt eine Datenbankoperation an, mit der das Datumsformat in der Quelltabelle in das der temporären Tabelle umgerechnet werden kann.
hasValueColumn	Gibt die Spalte an, in der der eigentliche Wert steht.
hasStringValue	Weist einem String-Zwischenknoten einen Datenwert zu (siehe Abschnitt 3.4.3 auf Seite 47).
isResultOfExpression	Weist einem Zwischenknoten für Debugging-Zwecke den bearbeiten Teilausdruck in QuickMapp-Syntax zu.

**Tabelle B.10:** Datatype-Properties der Ontologie OntoMappingSystem.owl.

### B.3 Testdatensatz

Sheet ID	MySheet Date	PatID	Sex	Age	Test Score 1	Test Score 2	Lab Value	Date of Lab Value	Patient Notes
DocumentID	GlobalDate	PatientID	Integer	Integer	PosFloat	Date	String		
1	01.06.10	100001	Male	10	4	0,2	01.01.10	Lorem ipsum dolor	
2	01.06.10	100002	Female	11	5	0,5	02.01.10	sit amet, consete	
3	01.06.10	100003	M	12	1	0,8	03.01.10	tur <b>Morleys</b> sadipsc	
4	01.06.10	100004	F	13	2	1,1	04.01.10	e et <b>lung cancer</b> d	
5	01.06.10	100005	M	14	3	1,4	05.01.10	umy <b>smoking</b> elim	
6	01.06.10	100006	F	3	4	1,7	06.01.10	invidunt ut labor	
7	01.06.10	100007	M	16	2	0,2	07.01.10	itr, sed diam non	
8	01.06.10	100008	F	17	3	1	0,5	08.01.10 aliquyam erat, se	
9	01.06.10	100009	M	18	4	0,8	09.01.10	d diam voluptua.	
10	01.06.10	100010	F	19	5	1,1	10.01.10	At <b>Hello World</b> eos	
11	01.06.10	100011		20	1	4	11.01.10	cusam <b>Hallo DPKK</b>	
12	01.06.10	100012	F	21	2	1,7	12.01.10	o <b>[Huhu IMI]</b> et ea	
13	01.06.10	100013	M	22	3	0,2	13.01.10	ebum. Stet clita	
14	01.06.10	100014	F		4	0,5	14.01.10	kasd gubergren,	
15	01.06.10	100015	M	24	5	0,8	15.01.10	no sea takimata s	
16	01.06.10	100016	F	25	1	1,1	16.01.10	anctus est Lorem	
17	01.06.10	100017	M	26		1,4	17.01.10	Lorem ipsum dolor	
18	01.06.10	100018	F	27		1,7	18.01.10	sit amet, consete	

Abbildung B.1: Inhalt der Datei Input.xlsx im Projekt „Nodes-Tests“ zum Testen der einzelnen Knotentypen.

## **ANHANG C**

---

Tabellen und Sonstiges

---

## C.1 Soarian-Export: Tabellenbeschreibungen

Spalte	Datentyp	Beschreibung
FormID	NUMBER	Eine ID, die der Form zugeordnet ist
Form_AdminName	VARCHAR	Administrativer Form-Name
Form_DisplayName	VARCHAR	Form-Name, wie er angezeigt wird
FormVersion	NUMBER	Form-Version
Staged	NUMBER	Gibt an, ob die Form vom Form-Bearbeiter (nicht Endanwender!) freigegeben ist und damit in der Soarian-Benutzeroberfläche zur Verfügung steht
ComponentID	NUMBER	Eine ID, die der Component zugeordnet ist
Comp_AdminName	VARCHAR	Administrativer Component-Name
Comp_DisplayName	VARCHAR	Component-Name, wie er angezeigt wird
ComponentVersion	NUMBER	Version des Component
Editor	VARCHAR	Widget-Typ, mit dem das Component-Name in der Benutzeroberfläche dargestellt wird (siehe Tabelle C.3 auf Seite 135)
CarryForward	VARCHAR	Gibt an, ob die Component mit Werten ausgefüllt werden soll, falls die Component in einer anderen Form bereits verwendet und ausgefüllt wurde. Mögliche Werte sind: None (nicht ausfüllen), Any (immer ausfüllen), Same (nur ausfüllen, wenn der Wert aus dem gleichen Form-Typ stammt).
Options_Int	VARCHAR	Laufende Nummer zur Durchzählung der Ausprägungen
Options_Exp	VARCHAR	Ausprägung in textueller Form, wie sie in der Soarian-Benutzeroberfläche angezeigt wird

