Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3527

Semantisches Wiki zur Erfassung von Design-Patterns

Norbert Fürst

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Frank Leymann
Betreuer/in:	Dipl.-Inf. Christoph Fehling
Beginn am:	4. Februar 2013
Beendet am:	6. August 2013
CR-Nummer:	D.2.1, D.2.2, D.2.11, H.2.1, H.2.7, H.3.5, H.5.2, I.5.0

Zusammenfassung

Patterns sind ein in unterschiedlichen Domänen verbreitetes Konzept, das häufig auftauchende Probleme in einem gewissen Kontext beschreibt und dazu eine Musterlösung bietet. Neue Patterns werden aus der Abstraktion von konkreten oft auftauchenden Problemen und deren Lösungen von Pattern Autoren entdeckt. Ein in allen Domänen auftauchendes Problem der Pattern Community ist der fehlende IT-Support bei der Verwaltung der Patterns. Im Rahmen dieser Diplomarbeit wurde daher ein Pattern Repository auf Basis von semantischen Wiki-Technologien entworfen, das die Erstellung, Verwaltung und Suche von Patterns ermöglicht. Dabei ist das Patternformat frei konfigurierbar, so dass die Anwendung für unterschiedliche Domänen angepasst werden kann. Der Entwurf wurde auf Basis von Semantic Mediawiki implementiert und zuletzt in einem Vergleich zu anderen Pattern Repositorys evaluiert.

Abstract

Patterns are a concept spread in different domains that describes problems in a certain context and offers a sample solution. Pattern authors discover new patterns by abstracting them from reoccurring actual problems and their solutions. A problem concerning pattern communities of all domains is the missing IT-support for managing patterns. This diploma thesis designs a pattern repository based on semantic wiki-technology, enabling the creation, management and search of patterns. To adapt the application for different domains, the pattern format is freely configurable. The design was implemented based on Semantic Mediawiki and evaluated by the comparison with other pattern repositories.

Inhaltsverzeichnis

1 Einleitung **9**
- 1.1 Ziel . . . 10
- 1.2 Gliederung der Arbeit . . . 10

2 Grundlagen **13**
- 2.1 Repository . . . 13
- 2.2 Patterns und Patternsprachen . . . 14
  - 2.2.1 Patternformat . . . 15
  - 2.2.2 Pattern Auffindung und Anwendung . . . 17
- 2.3 Semantic Web . . . 18
  - 2.3.1 RDF und RDFS . . . 19
  - 2.3.2 OWL zur Ontologiemodellierung . . . 21
  - 2.3.3 Abfragen mit SPARQL . . . 24
  - 2.3.4 Triplestores . . . 24
  - 2.3.5 Ontologie-Modellierung mit Protégé . . . 25
  - 2.3.6 Semantic Mediawiki . . . 26
  - 2.3.7 Erweiterungen von Semantic Mediawiki . . . 28

3 Anforderungen **33**
- 3.1 Funktionale Anforderungen . . . 33
- 3.2 Nichtfunktionale Anforderungen . . . 36
- 3.3 Use Cases . . . 36
- 3.4 Gründe für Verwendung semantischer Technologie . . . 38
- 3.5 Evaluation von vorhandenen semantischen Wikianwendungen . . . 39
  - 3.5.1 Vergleichstabelle . . . 41

4 Konzeptionelles Design **45**
- 4.1 Eingabe von Patterns . . . 46
  - 4.1.1 Formular . . . 47
  - 4.1.2 Templates . . . 48
- 4.2 Annotation . . . 50
  - 4.2.1 Semantische Attribute - Linktypen . . . 51
  - 4.2.2 Zieleigenschaften . . . 54
  - 4.2.3 Formular . . . 56
- 4.3 Verwendung des Pattern Repositorys . . . 57
  - 4.3.1 Einstieg in die Patterndomäne . . . 58
  - 4.3.2 Assistent . . . 59

4.3.3 Weiteres Navigieren . . . 62
4.4 Resultierendes Datenmodell . . . 63
4.4.1 OWL-Ontologie: Kerndatenmodell . . . 65
4.4.2 OWL-Ontologie: Zieleigenschaften . . . 73
4.4.3 Abbildung auf Semantic Mediawiki-Elemente . . . 74

5 Implementierung des Pattern Repositorys 79
5.1 Web-Entwicklung mit dem Mediawiki-Framework . . . 79
5.1.1 Technologien des Mediawiki-Frameworks . . . 79
5.1.2 Entwicklung von Erweiterungen für das Mediawiki-Framework . . . 80
5.2 Systemarchitektur . . . 82
5.3 Import des Datenmodells . . . 85
5.4 Das Patternformular . . . 87
5.5 Die Erweiterung - Pattern Repository . . . 88
5.5.1 Annotationsunterstützung - Semantic Textarea . . . 89
5.5.2 Auswahl semantischer Propertys - Property Dropdown . . . 93
5.5.3 Zahlenauswahl - Number Slider . . . 94
5.5.4 Ausblendbare Querys - Query Tabs . . . 95
5.5.5 Visualisierung von semantischen Prädikaten im Kontext von Kategorien - Property Visualizer . . . 97
5.5.6 Suchassistent - Wizard . . . 99

6 Ergebnis und Evaluation 105
6.1 Das Pattern Repository . . . 105
6.1.1 Infrastruktur . . . 107
6.1.2 Administration . . . 108
6.2 Evaluation . . . 109
6.2.1 Beispieldomäne Cloud Computing Patterns . . . 110
6.2.2 Beispieldomäne Kostüm-Patterns . . . 110
6.2.3 Vergleich mit anderen Pattern-Repositorys . . . 111
6.2.4 Vorteile und Nachteile des Pattern Repositorys . . . 117
6.3 Rückblick auf Verlauf der Arbeit . . . 118

7 Zusammenfassung und Ausblick 121
7.1 Zusammenfassung . . . 121
7.2 Ausblick . . . 122

Literaturverzeichnis 125

Abbildungsverzeichnis

2.1 Der Pattern-„Lebenszyklus“ (nach [SBLE12, S. 3]) 17
2.2 Mehrere RDF-Tripel in Graphdarstellung mit Literalen. Legende: Kreise=*Ressourcen*, Rechtecke=*Literale*, Pfeile=*Beziehungen* 19
2.3 Klassenhierarchie der Beziehung zwischen OWL/RDF(S) (nach [AVH04, S. 119]). Legende: Rechtecke=*Elemente aus OWL/RDF(S)*, Pfeile=*Generalisierung* . . 22
2.4 Beispiel einer einfachen OWL-Ontologie 23
2.5 Die graphische Darstellung einer OWL-Ontologie 25

3.1 Use Cases des Pattern Repository 37

4.1 Arbeitsablauf beim Verwenden des Pattern Repositorys 45
4.2 Mockup einiger Formularelemente 48
4.3 Zusammenhang zwischen Formular und Templates 51
4.4 N-äre Relation in einem Entity-Relationship Diagramm 55
4.5 N-äre Relation in OWL 55
4.6 Datenmodell - Kategorien 65
4.7 Datenmodell - Visualisierung 66
4.8 Datenmodell - Übersicht über die Abschnittsbeziehungen 68
4.9 Datenmodell - Abschnitte 69
4.10 Datenmodell - Eingabetypen des Formulars 70
4.11 Datenmodell - Semantische Attribute 72
4.12 Allgemeine Ontologie zur Einbindung von Zieleigenschaften mit Legende . . 74
4.13 Mapping der OWL-Ontologien durch den Importvorgang. Legende: Gestrichelter Pfeil=*Mapping* 75

5.1 Systemarchitektur mit angebundenem Triplestore. Auf der linken Seite ist das allgemeine Schichtenmodell zu sehen und rechts die konkrete Ausprägung des Mediawiki-Stacks mit den verwendeten Technologien (nach [Kar11]) . . . 83
5.2 Aufbau von DataWiki. Legende: Gestrichelter schwarzer Pfeil=*Abhängigkeit*, Braune Linie=*Verbindung zu MySQL-Datenbanktabelle*, Lila Linie=*Für den Triplestore benötigte Verbindung* 84
5.3 Interne Architektur der Erweiterung Pattern Repository. Legende: Schwarz gestrichelter Pfeil=*Zur Realisierung verwendet für*. Orange gestrichelter Pfeil=*Abhängigkeit* 90
5.4 Auswahl des zu annotierenden Wortes in Semantic Textarea 91
5.5 Annotation mithilfe von Semantic Textarea 91
5.6 Anschließende Darstellung der Annotation in Semantic Textarea 91

5.7 Eingabe eines Zahlenwerts durch den Number Slider 94
5.8 Ausgeblendete Querys 96
5.9 Eingeblendete Query 96
5.10 Visualisierung eines semantischen Linktyps durch den Property Visualizer . . 98
5.11 Das Hauptmenü des entwickelten Pattern Wizard - Links das Auswahlmenü der Empfehlungsfunktionen und auf der rechten Seite die Leseliste 99
5.12 Auswahl eines bekannten Anwendungsfalls im Wizard - Auf der linken Seite geschieht die Auswahl, in der Mitte werden die Empfehlungen angezeigt und rechts unten ist die Leseliste zu sehen 101
5.13 Auswahl grundlegenden Patterns im Wizard - Auf der linken Seite geschieht die Auswahl und rechts ist die Leseliste zu sehen. Die abgebildeten Beispielspatterns sind aus [FLR+13] 101
5.14 Auswahl einer Benutzerrolle im Wizard - Auf der linken Seite geschieht die Auswahl und rechts ist die Leseliste zu sehen 102
5.15 Vorschläge anhand der Leseliste im Wizard - Auf der linken Seite sind die Vorschläge abgebildet und rechts ist die Leseliste zu sehen 103

6.1 Das Hauptmenü des Pattern Repositorys 106

Tabellenverzeichnis

3.1 Vergleichstabelle der Evaluation von Alternativen 42

4.1 Abbildung von OWL- auf Semantic Mediawiki-Elemente 64

6.1 Vergleichstabelle von alternativen Pattern Repositorys 115

Verzeichnis der Auflistungen

2.1 Eine einfache SPARQL-Query 24

1 Einleitung

Seit dem Beginn des Informationszeitalters nimmt die Spezialisierung in unterschiedlichen Wissenschaftsbereichen zu, was Experten mit sehr spezifischem Fachwissen hervorbringt [Haro5]. Um solches domänenspezifisches Expertenwissen in Form von leicht konsumierbaren Wissensportionen für Andere zugänglich zu machen, gibt es das Konzept der „Patterns“ (Muster). Patterns sind Problem-Lösungspaare, die ein Problem in einem gewissen Kontext beschreiben und dazu eine Musterlösung bieten. Wenn es auf einem Gebiet eine gewisse Anzahl von Patterns gibt, können diese miteinander vernetzt werden, und bilden anschließend eine Patternsprache. Die Vernetzung geschieht dabei durch die unterschiedlichen Beziehungen, welche die Patterns untereinander aufweisen, z.B. kann ein Pattern eine Spezialisierung eines allgemeineren Patterns darstellen. Die Patternsprache kann für den Leser einer Sammlung von Patterns eine Reihenfolge vorgeben, in welcher sie betrachtet werden. Neue Patterns werden entdeckt, indem zunächst sich oft wiederholende ähnliche Problemstellungen und jeweils konkrete Lösungen in existierenden Anwendungsfällen von einem Experten des Gebiets analysiert werden. Anschließend wird versucht, einen gemeinsamen Kern des Problems zu abstrahieren und schließlich eine aus der Erfahrung heraus als gut befundene Lösung dafür zu beschrieben. Anhand einer Sammlung von Patterns können weniger erfahrene Anwender des Gebiets von der Erfahrung von Experten profitieren, indem sie Patterns auf ihre konkreten Entwurfsprobleme anwenden und individualisieren.

Als Wegbereiter für die Verbreitung von Patterns kann der Architekt Christopher Alexander betrachtet werden [AIS+77]. Die Verwendung von Patterns wurde in anderen Gebieten übernommen, und so gibt es neben Werken im IT-Bereich über objektorientierte Programmierung [JGVH95] oder Cloud Computing [FLR+13] auch Anwendungen im Bereich des Kostümmanagements [SBLE12] oder der Pädagogik [Köp13]. Ein Problem, welches in den Patterngemeinden aller Gebiete auftritt, ist die fehlende Unterstützung durch IT-Werkzeuge bei der Erstellung, Verwaltung und Suche von Patterns. Dies hat einige negative Folgen, wie z.B. die oft schlechte Dokumentation, aus welchen Quellen ein Pattern abstrahiert wurde. Patternautoren müssen zudem ohne die Unterstützung durch benutzerfreundliche Formulare oder ähnliches manuell sicherstellen, dass ein konsistentes Patternformat im ganzen Patternkatalog eingehalten wird. Für Benutzer eines Patternwerks ergibt sich oft das Problem, in einer unübersichtlichen Menge von Patterns eine geeignete Lösung für sich zu finden. Mithilfe eines IT-Werkzeugs könnte man den Benutzer durch einen Assistenten unterstützen, der bei der Auffindung von Patterns durch Empfehlungen hilft. Sowohl für Autoren als auch Benutzer wäre die Visualisierung der Patternsprache aufgrund der Beziehungen der Patterns zueinander zusätzlich ein nützliches Hilfsmittel, um die Vernetzung zu verbessern und um das Verständnis zu vertiefen.

Ein Patternkatalog in elektronischer Form wird als Pattern Repository bezeichnet. Es gibt bereits einige Pattern Repositorys [qui, Yah, SBS]; allerdings sind dies meistens von den Autoren eines Patternbuchs betriebene Webseiten, die keine weiteren Hilfsmittel bieten und spezifisch für eine konkrete Patterndomäne entwickelt wurden. Es fehlt eine Anwendung, die Patternautoren von Anfang an unterstützt, ein benutzerfreundliches Interface für die Benutzer des Patternkatalogs bietet und dabei frei konfigurierbar für unterschiedliche Patternformate ist.

1.1 Ziel

Das Ziel dieser Diplomarbeit ist die Entwicklung eines auf semantischen Wikitechnologien basierenden Pattern Repositorys, das die unterstützte Erstellung, Suche und Empfehlung von Design Patterns unterschiedlichster Domänen ermöglicht. Für eine möglichst komfortable Erstellung bzw. Eingabe von Patterns sollen dafür Dokumentvorlagen erstellt werden, so dass dem Benutzer ein gewisses Patternformat vorgegeben wird. Um eine erweiterte Suche und die Verwendung von semantischen Querytechniken zu ermöglichen, soll es auf benutzerfreundliche Art möglich sein, Patterns untereinander und Patterns mit Anwendungsfällen mithilfe von semantischen Annotationen zu verknüpfen. Neben den umfangreichen Querytechniken, die dem Benutzer dadurch zur Verfügung stehen, soll es einen Assistenten geben, der dem Benutzer sinnvolle Patterns empfehlen kann. Zum Abschluss der Arbeit soll eine Evaluation des resultierenden Pattern Repositorys geschehen, indem es an zwei Beispieldomänen getestet wird. Dabei handelt es sich um sehr unterschiedliche Domänen, so dass überprüft werden kann, ob das Pattern Repository flexibel an unterschiedliche Patternformate angepasst werden kann.

1.2 Gliederung der Arbeit

Im Folgenden soll ein kurzer Überblick gegeben werden, aus welchen Kapiteln die Diplomarbeit besteht und was jeweils inhaltliche Schwerpunkte sind.

Kapitel 2 - Grundlagen In diesem Kapitel werden die benötigten Grundlagen zum Verständnis dieser Arbeit in der benötigten Tiefe erläutert. Neben einer allgemeinen Einführung in Patterns und Patternsprachen gibt es einen Überblick der Semantic Web Technologien, die im Rahmen dieser Arbeit zum Einsatz kommen. Dabei werden viele Referenzen zu vertiefender Literatur geboten, um dem geneigten Leser Möglichkeiten zu bieten, sich tiefer mit der Materie zu beschäftigen.

Kapitel 3 - Anforderungen In den Anforderungen wird das in diesem Kapitel beschriebene Ziel in detaillierte Anforderungen in Form von User Stories aufgespalten, wie man sie aus der agilen Softwareentwicklung kennt. In den darauf folgenden Use Cases werden konkretere Einblicke in die Anwendungsaspekte des Systems gegeben. Neben einer Erläuterung,

weshalb semantische Technologie in der Arbeit verwendet wurde, gibt es auch eine Evaluation, die zu der Entscheidung geführt hat, Semantic Mediawiki als Basis für diese Arbeit zu verwenden.

Kapitel 4 - Konzeptionelles Design Das konzeptionelle Design beschreibt zunächst auf einer abstrakten Ebene, wie einzelne Verwendungsaspekte der entwickelten Software realisiert werden sollten. Anschließend folgt eine konkrete Umsetzung in Form eines Datenmodells, das in Ontologiesprache OWL modelliert wurde und sehr ausführlich beschrieben wurde. Am Ende des Kapitels gibt es eine Abbildung des Datenmodells auf Elemente von Semantic Mediawiki.

Kapitel 5 - Implementierung des Pattern Repositorys Die Implementierung stellt die Systemarchitektur des entworfenen Systems auf mehreren Ebenen dar. Es folgt eine Beschreibung des Programms, welches zum Import des im konzeptionellen Design beschriebenen Datenmodells dient. Die Beschreibung des Patternformulars geht auf die Details bei der Umsetzung ein und zuletzt folgt nach einer kurzen Einführung in die Begriffe der Mediawiki Erweiterungsentwicklung die genaue Vorstellung der für die Diplomarbeit entwickelten Mediawiki Erweiterungen.

Kapitel 6 - Ergebnis und Evaluation Das Ergebnis der Diplomarbeit wird vorgestellt und kurz auf dessen Infrastruktur und Administration eingegangen. Es folgt eine Evaluation, in der zunächst die Fähigkeit des entwickelten Pattern Repositorys überprüft wird, sich für beliebige Patterndomänen konfigurieren zu lassen. Es folgt ein Vergleich mit anderen Repositorys, die online verfügbar sind, um mit einem Rückblick auf den Verlauf der Arbeit zu schließen.

Kapitel 7 - Zusammenfassung und Ausblick Die wichtigsten Ergebnisse der Arbeit werden kurz zusammengefasst und es wird ein Ausblick auf mögliche weitere Entwicklungen auf dem Gebiet gegeben.

2 Grundlagen

Nachdem nun auf die Motivation und die Zielstellung dieser Arbeit eingegangen wurde, sollen im folgenden Kapitel grundlegende Begriffe und Systeme erklärt werden, die zum Verständnis dieser Diplomarbeit nötig sind. In Abschnitt 2.1 soll dazu zunächst der Begriff „Repository“ definiert werden, um eine klare Vorstellung des Ziels der Arbeit zu ermöglichen. In Abschnitt 2.2 soll anschließend das Verständnis für Patterns im Allgemeinen verbessert werden, um die Anforderungen aus Kapitel 3 an ein Pattern Repository besser verstehen zu können. Abschnitt 2.3, welcher das Semantic Web behandelt, soll schließlich das nötige Wissen auf der technischen Seite vermitteln, um die Modellierung des Datenmodells in Kapitel 4 verstehen zu können, um schließlich ein Gesamtverständnis für das System mithilfe von Kapitel 5 zu erlangen. Hierbei soll insbesondere auf Abschnitt 2.3.5 verwiesen werden, der zum Verständnis der in der Arbeit sehr gehäuft auftauchenden Darstellungen von Ontologien notwendig ist.

2.1 Repository

Der Begriff *Repository* (deutsch: Sammelplatz, Aufbewahrungsort) bezeichnet ein Dokumentationssystem, welches die Ablage und Suche von Informationen und die Beschreibung der Beziehungen der abgelegten Elemente zueinander anhand von Metadaten ermöglicht. Habermann und Leymann definieren dies folgendermaßen:

> *Das Repository ist die zentrale Ablage von beschreibenden Informationen über alle Informationselemente einer Organisation und deren Benutzer.* [HL93, S. 11]
>
> *Funktional ist das Repository ein System, das Informationen über Objekte der Softwareproduktion [...], deren Beschreibungen und Beziehungen untereinander verwaltet, auswertet und bereitstellt*. [HL93, S. 15]

Eine sehr ähnliche Definition liefert Ortner:

> *Repositorien sind Dokumentationssysteme. In Repositorien werden auf einer Metasprachebene Sprachartefakte [...] strukturiert beschrieben.* [Ort99, S. 236]

In den beiden zitierten Werken [HL93, Ort99] wurde der Begriff Repository dabei sehr beschränkt in dem Bereich des CASE[1] eingesetzt. Daher wurden in den Zitaten die zu CASE-spezifischen Stellen weggelassen. Dieser Diplomarbeit soll ein erweitertes Verständnis von

[1] Computer-Aided Software Engineering

Repositorys zugrunde liegen, so dass diese als Dokumentationssysteme für Informationen aller Art betrachtet werden können. In diesem Sinne lässt sich beispielsweise jedes Wiki als Wissensrepository betrachten, wie auch aus dem Zitat von García *et al.* ersichtlich wird:

> *This paper focuses on the use of wikis such as Web 2.0 knowledge repositories [...].* [GASB11]

Das im Rahmen dieser Arbeit entwickelte Repository wird als „Pattern Repository" bezeichnet, um auszudrücken, dass es sich dabei um ein auf Patterns spezialisiertes System mit Repository-Funktionalität handelt. Diese Bezeichnung wird ebenfalls von anderen Projekten verwendet, wie z.B. von [KE02], [por] und [Hee]. Welche Anforderungen an ein Pattern Repository es genau gibt, und damit auch die verfügbaren Funktionalitäten, wird ausführlicher in Kapitel 3 erläutert. Um überhaupt eine genaue Vorstellung vom Begriff der *Patterns* zu bekommen, folgt nun eine Definition und auch eine Erläuterung der damit in Verbindung stehenden *Patternsprachen*.

2.2 Patterns und Patternsprachen

Das Konzept der *Patterns* (deutsch: Muster) verdankt die mittlerweile große Verbreitung in unterschiedlichen Anwendungsgebieten dem Architekten, Architektur- und Systemtheoretiker Christopher Alexander. Alexander definierte Patterns in seinem klassischen Werk *A Pattern Language: Towns, Buildings, Constructions* wie folgt:

> *Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.* [AIS+77]

Christopher Alexander war zur Zeit des Verfassens dieses Werks bereits ein erfahrener Architekt und hatte festgestellt, dass ihm bei unterschiedlichen Projekten sich wiederholende Entwurfsprobleme begegneten. Diese mussten zwar jeweils individuell im Kontext des Gesamtprojekts gelöst werden, entsprachen dabei aber auf einer abstrakteren Ebene Problemen, welchen er schon bei früheren Projekten begegnet war. Daraus entsprang die Idee, für diese im Kern gleichen Probleme eine Lösung ebenfalls auf abstrakter Ebene anzugeben.

Mithilfe von Patterns, für die er ein festes Format in Form eines strukturierten Textes festlegte, können Best Practises von erfahrenen Experten einer Domäne festgehalten werden, die anschließend als leicht zugängliche „Nuggets of Advice" [FLR+13, S. 7] zur Verfügung stehen. Patterns helfen leicht variierende, aber im Kern gleiche Probleme nicht jedes Mal aufs Neue lösen zu müssen.

Um den Ansatz von C. Alexander etwas greifbarer zu machen, soll hier nun ein Beispielpattern kurz erläutert werden. Alexanders Patterns in [AIS+77] lösen architektonische Probleme auf der Ebene von Regionen, Städten, Stadtvierteln bis hin zu einer sehr detaillierten Stufe wie der Gestaltung von Türgriffen innerhalb eines Zimmers. Mit dem *Halb-versteckter Garten*-Pattern [AIS+77, S. 260-261] beschreibt er beispielsweise, wie der Grundentwurf eines Hauses

aussehen kann, das einen halb versteckten Garten besitzen soll. Zunächst wird beschrieben, mit welchen anderen Patterns das Halb-versteckter Garten-Pattern zusammenhängt, wie z.B. mit dem Häuserblock-Pattern. Anschließend wird die Tatsache, dass ein halb versteckter Garten an einem Haus die relative Position der Häuser innerhalb eines Blocks verändert, als Grund für die Verbindung mit dem Häuserblock-Pattern aufgeführt. Darüber hinaus wird das zugrundeliegende Problem beschrieben, dass der Garten weder voll von der Straße aus sichtbar sein darf, da dies nicht privat genug wäre, aber auch nicht zu weit weg von der Straße sein darf, da dies zu isoliert wäre. Es folgt ein Lösungsvorschlag, wie ein solch halb versteckter Garten in einen Entwurf integriert werden kann und auf was geachtet werden muss. Zusätzlich wird das Pattern mit anderen Patterns in Verbindung gebracht, die ebenfalls für diesen Entwurf relevant sein dürften, z.B. der Gartenmauer-Pattern. Anschließend wird auch noch auf Verfeinerungen des Patterns hingewiesen, wie z.B. den Dachgarten.

Laut Hohpe und Woolf [HW03, S. xli] entspricht jedes Pattern einer Entscheidung, die gemacht werden muss, sowie den Abwägungen, die in Betracht gezogen werden müssen, um die Entscheidung zu fällen. Eine sog. *Patternsprache* ergibt sich aus einem Netz von miteinander verknüpften Patterns, in dem jede Entscheidung zu weiteren verwandten Patterns führt. Auf dieser Weise ist es möglich, sich anhand des Expertenwissens durch die Patternsprache führen zu lassen. Bei Alexander ist die Mustersprache, welche durch die Relationen zwischen den Patterns gebildet wird, in eine hierarchische Baumstruktur unterteilt. Sie beginnt, wie bereits oben erwähnt, mit Patterns für die Gestaltung von Regionen und wird immer detaillierter bis zu dem Grad der Gestaltung von einzelnen Objekten in einem Haus. Dabei enthalten Patterns auf höherer Ebene Patterns auf niederer Ebene, oder die Patterns niederer Ebenen werden durch die Spezialisierung von allgemeineren Patterns gebildet. Auf gleicher Ebene gibt es Patterns, welche Alternativen zueinander darstellen und es auf diese Weise zulassen, unterschiedliche Wege einzuschlagen.

Der Ansatz, Expertenwissen durch Patterns festzuhalten, wurde auch in anderen Bereichen als der Architektur übernommen, z.B. für pädagogische Zwecke [Köp13], dem Design von Filmkostümen [SBLE12] und nicht zuletzt im IT-Bereich. Das Werk, das den Grundstein für Design Patterns in der Informatik gelegt hat, ist *Design Patterns: Elements of Reusable Object-Oriented Software* [JGVH95]. Die Autoren des Buches, Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides werden als Parodie auf die chinesische Viererbande um Mao Zedong auch als *Gang of Four* bezeichnet. Das Buch beschreibt Design Patterns für objektorientierte Software und verfeinert dabei das originale Format von Alexander in einigen Aspekten, behält aber eine einheitliche Struktur der Patterns bei, um das Lernen, Vergleichen und Verwenden der Patterns zu vereinfachen [JGVH95, S. 16]. Auch das Bilden der Patternsprache funktioniert sehr ähnlich wie bei Alexander, mit dem Unterschied, dass anstatt Türen und Wänden nun Beziehungen aus der objektorientierten Programmierung verwendet werden.

2.2.1 Patternformat

Im Kern besteht ein Muster immer aus einem Problem-Lösungspaar, das aus vielen anderen Problem-Lösungspaaren abstrahiert wurde und in einen gewissen Kontext gesetzt wird

[Bus98, S. 1-4]. Dies stimmt auch mit Alexanders Definition in seinem zweiten Werk zu Patterns, *The Timeless Way Of Building*, überein:

> *Each pattern is a three part rule, which express a relation between a certain context, a problem, and a solution.* [Ale79, S. 247ff]

Dabei wird im Kontext die Situation beschrieben, in der das Problem auftritt, welches das Pattern löst und bietet so einen Rahmen. Im Problemabschnitt werden die unterschiedlichen Kräfte (*forces*) beschrieben, die Aspekte darstellen, die bei der Lösung des Entwurfsproblems zu berücksichtigen sind, wie z.B. bestimmte Anforderungen oder sonstige Randbedingungen. Die Lösung beschreibt das allgemeine Lösungsprinzip, mit dem das Problem auf einer abstrakten Ebene gelöst werden kann und versucht dabei, die Kräfte des Problems möglichst gut auszugleichen. Während dieses ursprüngliche Format zur Beschreibung von Patterns von Alexander in [AIS+77] verwendet wurde, wird es auch in neueren Werken, wie z.B. [MD97], verwendet. In [JGVH95, S. 16-18] wurde dieses Format feiner aufgegliedert und schließlich in [FLR+13, S. 10-11] sinnvoll weiterentwickelt, so dass dieses als Beispiel erläutert werden soll. Dabei sei angemerkt, dass sich Patternformate unterschiedlicher Werke meistens unterscheiden, im Kern dabei jedoch immer das oben erläuterte dreiteilige Format verwendet wird.

Patternname Der Name des Patterns stellt wohl in allen Patternsprachen das wichtigste Merkmal zur schnellen Identifizierung eines Patterns dar, und sollte daher prägnant gewählt werden.

Intent Die Absicht drückt kurz und knapp den Zweck und das Ziel eines Patterns aus.

Icon Dient als schnelles Identifizierungsmerkmal eines Patterns und kann in Darstellungen der Patternsprache verwendet werden.

Driving Question Die Leitfrage erfasst das Problem, welches dem Pattern zugrunde liegt.

Context Der Kontext beschreibt, wie oben bereits erläutert, die Umgebung und die Kräfte, mit welchen das Problem umgeben ist.

Solution Die Lösung skizziert kurz, wie das Pattern das in der Leitfrage beschriebene Problem löst.

Result Der Ergebnisabschnitt erklärt die bereits skizzierte Lösung in größerem Detail.

Variations Beschreibt leichte Abweichungen von dem beschriebenen Pattern, welche es nicht nötig machen würden, ein komplett neues Pattern zu verfassen.

Related Patterns Beschreibt Beziehungen zu anderen Patterns, beispielsweise mit welchen das aktuelle Pattern gut kombiniert werden kann, und welche ausgeschlossen werden. Auf diese Weise wird die Patternsprache gebildet.

Known Uses Beispiele für Anwendungsgebiete, in denen das Pattern verwendet wurde und eventuell als Quelle zur Abstraktion des Patterns genutzt wurden.

2.2.2 Pattern Auffindung und Anwendung

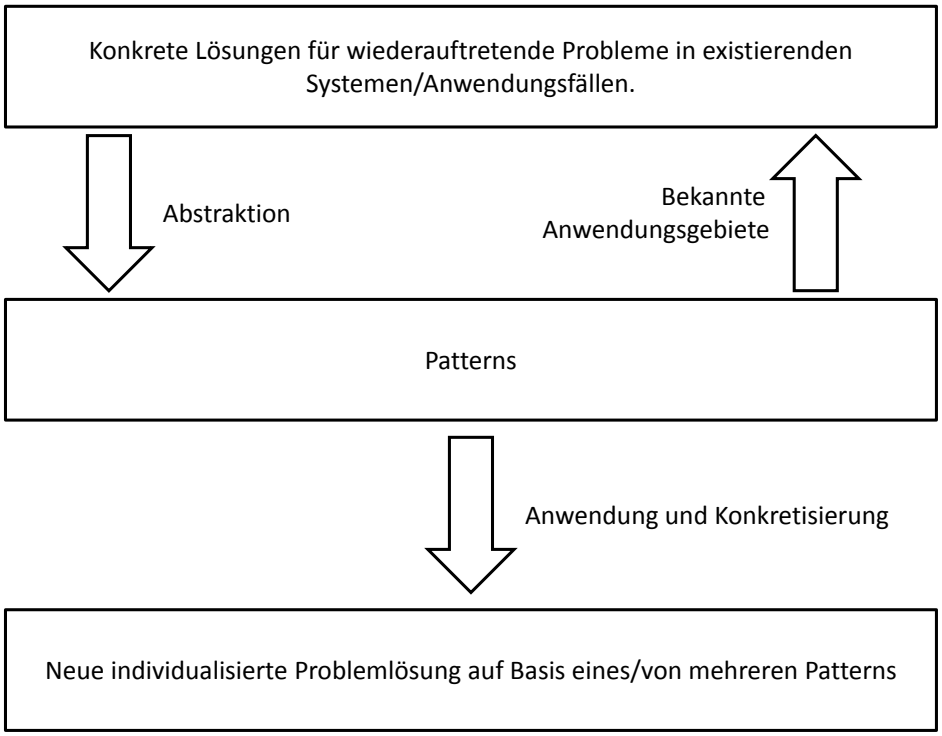

Abbildung 2.1: Der Pattern-„Lebenszyklus“ (nach [SBLE12, S. 3])

Das Verfassen von neuen Patterns geschieht nicht auf die Weise, dass diese neu „erfunden“ würden, sondern indem Sie von Experten *entdeckt* werden. Diese betrachten sehr viel ähnliche Anwendungsgebiete, abstrahieren Problem-Lösungspaare aus diesen und entdecken durch das Herausfiltern von Gemeinsamkeiten schließlich neue Patterns [Bus98, S. 2]. Wurden erst einmal eine Reihe von Patterns entdeckt, und diese zu einer Patternsprache vernetzt, stehen sie zur Anwendung bereit. Soll mithilfe von Patterns ein konkretes Problem gelöst werden, muss dieses unter Umständen erst soweit verallgemeinert werden, bis sich ein passendes Pattern zu dem verallgemeinerten Problem findet. Anhand dieses Patterns ist es nun möglich, entlang der Patternsprache zu navigieren und dadurch unterschiedliche Entwurfsprobleme des konkreten Anwendungsfalles zu lösen. Hierbei sei ein weiterer besonderer Nutzen des patterngestützten Entwurfs hervorgehoben: Werden Systeme mithilfe von Patterns entworfen, ist es möglich, innerhalb der Domäne auf ein gemeinsames Vokabular zurückzugreifen und Entwürfe vergleichbar zu machen [FEL$^+$12]. Dabei spielt es keine Rolle, ob nun ein Bauwerk anhand der Patterns von Alexander entworfen wurde, oder ein objektorientiertes Programm verfasst wurde; gewisse Charakteristika lassen sich nun durch Patterns vergleichen.

Hierzu soll Abbildung 2.1 einen Überblick bieten: die oberste „Ebene“ des Bildes stellen konkrete Anwendungsfälle dar, aus denen anhand von Problem-Lösungspaaren Patterns abstrahiert werden können. Dies ist in der Abbildung anhand des mit *Abstraktion* beschrifteten

Pfeils erkennbar. Von den Patterns, welche die zweite Ebene darstellen, gibt es wiederum einen Rückverweis auf bekannte Anwendungsgebiete (*Known Uses*) durch den Pfeil *Bekannte Anwendungsgebiete*. Die dritte Ebene zeigt die Anwendung von Patterns, indem sie für den konkreten Problemfall angepasst werden und so eine neue individuelle Lösung bilden, die aufgrund des patternbasierten Vorgehens trotzdem schnell fassbar und vergleichbar ist. Diese Beziehung ist in der Abbildung anhand des mit *Anwendung und Konkretisierung* beschrifteten Pfeils dargestellt.

Nachdem nun genauer auf die inhaltlichen Aspekte rund um Patterns und Patternsprachen eingegangen wurde, folgt eine Erläuterung von wichtigen Begriffen um das *Semantic Web*. Diese sind zum Verständnis der technischen Seite dieser Diplomarbeit und somit der folgenden Kapitel nötig.

2.3 Semantic Web

Das *World Wide Web* beeinflusst unser Leben wie sonst kaum eine Technologie, und entwickelt sich rasant weiter. Zunächst dominierten statische Webseiten das Netz, die von Einzelpersonen oder Firmen gepflegt wurden. Ab der Jahrtausendwende entwickelte sich eine Konzentration auf inhaltsbezogene Plattformen, auf welchen die Benutzer kollaborativ selbst neue Inhalte erstellen. Diese neuen das Web dominierenden Anwendungen wie Videoplattformen, soziale Netzwerke oder Wikis wurden unter dem weitreichenden Begriff des „Web 2.0“ zusammen gefasst. Das Aufkommen dieser Anwendungen hatte zur Folge, dass die verfügbaren Informationen immer weiter und schneller wuchsen. Dementsprechend erklärt sich auch die enorm gestiegene Bedeutung von Suchmaschinen und beispielsweise der unvergleichliche Aufstieg von *Google*[2]. Das Problem war nun, zu einer gewissen gewünschten Information guten Quellen aus den schier endlosen zu Verfügung stehenden Datenmengen herauszufinden und insbesondere Informationen mit der korrekten semantischen Bedeutung zu finden. Wird beispielsweise eine Google-Suche mit dem Begriff „Owl“ begonnen, und auf Informationen über die Web Ontology Language gehofft, werden sich viele Suchergebnisse auf die Eule als Tier beziehen, da das System nicht den *semantischen Zusammenhang* kennt.

Bei der großen unstrukturierten Datenmenge, welche das Web momentan darstellt, gibt es einige fundamentale Probleme, welche laut [HKRS08, S. 9-10] insbesondere die große Heterogenität der Daten, die dadurch nötige Informationsintegration und das Fehlen eines zugrunde liegenden maschinenlesbaren Datenschemas darstellen. Die unter dem Begriff des *Semantic Web* zusammengefassten Technologien versuchen dieses Problem zu lösen. Mithilfe von offenen Standards wird die Möglichkeit geschaffen, Informationen auf eine Weise zu beschreiben, dass sie von Maschinen verarbeitbar werden. Das World Wide Web Consortium[3] hat hierfür die Standards RDF(S) und OWL auf Basis von XML geschaffen. RDF(S) und OWL stellen formale Sprachen dar, die zur Beschreibung von Ontologien dienen können. [AVH04, S. 11] definiert den Begriff Ontologie nach R. Studer: *„An ontology is an explicit and formal*

[2] http://www.google.com/
[3] http://www.w3.org/

specification of a conceptualization.". Eine Ontologie stellt im Zusammenhang mit dem Semantic Web eine formale Beschreibung des Wissens über ein bestimmtes Anwendungsgebiet in Form eines RDF(S)- oder OWL-Dokuments dar. Dadurch wird es möglich, Informationen über semantischen Zusammenhänge in einer formalen maschinenlesbaren Sprache zu beschreiben. Unter Berücksichtigung dieser Zusammenhänge können dem Suchenden nun passendere Ergebnisse präsentiert werden, die mit der tatsächlich beabsichtigten Bedeutung übereinstimmen.

2.3.1 RDF und RDFS

Das *Resource Description Framework* (RDF) ist laut W3C ein standardisiertes Modell zum Datenaustausch im Web [W3C]. Den Hauptanwendungszweck von RDF sieht [HKRS08, S. 35] folgendermaßen: "Durch RDF sollen Anwendungen in die Lage versetzt werden, Daten im Web auszutauschen, ohne dass ihre ursprüngliche Bedeutung dabei verloren geht."Die grundlegende Dateneinheit in RDF stellen Aussagen im Subjekt-Prädikat-Objekt Format dar, welche als RDF-Tripel bezeichnet werden. Dabei sind Subjekt und Objekt beliebige Ressourcen, die durch *Uniform Resource Identifiers* (URI) identifiziert werden. Dabei ist nicht vorgeschrieben, ob es sich bei der URI um eine URL oder um eine beliebige andere einzigartige ID handelt. Zusätzlich lassen sich für Ressourcen unterschiedliche Datentypen angeben, und falls nun auf keine abstrakte Ressource, sondern auf ein konkretes Datenattribut z.B. des Typs `String` verwiesen werden soll, spricht man von einem *Literal*.

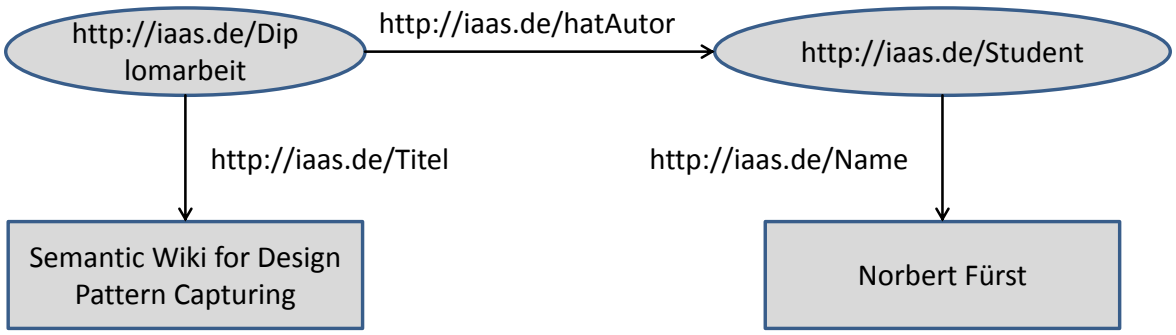

Abbildung 2.2: Mehrere RDF-Tripel in Graphdarstellung mit Literalen. Legende: Kreise=*Ressourcen*, Rechtecke=*Literale*, Pfeile=*Beziehungen*

RDF-Tripel beschreiben gerichtete Graphen, welche bei großen Datenmengen sog. „semantische Netze" bilden [AVH04, S. 69]. Ein Beispiel für einen solchen Graphen ist in Abbildung 2.2 abgebildet, welcher neben Ressourcen, die in Kreisen dargestellt sind, auch Literale in Form von Rechtecken enthält. Die Beziehungen zwischen Ressourcen und Literalen werden in der Grafik anhand von Pfeilen mit Bezeichnern dargestellt. Ein Baumformat wie bei klassischen XML-Dokumenten bietet sich bei solchen allgemeinen Ressourcen-Beziehungen insbesondere deswegen nicht an, da RDF nicht zur Beschreibung der Struktur hierarchischer

Dokumente dient. Die Unterordnung von verschiedenen Elementen ist bei RDF niemals eindeutig oder erwünscht, da bei einem Tripel nicht klar ist, ob sich nun das Subjekt dem Objekt unterordnen sollte oder umgekehrt. Des Weiteren ist ein wichtiger Aspekt von RDF, dass sich mehrere Graphen problemlos zu einem vereinen lassen, bei hierarchischen Strukturen dagegen ist dies sehr problematisch [HKRS08, S. 37].

In RDF werden keinerlei Annahmen über eine bestimmte Domäne gemacht, in welcher Ressourcen beschrieben werden, und keine Aussagen über die zugrundeliegende Semantik der beschriebenen Objekte. Für diesen Zweck wurde RDF Schema (RDFS) konzipiert, welches es erlaubt, ein *Vokabular* für eine bestimmte Domäne zu modellieren. Ein Vokabular bezeichnet dabei einen Satz von Bezeichnern für Individuen, Beziehungen und Klassen, festgelegte Bedeutungen für gewisse Bezeichner sowie Einschränkungen bei deren Verwendung. RDFS ermöglicht es nun, solches terminologisches Wissen über Begriffe eines Vokabulars zu spezifizieren [HKRS08, S. 67]. Auf syntaktischer Ebene ist jedes RDFS-Dokument in korrekter RDF-Syntax verfasst und beschreibt die neu eingeführten Konzepte konsequenterweise als RDF-Tripel. In RDFS werden folgende Konzepte zur Beschreibung der Semantik einer Domäne eingeführt:

Klasse Die Verwendung von Klassen erlaubt es, die grundlegende Unterscheidung von Individuen einer Domäne zu ermöglichen. Dabei gibt es die Beziehung zwischen Klasse und Instanz, wobei die Instanz ein konkretes Individuum einer Klasse ist. In dem Beispiel aus Abbildung 2.2 könnten die Ressourcen *Diplomarbeit* und *Student* beispielsweise Klassen darstellen, wohingegen *Semantic Wiki for Design Pattern Capturing* eine konkrete Instanz der Klasse *Diplomarbeit* wäre. Dabei ist es auch möglich, eine Klassenhierarchie zu definieren. Eine Überklasse für *Diplomarbeit* könnte demnach *Studentische Arbeit* sein, welche als Unterklasse ebenso *Studienarbeit* enthalten könnte. Weshalb eine solche Hierarchie nützlich ist, wird sich im nächsten Punkt zeigen.

Property Als Propertys werden diejenigen Ressourcen bezeichnet, die in Tripeln als Prädikate verwendet werden und dienen zum Ausdrücken von Relationen zwischen Subjekten und Objekten. Diese Relationen ähneln mathematischen Relationen insofern, dass sie eine Menge als Definitionsbereich (*Domain*) und eine Menge als Wertebereich (*Range*) besitzen. Die Mengen werden in RDFS dabei durch Klassen festgelegt, so dass bestimmt werden kann, welche Beziehungen zwischen welchen Klassen möglich sind. Domain schränkt dabei die Subjekte für Tripel ein, und Range dementsprechend die Objekte. Um zu dem obigen Beispiel zurückzukehren, könnte man nun die Property *hatAutor* so definieren, dass sie als Domain *Studentische Arbeit* zugewiesen bekommt, und als Range *Student*. Damit wäre es nun möglich, auch allen Unterklassen der *studentischen Arbeit*, wie z.B. der *Diplomarbeit* oder *Studienarbeit*, einen studentischen Autor zuzuweisen. Propertys sind vergleichbar zu Klassen, da sie Mengen von Individuenpaaren beschreiben, und verfügen ebenfalls über das Konzept der Subpropertys bzw. Unterpropertys. Die Beispielproperty *hatAutor* könnte dementsprechend eine Subproperty von *beschäftigtePersonen* sein, wovon eine weitere Subproperty beispielsweise *hatBetreuer* sein könnte.

2.3.2 OWL zur Ontologiemodellierung

Die *Web Ontology Language* OWL wurde vom W3C konzipiert, da die Ausdruckskraft von RDF(S) sehr begrenzt ist und sich bei der Modellierung von Ontologien für das Semantic Web einige mit RDF(S) nicht erfüllbare Anforderungen ergaben. Ein Anforderung war beispielsweise, Domain und Range einer Property lokal begrenzen zu können, so dass eine Range-Definition nur für eine eingeschränkte Menge von Klassen gilt. Eine weitere Anforderung war es, möglich zu machen, die Disjunktheit von Klassen ausdrücken zu können, was beispielsweise für die Klassen *Männlich* und *Weiblich* Sinn macht. Als letztes Beispiel für eine Anforderung, die in RDF(S) nicht erfüllt wurde, soll das Festlegen von Kardinalitäten dienen. Mit RDF(S) ist es nicht möglich, für eine Property *HatEltern* festzulegen, dass diese nur maximal zwei Werte annehmen kann.

Bei der Konzeption von OWL wurde versucht, einen Kompromiss zwischen Ausdrucksstärke und der Skalierbarkeit von logischem Schlussfolgern (*Reasoning*) zu gewährleisten. Deswegen wurde OWL in drei Teilsprachen unterteilt, die je nach den Anforderungen des Nutzers unterschiedliche Ausdrucksstärken und damit verbundene Komplexität von logischen Schlussfolgerungen besitzen [HKRS08, S. 151-154].

- *OWL Lite* ist die Teilsprache mit geringstem Umfang, ist daher auch am wenigsten ausdrucksstark, aber entscheidbar mit einer Zeitkomplexität von `ExpTime` im schlimmsten Fall. Diese Teilsprache besitzt auch die beste Unterstützung durch Softwarewerkzeuge, da sie am einfachsten zu Implementieren ist. Beispiele nicht verfügbarer Sprachkonstrukte sind das Fehlen von Klassendefinitionen durch Aufzählung oder Aussagen über die Disjunktheit von Klassen [AVH04, S. 118-119].
- *OWL DL* enthält OWL Lite, besitzt etwas mehr Ausdruckskraft und ist ebenfalls entscheidbar mit einer Zeitkomplexität von `NExpTime` im Worst-Case. OWL DL enthält die Ausdruckskraft von RDF(S) bis zu dem Grad, an dem die Sprache ansonsten unentscheidbar geworden wäre. Sie ist die am häufigsten verwendete Sprache zur Erstellung von Ontologien und besitzt eine gute Unterstützung durch Softwarewerkzeuge.
- *OWL Full* enthält OWL Lite und OWL DL. Es enthält RDF(S) als komplette Teilsprache und besitzt damit die größte Ausdruckskraft der drei Teilsprachen. Es enthält alle OWL-Sprachelemente und erlaubt, diese beliebig mit RDF(S)-Primitiven zu kombinieren und sogar deren Bedeutung zu verändern. Es ist ebenso keine Typentrennung zwischen Individuen, Klassen und Rollen nötig, was die Definition von Klassen ermöglicht und somit zur Meta-Modellierung verwendet werden kann. Dies führt dazu, dass die Sprache unentscheidbar wird und daher nur von sehr wenigen Softwarewerkzeugen teilweise unterstützt wird, da keine logische Inferenz mehr möglich ist sobald unentscheidbare Konstrukte auftauchen.