**Tabelle C.1:** Spalten der Tabelle DWH\_METADATEN\_ONTOLOGY.

Spalte	Datentyp	Beschreibung
PatID	NUMBER	Patienten-Nummer
FallNr	VARCHAR	Fall-Nummer
FormID	NUMBER	ID der Form, in die Component verwendet wurde
AssessmentID	NUMBER	Eine ID, die das ausgefüllte Formular eindeutig kennzeichnet
AssessmentStatus	NUMBER	Bearbeitungs-Status des Formulars
ComponentID	NUMBER	ID der Component, von der der Wert stammt
AdminName	VARCHAR	Administrativer Component-Name (entspricht Spalte Comp_AdminName in Tabelle C.1)
Value	VARCHAR	Wert
FormCreatedDt	TIMESTAMP	Zeitstempel, an dem die Form angelegt wurde
FormSavedDt	TIMESTAMP	Zeitstempel, an dem die Form gespeichert wurde
FormReferredDt	TIMESTAMP	
CompSavedDt	TIMESTAMP	Zeitstempel, an dem die Component gespeichert wurde

**Tabelle C.2:** Spalten der Tabelle DWH\_NUTZDATEN.

Component-Typ	Beschreibung
_DCSTime	Uhrzeit
_DCSNSLE	Zahlenwert
_DCSMSCheckboxList	Mehrfachauswahl
_DCSLargeTextEditor	Texteingabefeld
_DCSDateTime	Datum mit Uhrzeit
_DCSComboEdit	Auswahl-Liste (Benutzereingabe möglich)
_DCSCombo	Auswahl-Liste (Benutzereingabe nicht möglich)
_DCSCalculated	Aus anderen Feldern berechneter Wert
_DCSButton	Radio Button
_CCTTickBox	Check-Box
_CCTSLE	Einzeilen-Texteingabefeld
_CCTMLE	Mehrzeilen-Texteingabefeld
_CCTDateTime	Datum mit Uhrzeit

**Tabelle C.3:** Component-Typen in Soarian, wie sie in der Spalte „EDITOR“ in Tabelle C.1 zu finden sind.

FormID	Component-ID	Comp_DisplayName	Component-Version	Options_Int	Options_Exp	Editor
548	501	Fieber	1	1	nein	_DCSComboEdit
548	501	Fieber	1	2	37,5 - 38,0	_DCSComboEdit
548	501	Fieber	1	3	38,1 - 38,5	_DCSComboEdit
548	501	Fieber	1	4	38,6 - 39,0	_DCSComboEdit
548	501	Fieber	1	5	39,1 - 39,5	_DCSComboEdit
548	501	Fieber	1	6	39,6 - 40,0	_DCSComboEdit
548	501	Fieber	1	7	> 40,1	_DCSComboEdit
548	501	Fieber	1	8	konstant	_DCSComboEdit
548	501	Fieber	1	9	intermittierend	_DCSComboEdit
548	501	Fieber	1	10	...	_DCSComboEdit
548	501	Fieber	2	1	nein	_DCSComboEdit
548	501	Fieber	2	2	37,5 - 38,0	_DCSComboEdit
548	501	Fieber	2	3	38,1 - 38,5	_DCSComboEdit
548	501	Fieber	2	4	38,6 - 39,0	_DCSComboEdit
548	501	Fieber	2	5	39,1 - 39,5	_DCSComboEdit
548	501	Fieber	2	6	39,6 - 40,0	_DCSComboEdit
548	501	Fieber	2	7	> 40,1	_DCSComboEdit
548	501	Fieber	2	8	konstant	_DCSComboEdit
548	501	Fieber	2	9	intermittierend	_DCSComboEdit
548	501	Fieber	2	10	...	_DCSComboEdit
548	501	Fieber	4	1	Ja	_DCSButton
548	501	Fieber	4	2	Nein	_DCSButton

**Tabelle C.4:** Beispielhafter Auszug aus DWH\_METADATEN\_ONTOLOGY. Dargestellt ist die Component „Fieber“, wie sie in einem Formular mit FormID = 548 benutzt wird. Man beachte die farblich hervorgehobenen Versionssprünge und die Änderung des Widget-Typs (Spalte „Editor“, siehe Tabelle C.3). Die restlichen Spalten (aus Platzgründen hier nicht gezeigt) sind in Tabelle C.1 auf Seite 134 aufgelistet.

<b>PatID</b>	<b>FallNr</b>	<b>AssessmentID</b>	<b>FormID</b>	<b>ComponentID</b>	<b>AdminName</b>	<b>Value</b>
23123123	454356782	993423423	548	501	M1_AN_0062	37,5 - 38,0
23123123	454356782	123124153	548	501	M1_AN_0062	38,1 - 38,5
23123123	454356782	345616343	548	501	M1_AN_0062	38,6 - 39,0
56456456	146497945	343234511	548	501	M1_AN_0062	nein
67567567	179852034	345213453	548	501	M1_AN_0062	Nein
78678678	213206123	123784916	548	501	M1_AN_0062	Ja
89789789	246560212	234526764	548	501	M1_AN_0062	Ja
00900900	279914301	345342612	548	501	M1_AN_0062	Ja

**Tabelle C.5:** Beispielhafter Auszug aus DWH\_NUTZDATEN mit Werten zur Component „Fieber“. Die ersten drei Zeilen zeigen einen Patienten, für den drei Formulare gleichen Typs ausgefüllt wurden (gleiche Patienten- und Fall-Nummer, aber unterschiedliche Assessments). Hervorgehoben: „Nein“-Attribut aus unterschiedlichen Component-Versionen.

## C.2 i2b2-Tabellenbeschreibungen

Spaltenname	Datentyp (Oracle)	Beschreibung
encounter_num (PK)	NUMBER(38,0)	Fallnummer
concept_cd (PK)	VARCHAR2(50)	Concept Code
provider_id (PK)	VARCHAR2(50)	Bereitsteller der Daten, z. B. Arzt oder medizinisches Gerät
start_date (PK)	DATE	Startzeitpunkt der Untersuchung
modifier_cd (PK)	VARCHAR2(50)	Versionsnummer, vom Quellsystem vergeben
patient_num	NUMBER(38,0)	Patientennummer
valtype_cd	VARCHAR2(50)	Datentyp, siehe Tabelle C.7 auf Seite 139
tval_char	VARCHAR2(255)	Siehe Tabelle C.7 auf Seite 139
nval_num	NUMBER(18,5)	Numerischer Datenwert
valueflag_cd	VARCHAR2(50)	Siehe Tabelle C.7 auf Seite 139
units_cd	VARCHAR2(50)	Siehe Tabelle C.7 auf Seite 139
observation_blob	CLOB	Siehe Tabelle C.7 auf Seite 139
quantity_num	NUMBER(18,5)	Zähler für nval_num
end_date	DATE	Endzeitpunkt der Untersuchung
location_cd	VARCHAR2(50)	Code, der den Ort repräsentiert, an dem die Untersuchung stattgefunden hat
confidence_num	NUMBER(18,5)	Code, der die Korrektheit der Daten bewertet
update_date	DATE	Datum, an dem der Datensatz aktualisiert wurde
download_date	DATE	Datum, an dem der Datensatz aus dem Quellsystem geladen wurde
import_date	DATE	Datum, an dem der Datensatz nach i2b2 importiert wurde
sourcesystem_cd	VARCHAR2(50)	Code für das Quellsystem
upload_id	NUMBER(38,0)	ID, die beim Upload vergeben wird

**Tabelle C.6:** OBSERVATION\_FACT. Speichert sämtliche Untersuchungsergebnisse und andere Fakten, die von Patienten dokumentiert wurden.