Nun soll ein kleiner Überblick lediglich über diejenigen Sprachelemente folgen, die tatsächlich auch zur Modellierung der Ontologien in Kapitel 4 verwendet wurden. Dazu soll die Klassenhierarchie aus Abbildung 2.3 zunächst einen Überblick gewähren, wie OWL und RDF(S) in Verbindung miteinander stehen. In der Abbildung stehen Rechtecke für Elemente aus OWL/RDF(S), wobei die genaue Herkunft anhand des Namensraums im Bezeichner

erkennbar ist. Die Pfeile stellen eine Generalisierungsbeziehung dar, wie sie beispielsweise aus der objektorientierten Programmierung bekannt ist. Zu Modellierung der Ontologien dieser Diplomarbeit wurde OWL DL verwendet.

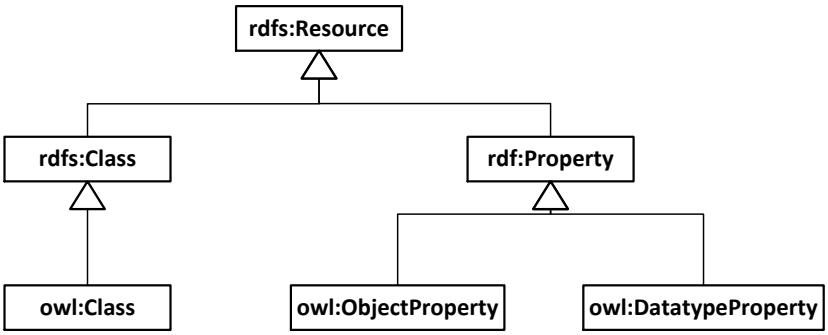

Abbildung 2.3: Klassenhierarchie der Beziehung zwischen OWL/RDF(S) (nach [AVH04, S. 119]). Legende: Rechtecke=*Elemente aus OWL/RDF(S)*, Pfeile=*Generalisierung*

Wie aus der Abbildung ersichtlich ist, wurde die RDF-Property in Object- und Data-Propertys verfeinert. *Object Propertys* verbinden dabei Objekte mit Objekten, und *Data Propertys* verbinden Objekte mit typisierten Datenwerten. In dem RDF-Beispiel aus Abbildung 2.2 wäre nun die Relation *HatAutor* eine Object Property, und *Titel* und *Name* wären Data Propertys. Es ist darüber hinaus möglich, die Eigenschaften einer Object Property anhand von sog. „Special Properties" festzulegen. Diese dienen zur Definition von Charakteristiken wie z.B. Transitivität oder Symmetrie. OWL bietet zur Typisierung von Data Propertys keine eigenen Datentypen, dafür aber standardmäßig eine Untermenge der XML Schema-Datentypen an. Eine weitere hinzugekommene Möglichkeit ist nun die Definition von mehreren Ranges oder Domains, was zu einer Bildung der Vereinigungsmenge der ausgewählten Klassen führt. In OWL gibt es zudem *Individuen*, welche Instanzen von Klassen klar als solche kennzeichnen. Wurde eine Object Property definiert, die zwei Klassen verbindet, bedeutet dies, dass Individuen der Klasse diese Beziehung aufweisen können, aber nicht müssen. Daher wird eine konkrete Beziehung zwischen Individuen in Form von *Object Property Assertions* ausgedrückt, welche aussagen, dass die Relation konkret zwischen den beiden Individuen gilt.

Logische Inferenzen in OWL

OWL DL lässt sich unter anderem auf eine Untermenge der Prädikatenlogik erster Stufe zurückführen und ermöglicht dadurch die Anwendung vieler gut erprobter Algorithmen aus diesem Bereich [HKRS08, S. 163]. Dadurch entfaltet sich ein großer Mehrnutzen aus der formalen Beschreibung und Klassifizierung einer Domäne anhand einer Ontologie in OWL. Aus der Ontologie lassen sich logische Schlussfolgerungen (*Inferenzen*) ziehen, die auf der einfachsten Ebene beispielsweise auf der Transitivität oder Symmetrie von Object Propertys basieren können.

Zu Veranschaulichung dieses Umstandes soll eine in OWL modellierte überarbeitete Version des obigen Beispiels aus Abbildung 2.2 verwendet werden, welche in Abbildung 2.4 dargestellt ist. Eine detaillierte Beschreibung aller Einzelheiten der Visualisierung einer OWL-Ontologie befindet sich in Abschnitt 2.3.5 und wird an dieser Stelle nicht für das Verständnis des Beispiels benötigt. Die beiden Beispielontologien unterscheiden sich insofern, dass nun statt Literalen zur Modellierung der Diplomarbeit und des Autors Individuen verwendet wurden. Zudem gibt es die bereits oben beschriebene Property-Hierarchie von beschäftigte-Personen mit den Unterpropertys *hatAutor* sowie *hatBetreuer* und die Klassenhierarchie zur Beschreibung der Diplomarbeit als studentische Arbeit. Die Object Property *hatAutor* ist nun asymmetrisch und funktional. *Funktionale* Object Propertys können nur genau auf ein Objekt zeigen, um den funktionalen Zusammenhang zwischen zwei Individuen zu verdeutlichen.

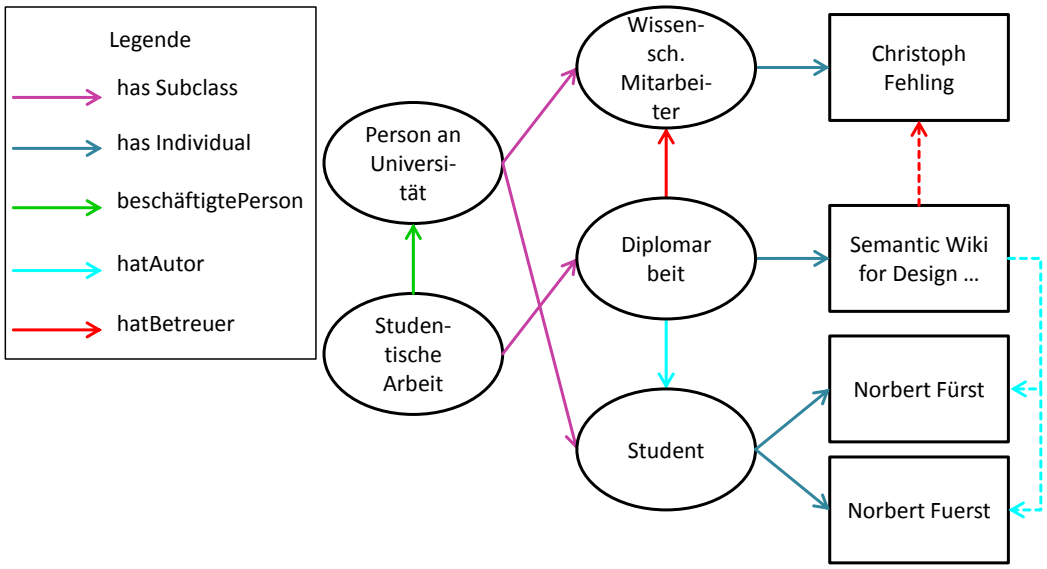

Abbildung 2.4: Beispiel einer einfachen OWL-Ontologie

Um implizites Wissen, welches in dieser Ontologie enthalten ist, explizit zu machen, gibt es semantische Inferenzprogramme (*Reasoner*), die logische Schlüsse aus explizit abgesicherten Aussagen (asserted statements) und Axiomen ziehen können. Ein solcher Reasoner könnte nun, auf die obige Ontologie angewandt, u.a. schlussfolgern, dass die *hatAutor*-Relation von Diplomarbeit zu den beiden Autoren eigentlich die gleiche Person meint. Dies lässt sich aus der Tatsache folgern, dass *hatAutor* eine funktionale Property ist, und dass es daher nur einen Autor geben kann. Aus der Asymmetrie dieser Property lässt sich außerdem folgern, dass es nie ein Tripel der Art `Student - hatAutor - Diplomarbeit` geben kann. Während diese Erkenntnisse innerhalb der kleinen Beispielontologie vielleicht trivial erscheinen, können durch Inferenzprogramme in Ontologien mit vielen tausend Begriffen durchaus interessante neue Erkenntnisse gewonnen werden.