<b>valtype_cd</b>	<b>tval_char</b>	<b>nval_num</b>	<b>valueflag_cd</b>	<b>units_cd</b>	<b>observation_blob</b>
N	E = größer, NE = ungleich, L = kleiner, LE = kleiner oder gleich, G = größer, GE = größer oder gleich	Der Wert (Zahl)	H = hoch, L = tief, N = normal	Einheit	Besondere verschlüsselte Information
T	Der Wert (Text)		A = nicht normal, N = normal	Einheit	Besondere verschlüsselte Information
B			X		Text
NLP			X		NLP-Ergebnis

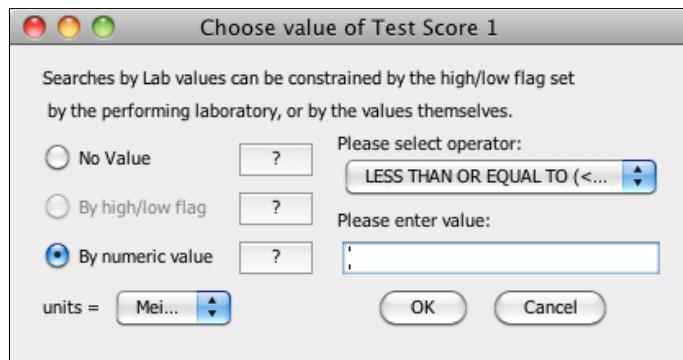
**Tabelle C.7:** Spalten in Tabelle OBSERVATION\_FACT, die in Abhängigkeit des Datenwertes (Spalte „valtype\_cd“) besondere Werte annehmen. Aus der i2b2-Dokumentation zu Version 1.5, CRC\_Design\_1-5.pdf, S. 9.

Spaltenname	Datentyp (Oracle)	Beschreibung
c_hlevel	INT	Hierarchielevel eines Eintrags
c_fullname	VARCHAR2(900)	Vollständiger Pfad im Baum, getrennt und eingefasst durch Backslashes
c_name	VARCHAR2(2000)	Bezeichnung, unter der das Element in der Workbench angezeigt wird
c_synonym_cd	CHAR(1)	Gibt an, ob es sich bei dem Eintrag um ein Synonym handelt ('Y') oder nicht ('N'). Synonyme besitzen den gleichen Concept Code (c_basecode) wie die ursprünglichen Terme.
c_visualattributes	CHAR(3)	Kombination aus drei Zeichen, die das Aussehen in der Workbench bestimmt. Mögliche Darstellungsformen sind F = Folder, C = Container, M = Multiple, L = Leaf (1. Zeichen) in Kombination mit den Attributen A = Active, I = Inactive, H = Hidden (2. Zeichen) und E = Editable (3. Zeichen).
c_totalnum	INT	Der Wert ermöglicht Optimierungen, um effizienteres SQL in der CRC-Zelle zu erzeugen.
c_basecode	VARCHAR2(450)	Concept Code
c_metadataxml	CLOB	Optionale Zusatzbeschreibung im XML-Format, die zur Wertebeschreibung für Laborwerte gebraucht wird
c_facttablecolumn	VARCHAR2(50)	Name der Spalte in der Fakten-Tabelle (OBSERVATION_FACT), die den Concept Code beinhaltet
c_tablename	VARCHAR2(50)	Name der Tabelle, die den Concept Path und Concept Code enthält (Normalerweise CONCEPT_DIMENSION)
c_columnname	VARCHAR2(50)	Spaltenname, der den Concept Path beinhaltet
c_columndatatype	VARCHAR2(50)	Datentyp: Text ('T') oder numerischer Wert ('N')
c_operator	VARCHAR2(10)	Operator für die SQL-Anfrage, typischerweise 'LIKE' oder '='
c_dimcode	VARCHAR2(900)	Pfad aus der Dimensionstabelle, der ein Mapping auf den Concept-Pfad ermöglicht
c_comment	CLOB	Optionales Feld für Kommentare
c_tooltip	VARCHAR2(900)	Tooltip, das in der Workbench angezeigt wird sobald die Maus über einem Element gehalten wird
update_date	DATE	Beschreibt, wann die Daten aktualisiert wurden.
download_date	DATE	Beschreibt, wann die Daten heruntergeladen wurden.
import_date	DATE	Beschreibt, wann die Daten importiert wurden.
sourcesystem_cd	VARCHAR2(50)	Quellsystem, von dem die Daten stammen.
valuetype_cd	VARCHAR2(50)	Gibt den Element-Typ an: Dok = Dokument oder Knoten, Lab = Labortwert oder Key = Schlüsselwort zur Verwendung mit den NLP-Funktionen.

**Tabelle C.8:** i2b2-Ontologie-Tabelle (üblicherweise Tabelle I2B2).

Spaltenname	Beschreibung
<Version>	Version des XMLs
<CreationDateTime>	Zeitstempel, an dem das XML angelegt wurde
<DataType>	Gibt den Datentyp des Datenelements an: PosFloat, Float, PosInteger, Integer, Enum, String
<Flagstouse>	Flags, die benutzt werden sollen. HL = High/Low: Schaltet den Radio-button „By high/low flag“ frei (siehe Abbildung C.1 auf Seite 142), in die Spalte „Valueflag_cd“ in OBSERVATION_FACT muss dann ein „H“ oder „L“ eingetragen werden. A=Abnormal: Markiert Ausprägungen von Enum-Datentypen als abnorm.
<Oktousevalues>	Ein „Y“ schaltet den Radio-Button „By numeric value“ frei (siehe Abbildung C.1 auf Seite 142).
<MaxStringLength>	Maximale Länge einer Zeichenkette
<LowofLowValue>	Werte, die kleiner als dieser Wert sind, werden als „sehr niedrig“ eingestuft.
<HighofLowValue>	Werte zwischen diesem Wert und LowOfLow werden als „niedrig“ eingestuft.
<LowofHighValue>	Werte zwischen diesem Wert und HighOfLow werden als „mittel“ eingestuft.
<HighofHighValue>	Werte zwischen diesem Wert und LowOfHigh werden als „hoch“ eingestuft. Werte, die größer sind als dieser Wert werden als „sehr hoch“ eingestuft.
<LowofToxicValue>	Der untere Grenzwert des toxischen Bereichs eines Medikaments
<HighofToxicValue>	Der obere Grenzwert des toxischen Bereichs eines Medikaments
<EnumValues>	Container für die einzelnen Aufzählungen, falls DataType „Enum“ ist (siehe Beispiel in Auflistung C.1)
<Val>	Wert einer Aufzählung
<UnitValues>	Container zur Angabe der Laborwert-Einheit (siehe Beispiel in Auflistung C.2)
<NormalUnits>	Die Bezeichnung der Einheit, mit der der Laborwert gemessen wurde
<EqualUnits>	Bezeichnung einer Einheit, die der Einheit in NormalUnits entspricht
<ExcludingUnits>	Bezeichnung einer Einheit, deren Datensätze bei einem Datenimport verworfen werden sollen
<ConvertingUnits>	Container mit Angaben zur Umrechnung in andere Einheiten
<Units>	Bezeichnung der Einheit
<MultiplyingFactor>	Multiplikationsfaktor

**Tabelle C.9:** Informationen für Laborwerte in der Spalte „c\_metadataxml“ der i2b2-Ontologietabelle.



**Abbildung C.1:** Fenster in der i2b2-Workbench, über das weitere Suchkriterien für numerische Datenwerte festgelegt werden können.

```

1  <EnumValues>
2    <Val>Ausprägung1</Val>
3    <Val>Ausprägung2</Val>
4  </EnumValues>
```