Listing 2.1 Eine einfache SPARQL-Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?subj
WHERE
{
?subj rdf:type <http://iaas.de/Diplomarbeit> .
}
```

2.3.3 Abfragen mit SPARQL

SPARQL (*SPARQL Protocol And RDF Query Language*) ist eine vom W3C standardisierte Abfragesprache für Daten, die im RDF-Format vorliegen [SP]. Sie ermöglicht das Abfragen von Daten mit einem Resultat als RDF-Graph oder als Ergebnismenge sowie die Manipulation von RDF-Daten. Es spielt dabei auch keine Rolle, ob die Daten tatsächlich in RDF vorliegen oder zunächst von einer Middleware in RDF transformiert wurden. SPARQL stellt mittlerweile den De-Facto Standard für Abfragen im Semantic Web dar und basiert im Kern auf RDF-Anfragen in Form von Graphmustern [HKRS08, S. 202].

Im Folgenden soll die Query aus Abbildung 2.1 als Beispiel für die Erklärung der grundlegenden Aspekte von SPARQL dienen. Mithilfe des Schlüsselwortes `PREFIX` werden Namensräume definiert, um nicht immer den vollen Namensraum ausschreiben zu müssen. `SELECT` dient zur Auswahl der Ergebnisspalten, bzw. gibt an welche Teile des unter `WHERE` angegebenen Graphmusters schließlich als Ergebnis gezeigt werden sollen. Die in Abbildung 2.1 dargestellte Query würde demzufolge alle Individuen des Typs Diplomarbeit als Ergebnis anzeigen, wenn sie auf die OWL-Ontologie auf Abbildung 2.4 angewandt würde.

2.3.4 Triplestores

Triplestores oder RDF-Datenbanken sind spezielle Datenbanken zur Ablage und Abfrage von RDF-Daten im Tripel-Format [Bec02]. Im Vergleich zu klassischen relationalen Datenbanken, in welchen zusammenhängende Daten durch Mengen von Tupeln beschrieben wurden, handelt es sich bei RDF-Graphen um Mengen von Tripeln. Dabei ergeben sich insbesondere bei großen Datenmengen Anforderungen, die mit klassischen RDBMS direkt nicht erfüllt werden können und daher spezielle Optimierungen für häufig vorkommende Operationen auf RDF-Datenbeständen nötig machen. Es gibt zur Umsetzung von Triplestores verschiedene Ansätze, die jeweils Vor- und Nachteile mit sich bringen [LH05]. Triplestores, die auf Basis von RDBMS entwickelt wurden, bieten gute Inferenzmöglichkeiten, skalieren ab einer gewissen Größe des Datenbestandes aber nicht mehr gut. Native Triplestores skalieren besser bei großen Datenbeständen, allerdings bieten sie zum Zeitpunkt des Erscheinens von [LH05] weniger gute Inferenzmöglichkeiten. Zuletzt gibt es die Möglichkeit, die Daten In-Memory zu halten, was bis auf der großen Hauptspeicherbedarf den anderen Möglichkeiten überlegen ist. Da es im Rahmen dieser Diplomarbeit nur sehr kleine Datenbestände gibt und somit

die Begrenzung des Hauptspeichers kein Problem darstellt, wurde die In-Memory-Variante ausgewählt.

Triplestores können in Zusammenarbeit mit einem Reasoner die Mächtigkeit von SPARQL-Abfragen insofern erhöhen, dass bei der Berechnung der Ergebnismenge logisch hergeleitetes Wissen miteinbezogen wird. So ist es möglich, dass nicht explizit angegebenes Wissen mit in der Ergebnismenge enthalten ist, wie z.B. Beziehungen über transitive Propertys.

2.3.5 Ontologie-Modellierung mit Protégé

Protégé ist ein weit verbreiteter Open-Source Ontologie Editor, der von der Stanford University School of Medicine in Java entwickelt wurde [pro]. Er verfügt über zwei eingebaute Reasoner (FaCT++ und HermiT), mit welchen die modellierte Ontologie direkt auf logische Widersprüche überprüft werden kann und implizit enthaltene Aussagen materialisiert werden können. Ein SPARQL-Endpoint erlaubt direkte SPARQL-Anfragen auf eine Ontologie, die in Protégé geladen wurde. Ebenso sind zwei Visualisierungsplugins enthalten, welche die graphische Darstellung von OWL-Ontologien ermöglichen. Für die Darstellung von OWL-Ontologien in dieser Arbeit wurde jedoch die manuelle Visualisierung durch selbst erstellte Grafiken gewählt, um übersichtlichere Resultate zu erhalten, die nur benötigte Elemente abbilden. Der Visualisierungsstil von Klassen, Individuen und Propertys ist grob an das Protégé-Plugin OntoGraf[4] angelehnt. Zum Besseren Verständnis der Ontologie-Abbildungen folgt nun anhand von Abbildung 2.5 eine Erklärung, wie einzelne Elemente dargestellt werden.

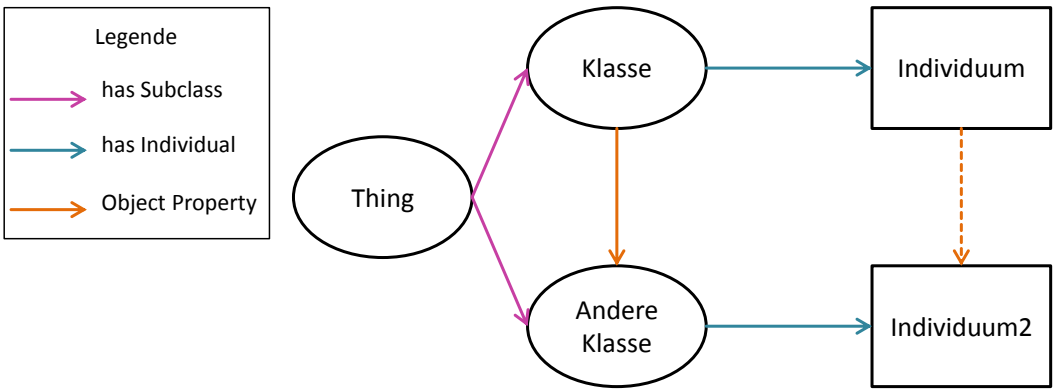

Abbildung 2.5: Die graphische Darstellung einer OWL-Ontologie

- **Klassen** werden als Ellipsen mit dem Klassennamen in der Mitte der Ellipse dargestellt. In Abbildung 2.5 sind *Thing*, *Klasse*, und *Andere Klasse* Klassen.

[4] http://protegewiki.stanford.edu/wiki/OntoGraf

- **Individuen** werden in Rechtecken mit dem Individuennamen in der Mitte des Rechtecks dargestellt. In Abbildung 2.5 sind *Individuum* und *Individuum2* Individuen.
- **Object Propertys** werden als farbige Pfeile dargestellt. Dabei entsprechen durchgezogene Pfeile zwischen Klassen Domain-Range Definitionen, und gestrichelte Pfeile zwischen Individuen Object Property Assertions. In der Abbildung gibt es eine Domain-Range Definition von Klasse nach AndereKlasse, was bedeutet, dass diese Object Property als Domain Klasse verwendet und als Range Andere Klasse. Von dem Individuum Individuum gibt es eine Object Property Assertion zu Individuum2, d.h. dass Individuum in der definierten Beziehung *Object Property* (siehe die Legende in Abbildung 2.5) zu Individuum2 steht.
- **Subklassen-** und **Klasse-Instanz-Beziehungen** werden ebenfalls als farbige durchgezogene Pfeile dargestellt. Die Beziehung *has Subclass* (deutsch: *hat Unterklasse*) ist in dem Beispiel dabei als pinker Pfeil dargestellt, und *has Individual* (deutsch: *hat Individuum*) als grünblauer Pfeil. Diese Farben werden in der kompletten Diplomarbeit konsistent verwendet, um ein schnelles Erfassen der Graphen zu ermöglichen.

2.3.6 Semantic Mediawiki

Semantic Mediawiki [KV] ist eine Erweiterung für Mediawiki [mwo], die die semantische Annotation von Seiten und deren Abfrage ermöglicht. Bevor auf die Besonderheiten von Semantic Mediawiki eingegangen wird, soll zunächst erklärt werden, aus welchen Grundbestandteilen Mediawiki besteht. Mediawiki ist das Open-Source Softwarepaket, mit welchem *wikipedia.org* betrieben wird, und basiert auf PHP und Javascript. Die grundsätzliche Bestandteile von Mediawiki, die zum Verständnis der Arbeit nötig sind, werden im Folgenden kurz erklärt:

- Seiten: Die Grundgranularität von Mediawiki stellen Seiten dar, auf welchen grundsätzlich jeder Benutzer Inhalte hinzufügen kann. Um eine gewisse Qualitätskontrolle zu ermöglichen, wird für jede Seite eine Versionsverwaltung des Inhalts geführt, so dass eine Seite auf frühere Stände zurückgesetzt werden kann. Seiteninhalte werden in der Auszeichnungssprache Wikitext (*Wiki markup*) verfasst, welches eine vereinfachte Form von HTML darstellt und eine grundlegende Textformatierung ohne HTML-Kenntnisse ermöglicht. Mithilfe von `[[Hyperlink|Interner Link]]` würde man z.B. einen Link auf die Wikiseite „Hyperlink“ verfassen, der als Linktext „Interner Link“ verwendet. `== Abschnitt ==` würde beispielsweise einen neuen Absatz mit der groß geschriebenen Überschrift „Abschnitt“ erzeugen.
- Namensräume (Namespaces): Namensräume dienen zur Abgrenzung von Seiten mit unterschiedlichen Verwendungszwecken. Dabei sollten Namensräume nicht mit Kategorien verwechselt werden, die zur inhaltlichen Abgrenzung von Artikeln dienen. Namensräume erkennt man daran, dass sie als Präfix vor dem eigentlichen Artikelnamen stehen, z.B. „Hilfe:Übersicht“.

- Kategorien: Die Kategorien in Mediawiki dienen dazu, Inhalte zu kategorisieren, strukturieren und somit leichter auffindbar und verständlich zu machen. Dabei ermöglicht es das Verfassen von Kategoriehierarchien Taxonomien abzubilden. Die Seiten, welche Kategorien definieren, befinden sich in dem eigenen Namensraum *Kategorie*. Zur Einordnung eines Artikels in eine Kategorie genügt es, `[[Kategorie:Beispiel]]` an eine beliebige Stelle des Artikels zu schreiben.
- Vorlagen (Templates): Vorlagen erlauben die Einbindung von vordefinierten Inhaltsbausteinen nach dem Transklusionsprinzip, d.h. durch die Angabe einer Referenz auf eine Vorlage wird der Inhalt der Vorlage an der entsprechenden Stelle eingefügt. Sie werden im eigenen Namensraum *Vorlage* definiert und bieten die Möglichkeit, benannte Parameter zu definieren. Auf diese Weise wird es möglich, standardisierte Vorlagen für gewisse Inhalte anzubieten, was ein einheitliches Layout von Artikeln ermöglicht. Ein Template kann durch `{{Templatename|Parameter1=a|Parameter2=b}}` auf einer Seite verwendet werden, wobei a und b als Eingabe für die vordefinierten Parameter verwendet würden.
- Spezialseiten: Der Namensraum *Spezial* wird hauptsächlich zu administrativen Zwecken genutzt und bietet die Möglichkeit, spezielle Funktionalitäten für Erweiterungen auf gesonderten Seiten anzubieten. Viele dieser Seiten sind nur für Benutzer mit Administrationsrechten verfügbar.

Im Folgenden sollen nun die wichtigsten Aspekte von Semantic Mediawiki näher erläutert werden. Dabei wird zunächst auf die Annotation von Artikeln mithilfe von semantischen Attributen eingegangen, anschließend auf die Abfrage von annotierten Inhalten und zuletzt auf in Seiten eingebettete Abfragen in Form von *Konzepten*.

Semantische Attribute

Semantische Attribute (*Semantic Propertys*) dienen zur Annotation von Artikeln mit semantischen Informationen. Als grundlegende Objekte muss man sich hier die Seiten vorstellen, welche mit OWL-Terminologie bezeichnet Individuen der Klasse *Seite* wären. Dabei gibt es sowohl die Möglichkeit, Relationen zwischen Seiten in der Entsprechung von OWL-Object Propertys auszudrücken, als auch Datenattribute pro Seite wie bei Data Propertys zuzuordnen. Hierbei wird bei Semantic Mediawiki allerdings immer von „Semantic Propertys“ gesprochen, die jedoch typisierbar sind. Ein einfacher semantischer Link zu einer anderen Seite wird erstellt, indem die beispielhafte semantische Annotation `[[semantischer Link::Andere Seite]]` auf einer Seite eingefügt wird. Damit wird das Tripel `Aktuelle Seite - semantischer Link - Andere Seite` geschaffen, sowie eine neue Seite im Namensraum *Attribut* (*Property*) mit dem Titel „semantischer Link“. Die Property *semantischer Link* hat dabei implizit den Typ *Seite* zugewiesen bekommen und wird daher als Object Property betrachtet. Sollte stattdessen beabsichtigt sein, dass es sich um eine Data Property handelt, versieht man die Seite der Property mit der Annotation `[[Hat Typ::String]]`. Dies hat zur

Folge, dass Andere Seite nun als semantisches Attribut vom Typ String betrachtet wird. Eine vollständige Liste der verfügbaren Datentypen ist unter [5] verfügbar.

Abfragen auf Attribute

Wurden in einem Wiki semantische Informationen annotiert, können diese über Querys abgefragt werden. Dafür gibt es eine spezielle Abfragesprache[6], die sich an der Notation des Wikitexts orientiert und so das Verfassen von Abfragen auch für technische Laien ermöglichen soll. In den Ergebnismengen dieser Querys entspricht grundsätzlich eine Reihe immer einer Seite im Wiki. Die Verwendung einer Abfrage kann entweder über die Seite „Spezial:Semantische_Suche“ geschehen, welche auch eine Unterstützung bei dem Verfassen von Querys bietet, oder über auf Seiten eingebettete Abfragen.

Die Abfrage `[[Category:Diplomarbeit]] [[hatAutor::+]] |?hatAutor=Autor |format=table` würde beispielsweise als Ergebnismenge alle Seiten der Kategorie *Diplomarbeit* liefern, für die ein beliebiger Autor annotiert wurde. Die Ergebnismenge würde als Tabelle dargestellt, in welcher die erste Spalte den Namen der Diplomarbeiten enthalten würde, und die zweite zu *Autor* umbenannte Spalte die Werte des semantischen Attributs *hatAutor*. Auf diese Weise lassen sich beliebig Werte von Propertys, Kategorien und Namensräumen abfragen, allerdings ist dabei immer das Subjekt der Tripelergebnismenge als Variable vorgegeben. Die Sprache unterstützt zudem Joins, Subquerys und Sortierungsoperationen auf der Ergebnismenge.

Konzepte

Die sog. *Konzepte*[7] (*Concepts*) bezeichnen Seiten im dem Namensraum *Konzept* und bieten die Möglichkeit, Querys zu speichern. Ein Konzept entspricht einer Query im Semantic Mediawiki-Format, und kann als eine Art dynamische Kategorie betrachtet werden. Auf einer Konzeptseite kann jeweils nur eine Abfrage gespeichert sein, im Gegensatz zu auf normalen Seiten eingebetteten Abfragen, von welchen beliebig viele auf einer Seite vorkommen können.

2.3.7 Erweiterungen von Semantic Mediawiki

Durch die modulare Architektur von Semantic Mediawiki gibt es eine große Entwicklergemeinde, die aktiv an Erweiterungen arbeitet. Die wichtigsten für diese Diplomarbeiten verwendeten Erweiterungen sollen im Anschluss kurz vorgestellt werden.

[5] http://semantic-mediawiki.org/wiki/Help:Properties_and_types
[6] http://semantic-mediawiki.org/wiki/Help:Selecting_pages
[7] http://semantic-mediawiki.org/wiki/Help:Concepts

DataWiki

Das Produkt DataWiki[8] der Firma DIQA[9], das ursprünglich unter dem Namen SMW+ entwickelt wurde, ist ein „semantisches Unternehmenswiki“. Es bereichert Semantic Mediawiki um eine Sammlung von Erweiterungen, die viele Änderungen und Verbesserungen an der Benutzeroberfläche und der Benutzerfreundlichkeit mitbringen. Der Kern von DataWiki stellt die *Halo Extension*[10] dar, welche unter anderem folgende Features liefert:

- Anbindung eines Triplestores: Ebenfalls von DIQA gibt es das Produkt *Triplestore Basic*[11], welches eine Anbindung von Semantic Mediawiki an den Jena In-Memory-Triplestore ermöglicht. Jena ist eines der führenden Frameworks zur Entwicklung von Semantic Web Anwendungen und ermöglicht die Verwendung von unterschiedlichen Reasonern [WSK+03]. Zudem bietet Triplestore Basic einen SPARQL-Web Endpoint an, was dadurch die Möglichkeit bietet, SPARQL-Querys auf dem Wiki-Datenbestand auszuführen und dabei die Unterstützung eines Reasoners zu verwenden.
- Erleichterte Annotation: Mithilfe der *Semantic Toolbar* wird die Annotation von Artikeln beispielsweise durch Autovervollständigung von Namen vereinfacht.
- WYSIWYG-Editor: Mithilfe des mitgelieferten WYSIWYG-Editors ist es möglich, Artikel ohne Kenntnisse von Wiki markup zu verfassen und dabei semantische Annotationen und Kategoriezuweisungen ebenfalls ohne Kenntnisse der Syntax einzufügen.
- Ontologiebrowser: Mithilfe dieser Spezialseite ist es möglich, die dem Wiki zugrunde liegende Ontologie zu visualisieren und effizient neue Propertys oder Kategorien anzulegen.

Neben der Erweiterung Halo sei ebenfalls die *Deployment Framework*-Erweiterung als Teil von DataWiki genannt. Sie ermöglicht die komfortable Administration des Wikis über ein Webinterface sowie das Verwalten von Erweiterungen auf Basis eines Repositorys.

Semantic Forms

Die Erweiterung *Semantic Forms*[12] erlaubt das Erstellen von auf Templates basierenden, bereits semantisch annotierten Seiten mithilfe von benutzerfreundlichen Formularen. Sie wurde hauptsächlich von Yaron Koren und Stephan Gambke entwickelt.

Dabei werden Assistenten zum Erzeugen von semantischen Propertys, Templates und Formularen geliefert, so dass das Entwerfen von Formularen sehr benutzerfreundlich geschehen kann. Bei dem Erstellen von Templates ist bereits die Angabe eines bestimmten Feldes möglich, dass verwendet werden soll, um einer semantischen Property einen Wert zuzuweisen.

[8] http://diqa-pm.com/de/DataWiki
[9] http://diqa-pm.com
[10] http://semanticweb.org/wiki/Halo_Extension
[11] http://diqa-pm.com/de/Triplestore_basic
[12] http://www.mediawiki.org/wiki/Extension:Semantic_Forms

Beim Erzeugen des Formulars werden nach Angabe der Templates, welche verwendet werden sollen, automatisch für alle erkannten Felder passende Eingabetypen vorgeschlagen und die Möglichkeit geboten, ein Label für jedes Feld zu bestimmen. Die benötigten Parameter zur Feinkonfiguration eines Eingabetyps können schließlich in einem ausklappbaren Fenster eingegeben werden. Semantic Forms bietet zudem die Möglichkeit, die erstellten Formulare lückenlos in ein Wiki zu integrieren. Dabei lassen sich zum einen ganze Namensräume einem gewissen Formular zuordnen, zum anderen auch einzelne Seiten durch die Annotation mit der semantischen Property `Page has default form`. Das zugeordnete Formular wird anschließend geöffnet, sobald der Benutzer den Editieren-Button verwendet.

Besonders erwähnenswert ist auch die offene modulare Architektur von Semantic Forms, welche die Entwicklung von zusätzlichen Eingabetypen sehr komfortabel erlaubt. Dies hat dazu geführt, dass es bereits ganze Sammlungen von zusätzlichen Eingabetypen gibt, wie z.B. in Form der Erweiterung *Semantic Forms Inputs*[13].

Category Tree

Die Erweiterung *Category Tree*[14] von Daniel Kinzler erlaubt es, die Struktur der Kategorien innerhalb des Wikis als Baum darzustellen.

Dabei muss zur Anzeige eines solchen Baumes lediglich ein Ausdruck der Art `<categorytree>Wurzelkategorie</categorytree>` auf einer Wikiseite eingefügt werden. Anschließend wird je nach ausgewähltem Modus nur die Unterkategorien der Wurzelkategorie, oder die Unterkategorien sowie deren Seiten angezeigt. Eine Unterkategorie entsteht, wenn die Seite, welche die Kategorie definiert, selbst mit einem Kategorie-Tag versehen wird, und somit in eine Überkategorie eingeordnet wird.

GraphViz

Die Erweiterung *GraphViz*[15] von Daniel Kinzler dient dazu, Graphen als eingebettete Grafiken im Wiki darzustellen. Dafür stehen die beiden Grapherzeugungsprogramme GraphViz [Per] und Mscgen[16] zur Verfügung. Mscgen dient zur Erzeugung von *Message Sequence Charts* (Nachrichten-Reihenfolge-Diagramme) und wurde im Rahmen der Diplomarbeit nicht verwendet.

Das Open-Source Programmpaket GraphViz ist in der Lage, aus Graphbeschreibungen im DOT-Format eine Vielzahl von unterschiedlichen Graphtypen zu erzeugen. DOT ist eine einfache Graphbeschreibungssprache, mit der in Klartext für Menschen und Maschinen lesbare Beschreibungen von Graphen verfasst werden können. Solche Graphbeschreibungen

[13] http://www.mediawiki.org/wiki/Extension:Semantic_Forms_Inputs
[14] http://www.mediawiki.org/wiki/Extension:CategoryTree
[15] http://www.mediawiki.org/wiki/GraphViz
[16] http://www.mcternan.me.uk/mscgen/

können mit der Mediawiki GraphViz-Erweiterung direkt auf Wikiseiten eingefügt werden. Im Rahmen der Erweiterung wird die Beschreibung mithilfe der auf dem Server vorhandenen GraphViz-Installation in ein Bild umgewandelt, welches anstatt des Textes auf der Seite erscheint. Die Möglichkeiten, die GraphViz dabei zur Gestaltung der Graphen bietet, sind sehr vielseitig. Neben der Bestimmung von Knotenfarben und -formen können Kanten unterschiedlich dargestellt und beschriftet werden, und zudem können in Knoten auch Links eingebettet werden.

Semantic Drilldown

Die Erweiterung *Semantic Drilldown*[17] bietet eine Spezialseite, auf der anhand von Kategorien und Filtern Drilldown-Operationen auf den Daten eines semantischen Wikis ausgeführt werden können. Dabei liegt der Fokus auf Filter, welche anhand von semantischen Propertys definiert werden. Die Erweiterung wird von Yaron Koren und David Loomer entwickelt.

Zu Beginn einer Drilldown-Operation auf der dafür bereit gestellten Spezialseite stehen alle Wurzelkategorien zur Verfügung, d.h. alle Kategorien, die selbst keine Unterkategorie einer anderen Kategorie sind. Wurde nun eine Wurzelkategorie ausgewählt, stehen alle Unterkategorien und für diese Kategorie definierten Filter als Drilldown-Merkmal zur Verfügung. Solange kein Drilldown-Merkmal ausgewählt ist, werden alle Seiten der Wurzelkategorie alphabetisch sortiert als Ergebnismenge angezeigt. Jede Auswahl eines zusätzlichen Drilldown-Merkmals schränkt die angezeigte Seitenmenge weiter ein. Wird beispielsweise eine Unterkategorie ausgewählt, werden anschließend nur noch Seiten der Unterkategorie angezeigt. Die bereits erwähnten Filter werden als Seiten des Namensraums *Filter* definiert und ermöglichen es, unterschiedliche Filterkriterien anzubieten. Eine Möglichkeit besteht beispielsweise darin, einen Filter mit einer semantischen Property zu verknüpfen. Zur Auswahl als Drilldown-Merkmal stehen nun alle Werte zur Verfügung, die das semantische Attribut im Wiki zugewiesen bekommen hat. Wird ein Wert ausgewählt, erscheinen nur noch Seiten, auf welchen die Property mit genau demselben Wert annotiert wurde. Ein andere Möglichkeit, einen Filter zu definieren, ist die feste Vorgabe von Werten, die zur Auswahl angezeigt werden sollen. Ein weiteres Feature von Semantic Drilldown besteht darin, bestimmte Filter erst anzuzeigen, wenn für einen anderen vorher definierten Filter bereits ein Wert ausgewählt wurde.

Nachdem nun die nötigen Grundlagen für das Verständnis sowohl der inhaltlichen als auch der technischen Aspekte dieser Diplomarbeit geschaffen wurden, folgt im nächsten Kapitel die Aufnahme der genauen Anforderungen an ein *Pattern Repository*. Dies ermöglicht eine Implementierung, die genau den festgehaltenen Zielen entspricht und anschließend auch eine exakte Evaluation, inwiefern die Ziele erreicht wurden.

[17] http://www.mediawiki.org/wiki/Extension:Semantic_Drilldown

3 Anforderungen

Im Folgenden Kapitel sollen zunächst in Abschnitt 3.1 die funktionalen, und in Abschnitt 3.2 die nichtfunktionalen Anforderungen an das Pattern Repository festgehalten werden, das im Rahmen dieser Diplomarbeit entwickelt werden soll. Nach der Beschreibung der relevanten Use Cases in Abschnitt 3.3 sollen in Abschnitt 3.4 die Vorteile der Verwendung semantischer Technologie für die Umsetzung vorgestellt werden. Zuletzt folgt in Abschnitt 3.5 eine Evaluation verschiedener Produkte, und eine Begründung der Entscheidung zugunsten von DataWiki.

3.1 Funktionale Anforderungen

In diesem Abschnitt sollen die funktionalen Anforderungen an das Projekt in Form von User Stories festgehalten werden, wie es beispielsweise in [LL07] beschrieben ist. Die funktionalen Anforderungen wurden aus unterschiedlichen Quellen erarbeitet, die sich mit der Beschreibung von Anforderungen an ein Pattern Repository beschäftigen. Weiss *et al.* beschreibt in [WB07] Anforderungen an ein auf einem Wiki basierenden Pattern Repository, welche sich aufgrund der großen Ähnlichkeit der Zielsetzung sehr gut zur Verwendung in dieser Arbeit eignen. Dabei werden Benutzer anhand ihrer Rollen in Pattern Autoren und Endnutzer unterschieden. Im Kern sollen Autoren in der Lage sein, Patterns und deren Beziehungen abzulegen. Dabei sollen es möglich sein, einzelne Pattern-Domänen anhand ihrer Patternsprache als Sammlungen zu speichern. Sowohl Endnutzer als auch Autoren können Tags an Patterns anbringen, um das spätere Auffinden zu erleichtern. Zuletzt soll zur einfacheren Suche die Möglichkeit bestehen, Metadaten über Patterns zu speichern und sie anschließend darauf basierend zu suchen.

Neben diesen Grundanforderungen wurden in [DKT05] eine Reihe Benutzerinterface Pattern Sammlungen verglichen und einige detailliertere Anforderungen an Pattern Repositorys aus diesem Bereich gesammelt. Sehr ausführlich wurde hier auf die Spezifikationen von Greene *et al.* [GMJ+03] und Gaffar *et al.* [GSJS03] eingegangen. Die Anforderungen sind in den drei Quellen nicht spezifisch auf diese Domäne zugeschnitten und können daher ebenfalls im Rahmen dieser Arbeit verwendet werden. Weitere Anforderungen konnten von Buschmann *et al.* übernommen werden, welcher die Anforderungen an ein Mustersystem für Softwarearchitektur beschreibt [BHS07, S. 360-361]. Diese sind bereits sehr allgemein gehalten, so dass sie sich sehr gut für die Verwendung in dieser Arbeit eignen.

Im Folgenden wurde die teils groben Anforderungen aus den unterschiedlichen Quellen in Form von User Stories ausformuliert und nach den Benutzerrollen angeordnet, die sie

betreffen. Die Benutzerrollen wurden aus [WB07] übernommen. Im Einzelnen ist angemerkt, aus welcher Quelle welche Anforderung übernommen wurde, wobei User Stories ohne Quellen aus der Zielsetzung dieser Arbeit abgeleitet wurden.

Pattern Autor

- Als Pattern Autor möchte ich die Struktur der Patterns im System passend zu meiner Patterndomäne konfigurieren können, so dass sie anschließend im kompletten System konsistent eingehalten wird. [DKT05, „Standardizing a Common Pattern Form“, S. 36], [Bus98, S. 361]
- Als Pattern Autor möchte ich neue Patterns mithilfe eines komfortablen Formulars hinzufügen können.
- Als Pattern Autor möchte ich vorhandene Patterns mit dem gleichen Formular editieren können, mit dem ich sie angelegt habe.
- Als Pattern Autor möchte ich Patterns vernetzten können, indem ich die Beziehungen angebe, welche die Patterns verbinden. [DKT05, „Relating Patterns“, S. 36], [Bus98, S. 361], [GSJS03]
- Als Pattern Autor möchte ich die semantische Annotation von Patterns auf WYSIWYG-Weise vornehmen können, und dabei von der verwendeten Software unterstützt werden.
- Als Pattern Autor möchte ich Patterns mit Zieleigenschaften aus einer Ontologie annotieren können, um später anhand dieser Zieleigenschaften filtern zu können. Unter dem Stichwort „Manipulating Forces“ in [DKT05, S. 36] wird die Anforderung beschrieben, dass Auswahl von Patterns anhand der Angabe von *Forces* möglich sein soll, welche sich teilweise mit Zieleigenschaften überschneiden.
- Als Pattern Autor möchte ich die Möglichkeit haben, Patterns mit den Anwendungsfällen zu vernetzen, aus welchen sie abstrahiert wurden. Dies ermöglicht die transparentere Diskussion über Patterns, da die Quellen nicht verloren gehen, aus welchen sie abstrahiert wurden. Die Vorteile von mehr Transparenz bezüglich der Patternquellen werden z.B. in [FEL+12] beschrieben.
- Als Pattern Autor möchte ich die Beziehungen zwischen Patterns visualisieren. [Sch03]
- Als Pattern Autor möchte ich die Möglichkeit der Versionierung von Patterns haben, um einen Überblick über den Entstehungsprozess mit zu dokumentieren. [DKT05, „Versioning Patterns“, S. 36]
- Als Pattern Autor möchte ich ein Pattern Repository, welches interoperabel mit anderen Repositorys eingesetzt werden kann. [Bir10, HL93, DKT05]

Endnutzer

- Als Endnutzer möchte ich ein Inhaltsverzeichnis der Patterns haben, aus dem ich komfortabel durch den Patternfundus navigieren kann. [GMJ+03, „Browsing“]
- Als Endnutzer möchte ich fortgeschrittene Suchmöglichkeiten wie z.B. eine Querysprache zur Verfügung haben, um Patterns zu finden, welche sehr spezifische Suchkriterien erfüllen. Diese Anforderung resultiert aus den in [Bir10] beschriebenen Schwierigkeiten bei der Patternsuche. In [GASB11] ist im Rahmen von Richtlinien für die Entwicklung eines Repositorys eine effiziente Suchfunktion als zentrale Anforderung angegeben.
- Als Endnutzer möchte ich eine oder mehrere Zieleigenschaften auswählen können, um eine Liste zu erhalten, in der die Patterns anhand ihrer Korrelationswerte zu dieser Zieleigenschaft sortiert sind. [GMJ+03, „Pattern ranking“]
- Als Endnutzer möchte ich direkt die Art von Beziehungen sehen können, die ein Pattern zu anderen Patterns hat. [Sch03]
- Als Endnutzer möchte ich von einem Ausgangspattern aus andere Patterns vorgeschlagen bekommen, die man gut mit dem Ausgangspattern kombinieren kann. Gleichzeitig möchte ich eine Anzeige von Alternativen, die ich verwenden kann, falls ich das Ausgangspattern nicht verwenden möchte. In [GMJ+03] wird unter dem Stichwort „Browsing“ zusammengefasst beschrieben, dass es unterschiedliche Möglichkeiten geben sollte, zwischen den Patterns zu navigieren.
- Als Endnutzer möchte ich die Möglichkeit eines einfachen Einstiegs in eine Patterndomäne haben, indem mir anhand von Pattern-User Stories Vorschläge gemacht werden, wo ich mit dem Lesen beginnen soll. Dieses Vorgehen ist in [JGVH95, S. 10] ,[FLR+13, S. 14-20] und [HW03, S. xlviii-xlix] zu beobachten und kann daher als Anforderung an ein Pattern Repository formuliert werden.
- Als Endnutzer möchte ich einen Assistenten zur Verfügung haben, der mir verschiedene Möglichkeiten vorschlägt, die dazu dienen, mich effizient mit der Patterndomäne zu beschäftigen. [Bir10, S. 8f]
- Als Endnutzer möchte ich ein tieferes Verständnis über die Patternsprache der entsprechenden Domäne gewinnen, wobei mir Visualisierungen helfen könnten. In [Sch03] wird erläutert, dass unterschiedliche Visualisierungsmöglichkeiten der Patternsprache eine wichtige Anforderung an ein Pattern Repository sind.
- Als Endnutzer möchte ich für jede Kategorie von Patterns innerhalb einer Domäne eine Übersichtsgrafik haben, anhand der ich durch die Inhalte der Kategorie navigieren kann.
- Als Endnutzer möchte ich einen Anwendungsfall auswählen, an dem ich besonderes Interesse habe, und mir zu diesem passende Patterns anzeigen lassen. [FEL+12]

3.2 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen an das Projekt sollen nach der Gliederung aus [Hoe] festgehalten werden. Sie wurden teilweise aus allgemeinen Anforderungen an Repositorys aus [HL93] übernommen, und teils für die Zielsetzung der Diplomarbeit entwickelt. Als Grundlage zur „Entwicklung“ der nichtfunktionalen Anforderungen wurde die in der Norm ISO25010 beschriebenen nichtfunktionalen Softwarequalitäten verwendet [ISO10].

Im Bereich der Produktqualität wird eine hohe *Usability* [ISO10] angestrebt, um auf Akzeptanz in der Pattern Community zu stoßen. Dies soll vor allem durch eine komfortable Bedienung der Software erreicht werden, wozu auch ein immer flüssig bleibendes Benutzerinterface der Software gehört. Zur Gewährleistung der *Portabilität* [ISO10] sollte eine Bindung an ein bestimmtes Betriebssystem möglichst vermieden werden, was auch bei der Verbreitung des Produkts helfen kann. Die *Plattformunabhängigkeit* ist eine Anforderung, die allgemein für Repositorys gilt [HL93].

Eine allgemeine Anforderung an den Entwicklungsprozess ist die Verwendung von Methoden der agilen Softwareentwicklung, um verwendbare Zwischenstände der Software zu Präsentationszwecken zur Verfügung zu haben. Auf diese Weise ist jederzeit ein Überblick über den Status der Diplomarbeit möglich. Während des Entwicklungsprozess soll bereits eine gute Dokumentation sowohl in Form eines Handbuchs als auch innerhalb des Codes entstehen, um so die *Wartbarkeit* und die *Erweiterbarkeit* [ISO10] der entwickelten Software zu gewährleisten.

Eine spezielle Anforderung, die sich aus der Verwendung von den Patterns aus [FLR+13] als Testdatensatz ergibt, ist die Möglichkeit der Zugriffsbeschränkung. Um rechtliche Probleme zu vermeiden, soll es zudem eine Möglichkeit geben, Teile des Datenbestandes vor der Öffentlichkeit zu verbergen. Diese speziellen Anforderungen lassen sich unter dem Bereich *Sicherheit* [ISO10] zusammenfassen.

3.3 Use Cases

Nachdem nun in den vorherigen Abschnitten die genauen Anforderungen an das zu entwickelnde System festgehalten wurden, folgt in diesem Abschnitt eine weitere Konkretisierung anhand von Use Cases. In Abbildung 3.1 ist ein UML-Use Case Diagramm der zu entwickelnden Software in UML-Notation abgebildet. Das System setzt sich aus drei großen Bestandteilen zusammen, welche im Rahmen eines Arbeitsflusses nacheinander verwendet werden und letztendlich das Pattern Repository hervorbringen. Die Teile des Systems sind jeweils durch einen großen Rahmen in der Abbildung dargestellt, der oben in der Mitte mit der Bezeichnung des jeweiligen Teilsystems beschriftet ist. Dabei werden die möglichen Benutzer des Systems in die Benutzerrollen technischer Administrator, Pattern Autor und Endnutzer eingeteilt.

Zunächst passt der technische Administrator das Datenmodell des Repositorys an das spezifische Patternformat an, welches verwendet werden soll, sowie die erlaubten Relationen

zwischen den Patterns. Dazu wird ein Ontologie-Editor verwendet, der auch dazu dient,

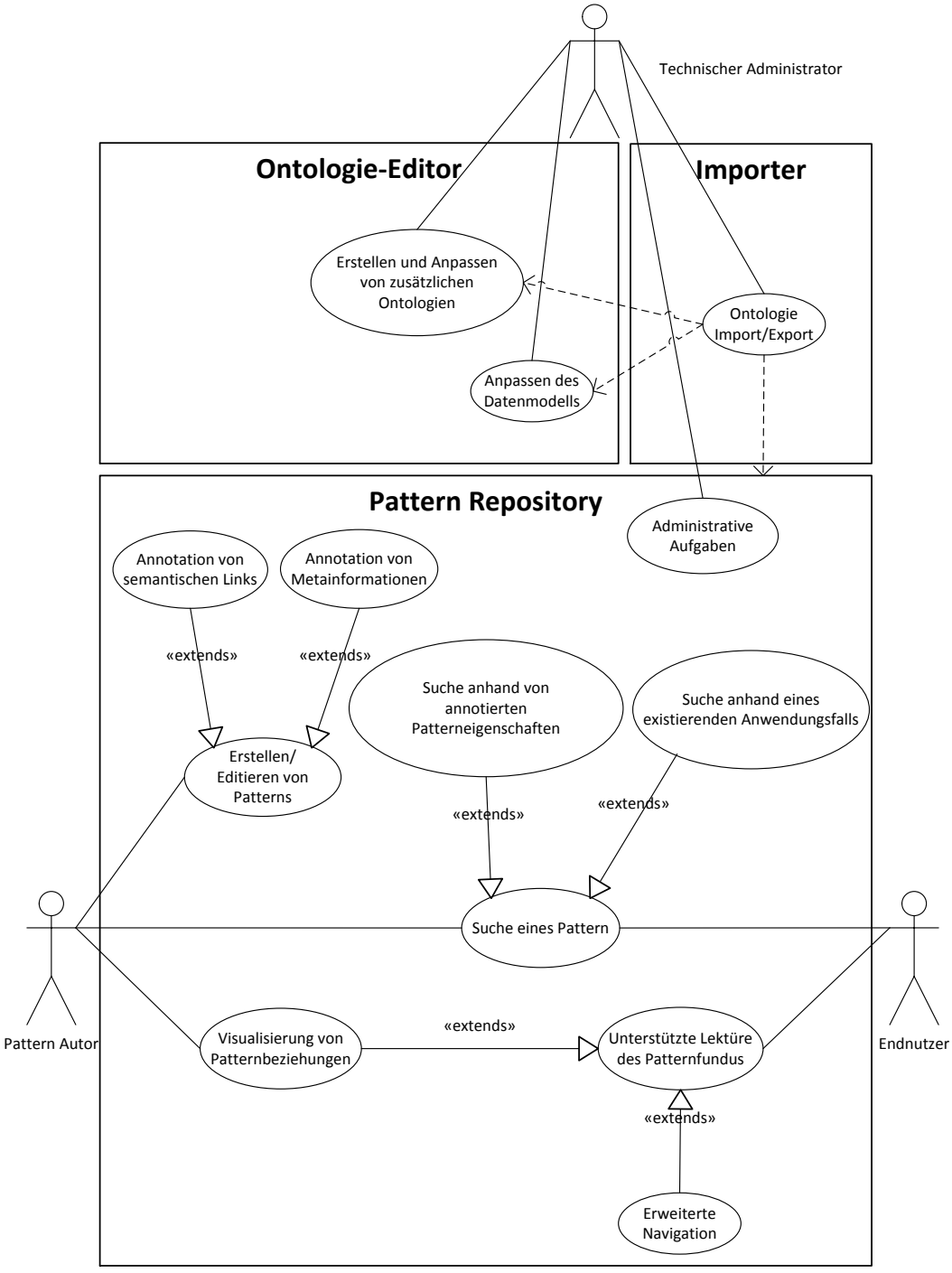

Abbildung 3.1: Use Cases des Pattern Repository

die in Abschnitt 3.1 erwähnten Zieleigenschaften für die entsprechende Patterndomäne anzupassen. Darauf folgt der Import dieser Ontologien in das eigentliche Pattern Repository mithilfe eines Importers, der das modellierte Datenmodell und sonstige Informationen auf das Datenmodell des Pattern Repositorys abbildet. Dieser Import ist in der Abbildung anhand von gestrichelten Linien angedeutet.

Nach der Übertragung des Datenmodells kann das Repository schließlich von Pattern Autoren und Endnutzern verwendet werden. Nachdem der Pattern Autor erste Patterns erstellt hat, kann er diese Annotieren und mithilfe von semantischen Links die Beziehungen zwischen den Patterns ausdrücken. Diese erweiterten Funktionalitäten wurden in dem Diagramm aus Abbildung 3.1 anhand der UML-Beziehung *extends* modelliert, welche für die Ableitung von spezifischeren Funktionalitäten basierend auf einer Basisfunktionalität verwendet wird. So stellt die Annotation von semantischen Links beispielsweise eine Spezialisierung des Editierens eines Patterns dar. Neben den Beziehungen können weitere Metainformationen, wie z.B. die oben erwähnten Zieleigenschaften, annotiert werden. Die Suche von Patterns wird sowohl von Pattern Autoren, als auch von Endnutzern verwendet. Als Spezialisierungen der Suche können annotierte Patterneigenschaften in den Suchkriterien angegeben werden, außerdem sind auch Suchen anhand von existierenden Anwendungsfällen möglich. Anhand dieser Suche können Patterns durch Anwendungsfälle gefunden werden, aus welchen sie abstrahiert wurden.

Für den Endnutzer ist insbesondere die Lektüre der vorhandenen Patterns wichtig, wofür unter der in der Grafik vorkommenden *Erweiterten Navigation* beispielsweise Vorschläge zählen, die durch einen Assistenten gemacht werden. Die Visualisierung von Patternbeziehungen kann nicht nur dem Pattern Autor helfen, seine Patternsprache besser zu entwerfen, sondern auch dem Endnutzer, um die Patternsprache besser zu verstehen. Detailliertere Beschreibungen der hier kurz angeschnittenen Zusammenhänge befinden sich in den Kapiteln 4 und 5.

3.4 Gründe für Verwendung semantischer Technologie

Im vorherigen Abschnitt wurden die Use Cases konkretisiert, die für das im Rahmen dieser Diplomarbeit zu entwickelnden Systems relevant sind. Nun soll eine Begründung folgen, weshalb es bei der Umsetzung des Systems von Vorteil wäre, semantische Technologien zu verwenden. Momentan verwenden viele Pattern-Autoren für die Veröffentlichung ihrer Patterns entweder nur ein Buch, oder begleitend eine Webseite (siehe [OOD], [HW03], [JGVH95]). Hierbei bleibt im Fall eines Buches dem Leser nur das zeitraubende Blättern im Buch, um von einem Pattern zu einem anderen zu springen, oder bei einer Webseite das komfortablere Verwenden von einfachen Links. Die Verwendung von semantischen Technologien ermöglicht es einem Pattern Autor, durch die Vernetzung der Patterns anhand von semantischen Linktypen zusätzliche Informationen über die Beziehungen zwischen Patterns anzugeben. Beispielsweise ermöglicht dies eine formale Beschreibung der Patternsprache (siehe Abschnitt 2.2) aufgrund der Relationstypen. Durch diese zusätzlichen semantischen

Informationen, die man an ein Pattern anhängen kann, sind nun erst die oben aufgezählten funktionalen Anforderungen an ein Pattern Repository erfüllbar.

Die semantischen Annotationen ermöglichen eine Führung des Endnutzers durch das Pattern Repository, indem er z.B. auf einen Blick sieht, dass aufgrund eines gewissen Linktyps auf ein anderes Pattern sich dieses Pattern gut mit diesem kombinieren ließe. Dazu lässt sich dynamisch über verschiedene Visualisierungen die Patternsprache darstellen, und eine interaktive Navigation ermöglichen. Auch der Einstieg in eine komplette neue Patterndomäne kann durch einen semantisch annotierten Patternfundus um ein vielfaches erleichtert werden. In gewissen Fällen hat der Endnutzer, wenn er ein Pattern Repository verwendet, bereits eine konkretes Problem im Hinterkopf, dass er mithilfe der Patterns lösen möchte. Wurden die Patterns zuvor vom Pattern Autor diesbezüglich annotiert, eröffnet sich die Möglichkeit, den Benutzer ein gewisses Problem auswählen zu lassen, zu dem er geeignete Patterns angezeigt bekommt.

Diese Beispiele für neue Anwendungsfälle zeigen, dass sich mithilfe von semantischen Annotationen ein großer Mehrwert für ein Pattern Repository ergeben kann. Während der tatsächlich erreichte Mehrwert direkt von der Qualität der Annotationen abhängt, ergeben sich stark erweiterte Abfragemöglichkeiten durch die Verwendung der Querysprache SPARQL (siehe Abschnitt 2.3.3).

3.5 Evaluation von vorhandenen semantischen Wikianwendungen

Nach der im vorherigen Abschnitt erläuterten Entscheidung, semantische Technologien für die Umsetzung des Pattern Repository zu verwenden, war der nächste Schritt die Auswahl eines bereits existierenden Produkts, welches möglichst vollständig den zuvor festgehaltenen Anforderungen entspricht. Aufgrund der speziellen Anforderungen an das Pattern Repository war von Anfang an auch die Erweiterbarkeit bzw. Anpassbarkeit ein wichtiges Merkmal des zu wählenden Basisprodukts. Bei der konkreten Auswahl gab es nun auf der einen Seite eine Reihe von semantischen Wikis, welche die Stärken von klassischen Wikis wie die einfache Verwendbarkeit, flexible Zusammenarbeit von Benutzern und Verlinken von Artikeln mit den Fähigkeiten des semantischen Webs wie strukturierte Inhalte, Wissensmodelle in Form von Ontologien und Reasoning vereinen [SBBK08]. Auf der anderen Seite gibt es eine Reihe von sehr ausgereiften Produkten aus dem Bereich des allgemeinen Content bzw. Knowledge Managements, welche ähnliche Grundfunktionalitäten wie Wikis aufweisen und ebenfalls immer mehr mit semantischen Funktionalitäten ausgerüstet werden.

Der erste Schritt stellte die Auswahl einer Menge von Kandidaten aus den vielen existierenden Produkten sowohl aus dem Semantic Wiki Umfeld, als auch aus dem Bereich der allgemeinen Knowledge Management Produkten, dar. Im Semantic Wiki Umfeld gab es eine Reihe von vielversprechenden Forschungsprojekten und Prototypen [SBBK08, VK06], die nicht mehr weiterentwickelt werden:

- IkeWiki [Sch06]: Entwicklung eingestellt zugunsten von KIWI, welches ebenfalls bereits seit dem Jahr 2010 inaktiv ist. Hätte die semantische Annotation anhand einer benutzerfreundlichen graphischen Oberfläche direkt im Text ermöglicht. Volle Reasoning-Unterstützung mit Ontologien im OWL-RDF und OWL DL-Format inklusive Import von OWL-Ontologien.
- WikSar [AA05]: Entwicklung eingestellt. War ein direkter Konkurrent von Semantic Mediawiki mit vielen ähnlichen Konzepten.
- SweetWiki [BGE+08]: Entwicklung eingestellt. Hatte den Fokus insbesondere auf WYSIWYG gelegt.
- Kaukulo [Dja05]: Scheinbar eingestellt, keine Updates seit 2007. Basierend auf JSP-Wiki, Annotation von belieben Teilen einer Seite statt Annotation, die sich auf die ganze Seite beziehen. Import von Ontologien und Reasoning-Unterstützung.

Eine Übersicht über den Status einzelner Projekte, welche auch zum Erstellen dieser Liste benutzt wurde, bietet [Sem]. Da es auch tatsächlich noch aktive Projekte gibt, die Semantic Wikis entwickeln, wurden die inaktiven Projekte zu Beginn von der Auswahl ausgeschlossen.

- Semantic Mediawiki [KV]: Open-Source Erweiterung zu dem bekannten Framework Mediawiki, welches als Grundlage für *wikipedia.org* dient. Benutzt eine Ontologie als Datenmodell, aber keine direkte Unterstützung für den Import von OWL Ontologien. Die große Entwicklergemeinde hat bereits viele Erweiterung für die unterschiedlichsten Anwendungsfälle mit zusätzlichen semantischen Daten entwickelt, und durch die Möglichkeit, einen Triple Store anzuschließen, volle Reasoning-Fähigkeiten.
- zAgile Wikidsmart [zAg]: Proprietäre Erweiterung zu Atlassian Confluence mit Ontolgie als Datenmodell, Import von OWL-Ontologien und voller Reasoning Unterstützung.