**Auflistung C.1:** Beispiel für den XML-Container <EnumValues>.

```

1  <UnitValues>
2    <NormalUnits>Einheit1</NormalUnits>
3    <EqualUnits>Einheit2</EqualUnits>
4    <EqualUnits>Einheit3</EqualUnits>
5    <ExcludingUnits>Einheit6</ExcludingUnits>
6    <ExcludingUnits>Einheit7</ExcludingUnits>
7    <ConvertingUnits>
8      <Units>Einheit4</Units>
9      <MultiplyingFactor>100.0</MultiplyingFactor>
10     </ConvertingUnits>
11     <ConvertingUnits>
12       <Units>Einheit5</Units>
13       <MultiplyingFactor>200.0</MultiplyingFactor>
14     </ConvertingUnits>
15   </UnitValues>
```

**Auflistung C.2:** Beispiel für den XML-Container <UnitValues>.

Spaltenname	Typ	Beschreibung
c_hlevel	A	
c_fullname	A	Ergibt sich aus der Klassenhierarchie
c_name	P	<b>mdr:hasNiceName</b>
c_synonym_cd	O	Die Synonym-Einträge müssen aus der Ontologie selbst abgeleitet werden können.
c_visualattributes	A	
c_totalnum	N/A	Nicht relevant für KAS-Datenelemente
c_basecode	P	<b>mdr:hasConceptCodePrefix</b> und <b>mdr:hasConceptCodeSuffix</b>
c_metadataxml		Siehe Tabelle C.11!
c_facttablecolumn	N/A	Es wird nur die Standard-OBSERVATION_FACT-Tabelle verwendet.
c_tablename	N/A	Es wird nur die Standard-OBSERVATION_FACT-Tabelle verwendet.
c_columnname	N/A	Es wird nur die Standard-OBSERVATION_FACT-Tabelle verwendet.
c_columndatatype	A	Ergibt sich aus den XML-Attributen.
c_operator	A	Ergibt sich aus den XML-Attributen.
c_dimcode	A	
c_comment	N/A	
c_tooltip	P	<b>mdr:hasDescription</b>
update_date	A	
download_date	A	
import_date	A	
sourcesystem_cd	A	
valuetype_cd	A	

**Tabelle C.10:** Ergebnis der Analyse in der i2b2-Ontologietabelle enthaltenen Informationen:  
A = kann durch die Export-Software (OntoExport) aus den anderen Informationen automatisch erzeugt werden, P = muss über eine Property abgebildet werden, N/A = nicht relevant, O = muss durch eine andere Ontologie abgedeckt werden.

<b>XML-Attribut</b>	<b>Typ</b>	<b>Beschreibung</b>
<Version>	A	
<CreationDateTime>	A	
<DataType>	P	<b>mdr:hasDataType</b>
<Flagstouse>	P	<b>mdr:hasFlagsToUse</b>
<Oktousevalues>	A	
<MaxStringLength>	?	
<LowofLowValue>	P	<b>mdr:hasLowOfLowValue</b>
<HighofLowValue>	P	<b>mdr:hasHighOfLowValue</b>
<LowofHighValue>	P	<b>mdr:hasLowOfHighValue</b>
<HighofHighValue>	P	<b>mdr:hasHighOfHighValue</b>
<LowofToxicValue>	P	<b>mdr:hasLowOfToxicValue</b>
<HighofToxicValue>	P	<b>mdr:hasHighOfToxicValue</b>
<Valdescription>	O	Die aufgezählten Ausprägungen müssen aus der Ontologie selbst abgeleitet werden können.
<NormalUnits>	P	<b>mdr:hasUnits</b>
<EqualUnits>	O	Beziehungen zwischen verschiedenen Einheiten müssen aus der Ontologie selbst abgeleitet werden können.
<ExcludingUnits>	O	Beziehungen zwischen verschiedenen Einheiten müssen aus der Ontologie selbst abgeleitet werden können.
<Units>	O	Beziehungen zwischen verschiedenen Einheiten müssen aus der Ontologie selbst abgeleitet werden können.
<MultiplyingFactor>	O	Beziehungen zwischen verschiedenen Einheiten müssen aus der Ontologie selbst abgeleitet werden können.

**Tabelle C.11:** Ergebnis der Analyse der Informationen des XMLs in der i2b2-Ontologietabelle:  
A = kann durch die Export-Software (OntoExport) aus den anderen Informationen automatisch erzeugt werden, P = muss über eine Property abgebildet werden, N/A = nicht relevant,  
O = muss durch eine andere Ontologie abgedeckt werden.

### C.3 DPKK-Datensatz

Die folgenden Tabellen geben einen Überblick über den überarbeiteten DPKK-Datensatz. Die Änderungen entstanden in einer engen Zusammenarbeit zwischen der Urologie des Universitätsklinikums Erlangen, dem Lehrstuhl für Medizinische Informatik der Universität Erlangen-Nürnberg und dem Institut für Medizinische Informatik an der Universität Münster. Die Datenelemente in Tabelle C.14 wurden Mitte Dezember 2010 in den DPKK-Datensatz aufgenommen und fanden keine nähere Betrachtung im Rahmen dieser Arbeit. Die Erklärungen zu den Abkürzungen und den Kommentaren (1) bis (6) befinden sich unter den Tabellen.

<b>Nummer</b>	<b>Datenelement</b>	<b>Erlangen</b>	<b>Münster</b>
A1	Patientenpseudonym		
A2	Fallpseudonym		
A3	Geschlecht	entfällt (1)	entfällt
A4	Einschlussstudie(n)	OK	OK
A5	cT	OK	OK (2)
A6	cN	OK	OK (2)
A7	cM	OK	OK (2)
A8	Topographie	OK	OK
A9	Digital rektale Untersuchung	OK	OK
A10	Gleason-Score 1	OK	OK
A11	Gleason-Score 2	OK	OK
A12	Gleason-Score Summe	OK	OK
A13	Präoperativer PSA	OK	OK
A14	Postoperativer PSA	OK	Vermutlich
A15	Medikamentöse Tumortherapie vor OP?	OK	N/A
A16	Medikamentöse Tumortherapie nach OP?	(4)	OK
A17	Strahlentherapie vor OP?	OK	N/A
A18	Strahlentherapie nach OP?	(3, 6)	OK
A19	Zweitmalignom?	OK (4)	N/A
A20	Ganzkörperszintigramm vor OP?	OK (4)	N/A
A21	Ganzkörperszintigramm suspekt?	OK	N/A
A22	Datum der Biopsieentnahme	OK	OK
A23	Patientenalter bei Biopsieentnahme	(1)	OK
A24	Durchgeführte Operation	OK	OK
A25	Nerverhalt	OK	Unbekannt
A26	Falls Nerverhalt, wo?	OK	N/A
A27	Prostatagewicht	OK	N/A (nur Volumen)
A28	Anzahl untersuchter Lymphknoten links (5)	OK	nur rechts+links
A29	Anzahl befallener Lymphknoten links (5)	OK	nur rechts+links
A30	Anzahl untersuchter Lymphknoten rechts (5)	OK	nur rechts+links
A31	Anzahl befallener Lymphknoten rechts (5)	OK	nur rechts+links
A32	Histomorphologie des Prostatektomiepräparates	OK	OK
A33	Histologie-Grading (G1, G2, G3, X, O)	OK	Umrechnung von Helpap
A34	pT	OK	OK (2)
A35	pN	OK	OK (2)
A36	pM	OK	OK (2)

**Tabelle C.12:** DPKK-Datensatz und -Mapping Teil 1.

Nummer	Datenelement	Erlangen	Münster
A37	Gesamtbeurteilung Residualklassifikation (R0, R1, RX)	OK	OK
A38	Art der Probe	nur FFPE und Cryo	aus Excel-Datenquelle
A39	Lokalrezidiv	OK	OK
A40	Art des Lokalrezidivs	OK	N/A
A41	Ist Tumor in Negativgewebeprobe(n) enthalten?	N/A	OK

Tabelle C.13: DPKK-Datensatz und -Mapping Teil 2.