- Tiki Wiki CMS Groupware [tik]: Wikibasiertes Open-Source Content Management System, welches seit Version 3.0 typisierte semantische Links zwischen Artikel unterstützt. Keine Ontologie als zugrunde liegendes Datenmodell, dementsprechend auch keinen OWL Import oder Reasoning-Unterstützung.

Aus dieser Liste wurden für den finalen Vergleich letztendlich Semantic Mediawiki und Wikidsmart ausgewählt, da bei Tiki Wiki der Fokus nicht auf den semantischen Features liegt, und es auch von der Tiki Wiki Community keinerlei „Mods“ für die Ausnutzung von semantischen Daten gibt[1]. Semantic Mediawiki dagegen bietet eine vielfältige Auswahl von Erweiterungen und lässt sich durch die große Flexibilität des Mediawiki-Frameworks an alle Anforderungen anpassen. Wikidsmart ist ein sehr ausgereiftes Produkt, welches aber den großen Nachteil hat, dass es ein kommerzielles Produkt ist.

Als Konkurrenten für die beiden Semantic Wikis wurden Microsoft Sharepoint und IBM Mashup Center ausgewählt. Microsoft Sharepoint mag vorwiegend als Verwaltungssystem

[1] http://mods.tiki.org/

für Microsoft Office Dokumente bekannt sein, bietet darüber hinaus aber auch umfangreiche Knowledge Management Fähigkeiten inklusive semantischer Annotationen und wird von den Semantic Mediawiki-Machern selbst als großer Konkurrent gesehen [Smw]. Das IBM Mashup Center wurde als letzter Konkurrent gewählt, da auf Basis dessen Vorgängers QEDWiki im Rahmen der Dissertation von Olaf Zimmermann [Zim09] ein „Architectural Decision Knowledge Wiki“ entwickelt wurde, welches gewisse Anforderungen an das in dieser Arbeit entwickelte Pattern Repository teilt.

3.5.1 Vergleichstabelle

In Tabelle 3.1 wurde eine Reihe von Kriterien zum Vergleich der Kandidaten ausgewählt, welche im Folgenden jeweils kurz begründet werden sollen. Dabei wurden die funktionalen Anforderungen aus Abschnitt 3.1 zusammengefasst und recherchiert, mit welchen existierenden Technologien einzelne Anforderungen umgesetzt werden könnten. Auf diese Weise wurde eine tatsächliche Vergleichbarkeit der Kandidaten erreicht.

- Kennzeichnung von Problemen (Tagging): Resultiert aus der Anforderung, Patterns mit Metainformationen annotieren zu können, um später eine erweiterte Suche nach diesen Kriterien zu ermöglichen. Tagging dient hier auch als allgemeine Bezeichnung für die Anforderung an einen Mechanismus, Aussagen über Beziehungen zwischen Entitäten treffen zu können.
- Ontologie-Support: Resultiert indirekt aus der Anforderung nach der Möglichkeit, Metainformationen unterschiedlicher Art zu annotieren. Um dem Benutzer eine Auswahl von Begriffen z.B. bei der Beschreibung von Beziehungstypen zu ermöglichen, muss diese Information zuvor maschinenlesbar hinterlegt worden sein. Ontologien eignen sich sehr gut für diesen Zweck, siehe dazu auch 2.3.
- Constraints auf Kardinalität: Dient zur Verfeinerung der Aussage über Beziehungen zwischen Entitäten. Resultiert aus der Anforderung nach erweiterten Suchmöglichkeiten.
- Suche anhand semantischer Metainformationen: Die Suche anhand von Metainformationen ist eine Basisanforderung, die jede als Repository verwendete Software erfüllen muss.
- Einpflegen von neuen Patterns: Ebenfalls eine Basisanforderung, die für die Grundfunktionalität als Repository erforderlich ist.
- Import/Export von Daten und Ontologien: Um auf per Ontologie spezifizierte Daten zugreifen zu können, sollten diese auch in das Repository importiert werden können. Wurde die Ontologie innerhalb des Systems verfeinert, sollte auch ein Export möglich sein.
- Reasoning: Resultiert aus der Anforderung, eine erweiterte Suchfunktion inklusive einer Querysprache zur Verfügung zu haben. Da mithilfe von Reasoning die Funktionalität von Querys insbesondere z.B. durch die Ausnutzung von transitiven Beziehungen

Anforderungen				
	Semantic Mediawiki	MS Sharepoint	Confluence Wikidsmart	IBM Mashup
Kennzeichnung von Problemen (Tagging)	Beliebige Tags via Semantische Attribute	Out-of-the -box keine individuellen Metadata-Tags. Eigene Implementierung in Visual Basic notwendig.	Beliebige Tags, die pro Kategorie festgelegt sind, welche man selbst definieren kann.	
Ontologie-Support	Ja	Als Taxonomie/Term-Set	Ja	Nein
Constraints auf Kardinalität	Ja, z.B. mit [BDS+09]	Nein	Unterstützung für Attribut-Datentypen und Kardinalitäten	Nein
Suche anhand semantischer Metainformationen	Vielfache Möglichkeiten durch Extensions. In Seiten integrierte SMW/SPARQL-Querys möglich.	SQL-basierte Suche.	Semantic Repository extern aufrufbar. Ebensfalls in Seiten integrierte SPARQL-Querys möglich.	Frei konfigurierbare Suchfilter auf Datenfeeds wie z.B. SQL-Datenbanken.
Einpflegen von neuen Patterns	Umfangreiches Fomularwerkzeug Semantic Forms und Mediawiki Templates	Normale Standardformulare müssten manuell per Visual Basic angepasst werden.	Anpassbare Formulare und Templates	Realisierung als Datenfeed von Quelle, welche bereits das entsprechende Format sicherstellt?
Import/Export von Daten und Ontologien	Manueller Import/Export mit freiem Mapping via Wiki-API	Import von RDF-Daten ohne Hierarchie in SharePoint Term Set	Getrennte Datenbanken für Wiki Inhalte und extra Semantic Repository	Viele unterstützte Datenquellen, kein OWL.
Reasoning	Durch Support von Triple Store lassen sich über Triple Store-Connector beliebige Reasoner anschließen.	Nein, Eigenentwicklung eines Triple Store Connectors nötig.	Ja	Nein, Eigenentwicklung eines Triple Store Connectors nötig.

Tabelle 3.1: Vergleichstabelle der Evaluation von Alternativen

erweitert werden kann, ist dies als Anforderung bei der Abfrage von Patternbeziehungen notwendig.

Anhand der Vergleichstabelle 3.1 wurde nun die Entscheidung getroffen, als Grundlage für diese Diplomarbeit Semantic Mediawiki zu verwenden. Das IBM Mashup Center hat seine Stärken v.a. in der Integration von Datenquellen und dem einfachen Visualisieren der integrierten Daten, erfüllt aber zu wenige der bestehenden Anforderungen, die es für die Aufgabe als Pattern Repository gibt. Microsoft Sharepoint würde vermutlich genauso eine sehr gute Basis für Eigenanpassungen liefern, aber der Aufwand würde den Rahmen einer Diplomarbeit wohl deutlich sprengen. Zuletzt bleiben Mediawiki mit der Erweiterung Semantic Mediawiki und Atlassian Confluence mit der Erweiterung Wikidsmart. Während Semantic Mediawiki und Wikidsmart zwar beides Open-Source Erweiterungen sind, ist lediglich Mediawiki als Wikiplattform auch Open-Source, Atlassians Confluence dagegen ist eine proprietäre kostenpflichtige Software. Die deutlichste Stärke von Semantic Mediawiki ist wohl die Existenz von vielen Erweiterungen, die den Mehrwert der semantischen Annotationen um ein vielfaches erhöhen. Im Vergleich dazu weiß Wikidsmart v.a. mit seiner großen Ausgereiftheit und Produktisierung zu gefallen.

Ausschlaggebend für die Entscheidung für Semantic Mediawiki waren zuletzt zwei Faktoren. Zum einen ist Semantic Mediawiki frei erhältlich und kostet keine Lizenzgebühren etc. Zum anderen gibt es bei Wikidsmart keinerlei Unterstützung für Entwickler von Erweiterungen, bzw. nur gegen die Kosten einer professionellen IT-Beratung. Da es sich bei dieser Diplomarbeit um eine wissenschaftliche Forschungsarbeit an einer Universität und kein Projekt in einem Unternehmen handelt, und es von absoluter Notwendigkeit ist, selbst Erweiterungen für das Framework zu schreiben, wurde die Entscheidung zugunsten von Semantic Mediawiki gefällt.

Für Semantic Mediawiki ist zudem die sehr umfangreiche Erweiterung *DataWiki* verfügbar, welche es u.a. auch ermöglicht, mit geringem Aufwand einen Triplestore für die semantischen Daten anzuschließend. Dadurch erhält man mit sehr geringem Aufwand einen voll funktionsfähigen Reasoner für Semantic Mediawiki, was ein weiteres sehr starkes Argument für dessen Verwendung ist. Eine genauere Beschreibung von DataWiki gibt es in Abschnitt 2.3.7, sowie einen grafischen Überblick über die Systemarchitektur in Abschnitt 5.2.

4 Konzeptionelles Design

In diesem Kapitel wird das Design beschrieben, welches aus den in Kapitel 3 erläuterten Anforderungen resultiert. Dabei orientiert sich die Reihenfolge der Beschreibung des Entwurfs an dem natürlichen Arbeitsablauf beim Verwenden des Pattern Repositorys, welcher in Abbildung 4.1 dargestellt ist. Die fundamentalen Arbeitsschritte sind durch die Pfeile ganz links repräsentiert: Zuerst erfolgt die *Eingabe* von Patterns, anschließend die *Annotation* mit semantischen Informationen und zuletzt die tatsächliche *Verwendung*.

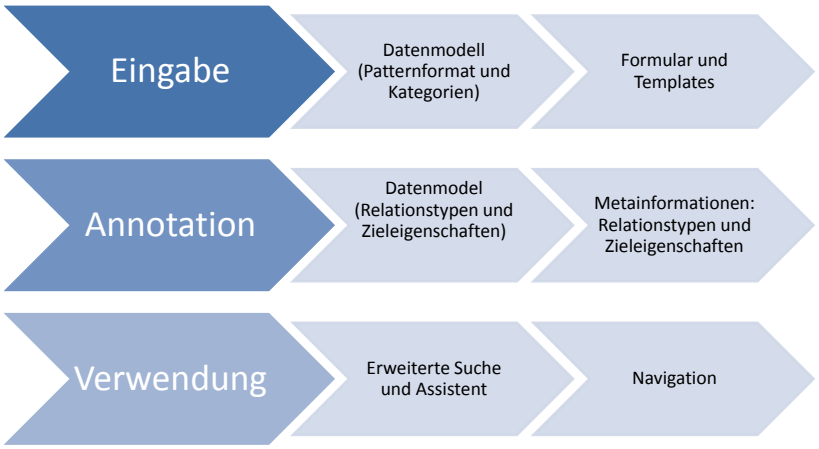

Abbildung 4.1: Arbeitsablauf beim Verwenden des Pattern Repositorys

Zunächst wird in Abschnitt 4.1 ein Entwurf für die Eingabe von Patterns in das System beschrieben. Wie in Abbildung 4.1 ersichtlich ist, geschieht dies anhand eines Formulars und Templates, die aufgrund eines Datenmodells erzeugt wurden. Es soll trotzdem zunächst der Grobentwurf für das Formular und die Templates beschrieben werden, um anhand der Kenntnis des Resultats schließlich die Beschreibung der Einzelheiten des für das Patternformats und der Kategorisierung nötigen Datenmodells in Abschnitt 4.4 besser verstehen zu können.

Anschließend oder bereits beim Einpflegen werden zusätzliche semantische Informationen annotiert, was in Abschnitt 4.2 genau beschrieben ist. In Abbildung 4.1 ist ebenfalls der Umstand dargestellt, dass für die Verfügbarkeit der Annotation von Metainformationen wie z.B. Relationstypen, diese zuvor im Datenmodell modelliert werden müssen. Die detaillierte Beschreibung der zugehörigen Teile des Datenmodells findet sich wiederum in Abschnitt 4.4. In Abschnitt 4.3 wird erläutert, welche Verwendungsmöglichkeiten es für ein Pattern

Repository geben kann und wie von der semantischen Annotation profitiert werden kann. In Abbildung 4.1 sind beispielsweise die erweiterte Suche anhand von Metainformationen sowie der Assistent aufgeführt, welche eine intelligente Navigation durch das Repository ermöglichen.

Zuletzt wird in Abschnitt 4.4 das vollständige resultierende Datenmodell aus den obigen Teilentwürfen genau beschrieben, welches als OWL-Ontologie umgesetzt wurde. Die detaillierte Erklärung des Datenmodells wurde bewusst am Ende des Kapitels platziert, um zunächst einen Überblick über die unterschiedlichen Funktionalitäten des Pattern Repositorys zu gewährleisten und anschließend einen Einblick in die genaue Modellierung zu erhalten.

4.1 Eingabe von Patterns

Der erste wichtige Arbeitsschritt ist die Eingabe von Patterns in das Repository. Hier sollte besonders beachtet werden, dass sich die Struktur von Patterns zwischen unterschiedlichen Patterndomänen deutlich unterscheiden kann, auch wenn viele Pattern-Autoren als Grundlage die Struktur von Christopher Alexander aus [AIS+77] benutzen. Um diesem Umstand gerecht zu werden, muss die Struktur bei der Einrichtung des Pattern Repositorys konfigurierbar sein, und anschließend konsistent in dem ganzen System eingehalten werden. Die Wichtigkeit des systemweiten Einhaltens eines festgelegten Patternformats liegt zum einen darin, dass eine gute Vergleichbarkeit innerhalb der Patternsprache und auch mit anderen Patternsprachen ermöglicht wird [DKT05, S. 32]. Zum anderen wird ein gemeinsames „Vokabular“ geschaffen, dass das kollaborative Erarbeiten von Patterns und den Austausch von Experten der Domäne untereinander erleichtert [FEL+12, S. 7].

Es muss zwischen zwei separaten Bestandteilen unterschieden werden: Einerseits dem Format des Formulars, welches zur Eingabe und der semantischen Annotation dient, andererseits dem tatsächliche Resultat, d.h. wie ein Pattern dem Nutzer präsentiert wird. Die tatsächliche Struktur des Patterns spiegelt sich dabei in beiden Bestandteilen wieder. Der Unterschied ist, dass im Eingabeformular evtl. gewisse technische Hilfsmittel zur Verfügung gestellt werden, um die Eingabe möglichst komfortabel zu machen, und in der Vorlage für die Ausgabe entsprechende Text oder Bild-Formatierungen hinterlegt sein müssen, um ein einheitliches Darstellungsformat zu erhalten. Diese Vorlagen zum festlegen des Ausgabeformats werden im weiteren Lauf der Arbeit als *Templates* bezeichnet.

In dieser Arbeit wird zum Erhalten einer konfigurierbaren Struktur zunächst versucht, eine minimale Menge von Inhaltselementen zu identifizieren, und diese Elemente mit einer minimalen Menge an Attributen zu versehen, um das Konfigurieren von beliebigen Patternformaten zu ermöglichen. Als Inhaltselement reicht ein Typ namens **Abschnitt** aus. Dieser genügt, um Text, der innerhalb eines Abschnitts enthalten ist, zu formatieren, ein Bild einzubetten, aber auch um komplexe semantische Beziehungen aus semantischen Annotationen zu beinhalten. In den meisten Fällen ist es sogar möglich, die originalen Pattern-Abschnitte direkt auf die Struktur-Abschnitte des Formats im Repository abzubilden. Diese Abschnitte dienen zudem als Basis dafür, bestimmte Relationstypen nur von bestimmten Abschnitten aus zu erlauben. Sowohl das Formular als auch die Templates sind anhand

dieser Struktur-Abschnitte gegliedert und verbunden, wobei das Formular für die Eingabe dient und die Templates für die korrekt formatierte Ausgabe der Eingaben.

4.1.1 Formular

Das Eingabeformular sollte das Patternformat möglichst direkt widerspiegeln, allerdings auch Unterstützung bei der Eingabe von Patterns bieten. Daher sollten die Abschnitte möglichst in der gleichen Reihenfolge im Formular auftauchen, wie sie auch in dem aus dem Formular generierten Patterntext im Repository auftauchen werden. Zudem sollte es möglich sein, für unterschiedliche Inhaltstypen eines Abschnitts passende Eingabetypen anzugeben. So bietet sich für die Eingabe eines Bildes evtl. ein anderer Eingabetyp an als für die Eingabe eines Textes. Es folgt eine Liste von möglichen Inhaltstypen und Vorschläge für benutzerfreundliche Eingabemöglichkeiten:

- Text: Für die Texteingabe genügt im Normalfall ein normales Textfeld. Formatierungsmöglichkeiten wie fette oder kursive Schrift sind dabei nicht erwünscht, da solche Möglichkeiten immer pro Abschnitt vorgegeben sind. Eine Besonderheit ist in diesem Fall das semantische Annotieren. Hierfür sollte es möglich sein, Text zu markieren und anschließend auszuwählen, mit welchem semantischen Relationstyp auf ein anderes Pattern oder eine Lösung gezeigt werden soll. Annotierte Textstücke sollen zudem als solche erkennbar sein. Annotationen sollten auch wieder rückgängig gemacht werden können.
- Bild: Um ein Bild zu einem Pattern hinzuzufügen, sollte es möglich sein, entweder neue Bilder in das System hochladen zu können oder auf bereits im System vorhandene Bilder zu verweisen. Für bereits vorhandene Bilder sollte der Benutzer per Autovervollständigung bei der Eingabe unterstützt werden.
- Zahlen: Für die Eingabe einer beliebigen Zahl bietet sich ein einfaches Textfeld an. Für die Auswahl einer Zahl aus einem gewissen vorgegebenen Bereich bietet sich ein Regler an, der nur Werte aus dem vorgegebenen Bereich zulässt. Für eine präzise Auswahl sollte es zusätzlich noch ein Textfeld geben, für eine unpräzise Auswahl dagegen genügt der Regler allein.
- Relation zu anderem Pattern: Zum einen muss es - wie im Punkt *Text* schon erwähnt - die Möglichkeit geben, in normalen Texten Relationen zu anderen Patterns zu annotieren, wenn diese im Fließtext anhand des Kontexts enthalten sind. Zum anderen gibt es in vielen Pattern-Formaten einen extra Abschnitt, in dem lediglich Verweise zu anderen Patterns aufgeführt sind. Um auf komfortable Weise für diese Verweise semantische Attribute auswählen zu können, sollte es ein spezielles Eingabewerkzeug geben, das die schnelle Auswahl aller verfügbaren semantischer Annotationstypen ermöglicht. Des Weiteren sollte die Auswahl von Zielpatterns eine Unterstützung für den Nutzer bieten, welche es ermöglicht, im System vorhandene Patterns schnell auswählen zu können. Eine Möglichkeit hierfür wäre ebenfalls die Autovervollständigung des Patternnamens.

- Relation zu Inhalten einer beliebigen Ontologie: Neben den Relationen zu anderen Patterns im System sollte es auch möglich sein, auf Inhalte von beliebigen zusätzlichen Ontologien verweisen zu können. Wie bereits im Rahmen von Kapitel 3 beschrieben, dienen Ontologien hier zur Vorgabe von zusätzlichen für die Annotation verfügbare Metainformationen. Zwei denkbare Einsatzzwecke wären hier zum einen eine Ontologie, die Zieleigenschaften enthält, auf die mit der Verwendung eines Patterns hingesteuert werden kann und zum anderen eine Ontologie, die mögliche Lösungen enthält, aus welchen die Patterns abstrahiert wurden oder die aufgrund der Auswahl einer bestimmten Patternmenge resultieren. Als Besonderheit sollte hier erwähnt werden, dass es in diesem Kontext wünschenswert ist, den Relationen zusätzliche Attribute zuweisen zu können. Dies wird ebenfalls eine gesonderte Eingabemaske nötig machen.

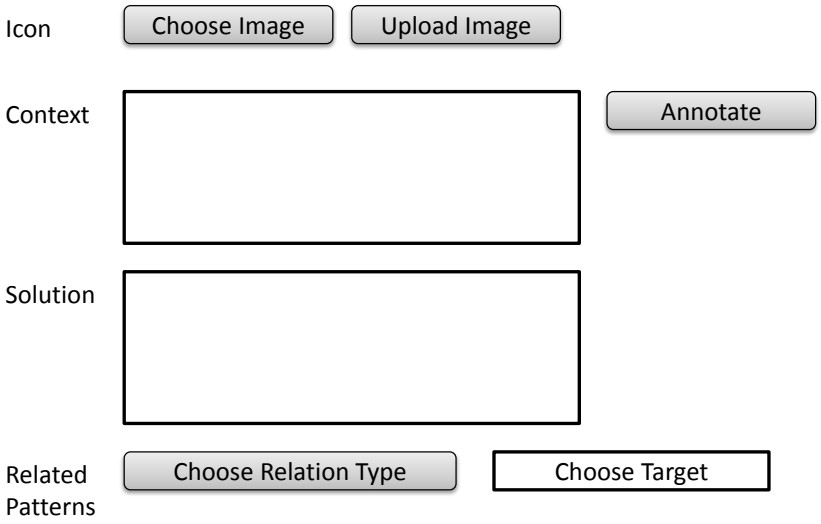

Abbildung 4.2: Mockup einiger Formularelemente

Das in Abbildung 4.2 dargestellte Mockup einiger oben vorgestellter Eingabetypen soll demonstrieren, wie ein solches Formular aussehen könnte. Links im Bild sind jeweils die Bezeichner der entsprechenden Patternabschnitte, und rechts die dazugehörigen Eingabetypen. *Icon* ist ein Beispiel für den Eingabetyp Bild und bietet Buttons zur Auswahl von vorhandenen Bilder und zum Hochladen von neuen Bildern. *Context* und *Solution* sind jeweils Texteingabefelder, wobei Context noch ein Button zur semantischen Annotation im Fließtext bietet. Bei *Related Patterns* findet sich ein Vorschlag für die Eingabe von semantischen Verweisen auf andere Patterns, wobei das Ziel in dem kleinen Textfeld rechts eingegeben werden kann.

4.1.2 Templates

Mithilfe der Templates werden die Informationen, welche im Formular eingegeben wurden, entsprechend aufbereitet und fertig für die Ausgabe formatiert. Anhand der Patternfor-

mate aus [FEL+12] und [SBLE12] wurde in einem ersten Arbeitsschritt untersucht, welche Formatierungsmöglichkeiten nötig sind. So sollte es für jeden *Abschnitt* folgende Formatierungsmöglichkeiten geben:

- Für die Überschrift: Zunächst sollte wählbar sein, ob ein Abschnitt eine Überschrift braucht. Neben der Überschrift selbst sollte darüber hinaus auswählbar sein, ob und wie sie formatiert ist. Also verschiedene Schriftgrößen sowie fett, kursiv und unterstrichen.
- Für den Text: Neben den Standard-Textformatierungen wie fett, kursiv und unterstrichen sollten auch Schriftfarbe und Schrifthintergrundfarbe wählbar sein. Zudem sollte, falls der Textabschnitt nach einem Bild kommt und keine Überschrift hat, wählbar sein, ob er als Fließtext neben dem Bild platziert werden soll oder danach.
- Für Bilder: Für Bilder sollten die Größe, die Position im Text (linksbündig, zentriert, rechtsbündig) und das Vorhandensein einer Bildunterschrift auswählbar sein. Des Weiteren sollte wählbar sein, ob das Bild eine Einrahmung oder sonstige Abhebung vom Text erhalten soll.
- Für semantische Annotationen: Falls ein Abschnitt speziell als Container für semantische Annotationen dient, sollten die Linktypen der Relationen auf einen Blick erkennbar sein. Daneben sollte zusätzlich zu der Verlinkung und dem Linktyp ein Beschreibungstext möglich sein, um zu begründen, weshalb die referenzierten Patterns in einer gewissen Beziehung zu dem aktuellen Pattern stehen. Handelt es sich bei der semantischen Relation um eine Relation mit Attributen, sollten die Attribute in einer übersichtlichen Form dargestellt werden. Unter Umständen wird es sich an manchen Stellen sogar anbieten, statt der semantischen Annotation direkt eine Query zur Darstellung der Beziehungen des Patterns zu anderen Patterns einzubetten.

Des Weiteren können Patterns anhand von Templates ein gewisses Bedienungslayout bzw. Interface erhalten. So wäre es beispielsweise sinnvoll, am Ende jedes Patterns einen Abschnitt zum weiteren Navigieren des Patternfundus einzublenden, der Querys auf bestimmte semantische Attribute enthält. Am naheliegendsten wären hier folgende Querys:

- Kombinierbare Patterns: Zeigt eine Liste der Patterns an, die entweder direkt vom aktuellen Pattern aus eine Verlinkung vom Typ *Kombinierbar* erhalten haben, aber auch Patterns, die auf die aktuelle Seite mit dem entsprechenden Linktyp zeigen. In OWL-Terminologie wären das alle Objekte, die mit dem aktuellen Subjekt das Prädikat Kombinierbar haben. Zudem alle Subjekte, die als Objekt das aktuelle Pattern haben und als Prädikat ebenfalls Kombinierbar. Dies entspricht im Prinzip einer Empfehlungsmatrix, wozu mehr im Abschnitt 4.3.2 unter *Empfehlungen aufgrund der Leseliste* erklärt ist und wozu die Grundidee aus [FLR+11] stammt.
- Kombinierbare Patterns höheren Grades: Dies sollte eine optional zuschaltbare Query sein, welche die Transitivität der Kombinierbarkeit ausnützt und dadurch solche Patterns anzeigt, welche als kombinierbare Partner der direkten Partnerpatterns des aktuellen Patterns annotiert sind. Dies könnte eine relativ große Ergebnismenge zur Folge haben.

- Sich ausschließende Patterns: Zeigt eine Liste der Patterns an, die nicht mehr sinnvoll verwendbar sind, sollte man sich für das aktuelle Pattern entscheiden. Die Liste soll Patterns enthalten, die entweder direkt von dem aktuellen Pattern aus eine Verlinkung vom Typ *Konkurrierende Alternativen* erhalten haben, aber auch Patterns, die auf die aktuelle Seite mit dem entsprechenden Linktyp zeigen.
- Patterns, welche die gleichen Ziele verfolgen: Aufgrund der Möglichkeit, beliebige Ontologien in das Pattern Repository mit einzubinden und die Patterns mit Bestandteilen der Ontologien zu annotieren, wäre es auch umsetzbar, Gemeinsamkeiten bei diesen zusätzlichen Annotationen aufzuzeigen. Ein mögliches Beispiel für eine Ontologie, die z.B. Zieleigenschaften enthält, folgt in Abschnitt 4.2.2. Dies würde ermöglichen, dem Endnutzer weitere Patterns aufzulisten, die die gleichen Zieleigenschaften verfolgen wie das aktuelle Pattern. Hat nun der Endnutzer bereits das aktuelle Pattern aufgrund einer gewissen Zieleigenschaft aufgerufen, die er verfolgen möchte, könnte diese Auflistung sehr nützlich sein.

Der Zusammenhang zwischen Formular und Template wurde in Abbildung 4.3 dargestellt. Auf der linken Seite befindet sich das Formular und auf der rechten Seite die Darstellung eines Patterns im Pattern Repository. Die Templates werden nun bei der Übertragung der Eingaben aus dem Formular in das gewünschte Ausgabeformular verwendet, was in der Grafik als grau gestrichelte Pfeile dargestellt ist. In den Templates werden die Eingaben als Parameter an den vorgegebenen Stellen eingefügt, so dass letztendlich die rechts dargestellte fertig formatierte Darstellung resultiert.

4.2 Annotation

Nachdem nun die Eingabe der Patterns entworfen wurde, soll der Entwurf der Annotation von zusätzlichen Informationen folgen. Den Mehrwert für das Pattern Repository im Vergleich zu nicht-semantischen Lösungen stellen die semantischen Informationen dar, mit welchen man die Artikel annotieren kann. Dabei ist der tatsächliche Mehrwert davon abhängig, wie sinnvoll die möglichen Annotationen gewählt wurden und ob diese auch korrekt vom Pattern Autor verwendet werden. Denn selbst wenn die Annotationsmöglichkeiten optimal gewählt wären, könnten falsche Annotationsinhalte den ganzen Nutzen wieder umkehren und evtl. sogar für inhaltliche Widersprüche sorgen. Eine gewisse Absicherung der Annotationsqualität ist dadurch gegeben, dass mithilfe des Datenmodells ein einheitliches Modell von Relationstypen vorgegeben wird, so dass diese konsistent innerhalb der Patternsprache verwendet werden.
Zunächst sollen nun die semantischen Attribute genauer erläutert werden, die im Pattern Repository zur Verfügung stehen. Als Anmerkung sei hier noch erwähnt, dass diese Attribute frei konfigurierbar sein sollten. Denn je nach Patterndomäne können sich evtl. die Anforderungen nochmals deutlich unterscheiden, oder spezielle Linktypen Sinn machen, die in einer anderen Domäne keinen Nutzen bieten würden.

4.2.1 Semantische Attribute - Linktypen

Als Basis für die vorgegebenen semantischen Attribute, die im Pattern Repository zum Ausdrücken von Relationen der Patterns untereinander mitgeliefert werden, diente die Diplomarbeit von Philipp Grimm [Gri11] . Die hier vorgeschlagenen Relationstypen sind sehr ähnlich zu den in [HL93, S. 28f] beschriebenen semantischen Beziehungen zwischen allgemeinen Entitäten in Repositorys. Dies liegt daran, dass es sich dabei um Relationen handelt, welche strukturelle Beziehungen von miteinander kombinierbaren Objekten beschreiben. Die Beobachtung, dass Patternsprachen durch solche strukturelle Beziehungen gebildet werden können, macht auch Welie *et al.* und liefert in [WV03] eine entsprechende Kategorisierung. Dabei gleichen die grundsätzlichen Relationstypen *Aggregation*, *Assoziation* und *Spezialisierung* den Relationen, die aus der objektorientierten Programmierung bekannt sind [WV03, S. 2]. Im Folgenden sind sowohl die grundsätzlichen Relationstypen als auch deren Verfeinerungen aufgeführt. Zusätzlich wird zur exakten Definition auch bereits beschrieben, ob eine Relation symmetrisch oder transitiv ist. Dies spielt bei der Abfrage von Patternrelationen eine Rolle und kann die Semantik solcher Abfragen verändern.

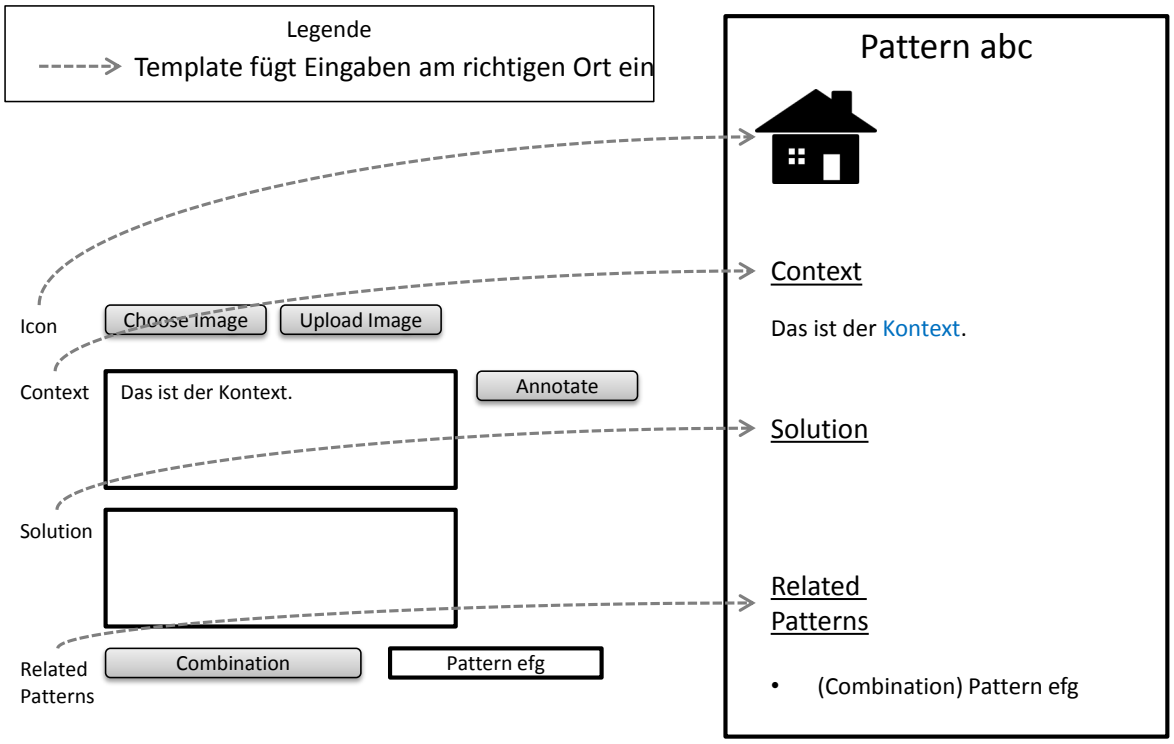

Abbildung 4.3: Zusammenhang zwischen Formular und Templates

Aggregation

Mithilfe des Relationstyps *Aggregation* lässt sich eine *hat ein*-Beziehung ausdrücken. Die Untertypen der Aggregation dienen vor allem zum Aufbau der Patternsprache (siehe Abschnitt 2.2 für eine Begriffserklärung) und drücken strukturelle Beziehungen zwischen den einzelnen Patterns aus. Von einem *aggregierenden* Pattern wird gesprochen, wenn das Pattern aus mehreren anderen Patterns besteht. Als Verfeinerung der Aggregationsbeziehungen werden die Vorschläge aus [Gri11] übernommen.

- Combining: Durch diesen symmetrischen transitiven Relationstyp lässt sich ausdrücken, dass sich die beiden verknüpften Patterns auf eine gewisse Weise kombinieren lassen, um in dieser Kombination entweder ein größeres Problem zu lösen oder um gemeinsam Teil eines größeren Systems zu sein. In gewissen Fällen kann es auch Situationen geben, in welchen sich mehrere Patterns kombinieren lassen, um die Funktionalität eines anderen Patterns zu imitieren. Dies ist in Abschnitt 7.2 näher ausgeführt.
- Completing: Durch diesen symmetrischen nicht transitiven Relationstyp lässt sich ausdrücken, dass sich die beiden verknüpften Patterns nicht nur kombinieren lassen (die Relation *Combining* ist implizit schon enthalten), sondern dass sie sich gegenseitig vervollständigen. Die beiden Patterns beinhalten Lösungen, die sich gegenseitig ergänzen und benutzt werden können, um ein größeres System aufzubauen. Es ist dabei denkbar, dass sich die beiden Patterns auf der gleichen Abstraktionsebene der Patterndomäne bewegen, aber auch dass eines der Patterns eine höhere Ebene anspricht, und das andere Pattern ein Problem auf einer niedereren Ebene löst.
- Competing: Durch diesen symmetrischen nicht transitiven Relationstyp lässt sich ausdrücken, dass zwei Patterns das gleiche Problem ansprechen und deswegen nicht gleichzeitig in einem System, welches durch die Patternsprache aufgebaut wird, verwendet werden können. Sie stellen Alternativen für die Problemlösung dar, wodurch dieser Relationstyp sehr wertvoll für den Endnutzer wird. Spricht aus einem beliebigen Grund etwas gegen die Verwendung eines gewissen Patterns, können dem Endnutzer durch diesen Linktypen verschiedene Alternativen aufgezeigt werden.

Spezialisierung

Die Beziehung *Spezialisierung* wird nicht weiter verfeinert, daher gibt es nur einen asymmetrischen nicht transitiven Relationstyp *Specialisation*. Dieser kann verwendet werden, um eine *ist-ein*-Beziehung zum Ausdruck zu bringen. Dies kommt insbesondere vor, wenn ein spezialisiertes Pattern die gleiche Grundidee wie ein anderes generelleres Pattern verwendet, aber sich in gewissen Aspekten des gelösten Problems eine spezifischere Lösung anbietet. Es erbt die Grundidee des allgemeineren Patterns und ermöglicht damit beim Bilden der Patternsprache die gleiche Vorgehensweise wie bei der objektorientierten Programmierung [WV03, S. 2]. Zunächst wird eine Basisklasse erstellt, in diesem Fall also ein Basispattern, und daraus Unterklassen abgeleitet bzw. spezialisierte Patterns verfasst.

Assoziation

Die Relationen vom Typ *Assoziation* dienen dem Aufzeigen von allgemeinen Beziehungen, die sich weder als Aggregation, noch als Spezialisierung beschreiben lassen. Die allgemeine Beziehung *Assoziation* dient zum Ausdrücken eines nicht näher beschreibbaren Zusammenhangs zwischen Patterns. Bei den Verfeinerungen der Beziehung *Assoziation* handelt es sich um keine strukturelle Beziehungen mehr, sondern z.B. um temporale Beziehungen wie eine zeitliche Reihenfolge der Anwendung von Patterns. Ebenfalls möglich ist eine Abstraktionsbeziehung, um Patterns mit den Anwendungsfällen in Verbindung zu bringen, aus welchen sie abstrahiert wurden. Für die Verfeinerung wurden lediglich der Untertyp *Association* aus [Gri11] übernommen, die restlichen Untertypen konnten aus den in Kapitel 3 beschriebenen Anforderungen entwickelt werden.

- Associaton: Hierdurch wird eine symmetrische transitive Beziehung ausgedrückt, die sich nicht genauer typisieren lässt. Im Prinzip entspricht dies einem Link ohne Typ, weswegen sich die Frage stellt, ob dieser Linktyp benötigt wird. Allerdings könnte er dazu dienen, unkompliziert ein gewisses Grundnetzwerk aufzubauen, welches später durch Linktypen verfeinert wird, welche mehr Aussagekraft haben.
- Depends On: Die asymmetrische nicht transitive *Depends On*-Beziehung drückt eine Abhängigkeit in der Art aus, dass ein Pattern allein, ohne das verlinkte Pattern, nicht verwendbar ist. Ein möglicher Fall könnte z.B. bei Patterns auf unterschiedlichen Abstraktionsebenen auftreten. So könnte erst ein Pattern auf einer höheren Abstraktionsebene überhaupt die Voraussetzungen für den Einsatz eines Patterns auf einer niedereren Abstraktionsebene schaffen, das eventuell ein sehr spezifisches Problem anspricht.
- Implemented By: Diese asymmetrische nicht transitive Beziehung ermöglicht es, Beziehungen von Patterns zu konkreten Lösungen oder Anwendungsgebieten des durch das Pattern angesprochenen Problems darzustellen. In einigen Patternsprachen, z.B. in [FEL+12], [SBLE12] und [JGVH95], werden diese Lösungen in einem extra Abschnitt *Known Uses* aufgeführt. Dieser Relationstyp ermöglich zum einen, konkrete Umsetzungen eines Patterns zu erhalten und andererseits den umgekehrte Weg. Dieser könnte vor allem bei Patterns auf höherer Abstraktionsebene Sinn machen, so dass der Endnutzer zunächst von einer Lösung ausgeht, dann entsprechende Patterns aufruft, die das Problem auf einer abstrakten Ebene lösen, und von dort über die Aggregationsbeziehungen weitere Patterns für seinen Anwendungsfall finden oder ausschließen kann. Dadurch wird die in Abschnitt 3.1 als Anforderung formulierte Transparenz ermöglicht sowie die Dokumentation der Quellen, aus welchen die Patterns abstrahiert wurden.
- Consider Before: Diese asymmetrische transitive Beziehung ermöglicht das Ausdrücken von temporalen Zusammenhängen. In vielen Situationen kann es sein, dass zwischen Patterns zwar keine direkte strukturelle Verknüpfung besteht, aber dafür eine temporale Beziehung in der Form, dass ein anderes Pattern erst dann sinnvoll einsetzbar ist, wenn das aktuelle Pattern bereits Verwendung findet. In gewisser Art überschneidet

sich dieser Relationstyp also mit Depends On; allerdings wird hier die Art der Abhängigkeit konkretisiert. Mithilfe dieses Beziehungstyps lässt sich eine Patternsprache mit „linearer Struktur“ modellieren [Sch03, S. 1]. Laut Schümmer trifft diese Linearität auf die meisten Patternsprachen zu, womit sich durch diesen Relationstyp eine gute Grundkompatibilität des Repositorys zu vielen Patternsprachen ergibt.

4.2.2 Zieleigenschaften

Die Annotation von Zieleigenschaften geschieht nach der Eingabe von Patterns und befindet sich dementsprechend auf der mittleren Ebene von Abbildung 4.1, die am Anfang dieses Kapitels zu finden ist. Mithilfe dieser Informationen können im letzten Schritt - der Verwendung des Pattern Repositorys - dem Benutzer im Rahmen eines Assistenten Vorschläge auf Basis der Annotationen gemacht werden.

Daher findet sich die Möglichkeit der Annotation von Patterns mit Metainformationen aus beliebigen Ontologien unter den in Abschnitt 3.1 beschriebenen Anforderungen. Ein sinnvoller Anwendungsfall für eine solche Ontologie könnte z.B. eine Ontologie von Zieleigenschaften für die entsprechende Patterndomäne sein. Denn die Ziele oder Eigenschaften, die mit einem Pattern korrelieren, sind oft vielschichtiger als nur das Lösen des spezifischen Problems, welches zugrunde liegt. Als Beispiel soll hier das Pattern *Eventual Consistency* aus der Domäne der Cloud Computing Patterns dienen [FLR+13]. Dieses Pattern beschreibt ein Vorgehen, welches bei der Verteilung von Datenreplicas über mehrere Orte die ständige Verfügbarkeit und Performance in den Vordergrund stellt. Dabei wird die strikte Konsistenz der Daten vernachlässigt, d.h. dass es Zustände geben kann, in denen inkonsistente Daten gelesen werden. Hier lassen sich drei Ziele identifizieren, welche dem Pattern zugrunde liegen: Verfügbarkeit, Performance und Datenkonsistenz. Wenn es nun eine Möglichkeit gibt, diese Zieleigenschaften anhand einer Ontologie zu annotieren, ist es im Nachhinein für den Endnutzer möglich, über die Angabe seiner Anwendungszieleigenschaften entsprechende Patterns vorgeschlagen zu bekommen.

An dem obigen Beispiel wird allerdings auch deutlich, dass eine einfach binäre Relation von dem Pattern zu der Ontologieentität nicht ausreicht, da z.B. nicht nur ausgedrückt werden soll, dass ein Pattern mit der Zieleigenschaft Verfügbarkeit verbunden ist, sondern auch auf welche Art die Verwendung des Patterns in einem System diese Zieleigenschaft beeinflusst. Diesbezüglich wäre auf der einfachsten Ebene denkbar, dass zusätzlich angegeben wird, ob die Verfügbarkeit angehoben oder gesenkt wird, und wie stark die Korrelation zwischen dem Pattern und der Zieleigenschaft ist. Um dies zu modellieren, ist es nötig, der Beziehung des Patterns zu der Zieleigenschaft weitere Eigenschaften in Form einer n-ären Relation hinzuzufügen, was bei der gewünschten Modellierung in OWL nicht direkt möglich ist. Während es z.B. bei Entity Relationship-Modellen ganz einfach möglich ist, n-äre Beziehungen darzustellen, muss bei RDF/OWL das Prinzip der *Reification* bzw. Vergegenständlichung angewandt werden. In [NRHW06] sind unterschiedliche Lösungswege zur n-ären Modellierung im Semantic Web beschrieben. Für die Modellierung im Rahmen dieser Arbeit hat sich am meisten der Weg über eine neu eingeführte Klasse angeboten, da der Use Case dieser Diplomarbeit genau mit dem in [NRHW06] beschriebenen übereinstimmt. Die neue

Klasse dient nur als leerer Containerknoten, um mit binären Relationen eine n-äre Relation darzustellen.

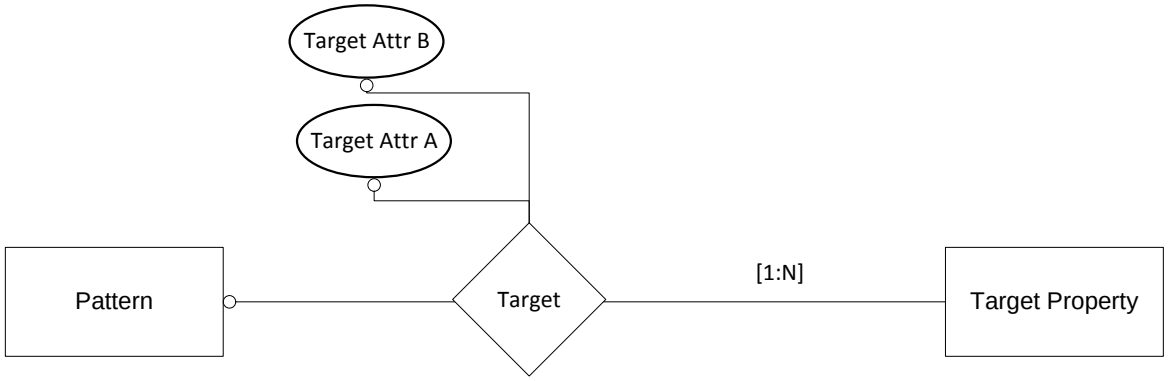

Abbildung 4.4: N-äre Relation in einem Entity-Relationship Diagramm

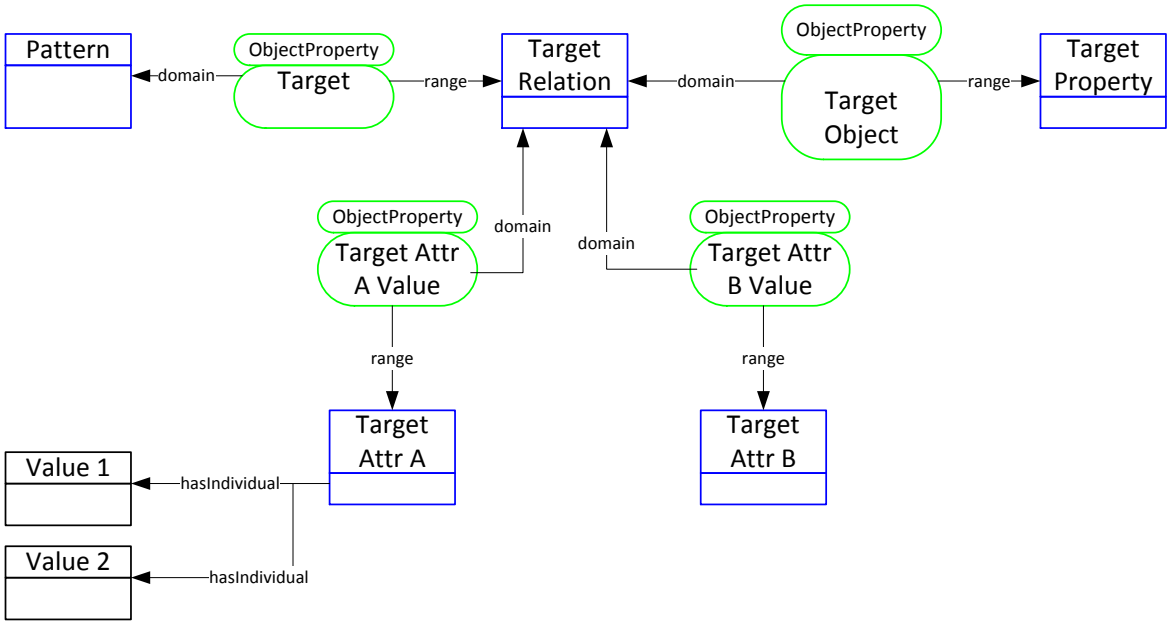

Abbildung 4.5: N-äre Relation in OWL

Zur Veranschaulichung dieser Modellierungsmethodik dient Abbildung 4.4, welche zunächst eine n-äre Relation mithilfe eines Entitiy-Relationship Diagramms darstellt und Abbildung 4.5, in der die gleiche Relation mit OWL dargestellt wurde. In dem Entitiy-Relationship Diagramm in Abbildung 4.4 sind die Entitäten *Pattern* und *Target Property*, welche als Rechtecke dargestellt sind, durch die Beziehung *Target*, welche als Diamant dargestellt ist, verbunden. Der Target-Beziehungen wurden nun die in Kreisen dargestellten Attribute

Target Attr A und *B* hinzugefügt. In der OWL-Ontologie in Abbildung 4.5 sind sowohl Entitäten als auch Attribute als Klassen modelliert, welche als blaue Rechtecke dargestellt sind. Beziehungen zwischen den Klassen werden als in der Abbildung grün dargestellte Object Propertys modelliert, wobei neben der ursprünglichen Beziehung Target auch noch Verbindungen zu den Attributen hergestellt werden müssen. Um einem Beziehungsattribut einen Wert zuzuweisen, müssen Individuen der Klasse mit dem gewünschten Wert erstellt werden, wie in der Abbildung anhand von Target Attr A demonstriert wird. Das Attribut Target Attr A bekommt die Werte *Value 1* und *Value 2* zugewiesen, indem diese als Individuen modelliert werden, welche in der Abbildung als schwarze Rechtecke dargestellt sind.

Beim Vergleich von Abbildung 4.4 und 4.5 fällt zunächst auf, dass sich durch die künstliche Einführung einer weiteren Entität in OWL die Komplexität des Diagramms erhöht. Statt nur einer Relation Target gibt es zusätzlich die Object Property Target, welche nur dazu dient, die Patternklasse mit der neuen *leeren* Relationsklasse *Target Relation* zu verbinden. Die tatsächliche Verbindung mit der Target Property wird nun erst hergestellt, indem die Target Relation-Klasse über eine neue eingeführte Object Property *Target Object* mit der Klasse der Zieleigenschaften verbunden wird. Zusätzliche Attribute der Relation benötigen in OWL pro Attribut eine extra Klasse und eine extra Object Property, um auf die Klasse zu verweisen. So entsteht zwar ein gewisser Overhead, aber es gelingt dadurch auch mit OWL n-äre Beziehungen darzustellen.

4.2.3 Formular

Die Annotation von semantischen Attributen stellt einen sehr zeitaufwendigen Prozess dar, welcher notwendig ist, um einen möglichen Mehrwert aus den zusätzlichen Informationen schöpfen zu können. Dabei stellt laut den Anforderungen aus Kapitel 3 nicht nur die Effektivität eine Anforderung dar, sondern auch die Effizienz. Daher wird die Annotation von semantischen Eigenschaften direkt in das Formular integriert, welches zum Erstellen von neuen Patterns dient.

Wie bereits in Abschnitt 4.1.1 erwähnt, werden dabei generell zwei Arten der Annotation unterschieden: Die Annotation direkt im Fließtext und extra Abschnitte, die speziell zum Annotieren dienen. Die Annotation im Fließtext soll für jedes Texteingabefenster möglich sein, allerdings in Abhängigkeit von dem entsprechenden Abschnitt, in welchem sich das Texteingabefenster befindet. Welche Relationstypen für den entsprechenden Abschnitt verfügbar sind, muss daher als zusätzliche Information in der Definition des Formular-Datenmodells geschehen. So können beispielsweise für einen Abschnitt, der Anwendungsfälle eines Patterns enthält, besondere Relationstypen zum Verweis auf eben jene Anwendungsfälle bereitgestellt werden. Die erlaubten Relationstypen müssen ebenfalls für Abschnitte definiert werden, deren Hauptzweck das Annotieren darstellt, da unter Umständen nicht erwünscht ist, dass in einem solchen Abschnitt komplett die gleichen semantische Prädikate zur Verfügung stehen als in anderen Volltext-Abschnitten. Hierfür gibt es zwei besondere Abschnittstypen, welche für alle Patterndomänen relevant sein dürften.

- Verwandte Patterns (*Related Patterns*): Dieser Abschnitt wird in allen der in dieser Arbeit referenzierten Patternkataloge in Buchform verwendet, und enthält Verweise auf verwandte Patterns innerhalb der Domäne. Da es mit dem Pattern Repository möglich ist, das Patternformat frei zu konfigurieren, und ebenso in welchen Abschnitten welche semantischen Linktypen verfügbar sein sollen, wird ein extra Eingabetyp benötigt, um diese Informationen aufzunehmen. Es soll eine Auswahl aller für den Abschnitt erlaubten semantischen Linktypen geben, sowie die Möglichkeit, pro Linktyp eine beliebige Anzahl von Linkzielen anzugeben. Dabei soll dem Endnutzer per Autovervollständigung bei der Eingabe geholfen werden. Zusätzlich soll pro semantischem Link auch eine Begründung bzw. Erklärung angegeben werden können, warum das aktuelle Pattern nun mit diesem semantischen Linktyp mit einem anderen Pattern vernetzt wird. Ebenso muss es möglich sein, beliebig viele dieser Annotationen pro Pattern hinzuzufügen.
- Zieleigenschaften (*Target Properties*): Dieser Patternabschnitt ermöglicht die Annotation der in Abschnitt 4.2.2 erläuterten Zieleigenschaften. Dabei ist die Grundidee gleich wie bei dem *Verwandte Patterns*-Abschnitt, mit dem Unterschied, dass hier zusätzliche Attribute, welche die Relation charakterisieren, abgefragt werden müssen. Ein weiterer Unterschied ist die Art der Auswahl einer entsprechenden Zieleigenschaft. Während die möglichen semantischen Linktypen im Verwandte Patterns-Abschnitt als flache Liste präsentiert werden können, da mehr als zehn Relationstypen im Normalfall nicht vorhanden sein sollten, ist dies bei unter Umständen sehr umfangreichen Ontologien für Zieleigenschaften nicht möglich. Hier sollte eine Auswahl im Baumformat geschehen, so dass jeweils nur gewisse Untermengen der Ontologie gleichzeitig angezeigt werden. Bei Patternrelationen sollte die Anzahl gering bleiben, da laut Miller [Mil56] der Mensch auf einen Blick nicht mehr als etwa sieben plus minus zwei Informationen gleichzeitig verarbeiten kann. Neben der unerwünschten Verlangsamung des Annotationsprozesses, die eine zu große Anzahl von semantischen Relationstypen zur Folge hätte, existieren zudem zum aktuellen Zeitpunkt keine Anwendungsfälle für eine noch feinere Aufgliederung von semantischen Relationstypen.

4.3 Verwendung des Pattern Repositorys

Die tatsächliche Verwendung des Pattern Repositorys stellt nach dem Erstellen und Annotieren von Patterns die letzte Stufe in der Übersichtgrafik 4.1 dar, welche am Anfang des Kapitels zu finden ist. Nachdem Patterns im ersten Schritt in dem Repository erstellt werden und im zweiten Schritt annotiert, gilt es, den Benutzer bei der Navigation durch den Patternfundus zu unterstützen. Dabei kann nun von den annotierten Metainformationen Gebrauch gemacht werden, indem dem Benutzer bei der Suche die semantischen Informationen als zusätzliche Suchkriterien angeboten werden, die sonst nicht möglich wären. Die erleichterte Suche von Patterns durch das Verwenden von zusätzlichen semantischen Informationen stellt somit eine Kernfunktionalität des Pattern Repositorys dar.

4.3.1 Einstieg in die Patterndomäne

Während sich bei Patternkatalogen in Buchform dem Leser zu Beginn die Frage stellt, ob er nun das Buch von Seite eins bis zum Ende linear durchlesen soll oder lieber bei einem Unterkapitel beginnen, tritt dieses Problem bei einem elektronischen Pattern Repository noch schneller auf. Denn ein Wiki lässt sich nicht von Beginn bis zum Ende durchblättern, so dass dem Endnutzer von der Startseite an eine Unterstützung geboten werden muss. Um dies zu einer Stärke des Repositorys auszubauen, soll zunächst untersucht werden, wie mit diesem Problem in einigen etablierten Patternkatalogen in Buchform umgegangen wird. Dazu gibt es verschiedene Ansätze, um dem Leser das Erschließen der Patterndomäne zu erleichtern und ihm zu helfen, das Patternbuch zu benutzen. Einige wiederkehrende Vorgehensweisen, welche in ähnlicher Form sowohl in [HW03], [FEL+12] als auch [JGVH95] zu finden sind, werden im Folgenden beschrieben.

Grundlegende Patterns

Viele Patterndomänen sind nach dem Vorbild der von Christopher Alexander in [AIS+77] entworfenen Patternsprache in ihrer Grundstruktur hierarchisch aufgebaut in dem Sinne, dass Entscheidungen auf einer hohen Abstraktionsebene mögliche Entscheidungen auf niedereren Abstraktionsebenen einengen. Der Begriff „Entscheidung“ wird hier als Synonym zu dem Begriff Pattern verwendet, da ein Pattern im Prinzip immer einer Entscheidung entspricht, die zur Lösung eines bestimmten Problems getroffen werden muss. Dementsprechend macht es in Patternsprachen mit solchen hierarchischen Beziehungen auch viel Sinn, bei der Lektüre auf der höchsten Abstraktionsebene zu beginnen und sich auf diese Weise durch die Patternsprache zu navigieren. Um diese hierarchische Struktur zugänglicher zu gestalten, bieten sich unterschiedliche grafische Aufbereitungsmöglichkeiten an. Zum einen wäre dies die Darstellung anhand einer Ordnerstruktur, die dem Inhaltsverzeichnis eines Buches gleicht, mit dem Unterschied, dass es Möglichkeiten zum Auf- und Zuklappen einzelner Ordner zur Erhöhung der Übersicht gäbe. Andererseits wäre dies die Darstellung als gerichteter Graph, in welchem die Kapitel bzw. Kategorien, zu welchen einzelne Patterns oder Unterkapitel bzw. Unterkategorien gehören, anhand von beschrifteten Umrandungen hervorgehoben werden würden. Der Graph sollte dabei gerichtet sein, um nicht die Information über die Richtung der Hierarchie zu verlieren.

Dabei sollen die leicht unterschiedlichen Ansätze der grafischen Aufbereitung aus [HW03] und [FEL+12] zu einer in unterschiedlichen Patterndomänen anwendbaren Lösung vereint werden. Aus [FEL+12] wird der Ansatz übernommen, für jedes großes Kapitel bzw. jede Kategorie von Patterns einen Graph zu erstellen, welcher die Abhängigkeiten innerhalb des Kapitels darstellt. Dabei sollen sowohl alle Patterns, als auch die Information der Zugehörigkeit zu eventuellen Unterkategorien erhalten und visualisiert werden. Dies kann, wie bereits oben beschrieben, durch einen gerichteten Graphen geschehen, in welchem Zugehörigkeiten zu Unterkapiteln durch beschriftete Umrandungen dargestellt werden. Bei einer Patterndomäne, in der noch nicht so viele Patterns entdeckt wurden und es dementsprechend auch noch nicht genügend Kapitel und Unterkapitel gibt, dürfte der

Ansatz aus [HW03, S. xlvii] nützlicher sein, bei dem eine große Übersichtsgrafik über den kompletten Patternkatalog existiert. Im Gegensatz zu [FEL+12], in welchem es auch eine Übersichtsgrafik gibt, ist diese in [HW03] in einer Art gehalten, wie Sie in anderen Domänen genau oder sogar noch besser verwendet werden kann. Auf einer sehr hohen Abstraktionsebene werden die Abhängigkeiten zwischen den Kapiteln skizziert, um so dem Leser einen schnellen Überblick über die Struktur der Patternsprache verschaffen zu können.

Diese Art des Einstiegs macht entsprechend am meisten Sinn, wenn ein Problem noch komplett ungelöst ist, wenn also nicht schon gewisse Teile des zukünftige Systems bzw. der Lösung des Problems feststehen. Wenn der Leser bzw. Benutzer noch völlig unvertraut mit dem Patternfundus ist, dürfte dies eine effiziente Einstiegsmöglichkeit sein, in dem Sinne, dass der Benutzer sich schnellstmöglich strukturiertes Wissen aus dem Gebiet aneignen kann.

Benutzerrollen

Ein völlig anderer Ansatz stellt das Vorschlagen von möglichen Startkategorien oder Startkapiteln aufgrund der Rolle dar, die der Benutzer beim Lesen des Patternkatalogs einnimmt. Dabei wird vorausgesetzt, dass sich jeder Leser in eine entsprechende Rollenkategorie einteilen lässt, und dass für einen Leser mit diesem Hintergrund die Lektüre von bestimmten Kapiteln vorteilhafter ist als die Lektüre von anderen Kapiteln. In [JGVH95, S. 10] geschieht dies nur sehr grob, so dass dem Leser im Fall, dass er unerfahren in objektorientiertem Design ist, der Beginn bei einem bestimmten Kapitel empfohlen wird. In [FLR+13, S. 14-20] und [HW03, S. xlviii-xlix] gibt es eine sehr viel feinere Unterscheidung der Kenntnisstände und Hintergründe der Leser, die dementsprechend spezifischer für das entsprechende Gebiet sind. Hier muss je nach Patterndomäne eine individuelle Lösung entwickelt werden, und es wird schwer sein, im elektronischen Format einen Mehrwert gegenüber einem Buch zu erzielen. Eine naheliegende Lösung dürfte das direkte Übernehmen der Rollen aus einer vorhandenen Buchvorlage sein.

4.3.2 Assistent

Nachdem nun bereits die in Büchern vorhandenen Einstiegshilfen für eine Patterndomäne erläutert wurden, soll nun untersucht werden, wie diese Punkte in einem Pattern Repository integriert werden können und welche weiteren Einstiegshilfen durch das elektronische Format möglich sind. Während bei einem Buch der Leser zunächst das Kapitel aufschlagen muss, in dem die Einstiegshilfen beschrieben werden, ist es bei einem Repository in Form einer Webanwendung sehr viel einfacher, den Benutzer zu führen. Diese Aufgabe soll ein Wizard bzw. Assistent übernehmen, in welchem der Benutzer neben den oben beschriebenen Einstiegspunkten auch noch weitere Möglichkeiten hat, sich effizient Wissen über die vorhandenen Patterns anzueignen.

Als Einstieg soll es zunächst eine Auswahl zwischen den möglichen Vorgehensweisen geben, bei welchen der Benutzer Schritt für Schritt von dem Assistent geleitet wird. Eine weitere Komfortfunktion, welche in dem Pattern Repository als Mehrwert im Vergleich zu einem normalen Buch integriert sein soll, ist das Vorhandensein einer „Leseliste“. Darin kann sich der Benutzer einzelne Patterns oder ganze Patternkapitel speichern, die er noch bearbeiten möchte, bzw. der Assistent kann dem Benutzer gewisse Vorschläge auf seiner Liste eintragen. Im Vergleich zu einem Buch fällt nun das wiederholte zurückblättern zu den Empfehlungen weg, und die Liste kann während des Lesens um weitere Einträge erweitert werden. Diese Leseliste kann auch als weiterer möglicher Einstiegspunkt verwendet werden, indem die semantischen Annotationen im Wiki für dynamisch zusammengestellte Querys verwendet werden. So können dem Benutzer intelligente Vorschläge auf Basis der in der Leseliste enthaltenen Patterns gemacht werden, welcher dieser im Rahmen eines iterativen Vorgehens erneut in seiner Leseliste abspeichern kann.

Die im Folgenden beschriebenen Einstiegspunkte wurden entwickelt, um den Benutzer im Falle von spezifischen Wünschen zu unterstützen und neue Perspektiven bei der Bearbeitung von Patterns zu bieten.

Zieleigenschaften

In Abschnitt 4.2.2 wurde bereits beschrieben, aus welchen Gründen es sinnvoll sein kann, Patterns mit Zieleigenschaften zu annotieren, mit welchen sie korrelieren. Um nun Vorteile aus diesen semantischen Informationen zu ziehen, soll der Benutzer eine Zieleigenschaft oder eine Reihe von Zieleigenschaften angeben können, zu welchen dann Patterns aufgelistet werden, welche möglichst stark mit den gewünschten Eigenschaften korrelieren. Da es die Möglichkeit gibt, beliebige Attribute für die Beziehung eines Patterns zu einer Zieleigenschaft zu definieren, müssen diese Attribute auch auswählbar sein. Das Ergebnis soll in tabellarischer Form präsentiert werden, so dass die Einträge nach der Korrelationsstärke zu den gewählten Zieleigenschaften mit den Attributen der Relation sortiert sind.

Dies ermöglicht eine völlig neue Herangehensweise für den Umgang mit einem Patternkatalog. Zuvor war der Leser bzw. Benutzer immer an die Kategorien und die Ratschläge gebunden, welche die Autoren der Patterns definiert haben. Dabei konnte es sehr schwierig sein, nützliche Patterns zu finden wenn man dabei nur an gewissen eingeschränkten Aspekten interessiert ist, welche die Patterns behandeln. Auch das Design eines kompletten neuen Systems anhand von Patterns wird im Normalfall eher durch gewisse Ziele definiert sein, als anhand von Problemen, über welche sich die Patterns definieren. Dabei ist es auch egal, ob das *System* nun ein neues Bauwerk sein soll [AIS+77], ein Kostüm für einen Film [SBLE12], eine objektorientierte Software [JGVH95] oder eine Anwendung im Cloud Computing Umfeld [FEL+12]. Allerdings muss die Ontologie der Zieleigenschaften immer an den Bereich angepasst sein, in welchem sich die Patterns ansiedeln. Da sich die Zieleigenschaften auf einer sehr hohen Abstraktionsebene befinden, dürfte es aber beispielsweise genügen, für Patterns unterschiedlicher IT-Bereiche die gleiche Ontologie von allgemeinen IT-Eigenschaften zu verwenden.

Anwendungsgebiete (Solutions)

Ein völlig entgegengesetzter Ansatz im Vergleich zu den bisher aufgeführten Einstiegspunkten ist die Auswahl eines Anwendungsgebiets, von welchem bekannt ist, dass das Pattern daraus stammt oder darin verwendet wird. Den Ansatz, von einer hohen Abstraktionsebene aus mit grundlegenden Patterns zu beginnen, oder auch aufgrund einer gewünschten Zieleigenschaft die Patterns auszuwählen, könnte man als „Top-Down"-Ansatz betrachten. Von grundlegenden Patterns auf einer hohen Abstraktionsebene bewegt man sich in Richtung niederer Abstraktionsebenen mit spezialisierten Patterns oder von einer Zieleigenschaft aus zunächst über abstraktere Patterns zu spezielleren Patterns hin.

Die Patternauswahl von den Anwendungsgebieten heraus stellt im Vergleich dazu den „Bottom-Up"-Ansatz dar. So werden Produkte und Lösungen betrachtet, die zuvor dazu verwendet wurden, durch Abstraktion des zugrunde liegenden gelösten Problems Patterns zu entdecken. Diese Anwendungsgebiete werden in einigen Patternkatalogen, wie z.B. in [FEL+12] und [HW03] in einem extra Abschnitt unter *Known Uses* erwähnt. Dabei gibt es nirgends einen Index, um über die Anwendungsgebiete alle Patterns zu finden, die darin verwendet werden bzw. daraus stammen. Es wäre allerdings durchaus denkbar, dass ein Benutzer an ein gewisser Produkt oder Anwendungsgebiet bereits gebunden ist, und zunächst wissen möchte, welche Patterns in dem Produkt verwendet werden und anschließend herausfinden möchte, welche weiteren Patterns sich besonders im Kontext mit diesem Produkt anbieten würden.

Als Beispiel hierfür soll ein Anwendungsgebiet bzw. ein Produkt aus [FLR+13] dienen: Amazon CloudFront[1]. Nehmen wir an, ein Entwickler einer Cloud Computing Anwendung ist aus einem bestimmten Grund, beispielsweise einer Partnerschaft seiner Firma mit Amazon, an dieses Produkt gebunden und möchte nun herausfinden, welche Patterns sich in diesem Umfeld verwenden ließen bzw. welche Patterns denn schon zum Einsatz kommen. Dann würde er durch die Verwendung des Pattern Repository-Assistenten herausfinden, dass CloudFront die Patterns *Content Distribution Network* und *Blob Storage* realisiert bzw. die beiden Patterns aus den konkreten Anwendungsfällen bei CloudFront abstrahiert wurden. Von diesen beiden Patterns aus kann der Benutzer entweder direkt über die Patternseiten oder über den Umweg der *Empfehlung aufgrund der Leseliste* (siehe folgenden Abschnitt) sich die nächsten Patterns anzeigen lassen, die bei eventuell auftretenden Problemen eine Lösung für ihn beinhalten könnten. Eine nötige Entscheidung wäre z.B., ob er sich für das Pattern *Strict consistency*, oder *Eventually consistency* entscheidet. Von diesen Patterns aus kann sich der Benutzer durch viele Einzelentscheidungen bei gewissen Designprobleme langsam ein System aus Patterns zusammenstellen. Zuletzt kann er von den Patterns aus wiederum nach möglichen Lösungen schauen, welche die ausgewählten Patterns realisieren.

[1] http://aws.amazon.com/de/cloudfront/

Empfehlungen aufgrund der Leseliste

Ein weiterer Ansatz, welcher bei der inkrementellen Zusammenstellung einer Leseliste sehr hilfreich sein kann, sind Empfehlungen auf Basis der Patterns, welche sich bereits in der Leseliste befinden. Dabei kann man die Leseliste als Hilfe für die Zusammenstellung eines schlüssigen Systems betrachten, zumindest falls der Benutzer keine sich widersprechende Patterns auf die Liste hinzufügt. Die Grundidee für diese Art, einen Patternfundus zu benutzen, stammt aus [FLR+11]. In diesem Paper wurde eine Entscheidungsempfehlungstabelle vorgestellt, welche eine Matrix darstellt, in welcher die Beziehungen von Patterns zueinander in Form von drei Beziehungstypen ausgedrückt werden: Starke Zusammengehörigkeit, Ausschluss und neutral. Übertragen auf das Pattern Repository wären diese Informationen in Form von semantischen Links auf den einzelnen Patternseiten gespeichert, wodurch man bei der semantischen Annotation implizit auch eine solche Entscheidungsempfehlungsmatrix erzeugt, welche aber mehr Beziehungstypen zulässt, und daher feinere Empfehlungen ermöglicht.

Basierend auf den Beziehungstypen *Combining*, welcher die starke Zusammengehörigkeit zweier Patterns ausdrücken kann, sowie *Competing*, welcher zum Ausdruck eines Ausschlusses dient, lassen sich anhand einer aggregierenden Query intelligente Empfehlungen aussprechen. Diese Query soll die Menge aller Patterns mit starker Zusammengehörigkeit zu den aktuellen Patterns in der Leseliste berechnen unter Entfernung solcher Patterns, welche sich gegenseitig ausschließen. Zusätzlich wäre es auch möglich, temporäre Abhängigkeiten in die Empfehlungsliste mit einfließen zu lassen, so dass lediglich Patterns empfohlen werden, welche eine direkte oder transitiv berechnete *Consider Before*-Relation zu den Patterns in der Liste besitzen. Diese Art der Empfehlung von weiteren Patterns dürfte viel benutzerfreundlicher sein, als das mühsame Absuchen einer riesigen Empfehlungsmatrix, was bei bereits vielen verwendeten Patterns beinahe unmöglich sein dürfte.

4.3.3 Weiteres Navigieren

Mithilfe dieser verschiedenen Einstiegsmöglichkeiten wird der Endnutzer des Systems wenig Schwierigkeiten haben, sich einen Einstieg in das Pattern Repository zu suchen. Sind die ersten Schritte geschafft, stellt sich die Frage, wie die weitere Navigation geschehen soll. Dafür ist es wichtig, das jedes Pattern in dem Repository über eine Art „Interface“ verfügt, welches sichtbar macht, wo sich das aktuelle Pattern in der Patternsprache befindet und wie man von dort aus weiter navigieren kann. Das Interface besteht zum einen aus der klassischen Art der Navigation in Form von Hyperlinks, wie sie auf jeder normalen Webseite auch verfügbar ist. Dies kommt vor allem in Abschnitten in Form von Prosatext vor, in welchen direkte Verweise auf andere Patterns enthalten sind, wofür auch keine semantischen Informationen benötigt werden. Die zusätzlichen Möglichkeiten der Navigation ergeben sich aus den semantischen Links auf andere Patterns. Diese können für jedes Pattern am Ende durch Querys gesammelt und in tabellarischen Form aufbereitet werden, so dass man beispielsweise auf einen Blick sieht, mit welchen Patterns sich das aktuelle Pattern denn gut kombinieren ließe, oder welche Patterns konkurrierende Alternativen sind. Dies ermöglicht

ein systematisches weiteres Vorgehen des Benutzers entlang der Patternsprache unter voller Ausnutzung der zur Verfügung stehenden semantischen Informationen.

Wenn man diese tabellarische Auflistung mit der Entscheidungsempfehlungsmatrix aus [FLR+11] vergleicht, entsprechen die unterschiedlichen Tabellen einer Zeile in der Matrix. Dabei ist die Auflistung in dem Pattern Repository insofern effizienter, da sie es ermöglicht, auf einen Blick alle Patterns mit starker Zusammengehörigkeit oder alle sich ausschließende Patterns anzuzeigen, ohne dabei erst nach den entsprechenden Zeichen in einer Matrix suchen zu müssen.

4.4 Resultierendes Datenmodell

Aus dem bisher beschriebenen Design resultiert ein Datenmodell, das in diesem Abschnitt detailliert beschrieben werden soll. Für die flexible Modellierung des Datenmodells werden mehrere OWL-Ontologien benutzt, zu deren besserem Verständnis eine Erklärung in Abschnitt 2.3.5 vorhanden ist. Im weiteren Ablauf werden die Ontologien durch ein Programm geparst, um in ein entsprechend vorbereitetes Repository importiert werden zu können. Erst nach dem Import ist die Funktionalität bezüglich dem Erstellen und Annotieren von Patterns möglich, was am Anfang des Kapitels in Abbildung 4.1 dargestellt ist. In dem Diagramm befindet sich das Datenmodell jeweils links von den jeweiligen Funktionen, welche durch das Modell beschrieben sind, da ohne das Datenmodell beispielsweise auch kein Formular erzeugt werden kann. Der Importvorgang ist detailliert in Abschnitt 5.3 beschrieben.

Das generelle Problem bei der Durchsetzung eines klar definierten Datenmodells in einem Wiki ist, dass Wikis ursprünglich für eine möglichst nicht in der Form eingeschränkte Sammlung von Informationen entwickelt wurden. Diesem allgemeinen Problem wirken nun zwei Faktoren entgegen. Zum einen wird durch die Verwendung von Semantic Mediawiki generell die Möglichkeit eingeführt, Daten zu strukturieren und dadurch in Querys verwendbar zu machen. Zum anderen gibt es in Mediawiki das Konzept der Templates, um das Datenmodell von Artikeln zu vereinheitlichen [HLS05]. Ohne die Verwendung von Templates kann es z.B. vorkommen, dass jeder Artikel, der eine Stadt beschreibt, ein leicht anderes Format aufweist. Mit der Verwendung von Templates, deren Funktionsweise detaillierter in Abschnitt 2.3.6 beschrieben ist, wird durch benannte Parameter ein konsistentes Format erreicht. Diese Möglichkeit ist bei der Verwaltung von Patterns in einem Wiki von großer Wichtigkeit, da Patterns einen großen Teil ihrer Zugänglichkeit durch ein konsistentes Format innerhalb der Patterndomäne erhalten. Dies wird beispielsweise in dem Paper [FLR+11] tiefer begründet, in welchem ein Patternformat für Cloud Computing Patterns vorgestellt wird, oder in [MD97], einem Standardwerk zum Verfassen von Patterns.

Es zeigt sich, dass das Datenformat im Wiki für die Patterns nicht direkt durch die Modellierung einer Tabelle für eine relationale Datenbank oder durch das direkte Importieren einer OWL-Ontologie festlegbar ist. Stattdessen muss das Wissen über das Datenformat in einem möglichst leicht maschinell lesbaren Format abgespeichert werden und dementsprechend auf die im Wiki zur Verfügung stehenden Entitäten übertragen werden. Auf welche Elemente des Wikis welche Elemente der Ontologie übertragen werden, soll nun

OWL	Semantic Mediawiki
Individual	Normale Seite in einem Namespace für Inhaltsseiten
Class	Seite im Namespace Kategorie
Subclass	Seite im Namespace Kategorie, mit einem Kategorieeintrag zu der Überkategorie
Object Property	Semantisches Attribut im Namespace *Property* mit dem Typ `Page`
Object Subproperty	Semantisches Attribut im Namespace *Property* mit dem Typ `Page` sowie semantisches Attribut `Subproperty Of` mit Verweis zum Vaterattribut.
Data Property	Semantisches Attribut im Namespace *Property* mit einem anderen Datentyp als `Page`, z.B. String oder Date

Tabelle 4.1: Abbildung von OWL- auf Semantic Mediawiki-Elemente

im Folgenden erörtert werden. In [VK06] wurde beschrieben, wie OWL-Ontologien in eine Semantic Mediawiki-Installation importiert werden können und insbesondere, welche „natürliche“ Entsprechungen es in Semantic Mediawiki für die Elemente einer OWL-Ontologie gibt. Seit dem Erscheinen des Papers 2006 hat sich jedoch das interne Semantic Mediawiki-Datenmodell verändert, weshalb in Tabelle 4.1 eine aktualisierte Tabelle aufgeführt ist. Im Vergleich zu der Originaltabelle in [VK06] hat sich insbesondere die Entsprechung von Object und Data Properties verändert, da es die zusätzlichen Namespaces *Relation* und *Attribut* mittlerweile nicht mehr gibt.

Wenn es für alle Standardelemente aus OWL Entsprechungen in Semantic Mediawiki gibt, stellt sich die Frage, warum nicht einfach die komplette Patterndomäne in OWL modelliert und direkt in das Wiki importiert wurde? Dies ist darin begründet, dass ein solcher direkter Import lediglich statische Daten auf Seitengranularität in das Wiki laden kann, aber nicht dafür geeignet ist, die Datenstruktur des Wikis auf besondere Bedürfnisse anzupassen. Die Ebene der Datenstruktur, die durch einen solchen Import von statischen Inhalten in das Wiki verändert werden kann, ist die Ebene der semantischen Attribute, Kategorien und Seiten, also auf sehr grober Granularität. Für die Anforderungen an das Pattern Repository ist dagegen eine Anpassung des Datenmodells auf einer viel niedereren Ebene bzw. feineren Granularität nötig: die Strukturierung von Daten innerhalb von Seiten durch verschiedene Templates und semantische Attribute sowie die automatische Erzeugung von Unterinhalten auf gewissen Seiten. Ein besonderer Anwendungsfall stellt hier auch die Konstruktion des Formulars dar, welche anhand der vorgegebenen Patternstruktur sehr flexibel sein muss, gleichzeitig aber eine komplexe Vielfalt von Möglichkeiten beinhaltet.

Darüber hinaus ist die Technologie OWL nicht dafür gedacht, großen strukturierten Prosatext aufzunehmen, sondern mehr um strukturelle Beziehungen zwischen Entitäten und Wissen über Taxonomien und deren Beziehungen auszudrücken. Im Fall des in dieser Diplomarbeit entwickelten Pattern Repository werden die OWL-Ontologien dafür genutzt, um das Wissen über die Form des Datenmodells abzuspeichern, welches von dem Importprogramm in das Wiki geladen wird. Es gibt dafür eine Ontologie, welche das Kerndatenmodell für die

Patterns sowie die semantischen Relationstypen zwischen den Patterns enthält, und eine andere optionale Ontologie, welche eine Taxonomie der möglichen Zieleigenschaften und eine Beschreibung der Attribute der n-ären Relation enthält.

4.4.1 OWL-Ontologie: Kerndatenmodell

Zunächst soll die Ontologie, die das Kerndatenmodell beschreibt, genauer betrachtet werden. Obwohl die ganze Ontologie über unterschiedliche Beziehungen sehr in sich vernetzt ist, soll versucht werden, die Betrachtung auf gesonderte einzelne Abschnitte zu lenken, um das Verständnis der Zusammenhänge zu vereinfachen.

Kategorien

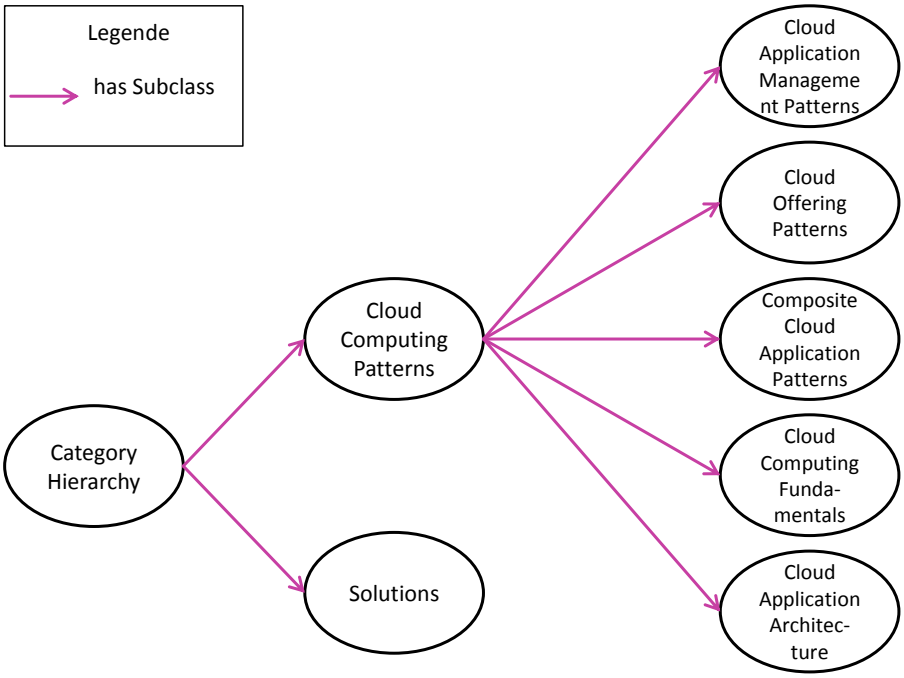

Abbildung 4.6: Datenmodell - Kategorien

Wenn für eine bestimmte Domäne bereits viele Patterns gefunden wurden, besteht im Normalfall auch bereits eine Kategorisierung der Patterns, d.h. bei Patternkatalogen in Buchform eine Einordnung der Patterns in Kapitel. Diese Kategorisierung lässt sich in der Klasse CategoryHierarchy der Ontologie direkt „einhängen“, d.h. die Taxonomie der Kategorien kann als Klassenhierarchie zu der Überklasse CategoryHierarchy hinzugefügt werden. In Abbildung 4.6 ist die oberste Ebene einer solche Taxonomie anhand eines Beispiels aus den Cloud Computing Patterns visualisiert [FLR+13]. Dabei dient die hier vorkommende

Klasse *Cloud Computing Patterns* als Überkategorie für die jeweilige Patterndomäne, wobei der Begriff „Cloud Computing“ durch die Domänenbezeichnung ersetzt wird. Die Klasse *Solutions* aus der Abbildung dient als Überkategorie für mögliche Anwendungsfälle von Patterns, die ins Repository eingepflegt werden können. Beim Import wird nun für jede vorhandene Klasse eine neue Kategorie im Wiki erzeugt, wobei die Information über die Unterklassen anhand der *has Subclass*-Beziehung durch Unterkategorien erhalten bleibt. Zusätzlich ist es möglich, über die Object Property *HasVisualizationGraph* festzulegen, dass auf einer bestimmten Kategorieseite einer der verfügbaren Visualisierungsgraphen verwendet werden soll.

Die Datenstruktur für die Zuweisung von Visualisierungsmöglichkeiten ist in Abbildung 4.7 abgebildet. Die Klasse *Visualization* dient der Speicherung von zur Verfügung stehenden Visualisierungsmöglichkeiten im Repository, wobei es in der Abbildung lediglich die Unterklasse *Graph* gibt. Momentan gibt es zwei unterstützte Graphtypen: Den PAREPTree, dessen Implementierung in 5.5.5 beschrieben ist, und den sog. Hypergraph aus der Semantic Graph-Erweiterung, welche in Abschnitt 2.3.7 beschrieben ist. Da dies konkrete Graphinstanzen sind, wurden sie als Individuen modelliert, was an der Darstellung als Rechteck erkennbar ist. Es sei angemerkt, dass beliebige weitere Graphtypen in die Ontologie eingefügt werden können, wenn entsprechend auch das Importprogramm erweitert wird, das für das Parsen der Ontologie zuständig ist.

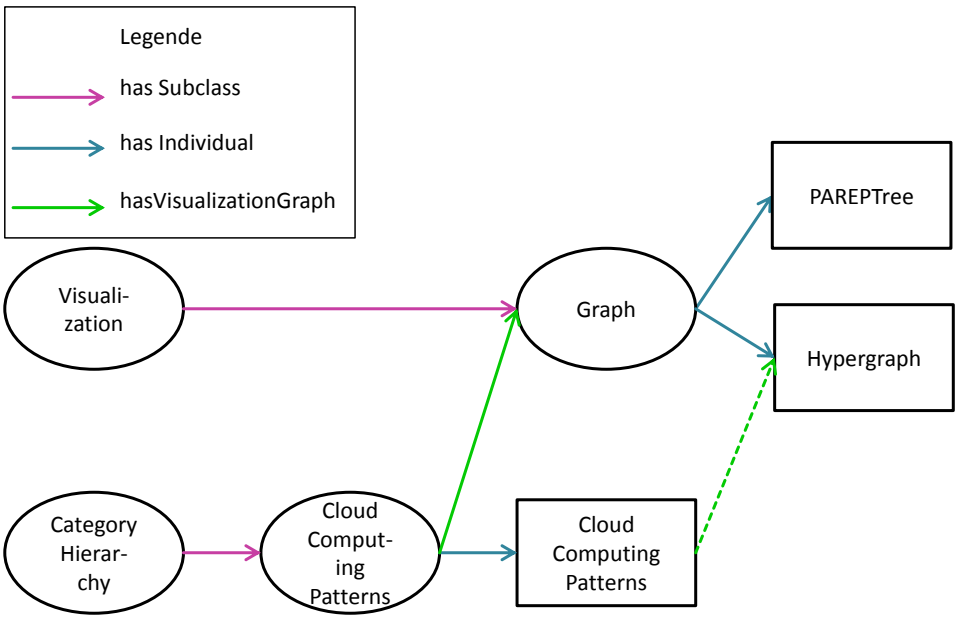

Abbildung 4.7: Datenmodell - Visualisierung

Für die Object Property HasVisualizationGraph wurde als Domain die Klasse Cloud Computing Patterns[2] gewählt, und als Range die Klasse Graph. Damit wird ausgedrückt, dass für Individuen der Klasse Cloud Computing Patterns die Möglichkeit besteht, einen Graph zur Visualisierung aus der Klasse Graph zu besitzen. Um nun für eine Kategorieseite einen Visualisierungsgraphen zu wählen, wird zunächst ein Individuum für die konkrete Seite erstellt. Für dieses kann anschließend in Form einer Object Property Assertion ein Graphindividuum aus der Klasse der verfügbaren Graphen ausgewählt werden. In der Abbildung gibt es beispielsweise die durch einen grün gestrichelten Pfeil dargestellte Object Property Assertion HasVisualizationGraph von der Kategorieseite Cloud Computing Patterns zu *Hypergraph*. Unterschiedliche Graphtypen benötigen im Normalfall auch verschiedene Eingabeparameter, welche für die Erzeugung des Wikicodes für den entsprechenden Graphen benötigt werden. Um eine Möglichkeit zu haben, solche zusätzlichen Parameter ebenfalls in der Ontologie abzulegen, wurde dafür eine Data Property *Style* eingeführt. Auf diese Weise können die nötigen Parameter über eine Data Property Assertion für das entsprechende Kategorie-Individuum gespeichert werden. Statt einer Data Property wäre auch eine Annotation Property zum Speichern dieser Art von zusätzlichen Informationen möglich gewesen. Diese Designentscheidung wurde aufgrund der Tatsache getroffen, dass sich Data Property Assertions komfortabler mit der für den Zugriff auf OWL-Daten verwendeten Jena RDF-API des Importprogramms parsen lassen.

Falls das Pattern Repository dazu benutzt werden soll, einen komplett neuen Patternfundus aufzubauen, ist es auch möglich, ohne einen bereits existierenden Kategoriebaum zu beginnen. Gewünschte Visualisierungsmöglichkeiten können entsprechend von Hand auf Kategorieseiten eingefügt werden.

Patternstruktur

Die Patternstruktur bzw. das Patternformat kann als Kern eines Patternkatalogs bezeichnet werden. Im Fall des Pattern Repositorys wird dieses Format über eine Reihe von Templates definiert, welche das Datenformat der Patterns definieren und dazu benötigt werden, die Eingabe über ein einheitliches Formular zu ermöglichen. Die Generierung der Templates und des Formulars wird über den Importer gesteuert und hängt von dem in der Ontologie konfigurierten Format ab. Neben den Informationen, welche Abschnitte es in einem Pattern geben soll, müssen auch Informationen über die Art der Abschnitte, über eventuelle Formatierungsoptionen für die Templates sowie für die korrekten Eingabetypen im Formular gespeichert sein.

Diese Informationen sind in dem Ausschnitt aus der Ontologie des Kerndatenmodells in Abbildung 4.8 modelliert. Die Klasse *Section* ist als Unterklasse von *ContentElement* modelliert, so dass es in Zukunft auch möglich ist, weitere Arten von Inhaltselementen einzuführen. Die tatsächlichen Abschnitte sind Individuen der Klasse Section, so dass sich

[2] In anderen Domänen würde die Klasse statt mit „Cloud Computing“ mit dem entsprechenden Domänennamen beginnen.

die nötigen Beziehungen zu anderen Klassen anhand von Object Property Assertions auf den Individuen darstellen lassen. Der orange Pfeil in der Abbildung steht für die Object Property *hasInputType* von der Klasse Section zu der Klasse *FormInputType*, in welcher verfügbare Eingabentypen für das Pattern-Eingabeformular gespeichert sind. Der grüne Pfeil von der Klasse *SemanticProperty* nach Section visualisiert die Object Property *allowsSection*, welche für die Zuweisung von Relationstypen zu Abschnitten benutzt wird. Anhand dieser Zuweisung wird bestimmt, welche Relationstypen für einen Abschnitt erlaubt sind, d.h. zur Annotation angezeigt werden, und welche nicht.

In Abbildung 4.8 ist beispielhaft anhand des Patternformats für Cloud Computing Patterns aus [FEL+12] dargestellt, welche Individuen die Klasse Section haben könnte. Um die Übersicht zu wahren, wurden lediglich für drei zufällig ausgewählte Individuen deren hasInputType-Beziehung angezeigt. Die Abschnitte Intent und Driving Question verwenden beide den gleichen Eingabetyp namens *Semantic Textarea*, was in der Abbildung anhand der von den beiden Individuen ausgehenden orange gestrichelten Pfeilen dargestellt ist. Der Abschnitt *Sketch* dagegen verwendet den Eingabetyp *ImageWithLabel*, was ebenfalls anhand eines orange gestrichelten Pfeils ersichtlich ist. Die Klassenzugehörigkeit der beiden Eingabetypen ist in dieser Abbildung nicht dargestellt. Dazu gibt es eine detaillierte Darstellung der Modellierung der Eingabetypen in Abbildung 4.10. Die näheren Beschreibungen zu dem semantischen Hintergrund der unterschiedlichen Typen sind in Abschnitt 4.1.1 zu finden. Die Implementierungen der Eingabetypen *Semantic Textarea*, *Number Slider* und *Target Props* sind in Abschnitt 5.5 detailliert beschrieben. Die Klasse FormInputType wurde als Unterklasse von *InputType* modelliert, um bereits einen möglichen Anknüpfungspunkt für die Definition von weiteren Eingabetypen für andere Entitäten innerhalb des Wikis zu bieten. Der orange Pfeil von Form zu FormInputType sagt aus, dass Instanzen der Klasse Form verschiedene Eingabetypen haben können. Zusätzlich erlaubt der Pfeil von Section zu FormInputType die individuelle Definition von Eingabetypen spezifisch pro Abschnitt.

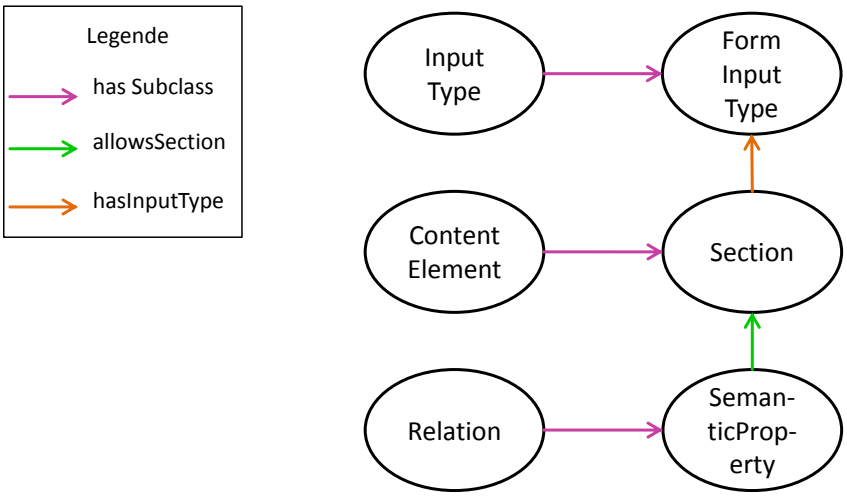

Abbildung 4.8: Datenmodell - Übersicht über die Abschnittsbeziehungen

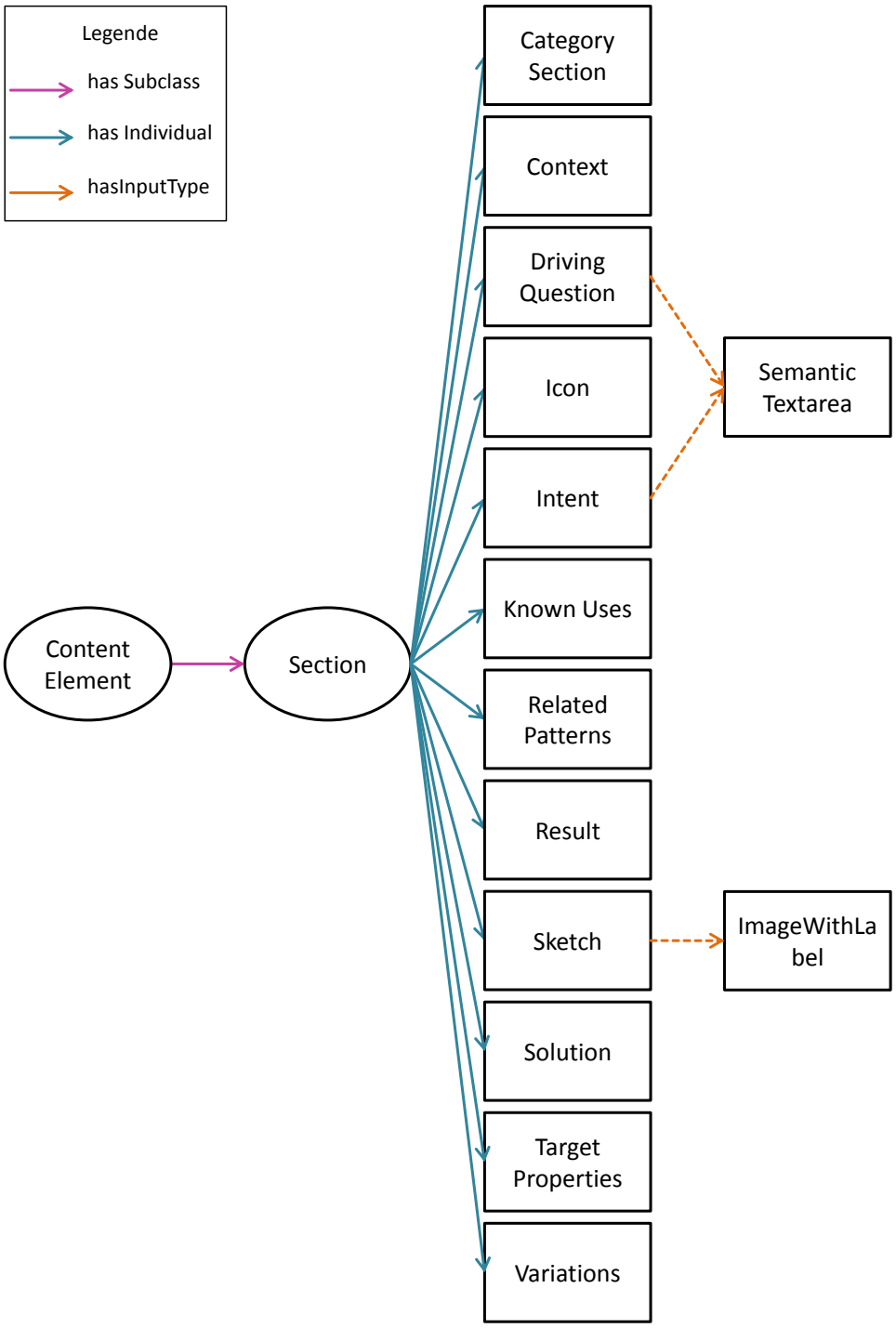

Abbildung 4.9: Datenmodell - Abschnitte

Die bis jetzt noch fehlenden Informationen betreffen die Darstellung der Patterns als fertige Seite, nachdem eine Eingabe über das Formular getätigt wurde. Neben der Angabe der entsprechenden Eingabetypen, soll es auch die Möglichkeit geben, für einzelne Abschnitte Formatierungsmöglichkeiten für die entsprechenden Inhalte angeben zu können. Darüber hinaus soll es auch möglich sein, die Reihenfolge der Abschnitte zu bestimmen, in welcher sie in dem resultierenden Element auftreten sollen. Da dies bei der Modellierung in OWL nicht direkt möglich ist, wurde der Umweg über eine Data Property gewählt. Die dafür eingeführte Data Property *OrderValue* vom Typ `postitiveInteger` wird daher benutzt, um jedem Abschnitt einen Index zuzuweisen. Konkret im Datenmodell ist dies als Data Property Assertion auf allen Individuen der Klasse Section modelliert. Eine Schwierigkeit ist hierbei,

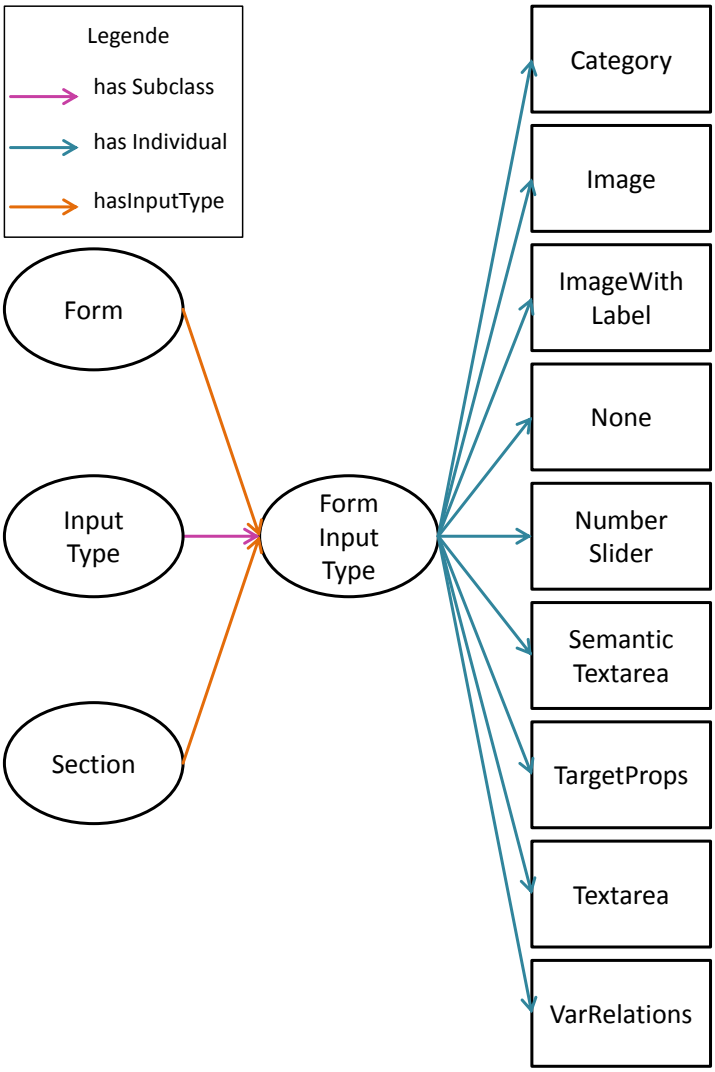

Abbildung 4.10: Datenmodell - Eingabetypen des Formulars

dass darauf geachtet werden muss, dass keine Indices doppelt verteilt werden, da dies beim Parsen dazu führen würde, dass eines der beiden Elemente mit gleichem Index ignoriert wird. Die Data Property Style wird schließlich dazu benutzt, um Formatierungen für Elemente angeben zu können. Dabei gibt es folgende gültige Optionen:

- `heading`: Gibt an, ob für den Abschnitt ein abgetrennter Textabschnitt mit fettgedruckter Überschrift erzeugt werden soll, oder lieber ein Abschnitt mit etwas kleinerer Überschrift, der keinen Absatz erzeugt. Der angegeben Wert entspricht der Überschrift.
- `headingStyle`: Gibt eine mögliche Textformatierung für die Überschrift an.
- `position`: Erlaubt es, für eingebundene Bilder die Position innerhalb der Seite zu bestimmen.
- `htmlElement`: Erlaubt die Angabe eines beliebigen HTML-Elements, in welchem der Eingabetext gespeichert werden soll.
- `htmlElementAttributes`: Erlaubt die Angabe von Attributen, falls ein HTML-Element angegeben wurde. Dabei sind nur solche Attribute gültig, die für das oben angegebene Element gültig sind.

Patternrelationen

Die Möglichkeit, Relationen zwischen Patterns durch unterschiedliche Linktypen auszudrücken, setzt voraus, dass zuvor die besonderen Charakteristiken dieser Relationen definiert wurden. Wieso es welche Linktypen gibt, wurde in Abschnitt 4.2.1 erklärt. Des Weiteren soll der Anforderung genügt werden, dass nicht alle semantischen Linktypen für alle Abschnitte zur Verfügung stehen. Dazu ist es nötig zu speichern, welcher Linktyp für welchen Abschnitt erlaubt ist. Diese Fülle an Informationen wurde in einem Teil der Kernontologie modelliert, welcher in Abbildung 4.11 dargestellt ist.

Die Klasse *Relation* in Abbildung 4.11 erlaubt es, allgemeine Charakteristika von Patternrelationen zu beschreiben, ohne dabei bereits spezifisch an eine Technologie gebunden zu sein. Die für Semantic Mediawiki spezifische Stufe beginnt dabei erst ab er Klasse Semantic Property, und spaltet sich von dort aus auf. Von der Klasse *Filter* aus gibt es eine Object Property *filtersRelation* zur Klasse Relation, um ausdrücken zu können, dass eine gewisse Relation filterbar ist. Dies ist als roter durchgezogener Pfeil dargestellt, drückt also eine Domain-Range-Definition aus. Das Filtern bezieht sich hierbei auf die Eignung für Drilldown-Operationen, wofür beispielsweise eine Verknüpfung des Filters mit einem semantischen Attribut bestehen muss. Von der Klasse Relation gibt es eine Object Property *hasRelationType* zur Klasse *RelationType*, welche dafür verwendet wird, Individuen der Klasse Relation einen Relationstyp zuweisen zu können. Momentan gibt es zwei unterschiedliche Relationstypen, die nach dem Vorbild von OWL-Object Properties ausgewählt wurden: *Symmetric* und *Transitive*, welche in der Abbildung als Individuen dargestellt sind. Auf weitere Charakteristiken, die es zwar für OWL Object Propertys gibt wurde verzichtet, da sie entweder nicht abbildbar auf Semantic Mediawiki sind oder nicht benötigt werden. Dabei wird von den Standardwerten dieser beiden Attribute ausgegangen, wenn einem Individuum der Klasse Relation keiner

der Relationstypen zugewiesen wird. Eine Relation ist dadurch standardmäßig asymmetrisch und nicht transitiv. Dies wurde nach dem Bestimmen der nötigen Charakteristiken für die gewählten semantischen Attribute so gewählt, da die Mehrzahl der Relationen asymmetrisch und nicht transitiv sind.

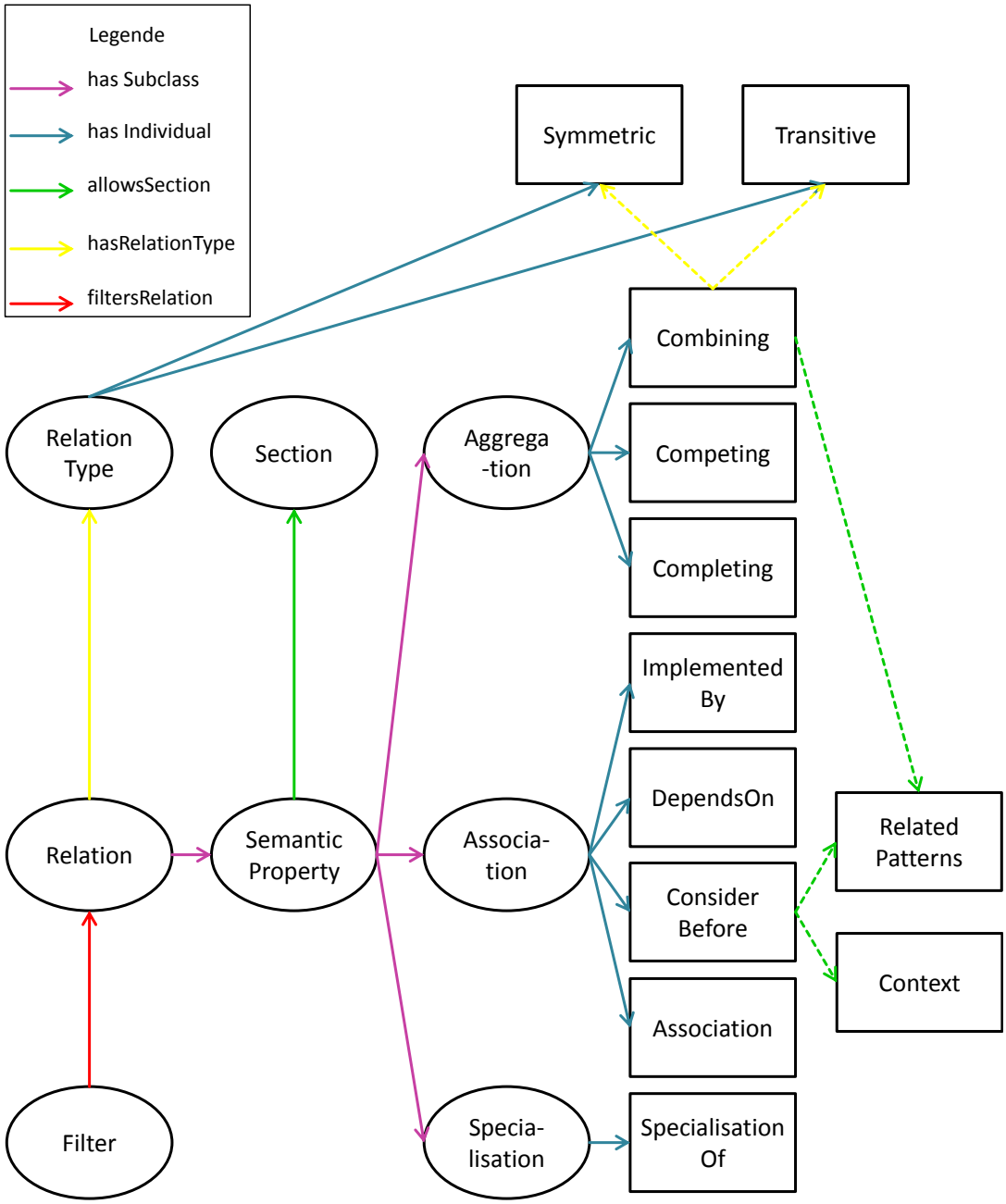

Abbildung 4.11: Datenmodell - Semantische Attribute

Ein Beispiel für eine solche Zuordnung findet sich in Abbildung 4.11 anhand des semantischen Attributs Combining, welches symmetrisch und transitiv ist. Dies wird in der Abbildung als gelb gestrichelter Pfeil dargestellte Object Property Assertion von hasRelationType modelliert, der Combining mit Symmetric und Transitive verbindet. Die nächste Information, welche für das Datenmodell benötigt wird, ist die Zuordnung von semantischen Relationen zu den entsprechenden Abschnitten des Patternformats. Bei der Annotation über das Patternformular sollen pro Abschnitt nur bestimmte semantische Attribute zur Verfügung stehen, und diese Zuordnung von Attributen zu Abschnitten muss im Datenmodell definiert werden. Dafür wurde die Object Property allowsSection eingeführt, welche zwischen den Klassen SemanticProperty und Section über Domain und Range definiert wurde. Dadurch ist es möglich, Individuen der Klasse SemanticProperty mit Individuen der Klasse Section über Object Property Assertions von allowsSection in Verbindung zu bringen. Ein Beispiel für die allowsSection-Relation in der Abbildung findet sich in Form der grün gestrichelten Pfeile. Dabei wurden jedoch bewusst nicht alle existierenden allowsSection-Relationen dargestellt, um die Übersicht zu waren. Aus der Abbildung lässt sich anhand dieser Relation beispielsweise entnehmen, dass das Individuum ConsiderBefore der Klasse SemanticProperty zur Annotation im Abschnitt Context zugelassen ist. Context ist dabei ein Individuum der Klasse Section, was jedoch aus Übersichtsgründen in der Abbildung ebenfalls nicht dargestellt ist.

4.4.2 OWL-Ontologie: Zieleigenschaften