Datenelement	Erlangen	Münster
cTNM Klassifikationsart	OK	OK
cM Orstangabe der Metastase(n)	Vermutlich	Unbekannt
cM Zeitabstand der Entdeckung zur OP	Vermutlich	Unbekannt
pTNM Klassifikationsart	OK	OK
pT (a, b, c)	OK	OK
pM Orstangabe der Metastase(n)	Vermutlich	Unbekannt
pM Zeitabstand der Entdeckung zur OP	Vermutlich	Unbekannt
PSA Rezidiv	N/A	OK
Tumor in Nativgewebeprobe(n) enthalten?	OK	N/A

Tabelle C.14: DPKK-Datensatz, Mitte Dezember ergänzte Datenelemente.

Ergänzungen zu den Einträgen:

**Leeres Feld:** Nicht relevant für den Mapping-Prozess dieser Arbeit.

**OK:** Das Mapping kann mit den Methoden dieser Arbeit hergestellt werden.

**N/A:** Datenelement wird im Quellsystem nicht dokumentiert.

- (1) Erfordert den Zugriff auf SAP/IS-H. Noch nicht umgesetzt.
- (2) In Münster wird entweder nur der klinische oder nur der pathologische TMN erfasst.
- (3) Hier muss geprüft werden, ob ein Bogen ausgefüllt wurde. Noch nicht umgesetzt.
- (4) Eine Auswahloption, wie z. B. „Unbekannt“, fehlt.
- (5) „Links“ und „rechts“ soll zusammengefasst werden.
- (6) Stark verzögerte Erfassung (ca. 12 Monate) in Erlangen.



## C.4 i2b2 QDC-XML-Nachricht

```

1  <ns4:request xmlns:ns2="http://www.i2b2.org/xsd/hive/pdo/1.1/" xmlns:ns4="http://
2    www.i2b2.org/xsd/hive/msg/1.1/" xmlns:ns3="http://www.i2b2.org/xsd/hive/plugin
3    /" xmlns:ns5="http://www.i2b2.org/xsd/cell/ont/1.1/">
4      <message_header>
5        <i2b2_version_compatible>1.1</i2b2_version_compatible>
6        <hl7_version_compatible>2.4</hl7_version_compatible>
7        <sending_application>
8          <application_name>i2b2 Workbench</application_name>
9          <application_version>1.2</application_version>
10         </sending_application>
11         <sending_facility>
12           <facility_name>i2b2 Hive</facility_name>
13         </sending_facility>
14         <receiving_application>
15           <application_name>QDC Cell</application_name>
16           <application_version>1.0</application_version>
17         </receiving_application>
18         <receiving_facility>
19           <facility_name>i2b2 Hive</facility_name>
20         </receiving_facility>
21         <datetime_of_message>2010-11-27T21:46:23.534+01:00</datetime_of_message>
22         <message_control_id>
23           <message_num>GThcYiuWbhgCrSoN8PhB</message_num>
24           <instance_num>0</instance_num>
25         </message_control_id>
26         <processing_id>
27           <processing_id>P</processing_id>
28           <processing_mode>I</processing_mode>
29         </processing_id>
30         <accept_acknowledgement_type>AL</accept_acknowledgement_type>
31         <application_acknowledgement_type>AL</application_acknowledgement_type>
32         <country_code>US</country_code>
33         <project_id>DPKKMuenster</project_id>
34       </message_header>
35       <request_header>
36         <result_waittime_ms>120000</result_waittime_ms>
37       </request_header>
38       <message_body>
39         <Recipients>
40           <Recipient>Muenster</Recipient>
41           <Recipient>Erlangen</Recipient>
42         </Recipients>
43         <PortalMessage>Your name:
44           Institution:
45           Phone:
46           e-Mail:
47           Research interest:
48           Notes:
49         </PortalMessage>
50         <QueryDefinition> ... </ns2:query_definition>
51       </message_body>
52     </ns4:request>

```

**Auflistung C.3:** XML-Nachricht im Standard-i2b2-Format des QDC-Views, wie sie an die QDC-Zelle übermittelt wird. Aus Platzgründen wurde die Query-Definition in Zeile 48 durch Punkte ersetzt.