Um die Annotation von Zieleigenschaften im Pattern Repository zu ermöglichen, muss es einen gewissen Rahmen geben, in welchen je nach Patterndomäne Ontologien von Zieleigenschaften eingehängt werden können. Zusätzlich soll auch die Beziehung zu den Zieleigenschaften beliebige Beziehungsattribute besitzen können, weswegen diese Attribute ebenfalls frei in der Ontologie definierbar sein müssen.

Ein mögliches Design für eine solche Ontologie ist in Abbildung 4.12 dargestellt. Zunächst ist anhand der Klassenhierarchie in der Abbildung von *ContentContainer*, *Page* und *Pattern* modelliert, von welchem Inhaltselement auf Zieleigenschaften verweisen werden kann. Dies ist durch den grauen Pfeil von Pattern nach *TargetRelation* anhand der Object Property *target* festgelegt, so dass Patterns auf Zieleigenschaften verweisen können. Der Verweis geschieht aufgrund der in Abschnitt 4.2.2 erläutert Problematik von n-ären Beziehungen in OWL aber lediglich indirekt. Die leere zusätzliche Klasse TargetRelation wurde als Zwischenknoten eingeführt, welcher als Zielobjekte Individuen aus der Klasse Target Properties annimmt und weitere beliebige Attribute aufweist. Der tatsächliche Verweis auf die Zieleigenschaften geschieht über die in der Abbildung lila dargestellte Object Property *Target Object*. Als zusätzliches Beziehungsattribut gibt es die Klasse *TargetImpact*, welche zwei Individuen beinhaltet: *Decrease* und *Increase*. TargetImpact ist eine Unterklasse von TargetRelation, und steht über die Türkis dargestellte Object Property *Target_Impact* in Verbindung, was zum Verweis von Individuen von TargetRelation auf Individuen von TargetImpact nötig ist.

Auf diese Weise lässt sich zu einer konkreten Zieleigenschaft-Relation ein konkreter Einfluss zuordnen. Zusammen mit der Object Property *target_Correlation* soll es somit möglich sein, die Korrelationsstärke eines Patterns zu einer Zieleigenschaft anzugeben. Darüber

hinaus ist es durch die Angabe der Richtung des Einflusses möglich, Aussagen zu treffen wie: *„Pattern Xyz steht mit einer Korrelation von 75% mit einem Sinken der Verfügbarkeit in Verbindung."*. Die Object Property für die Zieleigenschaft-Korrelation ist in Abbildung 4.12 nicht abgebildet, da sie zwar als Domain die Klasse Target Relation verwendet, als Range aber die Data Property *Correlation Range*, welche einen Integer im Bereich von $0 - 100$ erwartet. Damit wird ausgedrückt, dass target_Correlation von einem Individuum der Klasse Target Relation auf eine Zahl im von Correlation Range definierten Wertebereich verweist. In der Grafik ist bereits sichtbar, dass unter der Klasse *Target Properties* in der Ontologie eine Klassenhierarchie von Zieleigenschaften eingehängt werden kann. Als Beispiel wurde hier *IT-Target Properties* gewählt, wobei hier eine beliebige für die Domäne modellierte Ontologie von Zieleigenschaften stehen könnte. Um vom Parser gelesen werden zu können, ist es notwendig, die Zieleigenschaften als Klassenhierarchie mithilfe der Beziehung *has Subclass* zu modellieren.

4.4.3 Abbildung auf Semantic Mediawiki-Elemente

Der nächste Schritt nach der Definition des Datenmodells in einem nicht systemgebundenen Format ist nun die Abbildung der Elemente aus der OWL-Ontologie auf Elemente, die es in Semantic Mediawiki gibt. Abbildung 4.13 stellt eine Übersichtsgrafik dar, in welcher die verschiedenen Zuordnungen der Elemente zueinander abgebildet sind. Auf der linken Seite finden sich die beiden Ontologien, die zum Zeitpunkt des Verfassens dieser Arbeit existieren

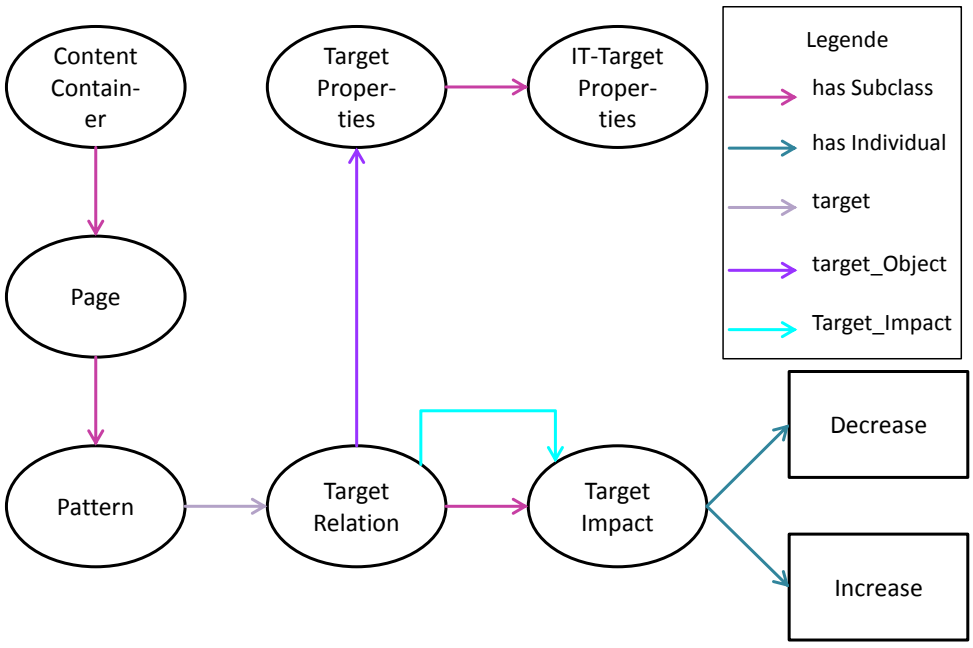

Abbildung 4.12: Allgemeine Ontologie zur Einbindung von Zieleigenschaften mit Legende

in Form von Rechtecken: Die *Kerndatenmodell Ontologie* und die optionale *Zieleigenschaften Ontologie*. Die Begriffe in Ellipsen stellen dabei OWL-Klassen dar. Auf der rechten Seite sind die Elemente innerhalb von Semantic Mediawiki abgebildet, auf welche die Ontologieinhalte nach dem Parsen durch das Importprogramm übertragen werden.

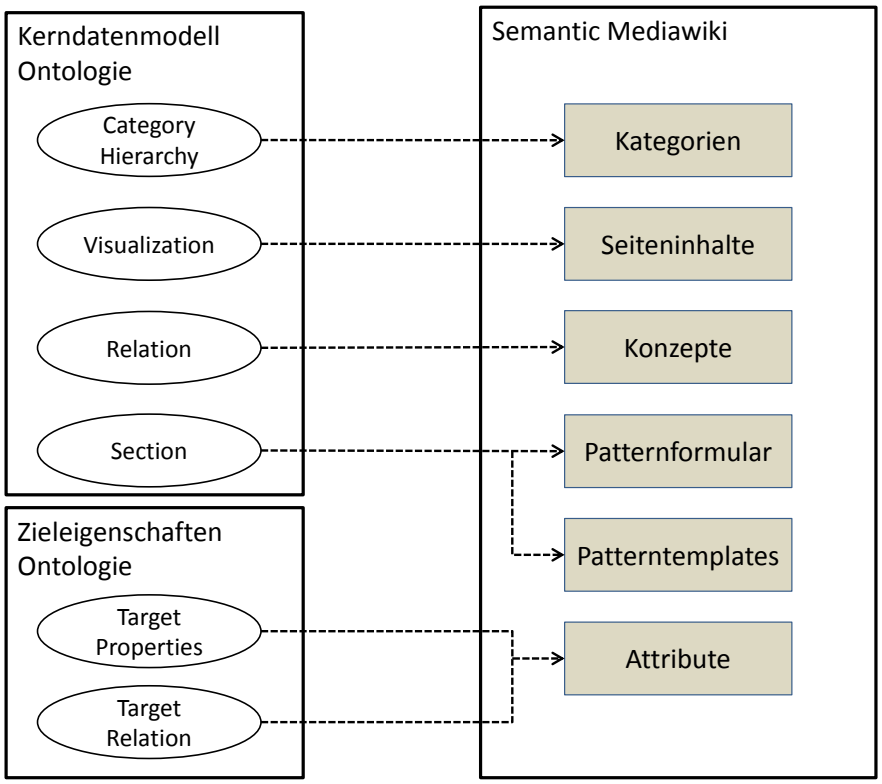

Abbildung 4.13: Mapping der OWL-Ontologien durch den Importvorgang. Legende: Gestrichelter Pfeil=*Mapping*

Die Frage, warum nun nicht direkt die Zuordnung aus Tabelle 4.1 verwendet werden konnte, wurde zu Beginn dieses Abschnitts bereits erörtert. In der folgenden Aufzählung wird erklärt, auf welche Elemente im Wiki die Inhalte der Ontologie Einflüsse haben. Sollten dabei auch Zuordnungen erwähnt werden, die aus Abbildung 4.13 nicht direkt ersichtlich sind, liegt das daran, dass zur Wahrung der Übersicht nur die größten Einflüsse abgebildet wurden. Die folgende Aufzählung der Abbildungen orientiert sich an den OWL-Klassen, die auf der linken Seite der Grafik dargestellt sind.

- **CategoryHierarchy:** Die Klassenhierarchie, welche in der Ontologie als Unterklassen von *CategoryHierarchy* gespeichert werden, wird auf die direkte Entsprechung im Wiki, den Kategorien, abgebildet. Dabei findet die Unterklasse-Beziehung in den Wiki-Unterkategorien ihre Entsprechung. Die Klassenhierarchie hat indirekt auch Einfluss auf das Patternformular, da dort eine Auswahl zur Einordnung des Patterns

in eine Kategorie zu Verfügung steht, welche dynamisch von den Unterkategorien der Kategorie CategoryHierarchy abgerufen wird.

- **Visualization:** In der Klasse Visualization sind mögliche Visualisierungsarten gespeichert, welche in dem Pattern Repository verfügbar sind. Über die Relation HasVisualisationGraph kann beispielsweise direkt für einer der Kategorien aus CategoryHierarchy ein Graph definiert werden, welcher den Inhalt der Kategorieseite darstellt.
- **Relation:** Aus den Unterklassen der Klasse Relation werden im Wiki semantische Attribute bzw. Linktypen erstellt, welche zur Vernetzung der Patterns verfügbar sein sollen. Die Information, für welche Patternabschnitte ein semantischer Linktyp zur Verfügung steht, wird über eine in einem Konzept gespeicherte Query dynamisch abgerufen. Dafür erhalten die Seiten, welche die semantischen Linktypen definieren, ihrerseits wiederum ein semantisches Attribut allowsSection vom Typ `String`, in welchem die erlaubten Patternabschnitte für den entsprechenden Linktyp hinterlegt sind. Die erstellten Konzeptseiten werden später von Funktionsmodulen der Erweiterung Pattern Repository zur Annotation (Semantic Textarea und Property Dropdown, siehe Abschnitt 5.5) verwendet, um nur erlaubte Linktypen anzuzeigen.
- **Section:** Hier wird das eigentliche Patternformat gespeichert. Dabei beinhalten die OWL-Klassen und Individuen aus der Klasse Section sowohl die Informationen über das Format des Patternformulars, als auch die Informationen zur Generierung der Templates, welche das Erscheinungsbild der eingegebenen Daten des Formulars steuern. In den Templates gibt es dabei für jeden Patternabschnitt einen benannten Parameter zur Platzierung der Eingaben aus dem Formular in der entstehenden Wiki-Seite. Zur Generierung des Formulars müssen von Semantic Forms akzeptierte Eingabetypen verwendetet werden, sowie eine Zuordnung zu den in den Templates verwendeten Parametern. Gäbe es in dem Formular keine Möglichkeit, ein Template mehrmals zu instanziieren, würde ein einziges Template für das Patternformular genügen. Da es für das Patternformular allerdings die Anforderung gibt, gewisse Inhaltselemente wegzulassen oder beliebig oft anzugeben, müssen die Templates für das Formular entsprechend aufgeteilt werden. Die Verteilung muss dabei so geschehen, dass jeder Inhaltsblock, welcher beliebig oft im Formular verwendet werden kann, ein vollkommen eigenes Template erhält. Ein Beispiel für solch einen Abschnitt ist Related Patterns, der zur Annotation beliebig vieler verwandter Patterns dient.
- **TargetProperties:** In dem Fall, dass eine extra Ontologie mit Zieleigenschaften zur Verfügung gestellt wird, werden die Unterklassen von TargetProperties, eventuelle Unterklassen von TargetRelation und dessen Individuen auf ein Geflecht von Attributen, Kategorien, Konzepten und normalen Wikiseiten abgebildet. Für jede Unterklasse der TargetProperties wird eine Kategorieseite und eine normale Wikiseite im Namespace *Ontologie* erstellt, welche dazu dienen kann, eine Beschreibung zu der entsprechenden Zieleigenschaft zu hinterlegen. Dabei werden alle Zieleigenschaftenseiten in eine Überkategorie TargetProperties eingeordnet, über welche mithilfe einer Konzeptseite eine Liste aller vorhandenen Zieleigenschaften dynamisch über eine Query geführt wird. Zum Verweisen auf diese Zieleigenschaften wird ein semantischer Linktyp Target eingeführt, der Patternseiten mit Zieleigenschaften verbindet, mit welchen

diese korrelieren. Mithilfe der Klasse TargetRelation ist es möglich, Eigenschaften der Relation zwischen Pattern und Zieleigenschaft in der Ontologie zu modellieren. Ins Wiki werden diese anschließend abgebildet, indem für jede Kombination aus Zieleigenschaft und Relationsattribut eine eigene semantische Eigenschaft erstellt wird, was auch in Abschnitt 4.2.2 beschrieben ist. Gibt es zum Beispiel eine Zieleigenschaft „Performance“, wird es auch ein semantisches Attribut „PerformanceCorrelation“ geben, falls ein Relationsattribut „Correlation“ existiert.
Des Weiteren hat die Zieleigenschaftenontolgie auch Einfluss auf die Templates und das Patternformular. Falls eine Ontologie mit Zieleigenschaften eingebunden wird, muss es ein zusätzliches Template geben, welches einen Inhaltsbaustein enthält, welcher einer Zieleigenschaftsannotation entspricht. Darüber muss dieses Template so im Formular eingebunden werden, dass kein oder beliebig viele dieser Inhaltsbausteine erstellt werden können.

5 Implementierung des Pattern Repositorys

Nach der Beschreibung des konzeptionellen Designs im vorherigen Kapitel, soll in diesem Kapitel auf die Implementierung des Pattern Repositorys eingegangen werden, also der Umsetzung des Entwurfs. Zunächst wird in Abschnitt 5.1 ein kurzer Überblick über im Mediawiki-Framework verwendete Technologien und die Besonderheiten bei der Entwicklung von Erweiterungen gegeben. Abschnitt 5.2 gibt eine Übersicht der Systemarchitektur, um die Anordnung der einzelnen Komponenten anhand eines Schichten-Modells deutlicher zu machen. Im darauf folgenden Abschnitt 5.3 wird der Import des in Kapitel 4 beschriebenen Datenmodells beschrieben. In Abschnitt 5.4 gibt es eine detaillierte Ausführung über das durch den Import erzeugte Patternformular. Zuletzt wird in Abschnitt 5.5 auf das eigentliche Herzstück des entwickelten Systems eingegangen, der Semantic Mediawiki-Erweiterung *Pattern Repository*.

5.1 Web-Entwicklung mit dem Mediawiki-Framework

In diesem Abschnitt soll zunächst ein kurzer Überblick über die verwendeten Technologien gegeben werden, die beherrscht werden müssen, um mit dem Mediawiki-Framework entwickeln zu können. Insbesondere Abschnitt 5.1.2 ist hilfreich, um Abschnitt 5.5 besser nachvollziehen zu können, in welchem auf die Implementierung der für das Pattern Repository entwickelten Erweiterung *Pattern Repository* eingegangen wird.

5.1.1 Technologien des Mediawiki-Frameworks

Mediawiki basiert auf der Skriptsprache PHP und verwendet als Datenbank MySQL. Mediawiki selbst, sowie die meisten Erweiterungen, verwenden die Möglichkeit objektorientiert mit PHP zu programmieren, wofür es seit Version 5 ein vollständiges Objektmodell in PHP gibt. Für eine effiziente Entwicklung mit Javascript auf der Seite des Clients wird in aktuellen Mediawiki-Versionen immer häufiger auf die umfangreiche Javascript-Bibliothek jQuery[1] gesetzt. jQuery vereinfacht viele, häufig bei der Javascript-Programmierung auftretende Aufgaben, wie die Traversierung und Manipulation von HTML-Dokumenten, Event Handling, Animationen und AJAX durch eine benutzerfreundliche API. Dabei verwendet jQuery CSS3-kompatible CSS-Selektoren und funktioniert in den meisten Browsern auf die

[1] http://jquery.com/

gleiche Weise, was bei Javascript nicht gegeben ist. AJAX[2] steht für *Asynchronous JavaScript and XML* und ermöglicht asynchrone Datenübertragung zwischen Browser und Server, so dass Seiteninhalte einer scheinbar bereits geladenen Seite transparent nachgeladen werden können. Der Benutzer bemerkt davon in den meisten Fällen nichts, sondern denkt, dass bereits alles beim Öffnen der Seite geladen wurde. Diese Technologie zur Vermeidung von unnötigem Datenverkehr wird beispielsweise von der Erweiterung *Category Tree*, die in Abschnitt 2.3.7 vorgestellt wurde, verwendet, um Teile des Kategorienbaums erst zu laden, wenn der Benutzer den entsprechenden Knoten aufklappt. CSS-Selektoren[3] werden dazu verwendet, bestimmte Elemente in einem HTML- oder XML-Dokument anhand eines angegebenen Musters auszuwählen. Dabei entspricht der Selektor einer Bedingung, die für jedes Element innerhalb des Dokuments geprüft wird und eine Ergebnismenge der Elemente zurückliefert, bei welchen die Bedingung zu „wahr“ ausgewertet werden konnte. Zu Beginn der meisten jQuery-Ausdrücke steht daher ein CSS-Selektor, um auszuwählen, auf welchem Element gearbeitet werden soll.

5.1.2 Entwicklung von Erweiterungen für das Mediawiki-Framework

Das Mediawiki-Framework bietet eine Reihe von Erweiterungsmöglichkeiten, die die Entwicklung von sehr unterschiedlichen Erweiterungen ermöglicht. In [Lero6] werden dazu die absoluten Grundlagen beschrieben und auf [Meda] wird ein detailliertes Handbuch zu den unterschiedlichen Aspekten der Erweiterungsentwicklung zur Verfügung gestellt. Im Folgenden sollen diejenigen Erweiterungsmöglichkeiten kurz erläutert werden, welche im Rahmen dieser Diplomarbeit bei der Entwicklung *Pattern Repository* verwendet wurden.

Wiki Markup-Erweiterungen erweitern die Syntax von Wikitext um zusätzliche Tags, die mit beliebigen Funktionen versehen werden können. Wikitext-Tags entsprechen XML-Tags in der Art von `<Beispieltag>Eingabetext</Beispieltag>`. Die Registrierung des Paares aus Tag-Bezeichner und auszuführender Funktion geschieht in der Instanz des Wikitextparsers, welcher Wikitext zu HTML umwandelt. Die Funktion, die mit dem Tag registriert wird, erhält den im Tag eingeschlossenen Text als Parameter übergeben und soll validen HMTL-Code zurückgeben. Falls in dem von den Tags umschlossenen Text weitere Wikitags oder Parserfunktionsaufrufe enthalten sind, müssen diese vor der Rückgabe explizit durch den Aufruf der entsprechenden Parserfunktion ebenfalls zu HTML umgewandelt werden.

Parserfunktionen stellen eine Möglichkeit dar, um eng mit dem Wikiparser integriert, direkt eigene Funktionen aufzurufen. Sie sind schwergewichtiger als Markup-Erweiterungen, bieten dafür jedoch den Vorteil, dass von Parserfunktionen erzeugte Ausgaben, vor dem eigentlichen Parsen des Wikitexts, in die Seite eingefügt werden. Dies ermöglicht eine Interaktion mit anderen Wikielementen, wie z.B. das Verwenden der Ausgabe

[2] http://www.w3schools.com/ajax/
[3] http://www.w3.org/TR/selectors/

der Parserfunktion als Eingabe für Wikitext Tags. Eine Parserfunktion wird im Wikitext in folgender Form verwendet: `{{#funktBezeichner:param1=a|param2=b}}`. Die Bezeichnung der Funktion muss dabei in einer Übersetzungsdatei hinterlegt sein und mindestens in Englisch vorliegen sowie als *Magic Word* registriert sein, um vom Mediawiki-System fehlerlos erkannt zu werden. Magic Words sind Strings, die vom Mediawiki-System nicht direkt ausgegeben werden, sondern stattdessen wird der Rückgabewert der Funktion ausgegeben, die mit dem Magic Word assoziiert ist. Die Übersetzungsdatei muss in dem Wiki-System registriert werden, ebenso wie der Funktionsbezeichner und die aufzurufende Funktion. Die Funktion hat über ein assoziatives Array Zugriff auf benannte Parameter und kann Wikitext oder HTML-Code zurückgeben. Die Rückgabe von Wikitext stellt kein Problem dar, da wie bereits oben erläutert, die Ausgabe der Parserfunktion nochmals komplett von dem Wikitextparser bearbeitet wird.

Hooks an Events ermöglichen es, eigene Event Handler mit beliebigen Mediawiki-Events zu verknüpfen. Dabei gibt es eine Liste[4] der momentan über 400 verfügbaren Hooks, an welchen Funktionen registriert werden können. Dabei ist es auch möglich, an einem Hook mehrere Funktionen anzuhängen. Die Funktionen werden in diesem Fall nacheinander ausgeführt und Änderungen werden von Funktion zu Funktion weitergereicht. Ein Beispiel für einen in *Pattern Repository* verwendeten Hook ist *BeforePageDisplay*. Damit verknüpfte Funktionen werden jedes Mal aufgerufen, bevor eine Seite frisch geladen wird.

Spezialseiten sind Seiten im gesonderten Namensraum *Spezial* und dienen zur Ausführung von spezifischen Funktionen. Spezialseiten können nicht direkt von Benutzern editiert werden und werden auf einer speziellen Seite des Wikis aufgelistet. Darüber hinaus können explizit die Benutzerrechte festgelegt werden, die zum Verwenden einer Spezialseite nötig sind. Dies ist insbesondere nötig, da viele Spezialseiten zu administrativen Zwecken dienen und daher nur von Administratoren verwendet werden sollten. Spezialseiten werden von der Klasse *SpecialPage* abgeleitet und durch Überschreiben der Ausgabefunktion soll valider HTML-Code zur Erzeugung der Spezialseite zurückgegeben werden. Falls es erwünscht ist, dass die Seite in der Auflistung aller Spezialseiten erscheint, müssen dafür Einträge in einer zuvor registrierten Übersetzungsdatei erstellt werden.

Darüber hinaus wurde versucht, den Best Practises bei der Entwicklung von Mediawiki-Erweiterungen zu folgen und so entspricht der Aufbau exakt den Mustervorgaben aus [Meda]. In der Hauptdatei „PatternRepository.php“ werden alle Teilerweiterungen, entsprechend der jeweils verwendeten Methode im Mediawiki-System registriert, die benötigten PHP-Klassen in den sog. *Autoloader* von Mediawiki hinzugefügt, die Übersetzungs- und Magic Word-Dateien registriert sowie alle Module für den *Resource Loader* zusammengestellt. Der Autoloader von Mediawiki sorgt dafür, dass alle benötigten PHP-Klassen geladen werden, die zu der jeweiligen Erweiterung gehören. Der Resource Loader ist ein relativ

[4] http://www.mediawiki.org/wiki/Manual:Hooks

neuer Bestandteil von Mediawiki, der eingeführt wurde, um die Leistung und das Anwendererlebnis der Weboberfläche zu verbessern. Durch die verstärkte Verwendung von Javascript, CSS und zu übersetzender Textbestandteile entwickelte sich das Problem, dass pro Seitenaufruf viel unnötiger Javascript-Code zum Benutzer übertragen wurde. Der Resource Loader versucht dieses Problem zu entschärfen, indem nur tatsächlich auf der Seite benötigte Funktionalitäten geladen werden und möglichst viel Caching verwendet wird [Medb]. Zur Ermittlung der benötigten Ressourcen wird daher für jede Mediawiki-Erweiterung ein Resource Loader-Modul definiert, welches verwendete Javascript- und CSS-Dateien enthält sowie Abhängigkeiten zu Mediawiki-Standardbibliotheken, wie z.B. jQuery.

Das Installieren einer Mediawiki-Erweiterung sowie deren Konfiguration, findet in der Datei „LocalSettings.php“[5] statt. Zunächst wird die Hauptdatei der Erweiterung in den allgemeinen Initialisierungsvorgang von Mediawiki mit eingebunden und ist somit installiert. Sollen allgemeine Parameter zur Konfiguration an die Erweiterung übergeben werden, geschieht dies durch globale PHP-Variablen, die ebenfalls in dieser Datei definiert werden können.

5.2 Systemarchitektur

Im vorherigen Abschnitt wurden die Grundlagen und die Möglichkeiten der Entwickelung mit Mediawiki erläutert, wodurch in diesem Abschnitt auf die konkrete Systemarchitektur, des für diese Diplomarbeit verwendeten Systems, eingegangen werden kann. In Abbildung 5.1 ist die Systemarchitektur des verwendeten Mediawiki-Stacks dargestellt. Es handelt sich dabei um eine zwischen Client und Server verteilte 4-Schichten Architektur.

Auf Seite des Clients wird zur Präsentation der Browser benutzt, welcher die vom Wiki berechneten Seiten rendert. Dabei sind Teile der Anwendungslogik in Form von Javascript und insbesondere jQuery, AJAX und Java-Applets auf den Client ausgelagert. Dabei wird Javascript mit jQuery v.a. zur Reaktion auf Benutzereingaben und der Anpassung aufwendiger Teile der Benutzeroberfläche verwendet. AJAX wird u.a. zum dynamischen Nachladen von Inhaltsbausteinen benutzt, um nicht bereits zu Beginn große Datenmengen vom Server übertragen zu müssen, wenn danach nur ein Bruchteil tatsächlich benutzt wird. Java-Applets werden u.a. für das Rendering von Graphen benutzt, die interaktiv auf Eingaben des Benutzers reagieren sollen, was mit normalen Bildern in dieser Art nicht möglich wäre.

Auf Serverseite besteht die Präsentationsschicht aus den dynamisch generierten Webseiten aus PHP-Aufrufen, die vom Mediawiki-Framework, welches sich auf Ebene der Anwendungslogik befindet, berechnet wird. Hier ist die Abgrenzung der Schichten nicht eindeutig gegeben, da im Rahmen von Mediawiki die Generierung von HTML und tatsächliche Berechnungen Hand in Hand gehen. Eine eindeutige Trennung ist dafür in Richtung der Datenhaltungsschicht gegeben, welche sich horizontal wiederum in eine Daten-Interface-Schicht und Datenschicht sowie vertikal in eine relationale Datenbank und einen Triplestore,

[5] http://www.mediawiki.org/wiki/Manual:LocalSettings.php

aufteilen lassen. Die Daten-Interface-Schicht besteht aus einem PHP-MySQL-Treiber und einer extra Komponente, dem DIQA Triplestore Connector. Dieser stellt die Verbindung zwischen der Anwendungslogikschicht von Mediawiki und dem Apache JENA Triplestore her. Die genaueren Zusammenhänge innerhalb der Mediawiki-Architektur werden im Folgenden Abschnitt ausführlicher beschrieben.
Die Verteilung der Schichten in Abbildung 5.1 auf lediglich Client und einen Server ließe sich dabei noch auf weitere Systeme ausweiten. Während Präsentations- und Anwendungslogik auf dem Server nicht getrennt werden können, könnte die Datenhaltungsschicht auf ein weiteres System ausgelagert werden. Die vertikale Teilung der Datenhaltungsschicht würde es ermöglichen, die relationale Datenbank sowie den Triplestore, auf zwei getrennte Datenbankserver zu verteilen, um eine bestmögliche Skalierung zu erreichen.

Abbildung 5.2 ermöglicht einen genaueren Einblick in den Aufbau der Anwendungslogik und Datenhaltungsschicht sowie in dazugehörige Zusammenhänge und Abhängigkeiten. Die Ausführung der Anwendungslogik geschieht auf einem Apache Http-Server, welcher über einen PHP Interpreter verfügen muss und in der Abbildung als äußerstes, oberes Rechteck dargestellt ist. Den Rahmen für die Anwendung bildet das Mediawiki-Framework,

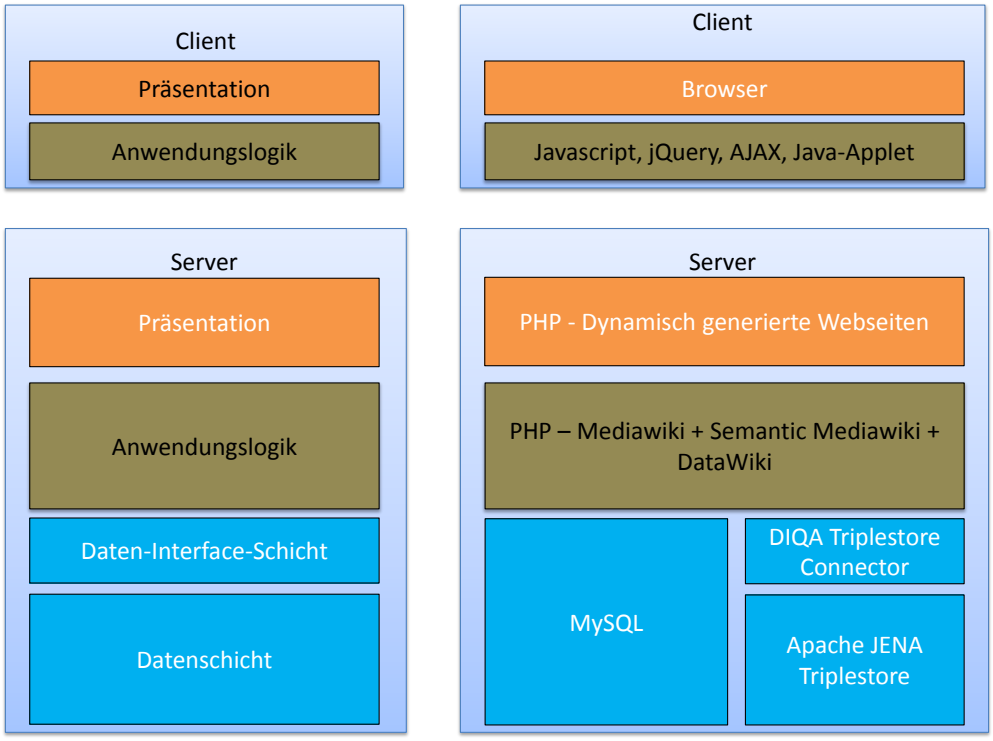

Abbildung 5.1: Systemarchitektur mit angebundenem Triplestore. Auf der linken Seite ist das allgemeine Schichtenmodell zu sehen und rechts die konkrete Ausprägung des Mediawiki-Stacks mit den verwendeten Technologien (nach [Kar11])

welches in der Abbildung als braunes Rechteck dargestellt ist. Es bietet eine große Vielfalt von möglichen Erweiterungspunkten, weswegen es viele Erweiterungen (*Extensions*) aufweist, die größtenteils als nicht kommerzielle Open Source-Projekte entwickelt wurden. Als Basis für DataWiki und das entwickelte Pattern Repository dient, das in der Abbildung als beiges Rechteck dargestellte, *Semantic Mediawiki*, welches Mediawiki um semantische Attribute und eine eigene Querysprache erweitert. Es handelt sich dabei um eine Mediawiki-Erweiterung, weswegen sich das Semantic Mediawiki-Rechteck in der Abbildung in dem hellen Rahmen für Extensions befindet. Mit Semantic Mediawiki als Basis wurden weitere Erweiterungen entwickelt, die dabei die möglichen Anwendungsgebiete der semantischen Attribute erweitern oder auch Teile des Semantic Mediawiki ersetzen. *DataWiki*, welches selbst wiederum aus einer Reihe von verschiedenen Erweiterungen besteht, erweitert Semantic Mediawiki um die Anbindung an einen Triplestore und um viele Komfortoptionen wie beispielsweise WYSIWYG-Eingabe von Wiki-Artikeln, ein Wiki Administration Tool zur komfortablen Installation von Erweiterungen aus einem Extension-Repository und weiteres. Die Abhängigkeit von DataWiki zu Semantic Mediawiki ist durch den gestrichelten Pfeil dargestellt, wobei die Abhängigkeit in Richtung des Pfeiles definiert ist. Die im Rahmen dieser Diplomarbeit

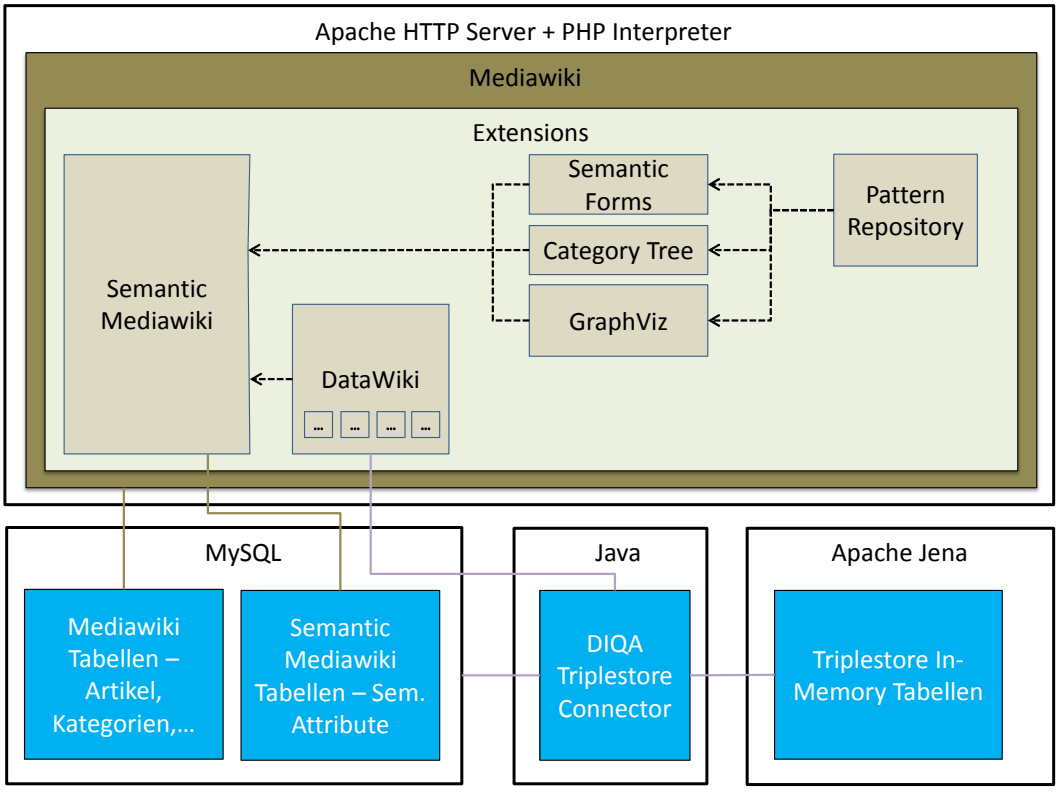

Abbildung 5.2: Aufbau von DataWiki. Legende: Gestrichelter schwarzer Pfeil=*Abhängigkeit*, Braune Linie=*Verbindung zu MySQL-Datenbanktabelle*, Lila Linie=*Für den Triplestore benötigte Verbindung*

entwickelte Erweiterung Pattern Repository wiederum besitzt Abhängigkeiten zu den Erweiterungen *Semantic Forms*, *Category Tree* und *GraphViz*, was wiederum durch die gestrichelten Pfeile in der Abbildung dargestellt ist. Welche Komponente von Pattern Repository dabei genau von welcher Erweiterung abhängig ist, wird in Abschnitt 5.5 genauer erläutert. Die verwendeten Erweiterungen wurden im Einzelnen bereits in Abschnitt 2.3.7 vorgestellt.

Die verschiedenen Verbindungen zu unterschiedlichen Datenbanken sind in Abbildung 5.2 anhand von farbigen Linien dargestellt. Die linke, braune Linie steht für die Verbindung von Mediawiki über die PHP-MySQL-Bibliothek zu den allgemeinen Tabellen in der relationalen MySQL-Datenbank. In diesen Tabellen sind beispielsweise Artikelinhalte, Kategorien, Namensräume oder auch Links der Artikel untereinander gespeichert. Bilder und sonstige Medien werden im Dateisystem gespeichert, so dass in der Datenbank lediglich Verweise darauf gespeichert werden müssen. Die Verbindung von Semantic Mediawiki zu den entsprechenden MySQL-Tabellen wird ebenfalls über die PHP-MySQL-Bibliothek hergestellt und ist in der Abbildung durch die rechte, braune Linie dargestellt. Dabei werden in der Mediawiki-Datenbank neue Tabellen für semantische Attribute, gecachte Ergebnisse von bereits ausgeführten Querys, URIs von Ontologie-Elementen und einige weitere Tabellen für interne Verwendungszwecke erstellt.

Sobald der DIQA Triplestore Connector mit dem Apache Jena Backend installiert wurde, gibt es die in der Abbildung durch lila Linien dargestellte Verbindungen. Die Verbindung des Wikis wird über die DataWiki-Komponente *Halo* hergestellt. Dabei agiert der Triplestore Connector als Webservice, der über eine RESTful Http-Verbindung aufgerufen wird und in der Abbildung als obere, in Lila dargestellte, Linie erkennbar ist. Beim Starten des Triplestores wird über einen Java-MySQL Connector (in der Abbildung dargestellt als die lila Linie zwischen Triplestore Connector und dem MySQL-Quadrat) zunächst ein komplettes In-Memory Abbild der MySQL-Datenbank erstellt. Dabei wird von dem relationalen Datenschema auf ein Tripelschema abgebildet, welches anhand der Verbindung zwischen dem Triplestore Connector und Apache JENA nun in dem für Triplestores typischen Graph-Format abgelegt wird. Wird in der Wiki-Anwendung eine Query auf einer Seite gespeichert, wird diese von der Halo Extension zunächst ins SPARQL-Format umgewandelt und dann über den Triplestore Connector an den Jena Triplestore geschickt, was in der Abbildung anhand der rechten lila Linie erkennbar ist. Dort wird mit Unterstützung eines Reasoners die Ergebnismenge berechnet und wiederum über den Connector an die Halo Extension zurück geschickt. Dort werden die Ergebnisse normalisiert, so dass das gleiche Ergebnisformat vorliegt, wie wenn die Query an die MySQL-Datenbank geschickt worden wäre.

5.3 Import des Datenmodells

In den beiden vorherigen Abschnitten wurden die Rahmenbedingungen erklärt, wie mit dem verwendeten System entwickelt werden kann und wie die zugehörige Systemarchitektur dazu aussieht. Damit kann nun auf die Implementierung des ersten Teils des Ergebnisses dieser Diplomarbeit eingegangen werden, dem Importer für das Datenmodell. Nachdem ein Datenmodell modelliert wurde, und ein entsprechend eingerichteter DataWiki-Stack zur

Verfügung steht, kann mit dem Import begonnen werden. Dies geschieht mithilfe eines Java-Programms, welches die Tripel aus den Datenmodell-Ontologien einliest und entsprechend parst, so dass die Zuordnung wie in Abbildung 4.13 dargestellt aus Abschnitt 4.4.3 geschieht. Die OWL-Ontologien sollten dabei in RDF/XML-Serialisierung vorliegen. Im Folgenden soll dargestellt werden, in welcher Reihenfolge das Parsing abläuft und welche Klassen daran beteiligt sind:

1. `DesignPatternWikiMaintenance`: Dies ist die Hauptklasse, die die *Main*-Methode enthält. Sie dient zum Verarbeiten der Eingabeparameter, wie z.B. Pfade zu den Ontologie-Dateien oder Login-Daten für das Wiki und organisiert das Zusammenspiel der einzelnen Hilfsklassen.

2. `WikiImporter`: Wurde als Wrapperklasse für *wiki-java*[6] geschrieben, um die Interaktion mit dem Wiki zu vereinfachen. Wiki-java ist ein Mediawiki-Bot-Framework, welches den Zugriff auf die von Mediawiki zur Verfügung gestellte API ermöglicht, und so das Erstellen und Editieren von Seiten im Wiki ermöglicht. Die Wrapperklasse `WikiImporter` wurde verfasst, um erweiterte Funktionalitäten, wie beispielsweise das Anhängen von Text an das Ende eines Artikels, zu ermöglichen. Beim Instanziieren der Klasse wird eine Verbindung zu dem Wiki mit zuvor angegebenen Logindaten hergestellt.

3. `TargetPropertiesImporter`: Im Fall, dass eine Ontologie mit Zieleigenschaften als Parameter an das Hauptprogramm übergeben wurde, dient diese Klasse zum Erzeugen der Kategorienhierarchie der Zieleigenschaften, zum Erzeugen der Zieleigenschaftenseiten im Namensraum *Ontology* sowie deren Vernetzung durch den semantischen Link *Has Subelement*. Darüber hinaus werden die semantischen Attribute erzeugt, welche zum Abbilden der n-ären Relation in Semantic Mediawiki nötig sind. Zudem werden die nötigen Daten für die Klasse `RepoImportUtil` gespeichert, die zur Generierung des Formulars und der Templates nötig sind. Zum Parsen der OWL-Ontologie wird die Jena RDF API benutzt [McB01]. Als Resultat erhält man Subjekt-Prädikat-Objekt Tripel, die sich je nach gesuchter Information weiterverarbeiten lassen. Die Hierarchie der Zieleigenschaften wird dabei zunächst in einem Graphen gespeichert, der anschließend rekursiv traversiert wird, um die Inhalte mithilfe der WikiImporter-Instanz in das Wiki zu übertragen.

4. `RepoImportUtil`: Diese Klasse dient zum Import der Ontologie, welche das Kerndatenmodell enthält. Die OWL-Daten werden ebenfalls mithilfe der Jena RDF API eingelesen und daraus werden Kategorienhierarchie, Konzeptseiten und semantische Attribute erzeugt. Anschließend wird die Klasse `TemplateFormImporter` verwendet, um die für das Patternformular benötigten Templates und das Formular selbst zu erzeugen. Falls zuvor bereits eine Zieleigenschaften-Ontologie eingelesen wurde, wird dafür ein zusätzliches Template erzeugt und die Eingabe in das Formular integriert. Zuletzt werden die Aufrufe zum Erzeugen von Visualisierungsgraphen auf den Wikiseiten eingetragen, falls dies in der Ontologie modelliert wurde.

[6] http://code.google.com/p/wiki-java/

5. `DatawikiCustomizer`: Diese Klasse dient hauptsächlich dazu, das Aussehen von DataWiki dem Benutzerinterface des Pattern Repositorys anzupassen. Dazu werden Einträge in der horizontalen oberen Hauptmenüleiste erzeugt, die Zugriffe auf einzelne Funktionalitäten ermöglichen. Darüber hinaus wird die Hauptseite mit Inhalt gefüllt, welche der Benutzer als Erstes beim Aufrufen des Wikis sieht und die Seite mit dem Inhaltsverzeichnis wird erzeugt.

5.4 Das Patternformular

Nachdem nun, mithilfe des im vorherigen Abschnitt beschriebenen Programms, das Datenmodell in das Wiki importiert wurde, steht nun das Patternformular zur Verfügung. Da dieses ein zentraler Bestandteil der Funktionalität zum Erstellen und Editieren von Patterns darstellt, soll es im folgenden Abschnitt detailliert beschrieben werden. Das Formular wird durch die Erweiterung *Semantic Forms* realisiert, wozu die Grundlagen des Funktionsumfangs in 2.3.7 beschrieben sind. Während es im Prinzip eine Eingabemaske für die Parameter der verwendeten Templates darstellt, sorgen insbesondere auf die Eingabedaten abgestimmte Eingabetypen für die nötige Benutzerfreundlichkeit bei der Bedienung des Formulars. In Abschnitt 4.4.1 werden anhand von Abbildung 4.10 die möglichen Eingabetypen beschrieben, wie sie im Datenmodell für das Wiki verfügbar sind. Es folgt eine Zuordnung zu den Semantic Forms Eingabetypen, auf welche die Eingabetypen beim Import aus der Ontologie im Formular abgebildet werden.

- Textarea: Entspricht direkt dem Semantic Forms-Eingabetyp *Textarea*, welcher ein einfaches, mehrzeiliges Feld zur Texteingabe darstellt. Als Parameter wird eine feste Größe übergeben, so dass Textarea-Felder und Semantic Textarea-Felder gleich groß im Formular dargestellt werden.
- Image: Entspricht dem Semantic Forms-Eingabetyp *Text with autocomplete*, welcher eine einzeilige Texteingabe ermöglicht und dabei per Autovervollständigung Werte vorschlägt. Welche Vorschläge gemacht werden sollen, wird über einen Parameter gesteuert, welcher als Eingabe die Anweisung erhält, alle bereits vorhandenen Bilder im System zur Autovervollständigung zu benutzen. Diese sind im Namensraum *File* gespeichert. Über den Parameter *uploadable* wird zudem erreicht, dass neben dem Texteingabefeld auch ein Link „Upload“ erscheint, der das Hochladen von neuen Bildern ermöglicht.
- ImageWithLabel: Zusätzlich zu dem Eingabetyp Image wird im Formular ein zusätzliches Feld zur Eingabe einer Bildbeschriftung erzeugt, welches den Semantic Forms-Eingabetyp *Text* verwendet. Als Eingabe wird ein einfacher Text erwartet, der im Template an der richtige Stelle als extra Parameter eingefügt wird.
- Category: Entspricht dem Semantic Forms-Eingabetyp *Category*, welcher die Erweiterung Category Tree benutzt, um die Auswahl einer Kategorie über einen Kategorienbaum zu ermöglichen. Als Parameter wird die Wurzel des Baumes benötigt, in diesem Fall die Kategorie *CategoryHierarchy*.

- Semantic Textarea: Wurde auf Basis des Semantic Forms-Eingabetyps Textarea entwickelt und ermöglicht das Annotieren von Semantischen Attributen direkt in einem Freitext. Details zur Implementierung werden in Abschnitt 5.5.1 erläutert.
- VarRelations: Obwohl *VarRelations* in der Ontologie als Eingabetyp für einen Abschnitt verfügbar ist, besteht die Darstellung im Formular aus drei Feldern sowie einem einzelnen Template nur für diesen Abschnitt. Für die Auswahl des zu annotierenden Relationstyps wird der neu entwickelte Eingabetyp *Property Dropdown* benutzt. Dieser erwartet als Parameter eine Konzeptseite, welche eine Aufzählung der zulässigen semantischen Relationstypen enthält, die dann zur Auswahl bereit stehen. Zur Eingabe des Zielpatterns wird ein Text with autocomplete-Feld benutzt, das seine Werte von einer Konzeptseite erhält, die alle verfügbaren Zielpatterns auflistet. Zur Angabe einer textuellen Erklärung der Verlinkung gibt es schließlich ein Textarea-Feld.
- TargetProps: Der *TargetProps*-Eingabetyp aus der Ontologie wird im Formular ebenfalls durch mehrere Felder repräsentiert sowie einem Template nur für diesen Abschnitt. Für die Auswahl der gewünschten Zieleigenschaft steht ein Dropdown-Menü zur Verfügung. Zusätzliche Attribute der Relation lassen sich über den Semantic Forms-Eingabetyp *Radiobutton* auswählen, wobei vorgegebene auswählbare Werte über den Parameter *Values* fest im Formular übergeben werden. Die Korrelation wird schließlich über den im nächsten Punkt genauer erläuterten Eingabetypen *Number Slider* ausgewählt.
- Number Slider: Dieser neu für Semantic Forms entwickelte Eingabetyp ermöglicht die Eingabe einer Zahl aus einem vorgegebenen Zahlenbereich heraus. Durch entsprechende Parameter kann der Zahlenbereich verändert werden, der zur Auswahl bereit stehen soll. Details zur Implementierung gibt es in Abschnitt 5.5.3.

Bevor die aufgelistete Zuordnung von Eingabetypen zu Feldern relevant wird, muss zunächst ein Template in das Formular eingebunden werden. Dies geschieht durch einen Aufruf über den Formularnamen, und die Zuordnung von Eingabetypen zu den im Template vorhandenen Feldern. Um es zu ermöglichen, ein Template beliebig oft in einem Formular zu *instanziieren* und dementsprechend den Template-Inhalt beliebig oft in das resultierende Dokument einzufügen, gibt es beim Aufruf eines Templates den Parameter *multiple*. Dieser Parameter bezieht sich jedoch immer auf ein komplettes Template und kann nicht für einzelnen Feldern definiert werden. Daraus resultiert die Notwendigkeit, die Templates für das Patternformular so aufzuteilen, dass mehrfach instanziierbare Inhalte gesonderte Templates erhalten. Dementsprechend erhalten die Eingabetypen VarRelations und TargetProps aus der Datenmodellontologie jeweils einzelne Templates.

5.5 Die Erweiterung - Pattern Repository

Der Funktionsumfang, welcher durch das entwickelte Pattern Repository zur Verfügung gestellt wird, besteht zunächst aus dem Datenmodell, welches in Kapitel 4.4 beschrieben wird, und dessen Import detailliert in Abschnitt 5.3 erläutert wurde. Mithilfe des Datenmodells als

Grundlage wurde nun möglichst viel Funktionalität durch bereits existierenden Mediawiki- und Semantic Mediawiki-Erweiterungen realisiert, insbesondere mit Semantic Forms. An einigen Stellen gab es jedoch Anforderungen, die dadurch nicht abgedeckt werden konnten, und diese fehlende Funktionalität wurde in Form der Erweiterung „Pattern Repository“ nachgeliefert bzw. eigenständig implementiert.

Die Integration der einzelnen Module der Pattern Repository Erweiterung geschieht über unterschiedliche Erweiterungspunkte, welche von Mediawiki bzw. den entsprechenden Mediawiki-Erweiterungen zur Verfügung gestellt werden. Eine Übersicht über die einzelnen Module, inklusive den benutzten Erweiterungspunkten und den Abhängigkeiten der einzelnen Teile, ist in Abbildung 5.3 dargestellt.

Die braunen Rechtecke auf der linken Seite stellen die verwendeten Erweiterungspunkte der einzelnen Erweiterungen dar. Eine Erklärung der allgemeinen Erweiterungsmechanismen gibt es bereits in Abschnitt 5.1.2. *SF Input Type* steht hier für *Semantic Forms Input Type* und stellt ein Interface dar, über welches es Semantic Forms erlaubt, beliebige neue Eingabetypen für die Formulare zu registrieren. Dazu muss lediglich die Klasse `SFFormInput` erweitert und ein von Semantic Forms angebotener Parser Hook verwendet werden, über welchen eigene Eingabetypen, aber auch neue Eingabetypen, eingelesen werden.

Die Rechtecke mit weißem Hintergrund in der Mitte repräsentieren jeweils den Ort, an dem das Modul eingesetzt wird. Dabei gibt es eine Gruppe, welche neue zusätzliche Eingabetypen für Semantic Forms bietet, zwei Module für die Verwendung auf „normalen“ Inhaltsseiten bzw. auch Kategorieseiten im Wiki und eine eigene Wiki-Spezialseite. Die orangen gestrichelten Pfeile, welche von den einzelnen Funktionsmodulen ausgehen, bedeuten Abhängigkeiten von anderen Mediawiki- oder Semantic Mediawiki-Erweiterungen und Javascript-Bibliotheken. Dabei stehen die dunkelgrünen Rechtecke für Mediawiki- und Semantic Mediawiki-Erweiterungen und die hellblauen Rechtecke für Javascript-Bibliotheken, wobei es sich außer bei jsTree bei den restlichen Bibliotheken um Module von jQuery handelt. jsTree [Boz] ist eine Bibliothek für die Darstellung von Bäumen mit Javascript. Um den Best Practises der Mediawiki-Entwicklung zu folgen [Medb], wurden die Abhängigkeiten der einzelnen Funktionsmodule in *Resource Modules* zusammengefasst, welche durch den Mediawiki Resource Loader verwendet werden können.

5.5.1 Annotationsunterstützung - Semantic Textarea

Die Anforderung, in einem bereits existierenden freien Text einzelne Wörter semantisch zu annotieren, ist in Semantic Mediawiki nur sehr unkomfortabel möglich. Dies soll anhand eines kleinen Beispiels demonstriert werden. Als Ausgangslage soll folgender Satz dienen:

```
Dieses Pattern stellt eine Spezialisierung von Pattern xyz dar.
```

Nun soll im Text eine `SpecialisationOf`-Verlinkung auf das Pattern *Xyz* stattfinden. Dabei darf auch nicht vergessen werden, dass in einem freien Fließtext das andere Pattern eventuell nicht exakt gleich benannt wurde, wie die tatsächliche Seite im Wiki für das Pattern. Mit den

mitgelieferten Mitteln von Semantic Mediawiki müsste die Annotation nun folgendermaßen geschehen:

```
Dieses Pattern stellt eine Spezialisierung von Pattern
[[SpecialisationOf::Xyz|xyz]] dar.
```

Durch den Ausdruck `SpecialisationOf::Xyz` wird die Verlinkung auf die korrekte Patternseite hergestellt, und durch den Zusatz `|xyz` wird nach wie vor „xyz“ im Fließtext angezeigt. Eine ausreichende Usability ist hier nicht gewährleistet. So erhält der Benutzer weder Unterstützung bei der Auswahl des semantischen Attributs (durch einen Tippfehler würde ein anderes Attribut verwendet werden), noch bei der Auswahl des korrekten Zielpatterns. Auch hier würde durch einen Tippfehler auf eine andere Entität verwiesen. Unter Berücksichtigung der Detailanforderungen, die sich aus der generellen Anforderung nach einem benutzerfreundlichen Annotationsmechanismus ergeben, ist ein neuer Eingabetyp für Semantic Forms entstanden, welcher *Semantic Textarea* genannt wurde.

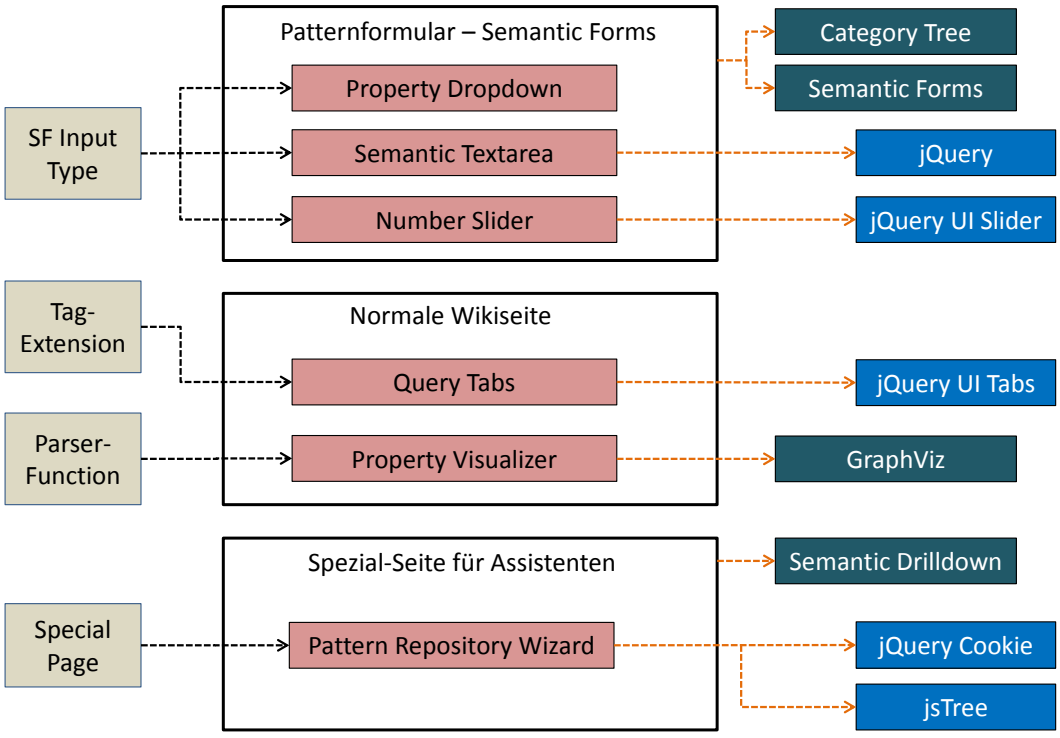

Abbildung 5.3: Interne Architektur der Erweiterung Pattern Repository. Legende: Schwarz gestrichelter Pfeil=*Zur Realisierung verwendet für*. Orange gestrichelter Pfeil=*Abhängigkeit*

Um ein Wort in einem Text zu annotieren, muss es zuerst mithilfe der Maus markiert werden, wie es in Abbildung 5.4 zu sehen ist. Nun kann der Button „Annotate“ betätigt werden, der das Annotationsinterface öffnet, welches in Abbildung 5.5 dargestellt ist.

Mithilfe des Dropdown-Menüs lassen sich semantische Attribute auswählen, welche zuvor im Datenmodell für den entsprechenden Abschnitt konfiguriert wurden. Auf diese Weise ist es nicht möglich, nicht existierende Attribute auszuwählen oder durch einen Tippfehler die Annotation nutzlos zu machen. In dem Textfeld kann schließlich ein Zielpattern ausgewählt werden, wobei der Benutzer hier mit einer Autovervollständigungsfunktion unterstützt wird.

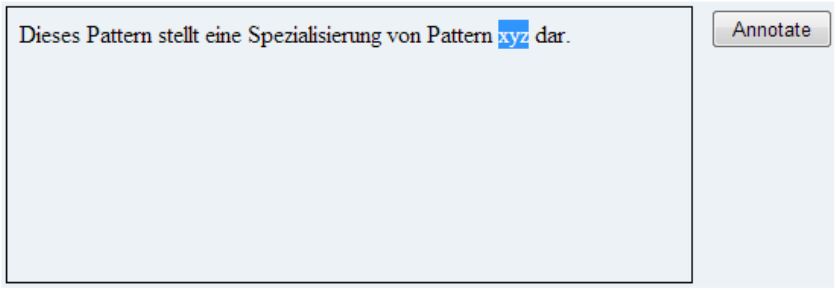

Abbildung 5.4: Auswahl des zu annotierenden Wortes in Semantic Textarea

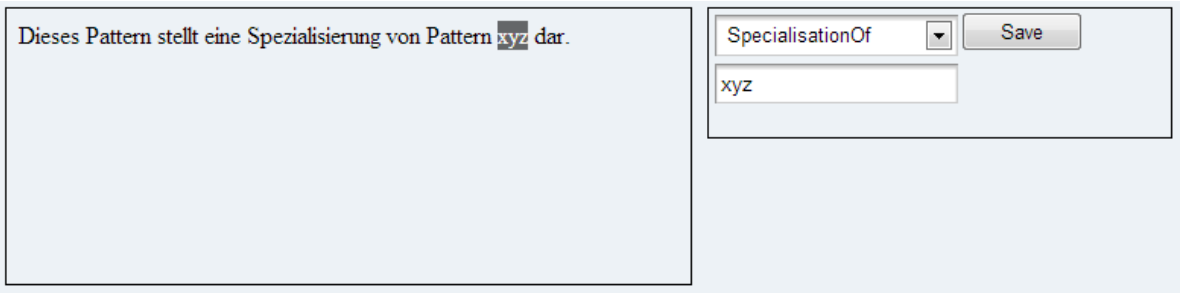

Abbildung 5.5: Annotation mithilfe von Semantic Textarea

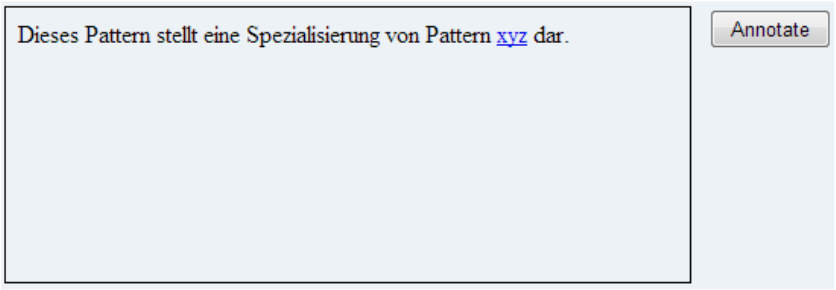

Abbildung 5.6: Anschließende Darstellung der Annotation in Semantic Textarea

Durch die Betätigung des Buttons „Save“ wird die Annotation gespeichert und auf gleiche Weise wie ein Hyperlink dargestellt, was in Abbildung 5.6 zu sehen ist. Das tatsächliche Ziel kann dabei auch jederzeit von dem annotierten Wort abweichen; das annotierte Wort muss trotzdem nicht verändert werden.

Die technische Umsetzung basiert auf einer Kombination eines iFrames, welches zur Darstellung des sichtbaren Textes und zur Verarbeitung von Eingabeevents mithilfe von jQuery dient und einer versteckten Textarea, welche den „tatsächlichen“ Text enthält, der im Wiki abgespeichert werden soll. Die Verwendung eines iFrames mit Aktivierung des `designMode=,,on‘‘`-Attributs zur Realisierung eines WYSIWYG-Editors ist ein gängiges Vorgehen, und dieser Teil der HTML-Spezifikation wird von folgenden Browsern in der jeweils neuesten Version unterstützt: Internet Explorer, Firefox, Opera, Safari und Chrome [Koc]. Die Verwendung des *Design Mode* ermöglicht es, auf einem ausgewählten Text innerhalb eines iFrames eine Liste von Befehlen über das Interface *execCommand* auszuführen, welche in [Koc] aufgelistet sind. Herausforderungen bei der Implementierung haben sich v.a. daraus ergeben, dass eine Art eigene „Markup-Sprache“ entwickelt werden musste, so dass zwar mithilfe des iFrames lediglich die Möglichkeiten von normalem HTML zur optischen Darstellung der Annotation ausreichen, im Endeffekt jedoch ein Text entsteht, der der Markup-Sprache von Sematic Mediawiki entspricht und somit vom System erkannt wird. Dieses Hinzufügen und Entfernen der Markupsprache bei der Übertragung von Text zwischen dem iFrame und der versteckten Textarea wird mithilfe von jQuery-Events realisiert.

Während dem Eintippen von Text wird nach jedem getippten Buchstaben der komplette Text übertragen und in die versteckte Textarea gespeichert. Ebenso wird bei jeder Übertragung die HTML-Darstellung aus dem iFrame in die Semantic Mediawiki-Markupsprache umgewandelt. Der gleiche Übertragungs- und Umwandlungsvorgang geschieht bei jeder Annotation, die mithilfe des Save-Buttons abgeschlossen wird.

Das Erstellen aller relevanten Elemente geschieht auf der Seite des Servers durch PHP, wobei bereits das Dropdown-Menü mit den semantischen Attributen gefüllt wird. Welche semantischen Attribute in dem Dropdown-Menü zur Verfügung stehen, wird dabei über eine Konzeptseite gesteuert. Dafür gibt es ein festes Mapping zwischen dem Namen des Abschnitts und einer Konzeptseite, welche nach folgendem Schema benannt wird: `Concept:Abschnittsname_Section`. Auf dieser Seite befindet sich eine Query, welche beim Import des Datenmodells angelegt wurde:

```
[[Property:+]] [[AllowedSection::Related Patterns]]
```

Diese Query sucht nach allen semantischen Attributen, die ihrerseits wiederum mit einem semantischen Attribut `AllowedSection` annotiert wurden. Dies dient zur Angabe für welche Abschnitte das Attribut zugelassen ist. Um schließlich das Ergebnis dieser Query für das Dropdown-Menü zu erhalten, wird eine Hilfsfunktion von Semantic Forms benutzt. CSS-relevanten Optionen wurden, um eine stärkere Modularisierung und einfacher wartbaren Code zu erhalten, in eine extra Datei ausgelagert. Die Textarea und das Annotationsfenster werden per `display: none` ausgeblendet, wobei bei der Betätigung des Annotate-Buttons

der Annotate-Button ausgeblendet wird und dafür das Annotationsfenster eingeblendet. Durch die Aktivierung des Save-Buttons wird der Vorgang wieder rückgängig gemacht.

Es folgt eine Übersicht, auf welche Dateien die Funktionalität verteilt ist:

- `includes/PAREP_SemanticTextarea.php`: Abgeleitet von der Semantic Forms-Klasse `SFFormInput`, dient zur Generierung des statischen HTML-Codes mit PHP-Funktionen und zur Füllung des Dropdown-Menüs. Hinzufügen einer Javascript-Initialisierungsfunktion mithilfe der Semantic Forms-Funktion `addJsInitFunctionData`, die ausgeführt wird, sobald der DOM-Tree geladen ist.
- `libs/semantictextarea.js`: Über eine Initialisierungsfunktion werden die nötigen Javascript-Event Listener mithilfe von jQuery-Selektoren an die entsprechenden HTML-Elemente angehängt, um dynamische Funktionalität zu realisieren. Umsetzung der Markup-Konversion von HTML nach Semantic Mediawiki-Markup und umgekehrt, wobei als Basis hier reguläre Ausdrücke benutzt werden, um Ausdrücke aus dem iFrame in Wikitext zu konvertieren.
- `skins/PAREP_SemanticTextarea.css`: Dient zur Steuerung welche Elemente versteckt sein sollen und welche angezeigt werden sollen. Korrekte Positionierung der Annotationsbox neben dem Texteingabefeld.

Wie die Funktionalität nun in Semantic Forms integriert wurde, wurde bereits in Abschnitt 5.5 beschrieben.

5.5.2 Auswahl semantischer Propertys - Property Dropdown

Das Property Dropdown-Menü wurde entwickelt, um eine komfortable semantische Verlinkung zu anderen Patterns zu ermöglichen. Dies stellt eine wichtige Anforderung dar, da die Vernetzung der Patterns zu einer Patternsprache als Kernaspekt vieler Patterndomänen betrachtet werden kann. In Semantic Forms existiert zwar ein Eingabetypen, welcher ein Dropdown-Menü zur Verfügung stellt, allerdings verfügt dieser nicht über alle benötigten Features.

Zum Auffinden der erlaubten semantischen Linktypen von dem Abschnitt ausgehend, in dem das Dropdown-Menü verwendet wird, soll eine Query ausgeführt werden. Diese Query befindet sich auf einer Konzeptseite, die entsprechend den Namen des Abschnitts trägt und während des Imports des Datenmodells erzeugt wurde. Für den Abschnitt *Related Patterns* würde die Konzeptseite beispielsweise „Concept:Related_Patterns_Section“ heißen. Würde nun der Standardeingabetyp von Semantic Forms verwendet, würden die gefundenen Linktypen jeweils ihren Namensraum vor ihrer Bezeichnung tragen, z.B. „Property:Combining“. Dies liegt daran, dass es keiner der normalen Anwendungsfälle von Semantic Forms ist, dynamisch einen semantischen Linktyp auszuwählen. Normalerweise wurden diese zuvor bereits statisch erzeugt, sodass anhand der Formulare lediglich Inhalte für die Linktypen eingegeben werden. Die dynamische Verwendung wird nun möglich, indem der Name der Property selbst einen Parameter in dem zugehörigen Template darstellt, ebenso wie der zugewiesene Wert. Für die Erzeugung einer gültigen semantischen Annotation der Form

`[[property::value]]` durch das Template ist es nötig, den Namensraum *Property* in dem Bezeichner zu entfernen.

Zu diesem Zweck wurde der Semantic Forms Dropdown-Eingabetyps `SFDropdownInput` so erweitert, dass als zusätzlicher Parameter angegeben werden kann, ob ein eventuell in den Einträgen des Menüs enthaltener Namensraum entfernt werden soll. An anderer Stelle wurde überflüssige Funktionalität entfernt, um den Programmcode schlank zu halten und da für allgemeine Zwecke weiterhin das Semantic Forms Dropdown-Menü verwendet werden sollte. Da das Menü wie ein Standard Dropdown-Menü aussieht, folgt keine Abbildung, sondern direkt die Erklärung, an welcher Stelle die PHP-Datei mit der Implementierung liegt.

- `includes/PAREP_PropertyDropdown.php`: Abgeleitet von der Klasse `SFFormInput`. Generiert den statischen HTML-Code für das Menü und führt die nötigen String-Operationen durch.

5.5.3 Zahlenauswahl - Number Slider

Der Number Slider wurde aufgrund der Anforderung entwickelt, Zieleigenschaften annotieren zu können. Wie in Abschnitt 4.2.2 beschrieben, ist es bei der Annotation von Zieleigenschaften erwünscht, die Relation mit zusätzlichen Attributen versehen zu können. Neben textuellen Attributen, welche im Formular über ein normales Textfeld eingegeben werden können, ist ein denkbarer Anwendungsfall die Angabe eines Korrelationswerts zwischen einem Pattern und einer Zieleigenschaft. Dieser Korrelationswert entspricht einer prozentualen Angabe von 0 bis 100. Um eine schnelle, nicht notwendigerweise sehr präzise Eingabe zu machen, biete sich hier ein Regler sehr gut an. Die Umsetzung ist in Abbildung 5.7 dargestellt, wobei es, wie in der Abbildung anhand des kleinen Textfeldes erkennbar, neben der Eingabe durch den Regler auch die Möglichkeit gibt, direkt eine Zahl einzutippen.

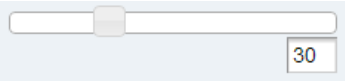

Abbildung 5.7: Eingabe eines Zahlenwerts durch den Number Slider

Da es sich bei dem Number Slider um einen Eingabetypen handelt, wurde er als zusätzlicher Eingabetyp zu Semantic Forms entwickelt. Es besteht eine Abhängigkeit zu Semantic Forms, was wiederum auch eine vollständige Integration ermöglicht. Zur Darstellung des Schiebereglers selbst wurde die jQuery UI-Bibliothek *Slider*[7] benutzt. Eine Besonderheit bei der technischen Umsetzung stellte die Anpassung des Schiebereglers für die *multiple Instance-Template*-Funktion von Semantic Forms dar. Diese Funktion ermöglicht es, wie bereits

[7] http://jqueryui.com/slider/

genauer in Abschnitt 5.4 beschrieben, ein Template mehrmals innerhalb eines Formulars zu instanziieren und so eine variable Anzahl von gewissen Inhaltsbestandteilen zu ermöglichen. Gerade bei der Annotation von Zieleigenschaften ist dies nötig, da es sein kann, dass der Benutzer gar keine Zieleigenschaften annotieren möchte oder eine beliebige große Anzahl. Das Erzeugen des jQuery-Elements *Slider* geschieht über eine Instanziierungsfunktion, welche zum Zeitpunkt des Erzeugens des Eingabetyps an das, den Eingabetyp umgebende, Div-Element angehängt wird. Um den Slider im richtigen Div-Element zu speichern, wird beim Erzeugen des HTML-Codes für den Eingabetyp eine statische ID vergeben. Die Instanziierung neuer Inhaltselemente wird so durchgeführt, dass zunächst ein Inhaltselement als Prototyp erzeugt und unsichtbar gemacht wird. Bei jeder Instanziierung wird nun eine Kopie des Prototyps erstellt, und im Rahmen der Instanziierungsfunktion wird eine neue ID für Semantic Forms-eigene Eingabetypen berechnet und zugewiesen. Da dies leider nicht dokumentiert ist, musste dieses Verhalten durch die Analyse des Semantic Forms-Code herausgefunden werden. Danach konnte die Instanziierungsfunktion des Number Sliders so angepasst werden, dass auch er eine neue eindeutige ID erhält, und so auch bei mehreren Instanzen des Schieberegler immer genau die ausgewählten Werte in der vom Formular generierten Seite gespeichert werden.

Es folgt eine Übersicht, auf welche Dateien die Funktionalität verteilt ist:

- `includes/PAREP_NumberSlider.php`: Abgeleitet von der Klasse `SFFormInput`. Generiert den statischen HTML-Code und hängt die Instanziierungsfunktionen an den Eingabetyp an.
- `libs/numberslider.js`: Erstellung des jQuery Sliders in dem dafür vorgesehenen Div. Erzeugung von dynamischen IDs bei multiple Instance-Templates.
- `skins/PAREP_NumberSlider.css`: Einbindung des für die jQuery-Bibliothek nötigen CSS-Codes.

5.5.4 Ausblendbare Querys - Query Tabs

Um die Navigation durch das Pattern Repository entlang der Patternsprache möglichst intuitiv zu gestalten, ist es nötig, vorhandene semantische Verlinkungen maximal auszunutzen, ohne den Endnutzer dabei durch zu viele Informationen zu verwirren. Die Patternsprache ist durch die semantischen Verlinkungen der Patterns zueinander in Form der semantischen Daten abgelegt und kann nun durch Querys abgerufen werden. Wenn nun pro Pattern eine große Informationsvielfalt vorhanden ist, könnte schnell die Übersichtlichkeit verloren gehen. Dies hätte den Effekt, dass der Benutzer noch unsicherer ist, wie er weiter navigieren soll. Daher muss ein Kompromiss zwischen dem Ausnutzen der Informationsvielfalt und dem Verhindern der Überforderung des Benutzers angestrebt werden. Ausblendbare Querys, die sich auf normalen Seiten integrieren lassen, bieten sich hierfür an.

In Abbildung 5.8 ist die Leiste von ausgeblendeten Querys zu sehen, die sich durch einen Klick auf die gewünschte Query öffnen lassen und ganz am Ende eines Patterns eingeblendet werden soll. Dabei können die Reiter der einzelnen Querys frei benannt werden, und die

Querys selbst sind beliebig komplexe einbettbare Semantic Mediawiki- oder SPARQL-Querys. In Abbildung 5.9 ist eine aufgeklappte Query mit einer einfachen Ergebnistabelle zu sehen, welche durch einen wiederholten Klick auf den entsprechenden Reiter auch wieder geschlossen werden kann.

Abbildung 5.8: Ausgeblendete Querys

Abbildung 5.9: Eingeblendete Query

Die technische Umsetzung von Query Tabs beruht auf einer Erweiterung der Wiki Markupsprache und benutzt zur Visualisierung die jQuery UI-Bibliothek *Tabs*. Es werden zwei neue Markupbefehle eingeführt, `<tabs>` als äußere Umgebung für Tabs und `<tab>` um den tatsächlichen Inhalt eines Tabs abzulegen, wobei zusätzlich noch ein Tabtitel angegeben werden kann. Die größte Schwierigkeit bei der Implementierung stellte die Verkettung der nötigen Befehle für die Übergabe an den Mediawikiparser dar. Zunächst muss nach dem Öffnen einer <tabs>-Umgebung der Mediawikiparser noch einmal aufgerufen werden, um die enthaltenen <tab>-Bestandteile zu erhalten und die damit verknüpften Funktionen aufzurufen. Zuvor muss ein Div-Container mit dem von jQuery Tabs benötigten Datenmodell erzeugt werden, in welchen die Tabs schließlich ihre Inhalte einfügen können. Bei der Verarbeitung der einzelnen Tabs wiederum muss darauf geachtet werden, dass der Parser den Inhalt nach Parserfunktionen durchsucht, welche Querys enthalten, so dass am Ende der Benutzer auch ein Queryergebnis angezeigt bekommt und nicht nur die Query als Text. Das Ergebnis wird dynamisch in den Div-Container eingefügt und zuletzt wird aus der fertigen Datenstruktur eine jQuery Tabs-Leiste erzeugt. Eine weitere Schwierigkeit stellte die Anwendung von Javascript-Initialisierungsfunktionen auf bestimmte Elemente des DOMs dar, deren vollständige IDs erst zur Laufzeit im PHP-Code erzeugt werden, indem an ein festes Präfix eine einzigartige MD5-Hashzahl angehängt wird. Dafür wurde die Lösung gewählt, dass an die Elemente des Div-Containers der Tableiste mithilfe von jQuery dynamisch Initialisierungsfunktionen angehängt werden, welche die richtigen IDs der Elemente enthalten. Nach dem Laden des Dokuments wird mithilfe von statischem Javascript nach Divs mit dem entsprechenden Präfix in der ID gesucht und alle angehängten Funktionen werden ausgeführt. Einen ähnlichen Mechanismus implementiert Semantic

Forms, so dass bei der Implementierung der anderen Semantic Forms Eingabetypen für das Pattern Repository dieses Problem nicht auftrat.

Es folgt eine Übersicht, auf welche Dateien die Funktionalität verteilt ist:

- `includes/PAREP_QueryTabs.php`: Enthält die Tab-Renderfunktionen, welche an die Aufrufe des Mediawikiparsers gekoppelt sind, sobald die entsprechenden Markup-Befehle geparst werden.
- `libs/querytabs.js`: Aufruf der an die Elemente angehängten Initialisierungsfunktionen. Erstellen des jQuery Tabs-Elements
- `skins/PAREP_QueryTabs.css`: Einbindung des für die jQuery-Bibliothek nötigen CSS-Codes.

5.5.5 Visualisierung von semantischen Prädikaten im Kontext von Kategorien - Property Visualizer

Zur Gewinnung eines entsprechenden Mehrwerts aus der semantischen Annotation von Patternrelationen bietet es sich an, diese neu beschriebenen Informationen zu visualisieren, um sie für den Menschen greifbarer zu machen und besser verständlich aufzubereiten. Wie bereits in Abschnitt 4.2.1 beschrieben, dienen solche Relationen der Patterns untereinander dem Aufbau einer Patternsprache, wodurch man durch die Visualisierung der Relationen eine Visualisierung des Aufbaus der Patternsprache erhält. Dies sind sowohl Anforderungen des Pattern Autors als auch des Endnutzers.

Hierfür wurde nun eine Erweiterung entwickelt, die es ermöglicht, unter Angabe einer Kategorie und eines semantischen Attributs, die Vernetzung der Patterns aller Unterkategorien durch die semantischen Links als gerichteten Graphen darzustellen. Ein mögliches Beispiel für einen solchen Graphen ist in Abbildung 5.10 dargestellt. Die grauen Rechtecke repräsentieren jeweils eine Kategorie von Patterns, wobei der weiße Schriftzug die Bezeichnung der Kategorie darstellt und die Patterns selbst in schwarz geschrieben sind. Die Pfeile stehen für semantische Links entsprechend dem Attribut, welches zuvor ausgewählt wurde. Voraussetzungen, um ein solches Ergebnis zu erhalten, sind die Gliederung einer Patterndomäne in Unterkategorien sowie die Existenz von gerichteten semantischen Links.

Die technische Umsetzung wurde durch das Hinzufügen einer Mediawiki-Parserfunktion realisiert, welche eine korrekte Eingabe im DOT-Format für die Mediawiki-Erweiterung GraphViz produziert, die bereits in Abschnitt 2.3.7 vorgestellt wurde. Die Parserfunktion `#pareptree` nimmt als Parameter eine Kategorie und das semantische Attribut, das visualisiert werden soll, entgegen. Nun wird zunächst intern ein mehrdimensionales Array der Kategorien, Unterkategorien und der dazugehörigen Seiten erstellt sowie ein mehrdimensionales Array, welches ein Array von ausgehenden Links pro Seite speichert. Mithilfe dieser Informationen wird ein mit `<graphviz>` umgebener Text im DOT-Format erstellt, der den abzubildenden Graph beschreibt. Damit als Resultat schließlich nicht dieser Text, sondern die gewünschte Grafik dargestellt wird, ruft die Property Visualizer-Parserfunktion den Mediawiki-Parser auf, der Wikitags auflösen soll. Dadurch wird die Erweiterung *GraphViz*

aufgerufen, welche ihrerseits wiederum davon abhängig ist, dass auf dem Server Graphviz installiert ist. Graphviz erzeugt eine Bilddatei, die ins resultierende Dokument eingebunden wird.

Es folgt eine Übersicht, auf welche Dateien die Funktionalität verteilt ist:

- `includes/PAREP_PropertyVisualizer.php`: Extraktion der Eingabeparameter, Aufruf der PAREPSemPropTree-Klasse und Aufruf des Mediawiki-Parsers.
- `includes/util/PAREP_SemPropTree.php`: Mithilfe von PAREPTreefunc wird ein Baum erstellt und schließlich in das Graphviz-Format geparst.
- `includes/util/PAREP_Treefunc.php`: Enthält nötige Logik für den komfortablen Umgang mit Bäumen.
- `languages/PAREP.i18n.magic.php`: Registrierung eines einzigartigen Namens für die Parserfunktion. Dabei ist es möglich, den Funktionsnamen an die jeweilige Sprache des Wikis anzupassen.

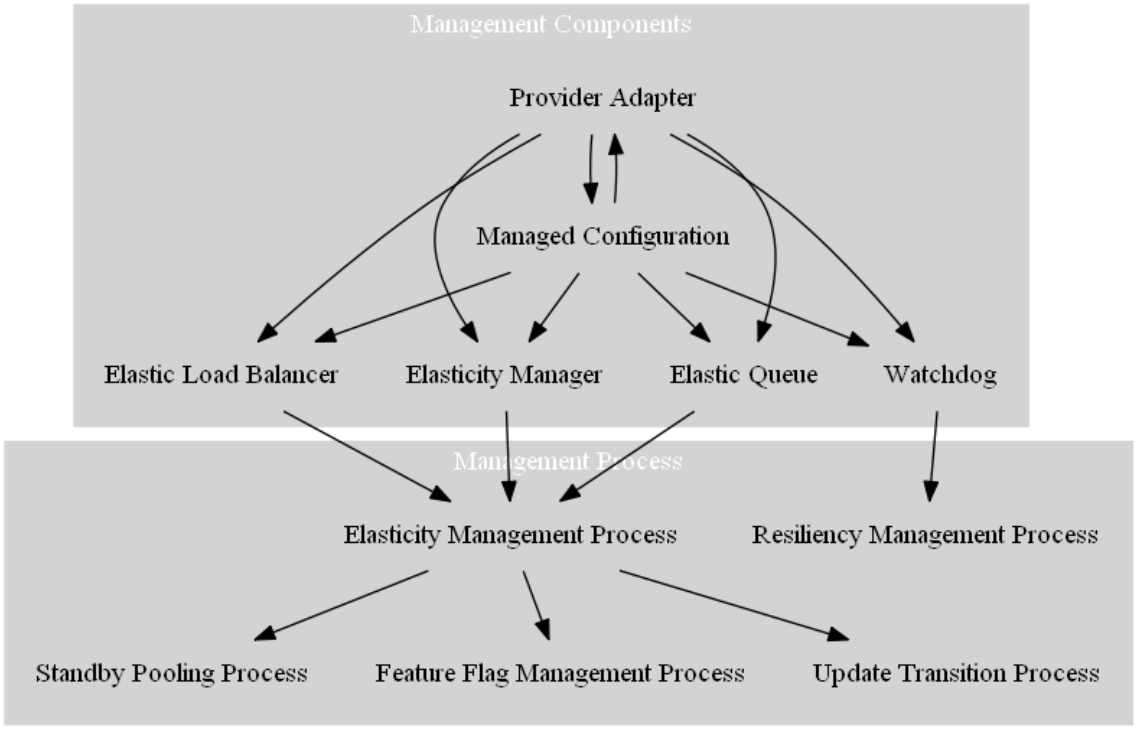

Abbildung 5.10: Visualisierung eines semantischen Linktyps durch den Property Visualizer

5.5.6 Suchassistent - Wizard

Als Kern für die Benutzung des entwickelten Pattern Repositorys kann der Suchassistent bzw. Wizard gezählt werden. Sobald das Repository zu einer Patternsprache vernetzte und durch semantische Annotationen angereicherte Patterns enthält, sind verschiedene Szenarien für die Führung des Benutzers durch Empfehlungen möglich. Dabei ist der Wizard insbesondere dann nützlich, wenn aufgrund der großen Komplexität einer Domäne der Endnutzer auf Probleme bei dem Einstieg in die Patterndomäne stößt oder Schwierigkeiten bei der Navigation durch die Patternsprache hat. Wegen diesen unterschiedlichen Anwendungsfällen wurde die Grundstruktur des Wizards so gewählt, dass der Benutzer zunächst auswählen kann, welche Art der Unterstützung benötigt wird bzw. welche Art von Vorschlägen erwünscht ist. Das Ziel liegt jedoch immer darin, dem Benutzer möglichst sinnvolle Patternvorschläge zu bieten. In dem Fall, dass der Benutzer eine ganze Liste von Vorschlägen erhält, die zwar alle interessant für ihn sind, aber die er nicht gleichzeitig bearbeiten kann, gibt es die in Abschnitt 4.3.2 beschriebene „Leseliste“. Diese stellt eine zusätzliche Funktion des Wizards dar und ermöglicht das Vormerken von Patterns, die der Benutzer später noch lesen möchte. Darüber hinaus lässt sich auf diese Weise ein System von Patterns anhand der Leseliste zusammenstellen, auf deren Basis weitere Empfehlungen für Patterns möglich sind, die sich mit den in der Liste vorhandenen Pattern kombinieren lassen.

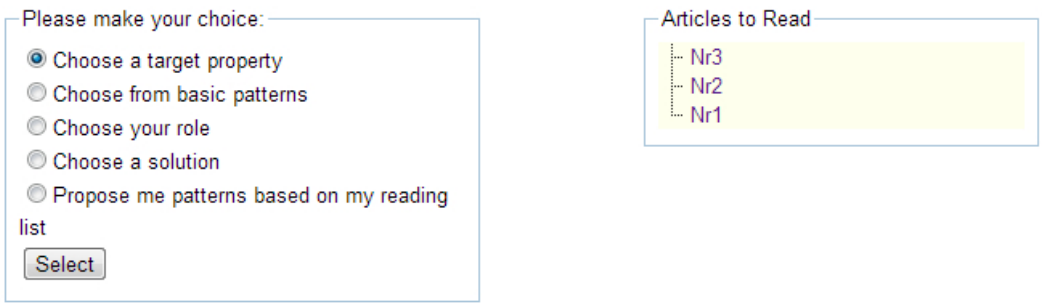

Abbildung 5.11: Das Hauptmenü des entwickelten Pattern Wizard - Links das Auswahlmenü der Empfehlungsfunktionen und auf der rechten Seite die Leseliste

In Abbildung 5.11 ist das Hauptmenü des Wizards dargestellt. Auf der linken Seite gibt es die verfügbaren Empfehlungsfunktionen, bei welchen eine getätigte Auswahl mit dem Button *Select* bestätigt werden kann. Auf der rechten Seite ist die Leseliste, mit einigen Beispielpatterns als Inhalt, dargestellt. Im Folgenden soll auf die Umsetzung der einzelnen Funktionen eingegangen werden. Bei der Beschreibung des konzeptionellen Entwurfs in Kapitel 4 wurde in Abschnitt 4.3.1 bereits erläutert, welche unterschiedlichen Einstiegsmöglichkeiten in eine Patterndomäne in einem Pattern Repository vorhanden sein sollten. Der Einstieg über grundlegende Patterns ist über den Menüpunkt *Choose from basic patterns* erreichbar und der Einstieg anhand von Benutzerrollen über den Menüpunkt *Choose your role*. In Abschnitt 4.3.2 wurden die weiteren Funktionen entworfen: *Choose a target property* ermöglicht die Empfehlung von Patterns anhand von ausgewählten Zieleigenschaften und

Choose a solution die Empfehlung anhand von bekannten Anwendungen. Der Menüpunkt *Propose me patterns based on my reading list* erlaubt die ebenfalls in Abschnitt 4.3.2 beschriebene Empfehlung aufgrund von vorhandenen Patterns in der Leseliste.

Der gesamte Wizard wurde in Form einer Mediawiki Spezial-Seite umgesetzt, die über URL-Parameter[8] gesteuert wird. Die genau Funktionsweise von Spezial-Seiten wurde bereits in Abschnitt 5.1.2 erläutert. Das Hauptmenü mit dem Select-Button wurde dabei durch ein HTML-Formular umgesetzt, welches bei Betätigung des Buttons eine GET-Anfrage sendet und die entsprechende Auswahl an die URL anhängt. Domänenspezifische Angaben über Kategorien und ähnliche von der Domäne abhängige Anpassungen können in der Mediawiki-LocalSettings.php vorgenommen werden. Ein großer Teil der sonstigen nötigen Benutzereingaben wurde über die auf jQuery basierende Javascript-Bibliothek jsTree [Boz] realisiert. Diese ermöglicht es, unter anderem aus JSON-Daten, eine Baumstruktur zu erzeugen, wobei Labels und hinterlegte Links frei wählbar sind. Sollen anhand der Auswahl in einem jsTree dynamische Empfehlungen angezeigt werden, wird zur Laufzeit eine SPARQL-Query aus der Benutzerauswahl erzeugt. Diese wird anschließend per AJAX-Anfrage an den Triplestore gesendet und das Ergebnis als Tabelle angezeigt. Zudem gibt es die Möglichkeit, Einträge der Ergebnistabelle per Drag & Drop in die Leseliste einzufügen. Es folgt eine kurze Beschreibung der einzelnen Funktionen und deren technische Umsetzung:

Auswahl eines bekannten Anwendungsfalls In Abbildung 5.12 ist die Empfehlungsfunktion anhand von bekannten Anwendungsfällen dargestellt. Die auf der linken Seite dargestellte Auswahl der Anwendungsfälle ist über jsTree realisiert und zeigt zunächst die Kategorienhierarchie an, die sich unter der in der „LocalSettings.php“ dafür konfigurierten Überkategorie befindet. Nun lassen sich anhand von Checkboxen mehrere Anwendungsfälle auswählen, an welchen der Benutzer interessiert ist. Bei Betätigung des *Propose Patterns*-Buttons wird per AJAX-Aufruf eine PHP-Funktion des Wizards aufgerufen, mit deren Hilfe dynamisch eine SPARQL-Query erzeugt wird. Die Query ruft alle Patterns ab, die eine *Implemented By*-Beziehung zu einem der ausgewählten Anwendungsfälle haben. Die fertig generierte Query wird in Textform über einen Aufruf an den Mediawiki-Parser zu dem Triplestore geschickt. Sobald der asynchrone Aufruf ein Ergebnis zurückliefert, wird dieses in Form einer Ergebnis-Tabelle dem Endbenutzer angezeigt. Auf die Tabelle wird zusätzliche eine Javascript-Funktion ausgeführt, die ihr eine bestimmte HTML-Klasse zuordnet. Aufgrund dieser Klasse ist es nun möglich, per Drag & Drop Ergebnisse aus der Tabelle in die Leseliste einzufügen.

Auswahl von Zieleigenschaften Die Auswahl von Zieleigenschaften ist weitestgehend gleich umgesetzt wie die Auswahl von bekannten Anwendungsfällen, weswegen hier keine zusätzliche Abbildung gezeigt werden soll. Die Umsetzung unterscheidet sich lediglich dadurch, dass die Überkategorie durch einen anderen Parameter gegeben ist und eine andere Query erzeugt wird. Pro Zeile wird ein Pattern sowie alle Zieleigenschaften angezeigt, mit welchen es annotiert wurde. Die möglichen Zieleigenschaften hängen dabei von der Auswahl der jsTree-Einträge durch den Benutzer ab. Zusätzlich gibt es für

[8] http://www.mediawiki.org/wiki/Manual:Parameters_to_index.php

jede Zieleigenschaft eine Spalte, welche die jeweilige Korrelation der Patterns zu dieser Zieleigenschaft anzeigt. Die letzte Spalte zeigt den berechneten Durchschnittswert aller Zieleigenschaftenkorrelationswerte eines Patterns an. Anschließend wird das Ergebnis anhand dieses Durchschnittswertes sortiert, so dass der Benutzer das Pattern an erster Stelle erhält, welches am meisten mit seiner Auswahl von Zieleigenschaften korreliert.

Auswahl eines grundlegenden Patterns In Abbildung 5.13 ist die Auswahl eines grundlegenden Patterns zu sehen, wobei die gezeigten Cloud Computing Patterns aus [FLR+13] stammen. Es gibt keine Auswahlmöglichkeiten für den Benutzer, stattdessen wird eine in der „LocalSettings.php"-Datei per Parameter festgelegte Kategorie von Patterns angezeigt. Dabei können die Patterns entweder direkt vom Benutzer angeklickt werden, um auf die Seite des entsprechenden Patterns zu gelangen oder per Drag & Drop in die Leseliste eingefügt werden.

Auswahl einer Benutzerrolle In Abbildung 5.14 ist der Auswahldialog zu sehen, der erscheint, falls sich der Benutzer für die Empfehlung anhand einer Benutzerrolle entschie-

Abbildung 5.12: Auswahl eines bekannten Anwendungsfalls im Wizard - Auf der linken Seite geschieht die Auswahl, in der Mitte werden die Empfehlungen angezeigt und rechts unten ist die Leseliste zu sehen

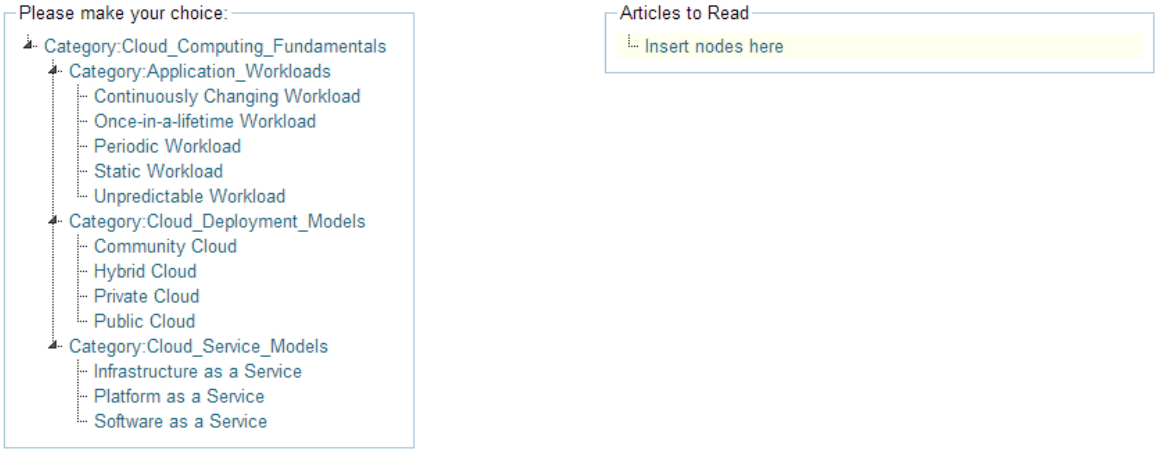

Abbildung 5.13: Auswahl grundlegenden Patterns im Wizard - Auf der linken Seite geschieht die Auswahl und rechts ist die Leseliste zu sehen. Die abgebildeten Beispielspatterns sind aus [FLR+13]

den hat. Nach der Bestätigung der Auswahl per Select-Button werden dem Benutzer die für diese Rolle empfohlenen Patterns, anhand eines jsTrees wie bei der Auswahl der grundlegenden Patterns, angezeigt. Die in dem Baum angezeigten Kategorien hängen dabei von den Empfehlungen für die jeweilige Rolle ab, wobei mehrere Kategorien auf Wurzelebene möglich sind.

Abbildung 5.14: Auswahl einer Benutzerrolle im Wizard - Auf der linken Seite geschieht die Auswahl und rechts ist die Leseliste zu sehen

Vorschlag aufgrund der Leseliste In Abbildung 5.15 ist die Empfehlungsfunktion der Leseliste dargestellt. Dabei ist in der Abbildung bereits ein Beispiel für ein Ergebnis sichtbar, welches nach der Betätigung des Propose Patterns-Buttons angezeigt wird. Zuvor erscheint in dem Rahmen *Propositions* die Aufforderung, den Button zu betätigen. Der Mechanismus für die Ergebnisanzeige funktioniert gleich wie bei der Auswahl der Anwendungsfälle, mit dem Unterschied, dass hier keine Benutzereingabe notwendig ist. Als Eingabe wird der Inhalt der Leseliste verwendet, um wiederum dynamisch eine SPARQL-Query zu erstellen. Die Query zeigt - wie in Abbildung 5.15 sichtbar - je Pattern die möglichen Kombinationen und konkurrierende Alternativen an, welche über die Relationstypen *Combining* und *Competing* abgefragt werden. Aus der Ergebnistabelle können schließlich alle gewünschten Patterns per Drag & Drop in die Leseliste hinzugefügt werden.

- `LocalSettings.php`: Enthält die Parameter zur Konfiguration von den unterschiedlichen Kategorien, die im Wizard verwendet werden sollen.
- `specials/PAREP_Wizard.php`: Enthält die Verarbeitung der übergebenen URL-Parameter, um je nach Benutzerauswahl die korrekte Seite anzuzeigen. Darüber hinaus sind hier die AJAX-Callbacks definiert, die zum Erzeugen der dynamischen SPARQL-Querys und dem Senden der Querys an den Triplestore verwendet werden.
- `util/PAREP_JsTree.php`: Dient zur Umwandlung von aus dem Wiki abgefragten Daten in die JSON-Datenstruktur, welche von jsTree zur Darstellung als Baum benötigt wird.
- `util/PAREP_SPARQLQueryBuilder`: Wird von den verwendeten AJAX-Callbacks aus `PAREP_Wizard.php` zur Erstellung der SPARQL-Querys abhängig von den erhaltenen Benutzereingaben verwendet.
- `libs/wizard.js`: Enthält die komplette Applikationslogik für die unterschiedlichen verwendeten jsTrees. Außerdem werden von hier aus die AJAX-Aufrufe für verschiedene Benutzereingaben gestartet und deren Ergebnisse verwaltet. Hier ist ebenfalls die persistente Speicherung der Leseliste als Cookie realisiert.

- `skins/PAREP_Wizard.css`: Steuert die Anordnung der einzelnen Elemente des Layouts, so dass sich diese beispielsweise nicht gegenseitig verdecken oder überschneiden.
- `languages/PAREP.i18n.php`: Enthält Titel für die Spezialseite des Wizards, unter welchem er in der Übersicht aller Spezialseiten erscheint.

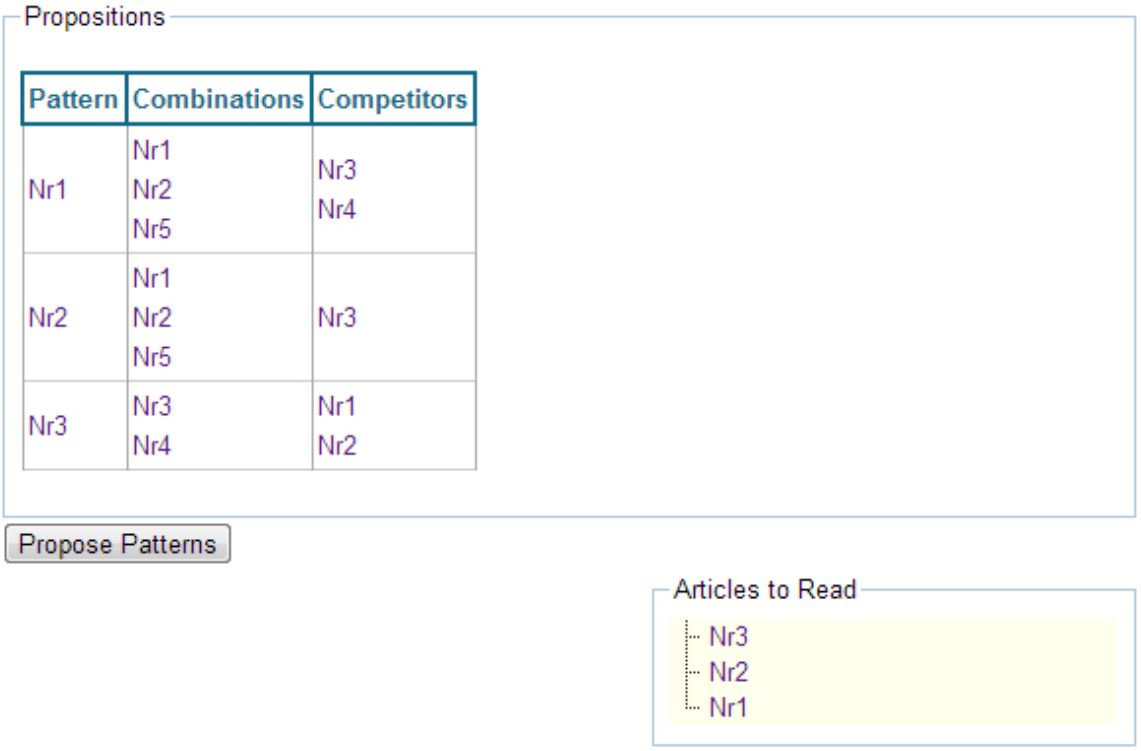

Abbildung 5.15: Vorschläge anhand der Leseliste im Wizard - Auf der linken Seite sind die Vorschläge abgebildet und rechts ist die Leseliste zu sehen

6 Ergebnis und Evaluation

Nach der genaueren Erläuterung relevanter Aspekte der Implementierung im vorherigen Kapitel soll nun anhand von Abschnitt 6.1 das Resultat dieser Diplomarbeit mit der verwendeten Infrastruktur und den Einstellungsmöglichkeiten vorgestellt werden. In Abschnitt 6.2 erfolgt eine Evaluation, welche die Anwendung an zwei Patterndomänen demonstriert und mit anderen Pattern Repositorys vergleicht. Abschließend erfolgt in Abschnitt 6.3 ein Rückblick auf den Verlauf der Arbeit, und eine kritische Bewertung im Hinblick auf Verbesserungsmöglichkeiten.

6.1 Das Pattern Repository

Das Resultat dieser Arbeit ist ein Pattern Repository, wobei die Repository-Funktionalität auf Semantic Mediawiki basiert, welches für die Verwendung mit Patterns angepasst und erweitert wurde. Eine grobe Übersicht der Bestandteile des Repositorys gibt es bereits in Abbildung 3.1 aus Abschnitt 3.3, woraus auch ersichtlich ist, dass es sich bei dem Pattern Repository im Prinzip mehr um ein Framework handelt, als um ein einzelnes Programm. Da die unterschiedlichen Teile des Systems nie gleichzeitig zum Einsatz kommen, soll im Folgenden die Reihenfolge der Verwendung beschrieben werden:

1. Modellierung des Datenmodells: Der erste Schritt zur Verwendung der Anwendung stellt das Editieren und Anpassen der mitgelieferten Datenmodell-Ontologien dar. Dabei ist es mindestens notwendig, die Ontologie des Kerndatenmodells auf die eigenen Ansprüche und insbesondere auf das gewünschte Patternformat anzupassen. Die Ontologie der Zieleigenschaften dagegen ist optional, und kann daher auf Wunsch ignoriert werden. Welche genauen Modellierungsmöglichkeiten es innerhalb der Ontologien gibt, wurde bereits ausführlich in Abschnitt 4.4 beschrieben. Als Modellierungswerkzeug kann jedes beliebige Programm verwendet werden, welches OWL-Dateien lesen kann und das Resultat als OWL in RDF/XML-Serialisierung speichern kann. Im Rahmen dieser Diplomarbeit wurde zur Modellierung das Programm Protégé verwendet, welches in Abschnitt 2.3.5 genauer beschrieben wurde.
2. Import des Datenmodells in eine DataWiki-Installation: Nachdem die Ontologien den eigenen Anforderungen entsprechend editiert wurden, können sie mithilfe des Java-Importers in eine Wiki-Installation übertragen werden. Die genauen Schritte zur Vorbereitung der Wiki-Installation sind in der technischen Dokumentation des entwickelten Pattern Repositorys hinterlegt. Inwiefern die Ontologie-Elemente auf

Elemente des Wikis abgebildet werden, wurde in Abschnitt 4.4.3 dargestellt. Der Import-Vorgang selbst und dessen Implementierung wurde in Abschnitt 5.3 erläutert.

3. Installation der benötigten Erweiterungen: Die für die volle Funktionalität des Repositorys benötigten Semantic Mediawiki-Erweiterungen müssen auf den Webserver kopiert werden. Zusätzlich ist es nötig, entsprechende Anpassungen in Konfigurationsdateien vorzunehmen. Die genauen Schritte sind auch hier ausführlich in der technischen Dokumentation erläutert.
4. Verwendung des Pattern Repositorys: Nun kann das Pattern Repository verwendet werden. Einen Überblick über die einzelnen Funktionen der Pattern Repository-Erweiterung wurde in Abschnitt 5.5 gegeben.

Abbildung 6.1: Das Hauptmenü des Pattern Repositorys

Eine besondere Rolle bei der Verwendung des Pattern Repositorys spielt das Benutzerinterface, welches hauptsächlich aus dem Hauptmenü besteht, das in Abbildung 6.1 dargestellt ist. Der Hausbutton ganz links öffnet die Hauptseite, welche einen Verweis zu dem Pattern Repository Assistenten enthält, sowie eine aus einer eingebetteten Query erstellen Tabelle aller Patterns inklusive deren Icons. Es folgt eine Beschreibung der einzelnen Menüpunkte und deren Untermenüs:

- Quickstart: Soll den schnellen Einstieg anhand einer Übersicht der Patterns ermöglichen. Dazu gibt es den Menüpunkt *Table of Contents*, der ein Inhaltsverzeichnis der Patterns in Form eines Kategoriebaums enthält. Der Menüpunkt *List of All Patterns* führt zu einer Seite mit der bereits beschriebenen dynamischen Auflistung aller existierender Patterns und deren zugehörigen Icons.
- Wizard: Dient einerseits über *Start Wizard* zum Aufrufen der Spezialseite für den Pattern Repository Assistenten. Andererseits lässt sich damit direkt auf die Leseliste zugreifen, welche mit dem Wizard erstellt werden kann, um daraus weitere vorgemerkte Patterns aufzurufen.
- Search: Dieses Menü enthält unterschiedliche Suchmöglichkeiten. Mit dem Menüpunkt *Semantic Search* lässt sich eine von DataWiki bereit gestellte Volltextsuche starten, die zudem auch Filterfunktionen zum Ausblenden gewisser Ergebnisteilmengen enthält. *Find Patterns By Target Property* öffnet die Spezialseite der Erweiterung Semantic Drilldown, welche einen semantischen Drilldown anhand von annotierten Zieleigenschaften ermöglicht.
- Create: Dieses Menü enthält nur den Eintrag *Create New Pattern* und dient zum Aufrufen des Patternformulars zum Erstellen von neuen Patterns.
- Help: Dieses Menü enthält Verweise zu kurzen Hilfestellungen bezüglich dem Erstellen und Suchen von Patterns sowie der Verwendung des Assistenten.

Der Button *New Page* neben dem Hauptmenü dient als weitere Möglichkeit, um entweder ein neues Pattern zu erstellen, oder um eine Seite ohne die Hilfe eines Formulars zu erstellen. Der Suchdialog ganz rechts bietet eine komfortable schnelle Suchfunktion mit Autovervollständigung sowie den schnellen Aufruf der Volltextsuche, die auch über das Menü *Search -> Semantic Search* aufgerufen werden kann. Die Volltextsuche kann dabei direkt über einen Klick auf das Lupensymbol für den eingegebenen Begriff aufgerufen werden.

6.1.1 Infrastruktur

Es ist möglich, einen Server für das Pattern Repository sowohl auf Windows- als auch auf Linux-Basis zu betreiben. Dafür wurden im Lauf der Diplomarbeit sowohl ein Windows Server 2008 R2-System vollständig konfiguriert, als auch ein Debian 6.0 „Squeeze"-System. Ein vollständig konfigurierter Server mit installiertem DataWiki stellt die Grundlage dar, um mit der weiteren Einrichtung fortzufahren, welche im vorigen Abschnitt beschrieben wurde.

Die Einrichtung der Infrastruktur auf dem Windows-System gestaltet sich dabei sehr unproblematisch. Seitens DIQA liegen dafür Installationsdateien mit einem sehr benutzerfreundlichen Installationsassistenten bereit, der DataWiki inklusive XAMPP und Solr sowie sinnvoller Voreinstellungen installiert. XAMPP[1] ist ein Paket, welches den Webserver Apache, die Datenbank MySQL, PHP und weiteres enthält, womit die Grundanforderungen von Mediawiki erfüllt sind. Solr[2] ist Teil des Apache Lucene-Projekts[3] und stellt ein Servlet für fortgeschrittene Suchoperationen bereit. Im Rahmen von DataWiki dient es zur Realisierung einer performanten Volltext-Suche. Die Installation des Triplestore Basic funktioniert ebenfalls unkompliziert über einen Installationsassistenten, der automatisch alle Einstellungen für die Integration mit DataWiki vornimmt.

Unter Linux gestaltet sich die Einrichtung eines mit Triplestore betriebenen DataWiki komplizierter, da keine automatischen Installationsassistenten zur Verfügung stehen. Dadurch besteht auch keine Bindung an XAMPP, sondern es muss nur dafür gesorgt werden, dass für Mediawiki möglichst ein Apache-Webserver mit PHP-Interpreter zur Verfügung steht, sowie eine MySQL-Installation als Datenbank. Um auf keine Probleme durch Inkompatibilitäten unterschiedlicher Versionen zu stoßen, wird für die Einrichtung unter Linux ebenfalls XAMPP in der identischen Version wie unter Windows verwendet. Nachdem die Webserver-Infrastruktur verfügbar ist, folgt die manuelle Installation der gleichen Mediawiki und Semantic Mediawiki-Versionen, wie sie unter Windows verwendet werden. Um die Vorteile von serverseitigem Caching ausnutzen zu können, sollte ebenfalls manuell der Cache *Memcached*[4] installiert werden. Für die Solr-Installation sollte man sich des Tricks bedienen, zuvor eine Windows-Installation einzurichten, und dann den Ordner von Windows nach Linux zu

[1] http://www.apachefriends.org/en/xampp.html
[2] http://lucene.apache.org/solr/
[3] http://lucene.apache.org
[4] http://memcached.org/

kopieren. Dies ist nötig, da die Solr-Version[5], die unter dem DataWiki-Sourceforgeprojekt erhältlich ist, keinerlei für die Integration mit DataWiki benötigte Einstellungen enthält. Nun kann mithilfe des *Wiki Administration Tools*[6], das Teil von DataWiki ist, die eigentliche Installation der restlichen DataWiki-Funktionen begonnen werden. Dies ist über das Webinterface des Wiki Administration Tools möglich, welches Zugriff auf ein Repository von Erweiterungen gewährt, aus welchem auch DataWiki ausgewählt werden kann. Für die Installation und Integration des Triplestore Basic müssen innerhalb der Mediawiki-Konfigurationsdatei „LocalSettings.php“ sowie in einer Triplestore Konfigurationsdatei zahlreiche Anpassungen gemacht werden. Eine ausführliche Anleitung der Einrichtung von DataWiki unter Linux ist in der technischen Dokumentation zu DataWiki enthalten.

Im Folgenden sind die genauen Versionen der einzelnen Programme und Frameworks aufgelistet, welche zur Installation einer DataWiki-Installation benötigt werden, die mit dem Pattern Repository kompatibel ist.

- XAMPP 1.7.4 Beta 2 (in dem DataWiki-Installationsprogramm enthalten)
- Mediawiki 1.17.0
- Semantic Mediawiki 1.7.1
- DIQA DataWiki 1.7.1 Build 134
- DIQA Triplestore Basic 1.7.3 Build 22

6.1.2 Administration

Die Administration des Pattern Repositorys sollte von einem Mediawiki-erfahrenen Administrator durchgeführt werden. Neben den üblichen Spezialseiten, die zur Administration von einzelnen Erweiterungen benutzt werden können, bietet DataWiki eine komfortable Seite, die alle diesbezüglich wichtigen Funktionen zusammenfasst. Sie ist über den Link *Administration* erreichbar, welcher links neben dem Login erscheint, sobald ein Benutzer mit Administratorrechten eingeloggt ist.

Von dieser Seite aus gibt es Zugriff auf graphische Werkzeuge zur Administration. Mithilfe des Wiki Administration Tool lassen sich unter anderem Backups der aktuellen Installation erstellen, oder anhand des bereits beschriebenen DataWiki Repositorys Erweiterungen komfortabel aus einer Liste installieren, ohne dass manuell Anpassungen an Konfigurationsdateien gemacht werden müssen. Die Triplestore Administrationsseite bietet Informationen über einen angeschlossenen Triplestore und informiert über den aktuellen Status. Neben der Verwaltung von Benutzer- und Benutzergruppenrechten ist es außerdem möglich, das DataWiki-Hauptmenü zu verändern, oder auch den Inhalt der Fußzeile anzupassen.

[5] http://sourceforge.net/projects/datawiki/files/DataWiki%201.7.1/SOLR/
[6] http://sourceforge.net/projects/datawiki/files/DataWiki%201.7.1/DataWiki%20extensions/

6.2 Evaluation

Nachdem vorhergehend das Ergebnis der Diplomarbeit beschrieben wurde, folgt nun dessen Evaluation und Bewertung. Dabei soll untersucht werden, inwiefern das in Abschnitt 1.1 definierte Ziel erreicht wurde und inwieweit es gelungen ist, die in Kapitel 3 formulierten Anforderungen zu erfüllen. Insbesondere soll in diesem Abschnitt auch überprüft werden, wie gut sich das entwickelte Pattern Repository im Umgang mit zwei Beispielpatterndomänen eignet.

Mit der Entwicklung der in Abschnitt 6.1 vorgestellten Anwendung wurde das Ziel dieser Diplomarbeit, ein Pattern Repository zu entwickeln, vollständig erreicht. Anhand des Repositorys ist es möglich, das Patternformat flexibel auf eine gewisse Patterndomäne anzupassen und man erhält anschließend ein benutzerfreundliches Formular zur Eingabe und zum Editieren von bereits vorhandenen Patterns. Die Anpassung des Patternformats geschieht mit einer OWL-Ontologie, was das Verwenden von beliebigen OWL-kompatiblen Editoren möglich macht. Durch das Formular ist das Patternformat fest vorgegeben, was unerwünschte Abweichungen vom „Standard“ ausschließt.

Das Annotieren von semantischen Links ist benutzerfreundlich mithilfe eines speziellen Werkzeugs des Patternformulars sowie innerhalb von Fließtext mithilfe eines WYSIWYG-Editors möglich. Dabei lassen sich ebenso bekannte Anwendungsfälle annotieren, um zu dokumentieren, aus welchen Quellen das Pattern abstrahiert wurde. Die Annotation von Zieleigenschaften, für welche ebenfalls ein spezielles Werkzeug für die Verwendung im Patternformular bereit steht, ermöglicht die zielgerichtete Auswahl von Patterns. Zur Unterstützung bei der Erzeugung einer Patternsprache gibt es unterschiedliche Visualisierungsmöglichkeiten, die eine Übersicht über die vorhandene Vernetzung bieten und daher eine gezieltere Feinvernetzung möglich machen.

Für die Verwendung des Pattern Repositorys durch den Endnutzer gibt es vielseitige Möglichkeiten. Anhand eines Inhaltsverzeichnisses im Baumformat, einer Liste aller Patterns mit deren Icons sowie von Visualisierungen erhält der Benutzer schnell einen Überblick über die Patterndomäne. Der Assistent ist dabei in der Lage, dem Benutzer Empfehlungen anhand von unterschiedlichen Kriterien auszusprechen. Neben auswählbaren Benutzerrollen können Zieleigenschaften, bekannte Anwendungsfälle oder eine abstrakte Startkategorie gewählt werden. Diese dienen anschließend als Grundlage für die Erstellung von dynamischen Querys, um dem Benutzer sinnvolle Empfehlungen zu geben. Die Leseliste, die dazu dienen kann, ein System von Patterns zu verfassen, ermöglicht die Empfehlung von Patterns, die sich gut in das zusammengestellte System einfügen lassen würden. Das Benutzerinterface in Form von ausblendbaren Querys, welches für die Patterns entwickelt wurde, ermöglicht ein Traversieren der Patternsprache von Patternartikel zu Patternartikel. Es folgt die Analyse der Eignung für zwei Beispieldomänen, für welche das entsprechende Patternformat komplett als Ontologie modelliert wurde. Darüber hinaus wurden jeweils der komplette zum Zeitpunkt des Verfassens dieser Arbeit vorhandene Patternfundus in das System übertragen, um einen Eindruck von dem Verhalten eines vollständig eingerichteten Pattern Repositorys zu erhalten.

6.2.1 Beispieldomäne Cloud Computing Patterns

Die Entwicklung der unterschiedlichen Repository-Bestandteile wurde jeweils mit Beispieldaten aus der Domäne der *Cloud Computing Patterns* aus [FLR+13] durchgeführt. Auf diese Weise konnte sichergestellt werden, dass bei der Konzeption möglichst allgemein gehaltene Mechanismen auch tatsächlich funktionieren. Daraus ergibt sich, dass das Pattern Repository sehr gut mit diesem Patternfundus funktioniert, da es wegen der Verwendung der Patterns als Testdatensatz auch gewissen Detailanpassungen speziell für diese Domäne gibt. Es wurde darauf geachtet, dass diese Detailanpassungen nicht in das Kerndatenmodell Einzug erhalten, und dadurch das ganze System speziell auf diese Domäne angepasst wäre.

Die Darstellung der Patternsprache für eine gewisse Patternkategorie wurde direkt aus dem Werk zu den Cloud Computing Patterns [FLR+13] übernommen und ist in Abschnitt 5.5.5 genauer beschrieben. Aufgrund der detaillierten Anpassungsmöglichkeiten des Patternformats und der gleichen grafischen Gestaltung der kategorieweisen Visualisierung der Patternsprache entspricht die optische Darstellung der Cloud Computing Patterns in dem Pattern Repository exakt der Darstellung im Buch. Zusätzlich wurde, um auch die Features betreffend der Zieleigenschaften nutzen zu können, eine Beispielontologie mit Zieleigenschaften modelliert und in das System eingebunden. Dies ermöglicht die Verwendung des vollen Funktionsumfangs des Pattern-Assistenten. Zusammenfassend lässt sich sagen, dass die vollständige Kompatibilität der Cloud Computing Patterns mit dem Repository ein gutes Indiz dafür ist, dass sich die entwickelte Anwendung für Patterns im IT-Bereich sehr gut eignet.

6.2.2 Beispieldomäne Kostüm-Patterns

Um sicherzustellen, dass das Pattern Repository tatsächlich frei für die unterschiedlichsten Patterndomänen konfigurierbar ist, wurden als zweite Evaluationsdomäne die Kostüm Patterns aus [SBLE12] gewählt. Für diese Patterndomäne gibt es zu dem Zeitpunkt des Verfassens der Arbeit keinen kompletten Fundus von Patterns, sondern dieses Projekt befindet sich gerade in dem Prozess des Entdeckens von Patterns. Die Sprache der Kostüm Patterns soll dabei helfen, den Entwurfsprozess für komplexe Kostüme in Filmen zu vereinfachen und damit das vorhandene Kostümmanagement auch durch bessere IT-Werkzeuge auf die nächste Ebene zu bringen.

Bei der Konfiguration des Patternformats anhand der OWL-Ontologie gab es keinerlei Schwierigkeiten, was insofern aber auch nicht verwunderlich ist, da das Kostüm Patternformat sehr ähnlich zu dem Format der Cloud Computing Patterns ist. Dementsprechend gab es auch keine Probleme bei der Generierung des Patternformulars anhand des Java-Importers.

Die Kostüminstanz des Pattern Repositorys wurde mit dem im Rahmen der Diplomarbeit von Daniel Kaupp entwickelten *semantischen Wiki zur Lösungsdokumentation und Musteridentifikation* integriert [Kau13]. Dieses System ermöglicht es, viele konkrete Problem-Lösungspaare in einem Wiki abzulegen, um auf diese Weise anschließend neue Patterns entdecken zu können. Mithilfe des Repositorys ist es bei der Eingabe von neuen Patterns möglich, diese

direkt mit allen zur Abstraktion verwendeten Lösungen zu vernetzen und von Grund auf eine dichte Patternsprache zu erzeugen. Es ist zu vermuten, dass ein bereits beim Erstellen der Patterns geführtes Pattern Repository sehr viele semantischen Annotationen enthalten wird, und dadurch potentiell sehr lohnende Ergebnisse vom Pattern-Assistenten produziert werden können. Zur Umsetzung des Systems von Daniel Kaupp wurde ebenfalls DataWiki verwendet, weswegen es bei der Integration der beiden Systeme nur wenige Probleme gab, die leicht lösbar waren.

6.2.3 Vergleich mit anderen Pattern-Repositorys

Nachdem die Evaluation anhand des Anpassens des entwickelten Pattern Repositorys an zwei Beispieldomänen sehr erfolgreich verlaufen ist, soll in diesem Abschnitt ein Vergleich der Anwendung mit anderen webbasierten Pattern Repositorys durchgeführt werden. Auf diese Weise soll eine Positionierung des neu entwickelten Repositorys in der aktuellen Pattern Repository Produktlandschaft ermöglicht werden. Zudem soll Lesern, die diese Diplomarbeit zur Evaluation von Pattern Werkzeugen verwenden, eine fundierte Entscheidung ermöglicht werden.

Im Folgenden sollen in dem Vergleich auftauchende Kandidaten kurz vorgestellt werden. Diese wurden aus der großen Fülle von Pattern-Webanwendungen ausgewählt, indem vorab überprüft wurde, ob die jeweiligen Anwendungen tatsächlich mehr sind als „flache“ Webseiten mit durch Hyperlinks verbundene Patterns. Aufgrund dieses Kriteriums ausgeschiedene Webseiten sind unter anderem `http://www.eaipatterns.com`, das Patternwerk zu [HW03], `http://www.welie.com` oder `http://ajaxpatterns.org/`, was als Grundlage für [Mah09] diente. Bedauerlicherweise behandeln die untersuchten Patternkataloge alle die gleiche Domäne: Design Patterns für Benutzerschnittstellen bzw. graphischer Oberflächen in der Webprogrammierung. Dies erklärt sich vermutlich aus der großen Nachfrage auf diesem Gebiet.

Quince Quince ist eine Patternbibliothek für interaktive Benutzererfahrungs- bzw. graphische Benutzeroberflächen-Design Patterns [qui] und wird von der Firma Infragistics[7] entwickelt. Neben der Möglichkeit, eine Liste aller Patterns anzuzeigen, gibt es die Herangehensweise, die Aufgabe anzugeben, die mithilfe des Patterns gelöst werden soll, um Vorschläge für passende Patterns zu erhalten. Patterns können in Quince mit Tags versehen werden, die einer Kategorisierung entsprechen. Anhand dieser Tags wird eine Übersichtskarte generiert, auf der die am meisten verwendeten Tags im Stil einer *Tag Cloud* am größten erscheinen, und die am wenigsten verwendeten Tags am kleinsten. Die Beziehungen zwischen Tags werden dadurch erzeugt, dass Tags, die gemeinsam für ein Pattern verwendet werden, eine Verbindung erhalten. Wird ein Tag ausgewählt, erscheinen alle Patterns, die damit versehen wurden. Mithilfe des sog. „Wireframe“ können Patterns anhand ihrer sinnvollen Verwendungsposition innerhalb einer „typischen“ graphischen Benutzeroberfläche ausgewählt werden. Benutzer

[7] `http://www.infragistics.com`

können zu allen Patterns Kommentare und Verbesserungsvorschläge beisteuern. Zur Eingabe neuer Patterns können registrierte Benutzer Vorschläge einreichen. Dafür lässt sich ein Dokument hochladen und Tags sowie die Position im Wireframe vorschlagen.

Ypatterns Die Yahoo! Design Pattern Library enthält Design Patterns für Benutzerschnittstellen von Webanwendungen und wird von Entwicklern der Yahoo!-Plattform betreut [Yah]. Neben einer Liste aller Patterns mit Icons sind die Patterns ebenfalls in Kategorien eingeteilt, aus welchen sie direkt ausgewählt werden können. Eine Besonderheit ist das Angebot eines *Stencil Kits*, welches Schablonen für unterschiedliche Designprogramme enthält. Die Schablonen entsprechen Patterns aus der Yahoo! Design Pattern Library und Codemodulen aus der Yahoo! User Interface Library[8], einer Bibliothek von interaktiven Webanwendungen. Mithilfe der Schablonen kann der Benutzer Entwürfe seiner Anwendungen erstellen und sieht direkt, welche Patterns er anschließend zur Umsetzung verwenden kann. Die Patternseiten enthalten neben den üblichen Abschnitten auch viele Verlinkungen zu Codebeispielen und zu ähnlichen Patterns aus anderen Benutzerschnittstellen-Design Pattern Repositorys. Die Patterns werden von einem Yahoo-Team bereitgestellt, allerdings werden dazu auf http://www.yuiblog.com teils Umfragen gestartet, um die Meinung der Entwicklergemeinde zu Patternentwürfen zu erfahren.

Usabilitypatterns UsabilityPatterns.de ist das Resultat einer studentischen Fachstudie und wurde von Yves Schubert, Uwe Breitenbücher und Jens Schumann entwickelt [SBS]. Die Seite enthält ebenfalls Design Patterns für Benutzerschnittstellen und befindet sich noch in der Beta-Phase. Das zugrundeliegende Patternkonzept unterscheidet sich dabei von dem klassischen auf Christopher Alexander basierenden Format. Dabei werden Patterns nicht mehr als feste Einheit gesehen; stattdessen sollen die starren Problem-Lösungspaare aufgespalten werden. Daraus resultiert ein Modell, das einer Lösung beliebig viele Probleme und Lösungsvorschläge zuordnen lässt. Bei der Suche nach Patterns gibt es einen Wizard, der anhand dieses Modells die Auswahl in vier Phasen aufteilt. In der Startphase, die momentan noch nicht implementiert ist, sollen später grobe Patternkategorien gewählt werden können. In der *Interaktionsartphase* kann die Art der Interaktion ausgewählt werden, die umgesetzt werden soll. Dies ist möglich, da Usabilitypatterns.de ebenfalls auf ein Tagsystem setzt, und dabei sogar semantische Tags erlaubt, die die Zuordnung des Tags zu einer der Phasen ermöglicht. In der *Typphase* werden nun alle Möglichkeiten aufgeführt, welche die beiden zuvor ausgewählten Tags und einen weiteren Tag besitzen, der als Typ angezeigt wird. In der *Zusatzattributephase* können die angezeigten Patterns anhand von weiteren verfügbaren Tags weiter eingeschränkt werden. Neben dieser assistentengestützten Suche ist das Stöbern durch den kompletten Patternfundus anhand der Auswahl von Tags möglich. Patterns können von registrierten Benutzern anhand von verschiedenen Kriterien bewertet werden. Für Implementierungen von Patterns gibt es eine extra Oberfläche, die neben Bewertungen auch das Schreiben von Kommentaren ermöglicht.

[8] http://yuilibrary.com/

Die vorgestellten Kandidaten sollen nun anhand von gewissen Qualitätsmerkmalen untereinander verglichen werden. Dabei werden die Systeme teils anhand der Anforderungen aus Kapitel 3 verglichen, denn diese haben, obwohl sie teils spezifisch für das neu entwickelte Repository sind, durchaus allgemeine Gültigkeit als Qualitätsmerkmale für Pattern Repositorys. Andererseits wurden Gemeinsamkeiten zwischen den Anwendungen gesucht, um überhaupt einen sinnvollen Vergleich möglich zu machen. Zunächst sollen nun die einzelnen Qualitätsmerkmale anhand von Fragen beschrieben werden, die zum Vergleich verwendet wurden.

- Visualisierung der Patternsprache: Gibt es eine Möglichkeit, die Beziehungen zwischen Patterns, die die Patternsprache bilden, graphisch darzustellen?
- Unterstützung bei der Eingabe von neuen Patterns: Gibt es ein Formular oder einen Assistenten, um den Benutzer bei der Eingabe von neuen Patterns zu unterstützen?
- Kollaborative Arbeit an Patterns: Ist es möglich, dass Patterns in gemeinschaftlicher Arbeit in dem System weiterentwickelt werden? Gibt es Möglichkeiten, Kommentare oder Verbesserungsvorschläge für Patterns anzumerken?
- Suche von Patterns: Können Patterns komfortabel gefunden werden? Wird der Benutzer dabei vom System unterstützt?
- Assistent zur Patternsuche: Gibt es einen interaktiven Assistenten, der dem Benutzer nach Abfrage von gewissen Kriterien sinnvolle Patterns vorschlägt?
- Führung des Benutzers: Ist ein Hilfedialog vorhanden, wenn der Benutzer das Pattern Repository aufruft? Wird erklärt, wie das Repository funktioniert?
- Darstellung der Patterns: Werden die einzelnen Patterns übersichtlich dargestellt, so dass der Inhalt schnell erfasst werden kann? Wie werden die Verweise auf andere Patterns dargestellt?
- Anwendungsperformance: Reagiert das System schnell auf Nutzereingaben? Gibt es lange Ladezeiten?
- Optik der Anwendung: Wirkt die Oberfläche eher wertig oder billig? Ist erkenntlich, dass es sich um ein zusammengehöriges System handelt, oder wirken die einzelnen Komponenten zusammengestückelt?
- Ausgereiftheit des Repositorys: Treten Bugs während der Verwendung des Systems auf? Wirken einzelne Aspekte optimiert, oder ist es spürbar, dass sich das System noch in der Entwicklung befindet?

Ein wichtiger Punkt, der in dieser Aufzählung fehlt, ist die Konfiguration für andere Patterndomänen. Dieser kann anhand dieser Evaluation leider nicht bewertet werden, da es keinen Zugriff auf den Quellcode der Projekte gibt und sie jeweils für eine bestimmte Patterndomäne konfiguriert wurden. In Tabelle 6.1 ist ein Vergleich der ausgewählten Kandidaten dargestellt. Um die einzelnen Pattern Repositorys anhand eines Merkmals bewerten zu können, wird eine Skala verwendet, die dem schulischen Notensystem ähnelt:

- **++**: Das Repository weist sehr gute Leistungen in dem mit zwei Plus bewerteten Qualitätsmerkmal auf, und übertrifft die meisten anderen Repositorys.
- **+**: Das Repository weist gute Leistungen in dem mit einem Plus bewerteten Qualitätsmerkmal auf, und kann sich dadurch eventuell leicht von den anderen Kandidaten abgrenzen.
- **o**: Das Repository weist befriedigende bzw. durchschnittliche Leistungen in dem mit einer Null bewerteten Qualitätsmerkmal auf.
- **-**: Das Repository weist ausreichende Leistungen in dem mit einem Minus bewerteten Qualitätsmerkmal auf, und ist damit etwas schlechter als die Konkurrenz.
- **- -**: Das Repository weist ungenügende Leistungen in dem mit zwei Minus bewerteten Qualitätsmerkmal auf.

Anhand dieser Bewertungsskala wird anschließend in der letzten Spalte der Tabelle eine Summe der Punkte für jede Anwendung gebildet. Dabei zählt jedes Plus als plus Eins, jede Null als Null, und jedes Minus als minus Eins.

Es folgt eine Begründung der Bewertungen in den jeweiligen Qualitätsmerkmalen der unterschiedlichen Repositorys.

- Visualisierung der Patternsprache: Das in dieser Arbeit entwickelte Pattern Repository erhält zwei Plus, da die Patternsprache anhand von auswählbaren Linktypen visualisiert werden kann. YPatterns erhält zwei Minus, da es keinerlei Visualisierung der Patternsprache gibt. Quince erhält zwei Plus für die übersichtliche Visualisierung in Form der Tagcloud, die Beziehungen enthält, wenn auch nur in Form eines Linktyps. UsabilityPatterns.de erhält ein Minus, das es zwar eine Visualisierung der Tags in Form einer Tagcloud gibt, diese aber keine Beziehungen enthält und nicht klar ist, wie die Tags innerhalb der Wolke angeordnet werden.
- Unterstützung bei Eingabe von neuen Patterns: Das Pattern Repository erhält zwei Plus, da das Patternformular viele den Benutzer unterstützende Hilfsmittel enthält. Die anderen Repositorys erhalten zwei Minus, da die Benutzer keine neuen Patterns hinzufügen können, und es daher auch keinen Zugriff auf ein Formular gibt. Es ist unbekannt, inwiefern die Eingabe der Patterns intern in den unterschiedlichen Anwendungen funktioniert.
- Kollaborative Arbeit an Patterns: Das Pattern Repository erhält ein Plus, da durch die Verwendung der Mediawiki-Plattform kollaboratives Arbeiten gut möglich ist. Zudem gibt es eine Diskussionsseite zu jedem Pattern, auf der Benutzer Kommentare zu den Patterns hinterlassen können. Die anderen Anwendungen erhalten jeweils eine Null, da sie zwar eine Möglichkeit für Benutzer bieten, Bewertungen oder Kommentare abzugeben, das gemeinsame Arbeiten an Patterns aber nicht vorgesehen ist.
- Suche von Patterns: Das Pattern Repository erhält zwei Plus, da eine Schnellsuche mit Autovervollständigung, eine Volltextsuche mit Filterfunktionen und eine Möglichkeit zum Drilldown durch den Patternfundus gegeben ist. YPatterns erhält zwei Minus,

	Pattern Repository	YPatterns	Quince	Usability-patterns
Visualisierung der Patternsprache	++	- -	++	-
Unterstützung bei Eingabe von neuen Patterns	++	- -	- -	- -
Kollaborative Arbeit an Patterns	+	o	o	o
Suche von Patterns	++	- -	o	++
Assistent zur Patternsuche	+	++	++	+
Führung des Benutzers	-	++	++	- -
Darstellung der Patterns	++	++	+	++
Anwendungsperformance	+	++	- -	++
Optik der Anwendung	o	++	++	+
Ausgereiftheit des Repositorys	-	++	+	- -
Summe	9	6	6	1

Tabelle 6.1: Vergleichstabelle von alternativen Pattern Repositorys

da es keinerlei Suchfunktion gibt und auch kein Anzeigen von Patterns anhand von Tags oder Ähnlichem. Quince erhält eine Null, da die Patternsuche anhand der Tagcloud sehr träge ist und es keine Schnellsuche mit Autovervollständigung gibt. UsabilityPatterns.de erhält zwei Plus, da die Suche anhand von Tags sehr komfortabel funktioniert und ein Filtern innerhalb der Tagcloud möglich ist. Für die Schnellsuche gibt es eine Autovervollständigung der Patternbezeichner.

- Assistent zur Patternsuche: Das Pattern Repository erhält ein Plus, da es einen Assistenten gibt, der bereits viele Funktionen hat, aber noch nicht so ausgereift ist. YPatterns wird mit zwei Plus bewertet für den sehr interaktiven Ansatz über die Stencil Kits, was ein intuitives Auffinden der Patterns ermöglicht. Quince erhält ebenfalls zwei Plus für den Ansatz mit dem Wireframe, wobei dieser sehr ähnlich zu dem YPatterns-Ansatz ist. Usabilitypatterns.de erhält ein Plus, da der Assistent momentan nicht sehr benutzerfreundlich zu bedienen ist und kaum brauchbare Ergebnisse liefert.

- Führung des Benutzers: In dem Pattern Repository gibt es lediglich auf der Hauptseite einen Verweis auf unterschiedliche Funktionalitäten, wobei der Benutzer diese aber auch übersehen könnte, daher gibt es ein Minus. Bei YPatterns hat der Benutzer direkt die Möglichkeit, eine Beschreibung der Yahoo Design Pattern Library zu öffnen und Hilfestellung zu erhalten, wofür es zwei Plus gibt. Quince erhält ebenfalls zwei Plus und schlägt sich bei der Führung des Nutzers am besten. Beim ersten Öffnen wird ein modaler Dialog angezeigt, der eine detaillierte Hilfestellung zu allen Funktionen enthält, der sich auf Wunsch aber bei zukünftigen Besuchen von Quince nicht mehr öffnet. UsabilityPatterns.de erhält zwei Minus, da beim Öffnen des Repositorys eine inhaltsleere Seite erscheint, und der Benutzer sich ohne Unterstützung zurechtfinden muss.
- Darstellung der Patterns: Das Pattern Repository erhält zwei Plus, weil die Verlinkung zu anderen Patterns und Anwendungsfällen sowie die eingebetteten informativen Querys ein sehr komfortables Navigieren durch die Patterns ermöglicht. YPatterns erhält ebenfalls zwei Plus, da sich die Darstellung der Patterns sehr übersichtlich gestaltet und alle verwandten Patterns, Lösungen und Implementierungen auf einen Blick erfasst werden können. Darüber hinaus gibt es hochwertige Icons sowohl für Patterns als auch für Kategorien. Quince erhält nur ein Plus, da das Scrollen durch die verschiedenen Abschnitte des Pattern sehr träge und damit nicht benutzerfreundlich ist; das ständige Einblenden von Beispielen dagegen sehr nützlich. UsabilityPatterns.de erhält zwei Plus, da die Abschnitte sehr übersichtlich sind, und alle wichtigen Informationen schnell erreichbar sind. Die zusätzliche Oberfläche für Implementierungen ist ebenfalls eine sinnvolle Idee.
- Gefühlte Anwendungsperformance: Das Pattern Repository erhält ein Plus, da die meisten Aktionen zwar ohne störende Wartezeiten durchgeführt werden können, aber manchmal kleinere Wartezeiten von ca. 2 Sekunden auftreten. YPatterns und UsabilityPatterns.de erhalten je zwei Plus, da beide Anwendungen sehr schnell auf Benutzeraktionen reagieren und es kaum Ladezeiten gibt. Quince erhält zwei Minus, da unabhängig von der Geschwindigkeit der Internetverbindung und des verwendeten Rechners extrem lange Ladezeiten auftreten. Bereits beim Öffnen der Applikation gibt es zunächst eine Ladezeit von ca. 10 Sekunden und auch das Öffnen von Patterns dauert jedes Mal 3-5 Sekunden.
- Optik der Anwendung: Das Pattern Repository erhält eine Null, da DataWiki durchaus ein ansehnliches Design hat, die entwickelten Erweiterungen im Rahmen der Diplomarbeit aber noch eine einheitliche optische Linie vermissen lassen. Quince und YPatterns erhalten je zwei Plus, da die kompletten Anwendungen sehr professionell erscheinen und ein einheitliches Design auf der kompletten Seite beibehalten. UsabilityPatterns.de erhält ein Plus, da die Oberfläche durchaus zu gefallen weiß, an einigen Stellen der Betastatus aber noch zu offensichtlich ist.
- Ausgereiftheit des Repositorys: Das Pattern Repository erhält hier ein Minus, da es zwar eine solide Grundlage darstellt, aber viele Funktionen noch detaillierter ausgearbeitet werden sollten. Zudem wurde es noch nie mit vielen Benutzern getestet. YPatterns ist eine sehr ausgereift professionelle Anwendung, die von einem ganzen Team von

Entwicklern gepflegt wird. Quince erhält ein Plus, da es zwar ebenfalls professionell gepflegt wird, die Performance aber noch viel Raum für Optimierungen offen lässt. UsabilityPatterns.de erhält zwei Minus, weil man der Anwendung an vielen Stellen ihren Beta-Status sehr deutlich anmerkt.

Zuletzt erfolgt ein Ranking anhand der Summenspalte aus Tabelle 6.1. Die Bildung der Summe wurde bereits oben erläutert, und daraus resultiert folgendes Ranking:

1. Pattern Repository
2. YPatterns und Quince
3. Usabilitypatterns

Um dieses Ergebnis etwas zu relativieren soll an dieser Stelle noch einmal betont werden, dass einige der verwendeten Qualitätsmerkmale aus dem Anforderungskatalog dieser Diplomarbeit stammen. Hätte man stattdessen die Anforderungen verwendet, die bei der Entwicklung von beispielsweise YPatterns formuliert wurden, hätte YPatterns vermutlich den Vergleich „gewonnen“. Auch die niedrige Punktzahl von UsabilityPatterns.de soll nicht aussagen, dass es sich dabei um ein schlechtes System handelt. Dies folgt vielmehr aus dem Beta-Status, der an vielen Stellen sehr deutlich spürbar ist.

6.2.4 Vorteile und Nachteile des Pattern Repositorys

In diesem Abschnitt sollen mögliche Vorteile und Nachteile aufgelistet werden, die sich aus den obigen Vergleichen mit anderen Pattern Repositorys ergeben haben. Zusätzlich wurden Bewertungen aus meiner Sicht hinzugefügt.

Vorteile:

- Benutzerfreundliche graphische Oberfläche, die dem Benutzer an vielen Stellen Unterstützung durch Autovervollständigung etc. anbietet.
- Benutzerfreundlicher Assistent hilft auch bei komplexen Suchszenarien. In Buchform gibt es keine Möglichkeit, den Benutzer auf diese Weise zu unterstützen.
- Effizientes kollaboratives Arbeiten an Patterns wird ermöglicht durch Verwendung von Mediawiki als Basis.
- Vielfältige Suchmöglichkeiten, um Patterns anhand von unterschiedlichsten Kriterien zu finden.
- Aussagekräftige Visualisierung der Patternsprache.
- Benutzerfreundliches Patternformular, das sowohl zur Eingabe von neuen Patterns dient, als auch zum Editieren von bereits vorhandenen.

- Komplette Flexibilität des zugrunde liegenden Datenmodells und insbesondere des Patternformats, womit sich das Pattern Repository für viele Patterndomänen sehr gut eignet. Für die Domäne der *Cloud Computing Patterns* aus [FLR+13] wurde die gute Eignung bereits erprobt.
- Erweiterungsmöglichkeiten des Pattern Repositorys durch die modulare Architektur und dem erweiterungsfreundlichen Design von Mediawiki.
- Problemlose Integration mit anderen auf Mediawiki basierenden Anwendungen möglich.
- Das Datenmodell lässt sich um beliebige Funktionalitäten erweitern, indem der Java-Importer ebenfalls angepasst wird.
- Einfache Installation der zugrundeliegenden Plattform unter Windows.

Nachteile:

- Der Nutzen der semantischen Annotationen kann nur so groß sein, wie die inhaltliche Qualität und Sinnhaftigkeit der Annotationen.
- Es sind noch keine komplexen Querys in dem System enthalten, so dass dementsprechend auch noch keine außergewöhnlichen neue Einsichten z.B. betreffend der Patternsprache gewonnen werden können. Allerdings kann nur spekuliert werden, ob dies durch komplexere Abfragen möglich ist.
- Wenig Führung des Nutzers durch das Repository.
- Die indirekte Abbildung der Datenmodellontologie auf das Wiki ist sehr komplex.
- Komplizierte Installation der zugrundeliegenden Plattform unter Linux.
- Breit gefächertes Wissen vom technischen Administrator des Pattern Repositorys gefordert: Wissen über Ontologiemodellierung in OWL, Java-Programmierkenntnisse, falls das Importprogramm angepasst werden soll, Administration eines Webservers und Mediawiki- sowie Semantic Mediawiki-Kenntnisse.
- Unsicherheit über die Zukunft von DataWiki. Falls die Entwicklung eingestellt werden sollte, muss anderweitiger Ersatz für die von DataWiki bereitgestellten Funktionen gefunden werden.

6.3 Rückblick auf Verlauf der Arbeit

Anschließend an die Vorstellung der Ergebnisse dieser Arbeit und deren Evaluation soll nun ein Rückblick auf den Verlauf der Arbeit erfolgen, um einen Überblick über das Vorgehen zu geben und es entsprechend zu analysieren. Am Beginn der Arbeit stellte sich zunächst die Frage des zu verwendenden semantischen Wikis. Dabei entstand bereits die in dieser Arbeit unter Abschnitt 3.5 aufgeführte Evaluation der alternativen Anwendungen, allerdings waren an vielen Stellen die Anforderungen an das benötigte Werkzeug nicht sehr klar formuliert.

Dies lag zum einen daran, dass sich viele Anforderungen erst aus Ideen entwickelten, die durch die Erforschung der Möglichkeiten von Semantic Mediawiki entstanden. Zum anderen hätte dies vielleicht früher forciert werden sollen.

Anschließend folgte die Untersuchung auf die Verwendbarkeit von unterschiedlichen Erweiterungen für Semantic Mediawiki, wobei unter anderem auch das sehr nützliche DataWiki entdeckt wurde. Es folgte nun die inkrementelle Entwicklung eines Datenmodells in Kapitel 4, wobei unterschiedliche Ansätze versucht wurden. Dabei wurde die Modellierung als UML-Metamodell oder als UML-Klassenhierarchie aber letztendlich wieder verworfen, weil die Modellierung als OWL-Ontologie sich schließlich als naheliegende Lösung herausstellte. Dies machte auch insofern Sinn, da in Semantic Mediawiki gewisse OWL-Konzepte direkte Entsprechungen besitzen.

Nach der Modellierung des Patternformats und der Kategorienhierarchie in OWL folgte die schrittweise Erweiterung des Datenmodells um Funktionen wie die Angabe von Visualisierungen auf Kategorieseiten und die Einbindung von Zieleigenschaften. Gleichzeitig wurde dabei jeweils das Parsen der neu hinzugekommenen Elemente in dem in Abschnitt 5.3 beschriebenen Java-Importer implementiert, so dass die jeweils erweiterten Datenmodelle auch ins Wiki übertragen werden konnten. Die unterschiedlichen Teile der Erweiterung Pattern Repository wurde jeweils nach der Integration der Funktionen im Datenmodell implementiert und sind in Abschnitt 5.5 beschrieben. Als letzter Schritt der Erweiterung wurde der Assistent entwickelt, der das Zusammenspiel aller bisher entwickelten Funktionalitäten benötigt.

Während der laufenden Entwicklung wurde auch damit experimentiert, das Pattern Repository auf unterschiedlichen Infrastrukturen zu betreiben, mit dem Resultat einer ausführlichen Installationsanleitung für Windows und Linux. Zu Evaluationszwecken wurde, wie in Abschnitt 6.2 beschrieben, das Datenmodell für die Patterndomäne der Kostüme konfiguriert und diese Kostüm-Instanz des Pattern Repositorys mit Daniel Kaupps System [Kau13] integriert. Im nächsten Kapitel folgt nun eine Zusammenfassung dieser Diplomarbeit.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde entsprechend der Zielsetzung ein Pattern Repository zur Verwaltung und Suche von Design Patterns unterschiedlicher Domänen entworfen. Die Anwendung wurde mithilfe von semantischen Webtechnologien auf Basis von Semantic Mediawiki und Erweiterungen für dieses Framework realisiert.

Zu Beginn wurde evaluiert, welches semantisches Wiki sich am besten eignen würde, um die zusätzlich benötigten Funktionen darauf zu implementieren. Die Evaluation ergab, dass Semantic Mediawiki als ausgereifte, den Anforderungen entsprechende Open-Source Anwendung eine gute Wahl dafür wäre. Semantic Mediawiki ist eine Erweiterung für Mediawiki, der Plattform für wikipedia.org, wodurch sich die Möglichkeit ergibt, zahlreiche bereits bestehende Erweiterungen für Mediawiki zu verwenden. Die Hauptanforderungen für das zu entwickelnde Werkzeug waren das benutzerfreundliche Erstellen von neuen Patterns, das Annotieren von semantischen Informationen zum Ausdruck der Beziehungen zwischen Patterns und die ausgeklügelte Suche nach Patterns. Zudem sollte das Werkzeug über einen Assistenten verfügen, der dem Benutzer sinnvolle Patterns vorschlagen kann.

Im Rahmen eines konzeptionellen Designs wurden die in der Zielsetzung geforderten Grundfunktionalitäten ausformuliert und konkretisiert. Insbesondere der Mechanismus zum Vernetzen der Patterns untereinander zur Bildung einer Patternsprache wurde sehr sorgfältig entworfen. Anschließend wurde das aus diesen detaillierten Beschreibungen resultierende Datenmodell in Form einer OWL-Ontologie modelliert und vorgestellt. Die nötige Abbildung der Elemente der Ontologie auf das Wiki, das für den Import des Datenmodells in Semantic Mediawiki nötig ist, wurde beschrieben und die Wahl von OWL als Modellierungssprache begründet. Nach der Fertigstellung aller nötigen Entwurfsschritte wurde mit der Umsetzung begonnen. Der Import des Datenmodells geschieht über ein Java-Programm, dass die Ontologie parst und in das Wiki überträgt. Auf diese Weise werden das Formular zur Eingabe von Patterns und die erlaubten semantischen Linktypen zwischen Patterns erzeugt, wobei das Formular mithilfe einer Semantic Mediawiki Erweiterung umgesetzt wird. Fehlende Funktionalitäten wie bestimmte Eingabetypen für das Formular oder Möglichkeiten zur Visualisierung der Patternsprache wurden durch die Umsetzung der Erweiterung „Pattern Repository“ in das Wiki integriert. Darin ist auch der Assistent implementiert, der Patterns anhand unterschiedlichster Kriterien vorschlagen kann, beispielsweise anhand von Zieleigenschaften, die durch die Wahl eines Patterns erreicht werden sollen. Die resultierende Architektur aus dem Zusammenspiel der Ontologie, des Java-Importers und der selbst

entwickelten Erweiterung wurde auf unterschiedlichen Detailstufen graphisch dargestellt und erläutert.

Zuletzt wurde das Pattern Repository als Ergebnis dieser Diplomarbeit anhand seiner Infrastruktur und Verwendung vorgestellt und durch die Konfiguration für zwei Pattern-Beispieldomänen evaluiert. Die Evaluation kam zu dem Ergebnis, dass sich das System sehr flexibel an unterschiedliche Domänen anpassen lässt und sehr gut für die beiden Beispieldomänen funktioniert. In einem Vergleich mit anderen Pattern Repositorys konnte sich das entwickelte Repository gegenüber seinen Konkurrenten behaupten, auch wenn es noch viel Verbesserungspotential gibt.

7.2 Ausblick

Wie bereits im Rahmen der Evaluation in Abschnitt 6.2 erwähnt, bietet der jetzige Implementierungsstand eine gute Grundlage für Erweiterungen der unterschiedlichsten Aspekte. Im Folgenden sollen die jeweiligen Grundideen der Vorschläge kurz umrissen werden.

Ausbau des Assistenten Der Assistent besitzt zum jetzigen Zeitpunkt bereits die unterschiedlichsten Ansätze zur Empfehlung von sinnvollen Patterns, allerdings ist keiner davon sehr detailliert ausgebaut oder optimiert. Bei der Empfehlung von Zieleigenschaften wäre es beispielsweise nötig, ein Eingabewerkzeug zur Verfügung zu haben, welches die Navigation durch den Ontologiebaum erleichtert. Bei der Auswahl über bekannte Anwendungsfälle wäre eine Art von Kategorisierung von diesen wünschenswert, um die Auswahl geordneter zu gestalten. Das größte Potential bietet sicherlich die Empfehlung anhand der Leseliste, für die ein intelligenter Algorithmus entwickelt werden sollte. Zumindest eine intelligentere Query wäre hier sehr wünschenswert. Neben dem Ausbau der bereits vorhandenen Punkte ist es auch problemlos möglich, den Assistenten um neue Empfehlungsarten zu erweitern.

Implementierungen an Patterns anhängen Momentan gibt es für das Anhängen von Implementierungen an Patterns zwar die Möglichkeit, diese in einem speziellen Abschnitt zu speichern oder auf externe Inhalte zu verlinken, aber gerade im IT-Bereich gäbe es hier Potential für erweiterte Funktionen. Als Vorbild könnte hier YPatterns [Yah] dienen, welches eine sehr enge Vernetzung von Patterns, Implementierungen und tatsächlich verwendbaren Beispielen umsetzt.

Interface der Patternseiten verbessern Die Art, wie durch eine Patternsprache navigiert wird, hängt sehr stark von dem Interface ab, das auf den einzelnen Patternseiten zu finden ist. In Form von ausblendbaren Querys wurde in dem Pattern Repository ein sehr mächtiges Werkzeug implementiert, wofür nur noch nützlichere Querys umgesetzt werden müssten.

Export der Ontologie aus dem Wiki Momentan ist lediglich der Import einer OWL-Ontologie in das Wiki implementiert, nicht aber der Export aus dem Wiki in eine

Ontologie. Dafür müsste das momentan verwendete Java-Programm entsprechend erweitert werden, um eine Abbildung auf die Kerndatenmodellontologie zu erhalten.

Integriertes Installationsprogramm Um die Benutzerfreundlichkeit des Repositorys auf die nächste Stufe zu bringen, müsste es ein Installationsprogramm geben, dass alle Schritte für die Einrichtung einer neuen DataWiki-Installation übernimmt. Dafür müsste das Programm den Java-Importer enthalten, und in der Lage sein, Mediawiki-Erweiterungen automatisch zu installieren.

Visualisierungen verbessern Bei Patternsprachen gibt es sehr unterschiedliche Aspekte, die visualisiert werden können. Besonders interessant wäre beispielsweise eine „grobe" Visualisierung, in der eine komplette Patterndomäne abgebildet wäre. Um die Übersichtlichkeit zu wahren, müsste dafür eventuell ein Algorithmus entwickelt werden, der bei zu vielen Pattern innerhalb einer Kategorie anschließend z.B. nur die Kategorie anstatt der Einzelpatterns in der Grafik anzeigt. Die Verwendung von z.B. Hypergraphen hat sich aufgrund der fehlenden Übersicht als nicht empfehlenswert herausgestellt.

Leseliste als Entscheidungsprozess Eine sehr interessante Idee, die eventuell eine tiefere Untersuchung Wert wäre, ist das Ausbauen der jetzigen „Leseliste" zu einer Möglichkeit, den Entscheidungsprozess anhand von Patterns abzubilden. Denn im Prinzip entspricht die Verwendung jedes Patterns dem Fällen einer Entscheidung, und somit dem Navigieren entlang der Patternsprache auf einem gewissen Pfad. Daher wäre es auch denkbar, diesen „Pfad" in der Leseliste abzubilden und es zu ermöglichen, ihn zu visualisieren. Dieser Pfad könnte man sich als Subgraph einer Patternsprache vorstellen, der alle verwendeten Patterns enthält.

Pattern Mitigation Pattern Mitigation beschreibt ein Vorgehen, um mit der Kombination von bestimmten Patterns auf einer hohen Abstraktionsebene Zieleigenschaften zu erreichen, für die normalerweise Patterns auf einer niedereren Abstraktionsebene benötigt würden. Diese Art der ausweichenden Modellierung kann sich anbieten, falls das spezifischere Pattern gewisse Nachteile mit sich bringt, die durch die ersatzweise Modellierung auf höherer Ebene nicht oder weniger stark auftreten, also *abgeschwächt* werden. Dies ist genauer in [FLR+13, S. 299-310] beschrieben. Um eine mögliche Pattern Mitigation in einem Pattern Repository abzubilden, würde dafür eine n-äre semantische Beziehung zwischen den beteiligten Patterns benötigt. Dafür wäre ein extra Abschnitt im Patternformular nötig, sowie eine Erweiterung des Pattern Interfaces und des Assistenten, um von den Annotationen letztendlich auch profitieren zu können. Da die Darstellung von n-ären Beziehungen bereits für die Annotation von Zieleigenschaften entwickelt wurde, müsste diese dafür verwendbar sein.

Literaturverzeichnis

[AA05] D. Aumueller, S. Auer. Towards a Semantic Wiki Experience - Desktop Integration and Interactivity in WikSAR. In *Semantic Desktop Workshop*. 2005. (Zitiert auf Seite 40)

[AIS+77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel. *A Pattern Language: Towns, Buildings, Constructions*. Oxford University Press, 1977. (Zitiert auf den Seiten 9, 14, 16, 46, 58 und 60)

[Ale79] C. Alexander. *The timeless way of building*, Band 1. New York: Oxford University Press, 1979. (Zitiert auf Seite 16)

[AVH04] G. Antoniou, F. Van Harmelen. *Semantic Web Primer*. the MIT Press, 2004. (Zitiert auf den Seiten 7, 18, 19, 21 und 22)

[BDS+09] J. Bao, L. Ding, P. R. Smart, D. Braines, G. Jones. Rule modeling using semantic mediawiki. 2009. (Zitiert auf Seite 42)

[Bec02] D. Beckett. SWAD-Europe Deliverable 10.1: Scalability and Storage: Survey of Free Software / Open Source RDF storage systems, 2002. URL http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_report/. (Zitiert auf Seite 24)

[BGE+08] M. Buffa, F. Gandon, G. Ereteo, P. Sander, C. Faron. SweetWiki: A semantic wiki. *Web Semant.*, 6:84–97, 2008. (Zitiert auf Seite 40)

[BHS07] F. Buschmann, K. Henney, D. Schmidt. *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages*. Pattern-Oriented Software Architecture. Wiley, 2007. (Zitiert auf Seite 33)

[Bir10] A. Birukou. A survey of existing approaches for pattern search and selection. In *Proceedings of the 15th European Conference on Pattern Languages of Programs*, S. 2. ACM, 2010. (Zitiert auf den Seiten 34 und 35)

[Boz] I. Bozhanov. JsTree. URL http://www.jstree.com/. (Zitiert auf den Seiten 89 und 100)

[Bus98] F. Buschmann. *Pattern-orientierte Software-Architektur*. Pearson Deutschland GmbH, 1998. (Zitiert auf den Seiten 16, 17 und 34)

[Dja05] B.-R. Djaloeis. *Kaukolu: Building a semantic wiki*. Diplomarbeit, University of Kaiserslautern, 2005. (Zitiert auf Seite 40)

[DKT05] J. Deng, E. Kemp, E. G. Todd. Managing UI pattern collections. In *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*, S. 31–38. ACM, 2005. (Zitiert auf den Seiten 33, 34 und 46)

[FEL+12] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rutschlin, D. Schumm. Capturing Cloud Computing Knowledge and Experience in Patterns. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, S. 726–733. IEEE Computer Society, Washington, DC, USA, 2012. (Zitiert auf den Seiten 17, 34, 35, 46, 49, 53, 58, 59, 60, 61 und 68)

[FLR+11] C. Fehling, F. Leymann, R. Retter, D. Schumm, W. Schupeck. An architectural pattern language of Cloud-based applications. In *Proceedings of the Conference on Pattern Languages of Programs (PLoP)*. 2011. (Zitiert auf den Seiten 49, 62 und 63)

[FLR+13] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2013. (Zitiert auf den Seiten 8, 9, 14, 16, 35, 36, 54, 59, 61, 65, 101, 110, 118 und 123)

[GASB11] J. García, A. Amescua, M.-I. Sánchez, L. Bermón. Design guidelines for software processes knowledge repository development. *Information and Software Technology*, 53(8):834 – 850, 2011. (Zitiert auf den Seiten 14 und 35)

[GMJ+03] S. L. Greene, P. Matchen, L. Jones, J. C. Thomas, M. Callery. Tool-based decision support for pattern assisted development. In *Proc. of CHI 2003 Workshop on User Interface Patterns*. 2003. (Zitiert auf den Seiten 33 und 35)

[Gri11] P. Grimm. *Metamodell und Plattform für Mustersprachen und Musterkataloge*. Diplomarbeit, Universität Stuttgart, 2011. (Zitiert auf den Seiten 51, 52 und 53)

[GSJS03] A. Gaffar, D. Sinnig, H. Javahery, A. Seffah. MOUDIL: A comprehensive framework for disseminating and sharing HCI patterns. In *CHI 2003 Workshop on Perspectives on HCI patterns: Concepts and Tools*. 2003. (Zitiert auf den Seiten 33 und 34)

[Har05] S. Hartmann. Transdisziplinarität - Eine Herausforderung für die Wissenschaftstheorie. *Homo Sapiens und Homo Faber. Festschrift für Jürgen Mittelstrass. Berlin & New York: Walter de Gruyter*, S. 335–343, 2005. (Zitiert auf Seite 9)

[Hee] U. van Heesch. Open Pattern Repository. URL http://www.patternrepository.com. (Zitiert auf Seite 14)

[HKRS08] P. Hitzler, M. Krötzsch, S. Rudolph, Y. Sure. *Semantic Web: Grundlagen*. Springer London, Limited, 2008. (Zitiert auf den Seiten 18, 19, 20, 21, 22 und 24)

[HL93] H.-J. Habermann, F. Leymann. *Repository - Eine Einführung*. R. Oldenbourg Verlag, 1993. (Zitiert auf den Seiten 13, 34, 36 und 51)

[HLS05] A. Haake, S. Lukosch, T. Schümmer. Wiki-templates: adding structure support to wikis on demand. In *Proceedings of the 2005 international symposium on Wikis*, WikiSym '05, S. 41–51. ACM, New York, NY, USA, 2005. (Zitiert auf Seite 63)

[Hoe] P. D. H. Hoepfner. Grundlagen des SE. (Zitiert auf Seite 36)

[HW03] G. Hohpe, B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. (Zitiert auf den Seiten 15, 35, 38, 58, 59, 61 und 111)

[ISO10] ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2010. (Zitiert auf Seite 36)

[JGVH95] R. Johnson, E. Gamma, J. Vlissides, R. Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. (Zitiert auf den Seiten 9, 15, 16, 35, 38, 53, 58, 59 und 60)

[Kar11] D. Karastoyanova. Services and Service Composition - Application Architecture and Application Integration, 2011. Vorlesungsfolien. (Zitiert auf den Seiten 7 und 83)

[Kau13] D. Kaupp. *Use of a Semantic Wiki for Capturing of Solutions and Pattern Discovery*. Diplomarbeit, Universität Stuttgart, 2013. (Zitiert auf den Seiten 110 und 119)

[KE02] D. M. Kienzle, M. C. Elder. Final technical report: Security patterns for web application development. *DARPA, Washington DC*, 2002. (Zitiert auf Seite 14)

[Koc] P.-P. Koch. ExecCommand compatibility. URL http://www.quirksmode.org/dom/execCommand.html. (Zitiert auf Seite 92)

[Köp13] C. Köppe. A Pattern Language for Teaching Design Patterns. In *Transactions on Pattern Languages of Programming III*, S. 24–54. Springer, 2013. (Zitiert auf den Seiten 9 und 15)

[KV] M. Krötzsch, D. Vrandecic. Semantic MediaWiki. URL http://semantic-mediawiki.org/. (Zitiert auf den Seiten 26 und 40)

[Ler06] R. M. Lerner. Installing and customizing MediaWiki. *Linux J.*, 2006(144):3, 2006. URL http://dl.acm.org/citation.cfm?id=1124506.1124509. (Zitiert auf Seite 80)

[LH05] B. Liu, B. Hu. An evaluation of RDF storage systems for large data applications. In *Semantics, Knowledge and Grid, 2005. SKG'05. First International Conference on*, S. 59–59. IEEE, 2005. (Zitiert auf Seite 24)

[LL07] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. Dpunkt-Verlag, 2007. (Zitiert auf Seite 33)

[Mah09] M. Mahemoff. *Ajax design patterns*. O'Reilly Media, 2009. (Zitiert auf Seite 111)

[McB01] B. McBride. Jena: Implementing the rdf model and syntax specification. 2001. (Zitiert auf Seite 86)

[MD97] G. Meszaros, J. Doble. A pattern language for pattern writing. In R. C. Martin, D. Riehle, F. Buschmann, Herausgeber, *Pattern languages of program design*, Kapitel A pattern language for pattern writing, S. 529–574. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. (Zitiert auf den Seiten 16 und 63)

[Meda] MediaWiki. Manual - Developing extensions. URL http://www.mediawiki.org/wiki/Manual:Developing_extensions. (Zitiert auf den Seiten 80 und 81)

[Medb] MediaWiki. Resource Loader. URL http://www.mediawiki.org/wiki/ResourceLoader. (Zitiert auf den Seiten 82 und 89)

[Mil56] G. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The psychological review*, 63:81–97, 1956. (Zitiert auf Seite 57)

[mwo] MediaWiki. URL http://www.mediawiki.org. (Zitiert auf Seite 26)

[NRHW06] N. Noy, A. Rector, P. Hayes, C. Welty. Defining n-ary relations on the semantic web. *W3C Working Group Note*, 12:4, 2006. (Zitiert auf Seite 54)

[OOD] Design Patterns - Object Oriented Design. URL http://www.oodesign.com. (Zitiert auf Seite 38)

[Ort99] E. Ortner. Repository Systems. Teil 1: Mehrstufigkeit und Entwicklungsumgebung. *Informatik-Spektrum*, 22(4):235–251, 1999. (Zitiert auf Seite 13)

[Per] D. Perry. Graphviz - Graph Visualization Software. URL http://www.graphviz.org. (Zitiert auf Seite 30)

[por] Portland Pattern Repository. URL http://c2.com/ppr/. (Zitiert auf Seite 14)

[pro] The Protégé Ontology Editor and Knowledge Acquisition System. URL http://protege.stanford.edu/. (Zitiert auf Seite 25)

[qui] Quince - UX Patterns Explorer. URL http://quince.infragistics.com. (Zitiert auf den Seiten 10 und 111)

[SBBK08] S. Schaffert, F. Bry, J. Baumeister, M. Kiesel. Semantic wikis. *Software, IEEE*, 25(4):8–11, 2008. (Zitiert auf Seite 39)

[SBLE12] D. Schumm, J. Barzen, F. Leymann, L. Ellrich. A Pattern Language for Costumes in Films. In *Proceedings of the 17th European Conference on Pattern Languages of Programs (EuroPLoP 2012)*. 2012. (Zitiert auf den Seiten 7, 9, 15, 17, 49, 53, 60 und 110)

[SBS] Y. Schubert, U. Breitenbücher, J. Schumann. UsabilityPatterns.de - Die Usability Pattern Sammlung. URL http://www.usabilitypatterns.de:8080. (Zitiert auf den Seiten 10 und 112)

[Sch03] T. Schuemmer. Seeking for structure in a Groupware Pattern Language. In *Workshop at CHI*. 2003. (Zitiert auf den Seiten 34, 35 und 54)

[Sch06] S. Schaffert. IkeWiki: A semantic wiki for collaborative knowledge management. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*, S. 388–396. IEEE, 2006. (Zitiert auf Seite 40)

[Sem] Semantic Wiki Projects. URL http://semanticweb.org/wiki/Semantic_wiki_projects#Active. (Zitiert auf Seite 40)

[Smw] Semantic Wikipedia FAQ - What are the alternatives to SMW? URL http://semantic-mediawiki.org/wiki/FAQ#What_are_the_alternatives_to_SMW.3F. (Zitiert auf Seite 41)

[SP] A. Seaborne, E. Prud'hommeaux. SPARQL Query Language for RDF. URL http://www.w3.org/TR/rdf-sparql-query/. (Zitiert auf Seite 24)

[tik] Tiki Wiki CMS Groupware. URL http://info.tiki.org/. (Zitiert auf Seite 40)

[VK06] D. Vrandecic, M. Krötzsch. Reusing ontological background knowledge in semantic wikis. In *Proceedings of the 1st Workshop on Semantic Wikis, Budva, Montenegro*. 2006. (Zitiert auf den Seiten 39 und 64)

[W3C] W3C. Resource Description Framework (RDF). URL http://www.w3.org/RDF/. (Zitiert auf Seite 19)

[WB07] M. Weiss, A. Birukou. Building a pattern repository: Benefitting from the open, lightweight, and participative nature of wikis. In *International Symposium on Wikis (WikiSym), ACM*, S. 21–23. 2007. (Zitiert auf den Seiten 33 und 34)

[WSK+03] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, et al. Efficient RDF storage and retrieval in Jena2. In *Proceedings of SWDB*, Band 3, S. 7–8. 2003. (Zitiert auf Seite 29)

[WV03] M. V. Welie, G. C. V. D. Veer. Pattern languages in interaction design: Structure and organization. In *Proc. Interact '03, M. Rauterberg, Wesson, Ed(s). IOS*, S. 527–534. IOS Press, 2003. (Zitiert auf den Seiten 51 und 52)

[Yah] Yahoo. Yahoo Design Pattern Library. URL http://developer.yahoo.com/ypatterns. (Zitiert auf den Seiten 10, 112 und 122)

[zAg] zAgile. Wikidsmart for Atlassian Confluence. URL http://www.zagile.com/products/wikidsmart.html. (Zitiert auf Seite 40)

[Zim09] O. Zimmermann. *An architectural decision modeling framework for service-oriented architecture design*. Dissertation, Stuttgart, Univ., Diss., 2009, 2009. (Zitiert auf Seite 41)

Alle URLs wurden zuletzt am 31. Juli 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift