Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3533

Kollaborative Musteridentifikation basierend auf Wiki-Technologien

Daniel Willig

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Frank Leymann
Betreuer/in:	Dipl. Inf. Christoph Fehling
Beginn am:	1. August 2013
Beendet am:	31. Januar 2014
CR-Nummer:	D.2.1, D.2.2, D.2.11, D.3.3, H.3.5, H.5.2, H.5.4, I.7.2

Zusammenfassung

Patterns haben sich als gute Möglichkeit herausgestellt, um wiederkehrende kontextbezogene Entwurfsprobleme zu lösen. In der Vergangenheit wurden diese Patterns meist in gedruckten Sammlungen veröffentlicht. Die sich daraus ergebenden Nachteile, wie etwa der feste Autorenkreis, die Beschränkung auf eine Domäne sowie die Unmöglichkeit nachträglicher Ergänzungen bzw. Korrekturen, wurden im Rahmen dieser Diplomarbeit durch die Entwicklung eines wiki-basierten Systems angegegangen. Dabei wurde zunächst versucht, einen bereits existierenden Prototypen auf eine wartbare und zukunftsfähige Basis zu stellen. Nach dem Scheitern dieses Vorhabens wurden die zentralen Anforderungen an eine solche Lösung herausgearbeitet und als Basis für einen eigenen Prototypen verwendet. Dieser wurde auf Basis des MediaWiki-Frameworks implementiert und bietet die Möglichkeit, Patterns mit semantischen Daten zu annotieren. Die Patterndokumente lassen sich mit Hilfe von frei konfigurierbaren Templates auf ein einheitliches Format bringen. Durch die Verwendung von semantischen Wiki-Erweiterungen wurde das Erstellen, Suchen und Betrachten von Patterns deutlich verbessert. MediaWiki als Grundsystem sorgt dafür, dass das System sich ideal für kollaboratives Arbeiten eignet.

Abstract

Patterns have been shown to be a good solution to recurring contextual design problems. In the past those patterns were often published within printed collections. Subsequent disadvantages like a fixed group of authors, the limitation to one specific domain plus the impossibility of future adjustments and corrections respectively, have been addressed in this diploma thesis by developing a wiki based solution. An initial attempt was made to put an existing prototype on a maintainable and sustainable basis. After the failure of this approach the

essential requirements of such a solution were carved out and used as a base for a new prototype. This prototype was implemented on top of the MediaWiki. It gives us the possibility to annotate patterns with semantic data. Pattern documents use freely configurable templates to ensure a common format. Due to the use of semantic MediaWiki extensions the process of creating, searching and displaying of patterns was strongly enhanced. Collaborative working was made easily available by the use of MediaWiki as a base.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel soll das Ziel der Arbeit erläutert werden. Zusätzlich wird ein Überblick über die Struktur der einzelnen Kapitel des Dokuments gegeben.

1.1 Ziel

Die Hauptaufgaben dieser Diplomarbeit umfassen die Erweiterung einer vorhandenen wiki-basierten Patternsammlung um kollaborative Funktionen sowie die Überführung des existierenden Quellcodes in ein Open-Source-Projekt. Dazu muss zunächst die existierende Code-Basis analysiert und im Hinblick auf Wartbarkeit bzw. Erweiterbarkeit bewertet werden. Zusätzlich müssen die vorhandenen Abhängigkeiten zur Middleware auf ihre Zukunftsfähigkeit hin beurteilt werden. Das Ziel ist es, zum Schluss eine Plattform zu erhalten, die leicht erweiterbar und testbar ist sowie einfach zu installieren und betreiben ist.

1.2 Kapitelübersicht

Es folgt eine kurze Übersicht der Kapitel dieser Arbeit. Dabei wird für jedes Kapitel der inhaltliche Schwerpunkt erläutert:

Kapitel 1 - Einleitung

Dieses Kapitel enthält das Ziel der Arbeit sowie eine kurze Übersicht der Kapitel.

Kapitel 2 - Grundlagen

Dieses Kapitel enthält eine Zusammenfassung der für die Arbeit zentralen Hintergrundinformationen. Dies umfasst verwendete Technologien sowie insbesondere eine allgemeine Betrachtung von Patterns.

Kapitel 3 - Anforderungsanalyse: Pattern-Collaboration-Tool

Dieses Kapitel enthält eine Anforderungsanalyse mit dem Ziel, die zentralen Anforderungen an ein kollaboratives Pattern-Verwaltungs-Tool herauszuarbeiten. Dabei wird zunächst ein passendes Rollenmodell ermittelt. Im Anschluss daran werden für die jeweiligen Rollen Anforderungen in Form von User-Stories festgehalten. Am Ende des Kapitels folgen allgemeine Anforderungen.

Kapitel 4 - Implementierung

Dieses Kapitel enthält zunächst die Betrachtung eines bereits existierenden Systems im Hinblick auf die in Kapitel 3 herausgearbeiteten Anforderungen. Im Anschluss daran folgt die Beschreibung der tatsächlichen Implementierung eines eigenen Prototyps.

Kapitel 5 - Zusammenfassung

Dieses Kapitel dient der Zusammenfassung des Verlaufs dieser Arbeit. Zusätzlich wird kurz auf die noch zu erledigenden Aufgaben eingegangen.

2 Grundlagen

Das folgende Kapitel liefert eine Zusammenfassung der für die Arbeit zentralen Hintergrundinformationen. Dies umfasst verwendete Technologien sowie insbesondere eine allgemeine Betrachtung von Patterns, die wiederum als Grundlage für die Analyse im Kapitel 3 dienen soll.

2.1 Patterns

Gemäß Hanmer [8] ist ein Pattern ein wiederkehrendes Design-Element und kann als die Lösung eines Entwurf-Problems angesehen werden. Zentral bei der Erstellung von Patterns ist nach Hanmer die sogenannte *Rule of Three*. Diese Regel besagt, dass eine Lösung mindestens dreimal erfolgreich angewandt werden muss, bevor sie als Pattern angesehen werden kann. Durch das Bestehen auf die Wiederholung soll laut Hanmer gezeigt werden, dass ein Pattern eine allgemeine Lösung darstellt, die immer wieder für gleiche bzw. ähnliche Probleme funktionieren wird. Hanmer führt weiter aus, dass Pattern sich immer dann ergeben, wenn mehrere Individuen vorhandene Lösungen betrachten und Gemeinsamkeiten erkennen. Falls nun jemand diese Gemeinsamkeiten extrahiert und in verständlicher und verwendbarer Form festhält, ergibt sich gemäß Hanmer ein Pattern, welches ab diesem Zeitpunkt von der Allgemeinheit verwendet werden kann, ohne dass diese die ursprünglichen Lösungen kennen muss.

An dieser Stelle seien kurz die sogenannten Anti-Patterns erwähnt. Der Begriff entwickelte sich nach Leitner [14] in der internationalen Software-Pattern-Community. Ein Pattern ist immer dann ein Anti-Pattern, wenn es mehr Probleme erzeugt als es löst [14]. Laut Leitner kann ein Anti-Pattern ursprünglich ein durchaus nützliches Pattern gewesen sein. Durch einen veränderten Kontext jedoch, ist es kontraproduktiv geworden [14]. Zusätzlich kann es sich nach Leitner schon von Anfang an um eine Scheinlösung gehandelt haben. Es herrscht also nur die Illusion, dass ein Problem gelöst wird. Nach Leitner entstand der Begriffe Anti-Pattern, aus der Beobachtung heraus, dass Patterns Nebenwirkungen haben können. Er nennt hier als drastisches Beispiel das Anti-Pattern *Krieg*. So wird durch den Krieg zwar vordergründig ein positives Ergebnis erzielt, indem nämlich der Sieger seine Interessen durchsetzen kann, in der Summe jedoch zerstört ein Krieg immer deutlich mehr Wert als an entwaigen Vorteilen gewonnen werden kann [14]. Es ist nach Leitner im Interesse der Menschen, wenn Anti-Patterns erkannt und vermieden werden können.

Zurück zu den Patterns, sie sind nützlich, da sie wiederkehrende Lösungen für Probleme darstellen [8]. Sie können oft direkt auf ein spezielles Problem angewandt werden. Oft ist allerdings auch eine Anpassung an die Problem-Domäne nötig. Dies ist aber nicht schlimm, denn die grundlegende Struktur bleibt die selbe [8]. Somit muss selten ganz von vorne begonnen werden und es kann auf die Erfahrungen anderer zurückgegriffen werden [8].

Ein weiteres wichtiges Merkmal des wiederkehrenden Designs ist nach Hanmer [8], dass es kommuniziert werden kann. Es ist deutlich einfacher eine Lösung anderen zu erklären, wenn dabei auf gebräuchliche und somit bekannte Patterns zur Beschreibung zurückgegriffen werden kann [8]. Ein gutes Pattern enthält alle nötigen Informationen, um einen Entwurf nachzubilden und zu verstehen, warum die präsentierte Lösung für eine konkrete Situation die beste Lösung darstellt [8]. Zusätzlich enthält ein gutes Pattern laut Hanmer ausreichend Informationen, um es an das eigene Problem anzupassen. Patterns beschrei-

ben also etwas und zeigen zugleich, wie dieses Etwas erschaffen werden kann [8]. Patterns sind nach Hanmer idealerweise modular, beschreiben also Teilprobleme und lassen sich somit zur Lösung eines Gesamtproblems kombinieren. Dabei sind sie oftmals domänenunabhängig. Betrachten wir z.B. das *Composite*-Pattern [5], lässt sich sehr leicht erkennen, dass sich eine Komposition von Elementen nicht nur in der Software-Entwicklung anbietet. Sie lässt sich vielmehr immer dann anwenden, wenn der Unterschied zwischen Kompositionen von Objekten und Einzelobjekten verborgen werden soll [5]. Patterns eigenen sich sehr gut, um das Wissen von Domänenexperten für andere (weniger Erfahrene) zugänglich zu machen (der sogenannte *Blick durch Expertenaugen*) [8]. Eine gute Möglichkeit, die eigenen Fähigkeiten zu verbessern, ist nach Hanmer also das Studium von Patterns. Der Leser lernt dabei wie Experten an bestimmte Probleme herangehen. Außerdem lassen sich die meisten neuen Probleme durch eine Kombination vorhandener Mittel und Wege lösen [8].

2.2 Probleme klassischer Patternsammlungen

Reiners et al. [20] bringen die Probleme klassischer, gedruckter Patternsammlungen auf den Punkt. So sind etwa die Hauptdomänen von gedruckten Werken Software Engineering, Notfallschutz, UI-Design und Web-Design. Im Vergleich zu Online-Sammlungen fehlen hier sehr viele Bereiche wie beispielsweise Architektur, allgemeine Prozesse, Bildung und Human-Computer-Interaction (HCI) [20]. Für Reiners et al. zählen vor allem Profis, Domänenexperten und Anfänger mit Vorkenntnissen zu der Zielgruppe von gedruckten Sammlungen. Im Gegensatz dazu liegt bei Online-Lösungen der Fokus auf dem Informationsaustausch, und zwar unabhängig von eventuellen Vorkenntnissen der Anwender [20]. Bei der Erstellung von gedruckten Sammlungen ergeben sich nach Reiners et al. inhärente Probleme bei der Zusammenarbeit und den Einflussmöglichkeiten. Sie lassen sich, wenn einmal veröffentlicht, nur sehr schwer erweitern.

Zusätzlich mangelt es an Transparenz, da sich der Entstehungsprozess bzw. die Änderungshistorie nicht nachvollziehen lässt. Oft ist auch nicht immer ganz klar, wer denn der usprüngliche Autor ist. Nach Reiners et al. enthalten klassische Patternsammlungen meist nur formale Konzepte. Es fehlen die nicht trivialen Angaben der *Bad Practices*. Ein weiteres großes Problem ist gemäß Reiners et al. die mühsame Erstellung von Patterns. Die Zeitdauer bis ein Pattern fertiggestellt ist und somit für die Allgemeinheit von Nutzen sein kann, ist meist recht hoch. Besser wäre es, wenn schon von Anfang an eine Beteiligung möglich wäre [20]. Im Gegensatz zu Online-Lösungen haben klassische Patterns immer einen festen und abgeschlossenen Autorenkreis. Die Experten einer Domäne formulieren ausgehend von ihrer persönlichen Erfahrung und Spezialisierung Patterns, die sie in einer individuellen Patternsprache festhalten. Dieser *Guru -Ansatz* hat den Nachteil, dass die allgemeine Sammlung und Formulierung von Wissen nur sehr selten durchgeführt wird [20]. Speziell die konzeptionellen Patterns würden laut Reiners et al. [20] von einer Wiederverwendung von vorhandenem Wissen aus anderen Domänen profitieren. So könnten etwa gute und schlechte Praktiken abstrahiert und für andere Domänen verfügbar gemacht werden. Ein weiteres großes Problem ist die mangelhafte Unterstützung beim Durchforsten einer Patternsammlung. Es gibt zwar meist eine Gruppierung von Hierarchieebenen, aber die (visuelle) Einordnung eines Patterns in die Gesamtstruktur fehlt komplett. Ein großes Problem ist auch, dass nachdem ein Pattern einmal veröffentlicht wurde, nachträgliche Erweiterungen bzw. Korrekturen nicht mehr oder nur sehr schwer möglich sind (z.B. erst bei einer Neuauflage des gedruckten Werks).

2.3 Patternerstellung

Wellhausen und Fießer [34] liefern uns einen Überblick über einen typischen Patternerstellungsprozess. So beginnt die Erstellung meist mit einem kurzen

Brainstorming mit dem Ziel, einen groben Überblick zu bekommen, welches Problem das Pattern denn eigentlich lösen soll [34]. Das Brainstorming soll uns Antworten auf einige wichtige Fragen liefern. Wir müssen uns klar machen, was genau das Pattern ausmacht, was darin enthalten ist und was nicht und schließlich wie es sich von anderen ähnlichen Ideen unterscheidet [34].

Im zweiten Schritt betrachten wir die Lösung und überlegen uns, was der Kern der Lösung ist [34]. Dazu gehört es auch herauszufinden, welche Aspekte die Lösung am besten beschreiben [34]. Am Ende soll eine Zusammenfassung entstehen, die es einem Domänenexperten ermöglicht, die Lösung zu verstehen [34]. Im dritten Schritt stellen wir uns die Frage, welches konkrete Problem durch das Pattern eigentlich gelöst wird [34]. An dieser Stelle warnen Wellhausen und Fießer davor, Sätze nach dem Muster *Wie mache ich XY?* zu erstellen. Das Problem soll stattdessen möglichst allgemein beschrieben werden, da sonst die Gefahr besteht, ein triviales Pattern zu erstellen [34]. Ein guter Ansatz ist, zu versuchen, eine Antwort auf die Frage *Warum ist die Welt eine schlechtere ohne meine Lösung?* zu finden [34]. Zum Schluss sollten wir uns fragen, ob die Lösung tatsächlich die Antwort auf die Problembeschreibung liefert [34].

Im nächsten Schritt betrachten wir die Konsequenzen, die durch die Anwendung des Patterns entstehen [34]. An dieser Stelle bietet es sich nach Wellhausen und Fießer an, zunächst die Vorteile bzw. den Nutzen zu erläutern und was passieren würde, wenn das Pattern nicht angewendet werden würde. Danach müssen die Einschränkungen der Lösung festgehalten werden [34]. Es muss klar sein, was für Nachteile sich durch das Anwenden des Patterns ergeben können [34]. Schließlich muss für den Anwender ersichtlich sein, was er beim Einsatz des Patterns beachten muss [34].

Nachdem nun ein Grundgerüst des Patterns steht, muss geklärt werden, welche äußeren Einflüsse auf das Problem wirken [34]. Wir müssen also erklären, warum das Problem schwer zu lösen ist bzw. welche Faktoren das Problem kompliziert machen [34]. Dieser Schritt ist laut Wellhausen und Fießer nötig,

um dem Leser klar zu machen, warum das Problem komplizierter ist, als er zunächst vermuten könnte. Um diese Einflüsse zu finden, schlagen Wellhausen und Fießer vor, an andere mögliche Lösungen für das Problem zu denken. Das Ziel hierbei ist es, Faktoren zu finden, die von unserer Lösung abgedeckt werden, aber nicht von den Alternativen [34]. Laut Wellhausen und Fießer sorgt ein Einflussfaktor dafür, dass eine Sache komplizierter wird. Dagegen wirkt eine dazugehörende Konsequenz in dem sie beschreibt, wie der entsprechende Faktor durch die Lösung behoben wird [34]. Nachdem die Einflussfaktoren gefunden wurden, sollte man also versuchen, jeden Faktor durch eine Konsequenz zu beseitigen [34].

Als nächstes muss der Problemkontext betrachtet werden [34]. Dieser liefert Anforderungen und Gesichtspunkte, ohne diese es gar kein Problem zu lösen gäbe [34]. Es ist gemäß Wellhausen und Fießer wichtig zu verstehen, dass der Kontext nicht durch das Anwenden einer Problemlösung verändert wird. Um einen passenden Kontext zu finden, können wir uns überlegen, wann und wie das Problem existiert [34]. Oftmals sind auch Einflussfaktoren, die im vorherigen Schritt identifiziert wurden, eigentlich Teil des Kontext [34]. Dies ist laut Wellhausen und Fießer immer dann der Fall, wenn ein Faktor das Problem zwar verkompliziert, aber nicht durch die Lösung neutralisiert wird.

Zu guter Letzt muss das Pattern noch einen Namen erhalten [34]. Der Pattern-Titel ist der wichtigste Teil wenn es darum geht, das Pattern zu kommunizieren [34]. Idealerweise enthält er bereits einen Teil der Lösung und sorgt somit dafür, dass der zentrale Aspekt des Patterns in Erinnerung bleibt [34].

Nachdem nun alle Teile des Patterns in einer ersten Version existieren, folgt die eigentlich wichtigste Arbeit. Das Pattern kann und muss über mehrere Iterationen hinweg kontinuierlich verbessert werden [34]. Erst nach diesen Korrekturschleifen kann man von einem fertigen Pattern sprechen.

2.4 Patternformat

Was in der Natur ohne Bewusstsein wie von selbst geschieht, vollzieht ein Gestalter laut Leitner [14] bewusst. Er sieht ein Problem und sucht nach einer Lösung [14]. Da die Wiederverwendung von bewährten Lösungen zu höherer Effizienz führt, ist ein Gestalter gemäß Leitner an allen Abstraktionen und am Lernen interessiert. Um jedoch Lösungen wiederverwenden zu können, müssen Situationen erkannt werden, die sich mit Vorhandenem vergleichen lassen [14]. Aus der Expertenkunst des Problemlösens wird durch geeignete Beschreibungen, die alle wesentlichen Informationen enthalten, ein einfacheres Handwerk. Man muss vorhandene Lösungen nur noch richtig anwenden können, um das Problem zu lösen [14]. Alexander [1] definiert ein Pattern als eine Lösung für ein Problem in einem bestimmten Kontext. Zur Beschreibung des Patterns wird also eine formale Menge an Eigenschaften, wie etwa *Titel* oder *Kontext* festgehalten [14]. Wir erhalten „*Eine Problemlösung in Form einer ausführlichen, verständlichen und methodischen Beschreibung* [14]“. Ein Pattern besteht nach Leitner z.B. aus den folgenden Eigenschaften:

Musternname (Bezeichnung)
Kontext (Anwendungszusammenhang)
Ausgangssituation
Problemlösung
Ergebnissituation (Resultat der Musteranwendung)
Wechselwirkungen (oder Kräfte)
Anschlussmuster
Zusatzinformationen (je nach Anwendungsgebiet)

Schauen wir uns zum Abschluss noch ein Beispiel-Pattern von Leitner an:

Musternname

Vier-Augen-Prinzip

Kontext

Entwicklung von Organisationen, aber auch von Menschen.

Ausgangssituation

Schwierige Entscheidungen verlangen besondere Maßnahmen zur Sicherung der Qualität.

Problem

Bei einsamen Entscheidungen von einzelnen Veranwortungsträgern passieren Fehler, die leicht vermeidbar scheinen.

Problemlösung

Man strebt an, dass bestimmte Entscheidungen nicht von einem Menschen alleine getroffen werden, sondern dass ein Konsens mit einer zweiten Person erforderlich ist.

Anmerkungen

Das Prinzip verlangt annähernde Gleichwertigkeit der beteiligten Personen. Die Qualität der Entscheidungen wird sich im Normalfall verbessern, die Enscheidungen brauchen aber mehr Zeit.

Anschlussmuster

Organisation, Beratung, Ehe

2.5 Verwendete Technologien und Begriffe

Es folgt eine Übersicht der in dieser Arbeit verwendeten Technologien sowie die Definition der wichtigsten Begriffe.

2.5.1 User-Story

Wirdemann [36] bezeichnet User-Stories als ein etabliertes und weit verbreitetes Konzept zur Beschreibung von Anforderungen in agilen Softwareprojekten. Dabei sind die Anforderungen im Vergleich zum klassischen Anforderungsmanagement deutlich unschärfer und offener formuliert [36]. Der Zweck von User-Stories ist es, die Anforderungen aus Sicht des Benutzers zu beschreiben [36]. Dabei ist die Unvollständigkeit laut Wirdemann Teil des Konzeptes und es wird akzeptiert, dass die meisten Kunden bzw. Benutzer erst sehr spät im Projekt wissen, was sie wollen [36]. User Stories besitzen einen konkreten und sichtbaren Mehrwert für den Kunden bzw. Benutzer [36].

2.5.2 Domäne

Wir halten uns bei dem Begriff der Domäne an die Definition des Duden. Eine Domäne ist ein Spezialgebiet, das heißt ein Gebiet auf dem sich jemand besonders betätigt bzw. besonders gut auskennt [2].

2.5.3 LAMP

LAMP ist ein Akronym und bezeichnet einen Software-Stack für den Betrieb und die Entwicklung von dynamischen Web-Anwendungen. Das Paket besteht in der klassischen Ausführung aus dem Betriebssystem Linux, dem Webserver Apache, der SQL-Datenbank MySQL und der Skriptsprache PHP. Das Erfolgsrezept des Pakets ist die einfache Installation bzw. Konfiguration, die auch von Anfängern ohne weiteres durchgeführt werden kann. Die einzelnen Bestandteile sind dabei aufeinander abgestimmt und sollen einen reibungslosen Betrieb garantieren. Oftmals enthalten die LAMP-Pakete eigene Tools, die den Administrator bei seiner Arbeit unterstützen.

2.5.4 XML

Die Extensible Markup Language (XML) ist eine textdateibasierte Auszeichnungssprache, die die Darstellung von hierarchisch strukturierten Daten ermöglicht. XML wurde mit dem Hintergedanken des einfachen Datenaustauschs vor allem über das Internet spezifiziert. Zusätzlich war eine Kernanforderung, dass es sowohl von Menschen als auch von Computern gelesen bzw. geschrieben werden kann. [37]

2.5.5 Semantic Web

Semantic Web [33] bezeichnet eine internationale Gemeinschaftsbewegung unter dem Schirm des World Wide Web Consortium (W3C), deren Ziel es ist, durch das Hinzufügen von semantischen Inhalten zu normalen Webseiten von klassischen unstrukturierten bzw. semi-strukturierten Dokumenten, hin zum *Web of Data* zu wechseln. Dabei basiert der *Semantic Web Stack* auf dem *Resource Description Framework (RDF)* des W3C.

Das aktuelle Web lässt sich als Mensch relativ einfach und zielführend durchsuchen. Es ist für die meisten von uns kein großes Problem, sich z.B. auf der Homepage der städtischen Bibliothek ein bestimmtes Buch zu reservieren. Für Computer jedoch ist bereits diese triviale Aufgabe kaum zu bewerkstelligen. Der Hauptgrund hierfür ist die Tatsache, dass heutige Webseiten entwickelt werden, um von Menschen gelesen zu werden und eben nicht von Maschinen. Das Semantic Web setzt an dieser Stelle an. Die ursprüngliche Vision beinhaltete ein System, das es Maschinen erlaubt, komplexe menschliche Anfragen zu verstehen und darauf zu antworten.

2.5.6 Triple-Store

Ein Triple-Store ist eine Datenbank, speziell geschaffen zum Abspeichern von sogenannten Triples. Ein Triple ist ein 3-Tupel, bestehend aus Subjekt, Prädikat und Objekt (z.B. *Hans kennt Markus*). [32]

2.5.7 SPARQL

Bei SPARQL handelt es sich um eine graphbasierte Abfragesprache für RDF [26]. Ein Beispiel:

```
PREFIX ns: <http://example.com/meineOntology#>
SELECT ?hauptstadt ?land
WHERE {
  ?x ns:stadtname ?hauptstadt ;
     ns:istHauptstadtVon ?y .
  ?y ns:landname ?land ;
     ns:istAufKontinent ns:Europa .
}
```

2.5.8 MediaWiki und Semantic-MediaWiki

Pages und Categories

Die grundlegende Struktur des Wikis setzt sich aus vielen sogenannten Pages zusammen. Diese Seiten lassen sich zunächst einmal von allen Benutzern editieren. Die MediaWiki-Software beinhaltet bereits eine Versionsverwaltung für Seiten, sodass Änderungen jederzeit nachvollzogen und bei Bedarf rückgängig gemacht werden können. Es gibt verschiedene Arten von Seiten. Die am häufigsten verwendete Form ist die klassische, einfache und benutzereditierbare Seite, die mit Hilfe von Wiki-Markup und/oder HTML mit Inhalt gefüllt werden kann. Ein weiterer häufig verwendeter Typ ist die Kategorie-Seite. Mit Hilfe

von Kategorien lassen sich Seiten in eine Hierarchie einordnen, somit strukturieren und leichter auffindbar machen. [10]

Special-Pages

Special-Pages sind Seiten, die von der MediaWiki-Software bei Bedarf generiert werden. Sie sind im Namespace *Special:* angesiedelt und sind nicht wie andere Seiten editierbar. Viele Special-Pages dienen dazu Informationen aus dem Wiki zu aggregieren und für den Benutzer aufzubereiten. Ein Beispiel hierfür ist die Spezial-Seite *Unused Categories*, welche alle unbenutzen Kategorien auflistet. Ein weiterer wichtiger Verwendungszweck von Special-Pages ist das Bereitstellen von Administrations- bzw. Konfigurationswerkzeugen, die für den Wiki-Betrieb nützlich bzw. notwendig sind. [28]

Namespaces

Einzelne Seiten im Wiki lassen sich zu sogenannten Namespaces gruppieren. Diese Namensräume dienen dazu, auf höchster Ebene den Zweck einer Seite zu unterscheiden. So sind etwa alle Hilfe-Seiten im Namespace *Help:* zusammengefasst. [16]

Templates

Templates dienen dazu wiederkehrende bzw. gleiche Inhalte in einfacher Form auf einer Wiki-Seite einzufügen. Dazu wird zunächst die Vorlage im Namensraum *Templates:* angelegt. Im Anschluss daran kann dieses Template dann auf einer beliebigen Seiten im Wiki referenziert und somit eingebunden werden. Statt für jede Seite den Inhalt manuell einzufügen, kann auf das Template referenziert werden. Dies spart Schreibaufwand und reduziert das Fehlerrisiko auf ein Minimum. Zusätzlich bekommen alle referenzierenden Seiten immer die aktuellste Version der Inhalte. Dies bedeutet, dass nur das Template aktualisiert werden muss, um gewünschte Änderungen auf allen Seiten durchzuführen. [30]

Infobox

Infoboxen sind eine von Wikipedia verwendetet Möglichkeit zur einheitlichen und einfachen Verwendung von Tabellen die grundlegende Daten enthalten. Dabei werden sie mittels Templates jeweils speziell für eine bestimmte Reihe von Artikeln (z.B. alle Städte, Berge, Länder, etc.) gestaltet. Infoboxen dienen als anschauliches Hilfsmittel zum Fließtext und sollen diesen ergänzen. Sie dienen als Übersicht der zentralen Daten eines Artikels. [12]

Semantic-MediaWiki

Semantic-MediaWiki (SMW) [23] ist eine Erweiterung für die MediaWiki-Software mit dem Ziel beim Durchsuchen, Organisieren, Taggen, Browsen und Auswerten von Inhalten zu unterstützen. Im Gegensatz zu traditionellen Wikis, die Texte enthalten, die Computer weder verstehen noch auswerten können, erlaubt es SMW dem Benutzer semantische Annotationen hinzuzufügen. Dies ermöglicht es Rechnern das Wissen, das im Wiki gesammelt ist, zu verarbeiten. So lässt sich etwa die Frage *Wieviele Einwohner hat Deutschland?* in einem entsprechend gepflegten Wiki mit Hilfe der SMW-Erweiterung direkt vom System beantworten.

Semantic-MediaWiki unterstützt mit einer Vielzahl verschiedener Mechanismen, die nachfolgend aufgelistet und erläutert sind:

Automatisch generierte Listen Listen und Tabellen sind ein natürlicher Weg, um Informationen auf einen Blick zu präsentieren. Klassische Auflistungen haben aber das Problem, dass sie manuell auf dem aktuellen Stand gehalten werden müssen. Dies ist ein sehr aufwändiger und fehleranfälliger Prozess. Zusätzlich ist die Anzahl an potentiell interessanten Listen riesig. SMW setzt hier an und erlaubt es dem Benutzer mit Hilfe spezieller Abfrage-Syntax (*#ASK-Queries*) Listen automatisiert zu generieren.

Visuelle Aufbereitung von Informationen Die Informationen lassen sich auf sehr viele Arten präsentieren. Je nach Anforderung können die Daten beispielsweise als Kalender, Karte oder Graph aufbereitet werden. Es gibt hier zahlreiche (Semantic-)MediaWiki-Erweiterungen, die weitere Formate bereitstellen.

Verbesserte Datenstrukturen Klassische Wikis sind meist sehr stark auf kategoriebasierten Strukturen aufgebaut. Die meisten dieser Kategorien ließen sich aber viel einfacher, natürlicher und weniger starr mit Hilfe von semantischen Werten ersetzen. Soll etwa eine Übersichtsseite mit allen Ereignissen des Jahres *1945* erstellt werden, reicht im SMW eine einfache Abfrage auf das Attribut *Jahr* und die Seite wird automatisch generiert. Sobald also die Inhalte annotiert sind, lassen sich beliebig komplexe Zusammenhänge einfach abfragen.

Externe Wiederverwendung Die Daten müssen nicht für alle Zeiten im Wiki gefangen sein. SMW erlaubt es, Inhalte mittels zahlreicher Formate zu exportieren. Neben dem statischen Export bietet SMW eine API, mit deren Hilfe Inhalte dynamisch abgefragt werden können. Dies hat zur Folge, dass das Wiki als Datenquelle für externe Anwendungen dienen kann. Eine doppelte Datenführung entfällt komplett.

Properties und Property-Types im SMW

Properties und Property-Types sind die grundlegenden Methoden, um semantische Inhalte zu einem SMW hinzuzufügen. Dabei können Properties als *Kategorien für Werte in der Wiki-Seite* verstanden werden. Sie werden mittels einfachem Markup definiert:

```
[[Hauptstadt::Berlin]]
```

Dieser Code-Schnipsel definiert den Wert *Berlin* für das Property mit dem Namen *Hauptstadt*. Auf der Wiki-Seite wird jedoch nur der Wert angezeigt. Die semantische Information ist in der Standard-Ansicht versteckt. Es ist also sehr ein-

fach möglich, vorhandene Links oder Werte in Properties umzuwandlen. Das weitaus größere Problem stellt die sinnvolle Benennung der Properties dar. Jede Property kann einen bestimmten Property-Typen annehmen. Beispieltypen wären *Boolean*, *URL* oder der am häufigsten verwendete Typ *Page*. Eine vollständige Liste aller eingebauter Property-Typen lässt sich unter [19] einsehen. Es ist möglich die erlaubten Werte, die ein Property annehmen darf einzuschränken. Dies ist z.B. sinnvoll, wenn die Wertemenge fest definiert und beschränkt ist. Beispielsweise könnten die erlaubten Werte für das Property *Country* gleich der Menge der bekannten Ländernamen sein. [19]

2.5.9 Semantic Forms

Semantic Forms (SMF) ist eine SMW-Erweiterung die es erlaubt, Formulare zum Hinzufügen, Editieren und Abfragen von Daten im Wiki zu verwenden. Diese Formulare können sowohl vom Administrator als auch von den Benutzern selbst erstellt werden. Die SMW-Erweiterung schreibt die Verwendung von Templates bei der Erstellung von semantischen Daten vor. Das direkte Editieren von semantischen Daten wird dabei nicht unterstützt. Stattdessen wird das Markup indirekt über Templates gespeichert. Ein Formular erlaubt dem Anwender eine konfigurierbare Menge an Templates und Abschnitten auszufüllen und somit den Inhalt einer Seite zu definieren. [22]

2.5.10 Semantic-Drilldown

Semantic-Drilldown (SMD) ist eine SMW-Erweiterung, die es ermöglicht, einen *Drill-down* auf die Daten eines Wikis durchzuführen. Dabei werden Kategorien und Filter auf semantische Properties verwendet. Der Benutzer beginnt seine Suche immer auf der *Browse-Date*-Seite. Dort werden alle Top-Level-Kategorien angezeigt. Eine Top-Level-Kategorie zeichnet sich dadurch aus, dass sie keine Unterkategorie einer anderen Kategorie ist. Von dieser Ansicht aus kann der

Benutzer die Ergebnismenge auf zwei verschiedene Arten einschränken. Die erste Möglichkeit ist die Auswahl von Unterkategorien. Es ist somit möglich, durch den gesamten Kategoriebaum zu navigieren. Die zweite Möglichkeit ist die Verwendung bzw. das Erstellen von Filtern auf semantische Properties. So lässt sich beispielsweise eine Antwort auf die Frage *Welche deutschen Städte haben eine Einwohnerzahl von mehr als 50000* einfach finden (natürlich unter der Voraussetzung, dass die entsprechenden Daten vorhanden sind). [21]

2.5.11 Wiki-Editor

Der Wiki-Editor ist eine MediaWiki-Erweiterung, die ein erweiterbares Framework zur Erstellung von feature-basierten Modulen bereitstellt. Diese Module dienen zur Verbesserung der Anwendererfahrung beim Editieren von Seiten. Mit Hilfe des Wiki-Editors muss der Anwender das Wiki-Markup nicht mehr von Hand schreiben, sondern kann durch Klicken eines Toolbar-Buttons einen geführten UI-Prozess auslösen. So kann beispielsweise ein zuvor selektiertes Textfragment in einen Hyperlink konvertiert werden. Der Benutzer wird hierzu vom Editor mit Hilfe eines Eingabedialogs nach der Zieladresse gefragt, welche dann automatisch zu korrektem Wiki-Markup umgewandelt wird. Der Wiki-Editor kann somit als die Vorstufe sogenannter *What You See Is What You Get (WYSIWYG)*-Editoren, die schon beim Editieren das fertige Formatierungsergebnis anzeigen können, verstanden werden. [35]

2.5.12 Semantic Result Formats (SRF)

SRF ist eine SMW-Erweiterung, die eine große Anzahl von Ergebnisformaten zu den *Inline-Queries* hinzufügt. Dabei lassen sich die gefunden Daten beispielsweise als Chart rendern, als Kalender anzeigen oder auch als Graph visualisieren. Die Erweiterung ist für einige Formate auf die *Graphviz*-Bibliothek angewiesen. [24]

2.5.13 Graphviz

Graphviz ist eine Open-Source-Software zur Visualisierung von Graphen. Graphviz besteht aus mehreren Komponenten, von denen die wichtigste das sogenannten Graphviz-Layout-Programm ist. Dieses wird mit Graphenbeschreibungen in einer einfachen, aber mächtigen Beschreibungssprache befüllt und erstellt daraus Diagramme in verschiedenen Formaten wie z.B. SVG, PDF oder Postscript. Diese Beschreibungssprache (genannt DOT) ermöglicht es sowohl die Struktur eines gerichteten oder ungerichteten Graphen zu beschreiben als auch die Visualisierung dieses Graphen zu steuern. [6]

2.5.14 Typen von MediaWiki-Erweiterungen

Parser-Funktion

Diese Art von Erweiterung integriert sich sehr eng in den Standard-Wikiparser. Sie kann im Gegensatz zur Tag-Erweiterung mit anderen Wiki-Elementen auf einer Seite interagieren. So kann z.B. die Ausgabe einer Parser-Funktion als Template-Parameter verwendet werden. [17]

Tag-Erweiterung

Es ist für angepasste Projekte meistens sehr sinnvoll, das eingebaute Wiki-Markup um eigene Tags zu erweitern. Mit Hilfe von Tag-Erweiterungen ist genau dies möglich. Um beispielsweise das Wiki um einen Spenden-Tag zu erweitern, muss nur eine Callback-Funktion definiert werden, die beim Parsen aufgerufen wird, sobald der <donation>-Tag erreicht wurde. Der Parser findet und ersetzt alle Vorkommen des Tags mit der von der Callback-Funktion gelieferten HTML-Ausgabe. [29]

Special-Page

Special-Pages sind Seiten, die vom MediaWiki bzw. den MediaWiki-Erweiterungen bei Bedarf generiert und angezeigt werden. Jede Special-Page hat dabei eine spezielle Funktion. So gibt es beispielsweise eine Special-Page, die alle Seiten mit mindestens einem ausgehenden Link anzeigt. Die Special-Pages sind in einem eigenen Namespace angesiedelt (*Special:*) und können nicht wie andere Seiten editiert werden. Zusätzlich können bestimmte Rechte erforderlich sein, um eine Special-Page zu verwenden (z.B. Adminrechte, um Benutzerkonten zu verwalten). Um eine Special-Page-Erweiterung zu erstellen, ist etwas mehr Aufwand nötig als bei anderen Erweiterungen. Die Minimalkonfiguration besteht aus einer Setup-Datei, einer Internationalisierungs-Datei, einer Alias-Datei und einer Datei mit der tatsächlichen Funktionalität. Die Setup-Datei ist dafür zuständig, die Pfade der verschiedenen Dateien der Erweiterung im System zu registrieren. Zusätzlich teilt sie der MediaWiki-Software mit, wie der Klassenname sowie der Anzeigename der Special-Page lauten. Bei Bedarf lässt sich hier eine Special-Page-Gruppe festlegen. Die Hauptdatei (*Body-File*) enthält minimal eine Unterklasse von *SpecialPage*, die die tatsächliche Funktionalität der Erweiterung enthält. Die Nachrichten- bzw. Internationalisierungs-Datei enthält den Titel sowie alle zusätzlichen Nachrichten und Anzeigetexte der Erweiterung in allen zu unterstützenden Sprachen. Mit Hilfe der Alias-Datei lässt sich der Name der Special-Page internationalisieren. Somit ist es bei mehrsprachigen Installationen möglich, die Special-Page in der jeweilig aktiven Sprache aufzurufen (z.B. *Spezial:MeineErweiterung* vs. *Special:MyExtension*). [27]

Hook

Mit Hilfe von Hooks kann eigener Code beim Auftreten bestimmter Events ausgeführt werden. Das MediaWiki definiert eine Vielzahl solcher Events, in die sich eine Erweiterung einhaken kann. Hooks und Event-Handler sind da-

zu gedacht, zusätzliche Aufgaben zu definierten Zeitpunkten bzw. Ereignissen zu erledigen. Beispielsweise könnte ein Event-Handler immer dann einen Log-Eintrag erstellen, wenn eine Wiki-Seite gespeichert wurde. [11]

Skin

Mit dieser Form von Erweiterung kann das Look-and-Feel der MediaWiki-Installation den eigenen Anforderungen angepasst werden. Dabei lassen sich sowohl einfache Anpassungen wie das Ändern von Farben als auch komplexe Änderungen wie das Umgestalten des Seitenlayouts durchführen. [25]

3 Anforderungsanalyse: Pattern-Collaboration-Tool

Sollen Patterns erstellt werden, haben sich einige grundsätzliche Vorgehensprozesse bewährt. So beschreiben etwa Wellhausen und Fießer [34] das Vorgehen, um aus einer speziellen Problemlösung ein allgemeingültiges Pattern zu entwickeln. Dabei werden Patterns üblicherweise in einem bestimmten Format, einer Sprache, formuliert. Meszaros und Doble [15] liefert uns eine Meta-Sprache, die die *Best Practices* der Patternerstellung erfasst. Normalerweise sind Patternsprachen domänenspezifisch [7]. Dies bedeutet, dass jede Domäne spezielle Anforderungen an die Präsentation der jeweiligen Patterns haben kann. Ein PCT muss also flexibel genug sein, diese Anforderungen zu erfüllen. Ein wichtiger Schritt bei der Patternerstellung ist das gegenseitige Review von Patternkandidaten [9]. Um die Qualität dieses Reviewprozesses möglichst hoch zu halten, ist es also sinnvoll, nach klar definierten Regel vorzugehen. Wir entwickeln die Anforderungen an ein PCT auf Basis der oben genannten Quellen. Zusätzlich werden weitere allgemeine sowie projektspezifische Anforderungen erarbeitet und festgehalten.

3.1 Rollenmodell

Im folgenden Abschnitt wird ein für ein PCT geeignetes Rollenmodell erarbeitet. Hierbei werden zunächst die einzelnen Rollen beschrieben. Im Anschluss

werden für jede Rolle spezifische Anforderungen in Form von User-Stories festgehalten. Abbildung 3.1 zeigt die Interaktionen zwischen den Rollen untereinandern und den Rollen und dem System.

3.1.1 Admininistrator

Die Rolle des Administrators umfasst in erster Linie Wartungsaufgaben. Ein Administrator ist zunächst einmal für die Installation des Systems zuständig. Er hat Grundkenntnisse in der Systemadministration, denn er muss erkennen können, ob sich eine Laufzeitumgebung zur Installation und den Betrieb der Software eignet, in dem er beispielsweise sicherstellt, dass die Anforderungen erfüllt sind, also auch eventuell notwendige Softwarepakete installiert werden können. Der Administrator ist neben der Einrichtung auch für einen reibungslosen Betrieb des Systems zuständig. Er ist in der Lage, Wartungsaufgaben wie etwa regelmäßige Systemaktualisierungen oder auch das Erstellen von Systemsicherungen durchzuführen. Für die anderen Benutzer des Systems ist der Administrator zentraler Ansprechpartner für alle technischen Probleme mit der Software.

3.1.2 Domänenexperte

Der Domänenexperte ist ein Spezialist eines Wissensgebietes. Er kennt Anforderungen und die typischen Fallstricke seines Bereichs und kann diese in verständlicher Form festhalten. Er kennt die Standard-Patterns seiner Domäne und ist darüber hinaus in der Lage, neue Patterns zu formulieren. Er weiß, wie sich die wirkliche Welt in die Anwendung abbilden lässt. Der Domänenexperte hat eine Vorstellung davon, wie sich die Patterns einer Domäne am Besten visualisieren lassen. Es obliegt ihm, geeignete Vorlagen zu entwickeln und diese im System für die Nutzer zu hinterlegen.

3.1.3 Autor

Autoren haben so viel Erfahrung in ihrem Fachgebiet erlangt, dass sie in der Lage sind, Muster zu erkennen und daraus Patterns zu entwickeln. Die Hauptaufgabe umfasst das Anlegen bzw. Überarbeiten von Patterndokumenten im System. Autoren sorgen durch die Pflege von semantischen Annotationen zwischen den einzelnen Patterns dafür, dass Benutzer des Systems bei ihrer Suche nach den passenden Patterns unterstützt werden. Durch gemeinsames Arbeiten an Patterndokumenten halten Autoren die Qualität der Patterns auf einem möglichst hohen Niveau. Sie erkennen Fehler und Probleme und eliminieren diese. Dies bedeutet, dass ein PCT ohne Autoren nicht funktionieren kann. Sie sind also zentral für den Erfolg bzw. Misserfolg verantwortlich.

3.1.4 Anwender

Anwender sind die tatsächlichen Nutzer eines PCT. Sie haben ein zumindest grundlegendes Verständnis der Domäne und möchten durch die Betrachtung von Patterns ihr Wissen erweitern und typische Anfängerfehler vermeiden. Ein Anwender möchte bei der Suche nach einem passenden Pattern möglichst einfach und schnell zum Ziel kommen. Er legt daher großen Wert auf eine hohe Benutzerfreundlichkeit und Nützlichkeit des Systems. Für Anwender ist es wichtig, einzelne Patterndokumente bewerten zu können bzw. sich die Bewertungen anderer anzuschauen.

3.2 User Stories

Typische Benutzeranforderungen an ein PCT werden in diesem Abschnitt mit Hilfe von sogenannten User Stories für die einzelnen Rollen beschrieben.

3.2.1 Administrator

Als Administrator kann ich das Grundsystem einfach installieren und einrichten

Reiners et al. [20] identifizieren die Möglichkeit des einfachen, flexiblen und zeitnahen Mitwirkens aller Beteiligten am Patternerstellungsprozesses als zentrale Anforderung. Um dem gerecht zu werden, darf das System z.B. keine besonderen Anforderungen an die Laufzeitumgebung stellen. Es muss domänenunabhängig sein und nach Möglichkeit auch von Nicht-Informatikern installiert werden können. Idealerweise lässt sich die Anwendung auf einem weit verbreiteten LAMP-Software-Stack betreiben.

Als Administrator kann ich alle administrativen Aufgaben direkt aus der Anwendung heraus tätigen

Bei der Anwendung der Lösung von Fürst [4] hat sich gezeigt, dass die Verwendung von externen Programmen zur Installation bzw. Konfiguration aufwändig, kompliziert und teilweise sehr fehleranfällig ist. Wir haben daher an unsere Lösung die Anforderung, dass das System in sich geschlossen sein muss. Dazu gehört, dass keine externen Tools benötigt werden, um administrative Aufgaben, wie z.B. das Konfigurieren des Anwendungsdesigns, durchzuführen. Zusätzlich müssen Importe und Exporte direkt aus der Anwendung heraus möglich sein.

Als Administrator kann ich Zugriffsrechte und Gruppenzugehörigkeiten der Benutzer setzen

Um das PCT z.B. vor Vandalismus zu schützen, sollte das System die Konfiguration unterschiedlicher Zugriffsrechte unterstützen. Der Administrator sollte also in der Lage sein, mit Hilfe eines Rechtesystems verschiedene Benutzer-

gruppen mit unterschiedlichen Zugriffsrechten zu pflegen. Ein Benutzer sollte dabei gleichzeitig in mehreren Gruppen sein können.

Als Administrator kann ich sämtliche Inhalte einfach Exportieren bzw. Importieren

Es muss mit Mitteln der Anwendung möglich sein, Datenexporte bzw. -importe durchzuführen. Hierbei sollten sich sowohl einzelne logische Einheiten (z.B. einzelne Dokumente oder aber auch alle Dokumente einer bestimmten Kategorie) als auch der komplette Datenbestand exportieren lassen. Das Datenaustauschformat sollte nicht proprietär sein. Vorteilhaft wäre es, wenn es sich um ein weitverbreitetes Format wie z.B. XML handeln würde. Um diese Funktionalität speziell auch für Backups nutzen zu können, müssen in einem Export alle nötigen Informationen zur Wiederherstellung enthalten sein. Siehe hierzu auch Unterabschnitt 3.3.5.

Als Administrator kann ich einen initialen Datenimport durchführen

Es muss möglich sein, eventuell vorhandene externe Daten in die Anwendung zu importieren. Hierfür sollte die Standard-Import-Funktion verwendet werden. An dieser Stelle ist es in Ordnung, einen externen Konverter vom Quellformat in das Zielformat zu verwenden, da die Zieldaten mit hoher Wahrscheinlichkeit domänenspezifisch sind (Templates etc.). Es muss also für jede Installation (und somit Domäne) ein Konverter erstellt werden bzw. ein vorhandener angepasst werden.

Als Administrator bin ich der zentrale Ansprechpartner bei technischen Fragen und Problemen mit der Anwendung

Der Administrator ist für den reibungslosen Betrieb der Anwendung zuständig. Es liegt daher nahe, dass er auch bei eventuellen Problemen und Fragen der Anwender zentraler Ansprechpartner und Unterstützer ist.

Als Administrator kann ich das Design der Anwendung anpassen

Tractinsky [31] zeigte in seiner Arbeit, dass es sehr starke Anzeichen dafür gibt, dass eine ästhetisch ansprechende Benutzeroberfläche beim Anwender ein Gefühl von höherer Benutzerfreundlichkeit auslöst. Er folgert daraus, dass die Ästhetik einen wesentlichen Einfluss auf die Akzeptanz eines Systems haben könnte [31]. Wir leiten hieraus die Anforderung der visuellen Anpassbarkeit des PCT ab. Es muss möglich sein, das Grunddesign der Anwendung an die gegebenen Anforderungen der Domäne anzupassen. Hier muss unterschieden werden zwischen einfachen Änderungen (z.B. Farben oder Schriften etc.) und komplexen Änderungen des Grundlayouts. Ersteres sollte einfach aus der Anwendung heraus möglich sein. Letzeres sollten über sogenannte Themes möglich sein.

Als Administrator kann ich allgemeine (domänenunabhängige) Templates pflegen

Die Anwender sollen bei der Inhaltserstellung durch Templates unterstützt werden. Zusätzlich soll mit Hilfe von Templates eine einheitliche Inhaltsdarstellung erreicht werden. Unter Templates verstehen wir hier Text-Fragmente in einer bestimmten Markup-Sprache, deren Aufruf einem Funktionsaufruf aus einer klassischen Programmiersprache ähnlich ist.

Als Administrator kann ich andere Benutzer verwalten

Ein Administrator soll jeden Benutzeraccount verwalten können. Dazu gehört beispielsweise das Freischalten von neuen Accounts oder das Zurücksetzen von Benutzerpasswörtern. Des Weiteren ist er dafür zuständig, Benutzer zu entsprechenden Rechtegruppen hinzuzufügen.

3.2.2 Domänenexperte

Als Domänenexperte kann ich die reale Welt in der Anwendung abbilden

Ein Domänenexperte legt fest, wie die reale Welt in der Anwendung repräsentiert wird. Wir haben die Anforderung Inhalte im System mittels semantischer Annotationen zu verknüpfen bzw. zu versehen (siehe Abschnitt 3.2.3). Der Domänenexperte ist dafür zuständig, die relevanten semantischen Eigenschaften zu identifizieren und diese im Anschluss im System zu konfigurieren.

Als Domänenexperte kann ich die Darstellung der Patterns mit Hilfe von Templates konfigurieren

Der Domänenexperte weiß am besten, wie sich die Patterns seiner Domäne sinnvoll visualisieren lassen. Er benötigt daher eine Funktion im System, um entsprechende Templates verwalten zu können.

Als Domänenexperte kann ich die Ansicht der Dateneingabemasken mit Hilfe von Templates konfigurieren

Um dem Anwender nicht zuzumuten, sich mit Markup-Syntax beschäftigen zu müssen sowie dafür zu sorgen, dass die Inhalte einheitlich dargestellt werden, soll der Großteil der Benutzereingaben über Formulare erfolgen. Da diese For-

mulare domänenspezifisch sind, braucht es eine Möglichkeit, sie z.B. mittels Templates zu konfigurieren.

3.2.3 Autor

Als Autor kann ich Patterndokumente einfach und schnell anlegen und editieren

Gemäß Reiners et al. [20] ist die Möglichkeit des einfachen, flexiblen und zeitnahen Mitwirkens am Patternerstellungsprozesses eine zentrale Anforderung. Ein Autor muss daher jederzeit und nach Möglichkeit ohne vorherige Registrierung in der Lage sein, Patterndokumente anzulegen bzw. zu editieren. Hierbei wird er mittels vorgefertigter Templates unterstützt. Dies dient in erster Linie auch dazu, eine einheitliche Darstellung der Inhalte zu fördern bzw. vorzuschreiben. Der Autor soll dabei nicht den Quellcode editieren müssen, sondern eine Formular-Eingabemaske präsentiert bekommen. Es kann Dokumente geben, die sich nur mittels bestimmter Zugriffsrechte editieren lassen.

Als Autor kann ich Patterndokumente semantisch annotieren bzw. verknüpfen

Sowohl Reiners et al. [20] als auch Pavlic, Hericko und Podgorelec [18] identifizieren die Möglichkeit der semantischen Annotation von Patterns als wichtige Anforderung. Die Verknüpfung von einzelnen Patterns soll im PCT mittels typisierten Hyperlinks erfolgen. Neben der reinen Verknüpfung von Patterns soll es auch möglich sein, semantische Annotationen zu einem Patterndokument hinzuzufügen. Diese Annotationen sollen in der Standardansicht des Patterns entweder unsichtbar sein oder in speziell formatierten Bereichen (*Infoboxen*) präsentiert werden. Es soll eine spezielle Ansicht geben die alle Annotationen in einem Patterndokument anzeigt.

Als Autor kann ich Aggregationsseiten erstellen und verwalten

Aggregationseiten sind immer dann nützlich, wenn der Anwender auf der Suche nach einer bestimmten Information ist bzw. sich einen Überblick über ein Themengebiet verschaffen möchte. In einem semantisch entsprechend annotierten System, lassen sich mittels Aggregationsseiten beispielsweise auch klassische (starre) Kategorieseiten ersetzen. Es soll daher möglich sein, Patterndokumente nach bestimmten semantischen Kriterien zu aggregieren. Beispiel: Zeige mir alle Patterns, die eine related-to-Beziehung zu Pattern XYZ besitzen. Diese Aggregationsseiten sollen persistent sein, sodass auch andere Benutzer die Informationen nutzen können. Typische Anwendungen sind z.B. das Erstellen von Statistikseiten.

Als Autor bleiben meine Urheberinformationen bei Änderungen eines Patterndokuments erhalten

Um Autoren etwas mehr Anreiz zur Beteiligung zu geben, sollen Autoreninformationen über alle Revisionen eines Patterndokuments hinweg erhalten bleiben [20]. Diese Informationen sollen leicht zugänglich sein, aber nicht zu aufdringlich präsentiert werden.

Als Autor kann ich bei der Pflege von Patterndokumenten beliebige Templates verwenden

Um die Pflege von Patterndokumenten so einfach und fehlerlos wie möglich zu gestalten, sollte den Autoren Templates zur Verfügung stehen. Jeder Autor soll die Möglichkeit haben, alle Templates die vorher durch einen Admin bzw. Domänenexperten angelegt wurden, zu nutzen. Um dem Community-Gedanken gerecht zu werden, wäre es vorteilhaft, wenn Autoren eigene Templates anlegen könnten, um diese dann für alle anderen Autoren verfügbar zu machen.

Als Autor kann ich bestimmte Dokumente (z.B. meine eigenen) vor Änderungen schützen

Um das PCT vor Vandalismus zu schützen soll es möglich sein, Änderungen an vorher speziell geschützten Dokumenten zu verhindern. Dies kann z.B. in der Form geschehen, dass diese Seiten nur für registrierte Benutzer editierbar sind. Alternativ können auch bestimmte Zugriffsrechte notwendig sein, um beispielsweise spezielle Übersichtsseiten ändern zu können.

Als Autor kann ich verwaiste Links in den Patterndokumenten einfach und zuverlässig finden

Nach Reiners et al. [20] ist es wichtig, dass Patterndokumente aktuell gehalten werden. Dazu gehört es unter anderem auch, dass verwaiste Links, die z.B. durch das Löschen von Patterndokumenten entstehen können, gefunden werden. Da diese Aufgabe kaum manuell zu bewältigen ist, ist es nötig eine spezielle Funktionalität in der Anwendung zu haben, die Autoren bei der Behebung dieser Inkonsistenzen unterstützt.

Als Autor kann ich mich bei Referenzierung meiner Patterndokumente vom System benachrichtigen lassen

Da sich die Pflege von Rückwärtsverknüpfungen in referenzierten Patterndokumenten nur sehr schwer automatisiert lösen lässt, muss es für Autoren die Möglichkeit geben, sich bei Referenzierung ihrer Patterndokumente vom System benachrichtigen zu lassen. Dies kann z.B. über das Versenden von E-Mails erfolgen.

Als Autor kann ich mich bei Änderungen an meinen Patterns vom System benachrichtigen

Ein Autor ist für die Qualität des Patterndokumentes verantwortlich. Es ist daher wichtig, dass er bei eventuellen Änderungen an seinen Patterns benachrichtigt wird. Er kann dann entscheiden, ob eine Änderung den Qualitätsstandards entspricht und sie entweder absegnen oder rückgängig machen bzw. anpassen.

3.2.4 Anwender

Als Anwender kann ich mittels einer Volltextsuche nach Patterns suchen

Das System muss eine Volltextsuche zur Verfügung stellen. Diese ermöglicht den Anwendern das Repository nach einzelnen Begriffen zu durchforsten. Die Suche sollte einfache logische Verknüpfungen zwischen den Suchbegriffen erlauben (AND, OR, NOT).

Als Anwender kann ich mit Unterstützung des Systems durch das Repository navigieren

Der Anwender soll bei seiner Suche nach einer geeigneten Lösung durch die Anwendung unterstützt werden. So sollen ihm etwa bei der Lektüre eines Artikels verwandte bzw. verknüpfte Themen vorgeschlagen werden. Zusätzlich soll es möglich sein, sich durch die Auswahl verschiedener Kriterien und der damit verbundenen Verfeinerung der Anforderungen bis zu einem möglichen Lösungs-Pattern durchzuhangeln (z.B. mittels schrittweiser Filterung). Hier sollte ein Algorithmus entwickelt werden, der für eine Menge von Patterns weitere vorschlägt.

Als Anwender kann ich Erfolgs- bzw. Misserfolgs-Beschreibungen zu einem Patterndokument hinzufügen

Um den Reifegrad eines Patterndokuments voran zu bringen, ist es laut Reiners et al. [20] unter anderem nötig, erfolgreiche Anwendungen dieses Patterns zu dokumentieren. Erfolgreiche Anwendungen erhöhen auch die Zuversicht anderer Anwender und zeigen in welchen Bereichen das Pattern funktionieren kann. Fast genauso wichtig ist nach Reiners et al. [20] auch die Dokumentation von fehlgeschlagenen Anwendungen, da sich dadurch die Bereiche zeigen für die ein Pattern eher ungeeignet ist.

Als Anwender kann ich Alternativen bzw. verwandte Themen zu einem Pattern suchen

Für die korrekte Auswahl eines Patterns für ein gegebenes Problem ist es wichtig, mögliche Alternativen und verwandte Themenbereiche zu kennen. Das System bietet daher idealerweise eine auf einen Blick einsehbare Liste mit alternativen und verwandten Patterndokumenten.

Als Anwender kann ich Patterns bewerten und kommentieren

Eine Bewertungsfunktion ist ein nützliches Werkzeug für die Community, um den Reifegrad und die Korrektheit eines Patterns zu bewerten. Da beispielsweise ein einfaches 5-Sterne-Rating hierbei kaum Informationen über gute und schlechte Aspekte des Patterndokuments liefert, ist es nötig, dass Anwender Kommentare hinterlassen können, mit deren Hilfe sie ihre Bewertung begründen können.

Als Anwender kann ich persönliche Leselisten bzw. Favoriten pflegen

Um bei vielen Patterndokumenten nicht den Überblick zu verlieren, ist es sinnvoll, eine Möglichkeit einzubauen, wie Anwender Referenzen auf Pattern-Do-

kumente mittels persönlicher Favoritenlisten verwalten können. Dabei soll es sehr einfach möglich sein, ein bestimmtes Pattern zu den Favoriten hinzuzufügen. Dies kann z.B. mittels speziellem Navigationselement auf der Pattern-Seite geschehen.

Als Anwender werde ich von der Anwendung bei der Betrachtung von Patterns mit nützlichen Zusatzinformationen versorgt

Reiners et al. [20] fordern intelligente Visualisierungen der vorhandenen Daten mit dem Ziel, den Benutzer zu unterstützen. Hierbei geht es z.B. darum, eine Übersicht des aktuellen Zustandes des PCT bezüglich der relativen und absoluten Position der einzelnen Patterns innerhalb der Anwendung zu erhalten. Zusätzlich sollen die Relationen der Patterns untereinander visualisiert werden (z.B. mittels eines geeigneten Graphs). Die einzelnen Visualisierungen sollen sich nach Möglichkeit konfigurieren lassen.

3.3 Sonstige Anforderungen

In diesem Unterkapitel sollen Anforderungen gesammelt werden, die sich nicht in die zwei vorherigen Abschnitte einsortieren lassen. Dazu gehören z.B. softwaretechnische Anforderungen, wie etwa die Wartbarkeit eines PCT.

3.3.1 Wartbarkeit

Im Folgenden sollen Anforderungen formuliert werden, die sich grob unter dem Begriff Wartbarkeit gruppieren lassen:

Wir möchten bei der Entwicklung eines PCT möglichst auf Open-Source-Lösungen setzen. Die Vergangenheit hat gezeigt, dass die Verwendung proprietärer Komponenten sehr schnell zu großen Problemen führen kann. So hatte etwa Fürst [4] große Schwierigkeiten, nachdem einige zentrale Komponenten

nur noch kommerziell erhältlich waren bzw. die Open-Source-Versionen nicht mehr aktualisiert wurden. Das Resultat war, dass das System nicht mehr mit vertretbarem Aufwand auf aktuelle Versionen aktualisiert werden konnte.

Das PCT soll auf einem komponentenbasierten System aufgebaut werden. Komponenten haben den Vorteil, dass sie einfach austauschbar sind sowie das System einfach zu erweitern ist und somit leicht an die eigenen Anforderungen anzupassen ist. Dabei soll möglichst wenig Neuentwicklung betrieben werden. Stattdessen wollen wir auf ein etabliertes Grundsystem und erprobte Komponenten setzten. Es dürfen keine Komponenten verwendet werden, die ein manuelles und unkontrollierbares Patchen der Kern-Anwendung voraussetzen. Die Anwendung soll so simpel wie möglich gehalten werden. Dazu gehört auch, dass das KISS-Prinzip [13] befolgt wird. Das PCT soll also nicht mehr können als tatsächlich gefordert. Dies bedeutet vor allem, dass keine unnötigen Funktionen und Komponenten eingebaut werden, da diese nur zu höherem Wartungsaufwand führen würden und ansonsten keinen Nutzen hätten.

Um den Kreis potentieller Entwickler nicht zu sehr einzuschränken, soll das Grundsystem in einer weitverbreiteten Programmiersprache entwickelt sein. Zusätzlich muss eine gute, das heißt eine nützliche, ausgereifte und verständliche, Dokumentation vorhanden sein. Da eine Dokumentation selten alle Bereiche der Entwicklung abdeckt, ist eine professionelle und hilfsbereite Community unerlässlich.

Das System soll robust gegenüber nachträglichen Änderungen der Domänenkonfiguration sein. Es sollte keine Neuinstallation nötig sein, um Anpassungen durchführen zu können. Zusätzlich darf es nicht zum Datenverlust bzw. zur Datenkorruption kommen, wenn das System umkonfiguriert wird.

3.3.2 Pattern Maturity

Reiners et al. [20] fordern ein flexibles System, welches eine Beurteilung des Pattern-Reifegrads mittels verschiedener Belege ermöglicht. Hierbei sollen so-

wohl unterstützende als auch widerlegende Faktoren mit einfließen. Die Gewichtung der einzelnen Aspekte soll anpassbar sein. Ein solcher Faktor kann z.B. ein erfolgreiches Praxisbeispiel sein.

3.3.3 Transparenz

Die Anwendung soll Änderungen an den Artikeln für alle Beteiligten nachvollziehbar und transparent präsentieren [20]. Dies kann z.B. mit Hilfe einer öffentlich einsehbaren Änderungshistorie geschehen. Zu diesem Punkt gehört auch die Anforderung, dass bisher vernachlässigte Patterns sichtbar gemacht werden können, um sie gegebenfalls neu zu evaluieren.

3.3.4 Semantische Verknüpfung

Es soll möglich sein, Verknüpfungen (= Hyperlinks) zwischen Artikeln zu erstellen, welche den Zusammenhang zwischen zwei Artikeln genauer beschreiben (z.B. Pattern A ist ein Spezialfall von Pattern B). Diese Links lassen sich durch Annotationen ergänzen, welche die Verknüpfung im Allgemeinen und/oder im Kontext des Patterns genauer beschreiben. Diese Zusatzinformation ist meist patternspezifisch und daher nicht direkt an den Link annotiert, sondern im Patterntemplate beschrieben. Links können zwar automatisch eingefügt werden, diese Zusatzinformationen aber nicht. Dies bedeutet, dass die Autoren der referenzierten Artikel ggf. benachrichtigt werden müssen. Bestimmte Linktypen sollen bidirektional sein (z.B. bidirektionale *related-to*-Beziehungen). Hier ist eine Konsistenzprüfung nötig um z.B. tote Links zu erkennen und evtl. automatisch zu entfernen.

3.3.5 Schnittstelle

Abgeschlossene Systeme haben immer den Nachteil, dass Daten nur durch die Anwendung direkt zugänglich sind. Dies bedeutet auch, dass man auf die im

System hinterlegten Auswertungen und Visualisierungen beschränkt bleibt. Gibt es etwa neue Anforderungen, muss die Anwendung angepasst werden. Um diesen Lock-In zu vermeiden, soll die Anwendung eine Daten-Schnittstelle nach außen mitliefern. Mit Hilfe dieser Schnittstelle sollen die vorhandenen Daten, evtl. unter Beachtung des Rechtesystems, exportierbar und somit extern auswertbar sein.

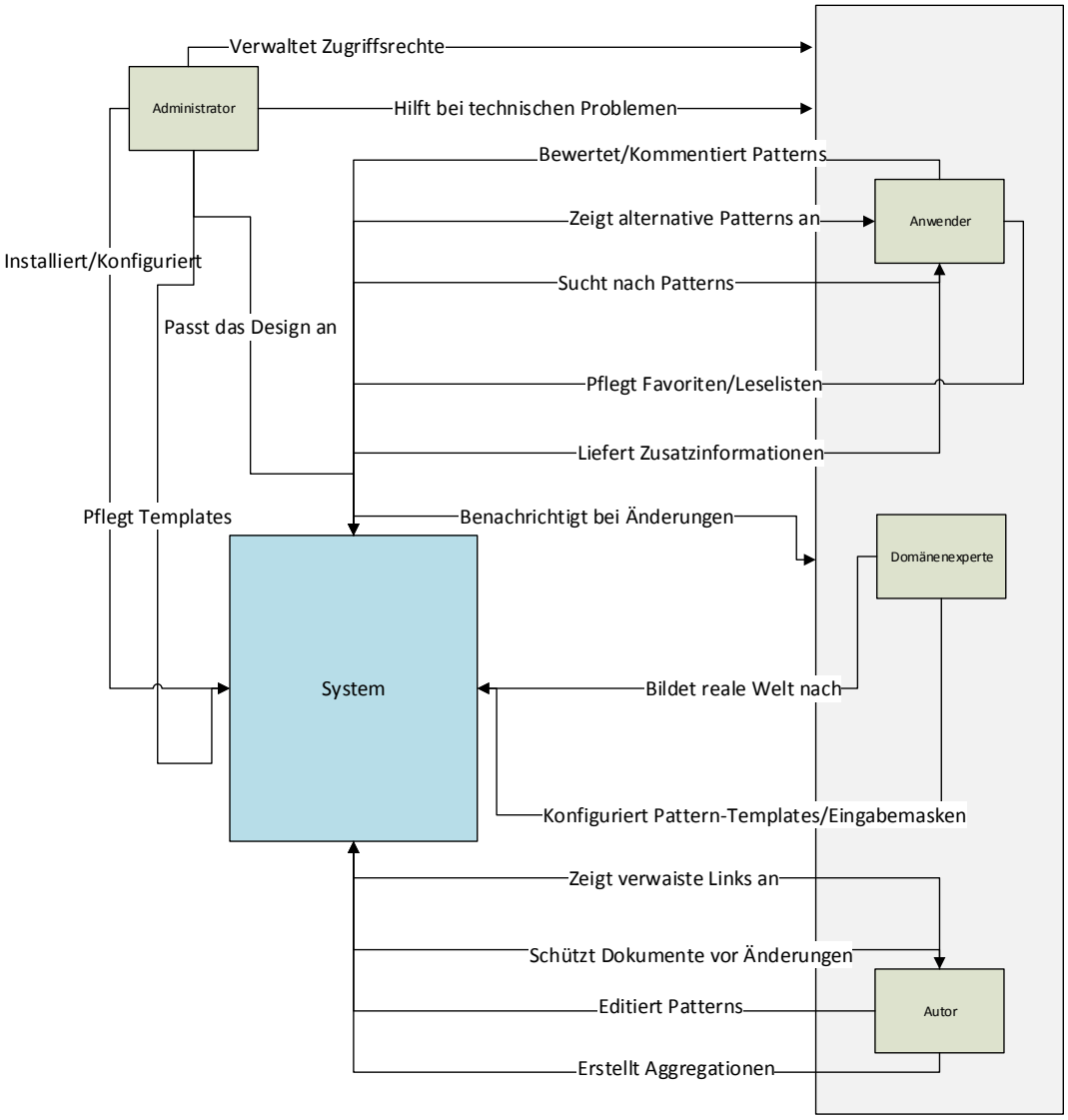

Abbildung 3.1: Interaktionen zwischen den Rollen untereinander bzw. den Rollen und dem System

4 Implementierung

In diesem Kapitel sollen zunächst die Anforderungen aus Kapitel 3 mit einem bereits existierenden System verglichen werden. Nach diesem Abschnitt folgt die Beschreibung der Design-Entscheidungen sowie die Beschreibung des tatsächlich realisierten Prototyps.

4.1 Analyse des vorhandenen Systems

In diesem Abschnitt betrachten wir die Lösung von Fürst [4] und arbeiten die Stärken und Schwächen im Hinblick auf unsere im Kapitel 3 definierten Anforderungen heraus. Fürst basiert seine Lösung auf dem MediaWiki in Kombination mit der DataWiki-Erweiterung der Firma DIQA. Abbildung 4.1 zeigt die grobe Architektur seiner Lösung. Beachtenswert ist vor allem der über den sogenannten *DIQA-Triplestore-Connector* angebundene *Apache Jena Triplestore*. Mit Hilfe des Connectors werden semantische Informationen aus dem Wiki in den Triplestore synchronisiert. Auf den Triplestore können im Gegensatz zu den normalen *#ASK-Queries* des SMW weitaus komplexere *SPARQL-Abfragen* aus dem Wiki heraus abgesetzt werden. Wie sich später gezeigt hat, liegt ein Großteil der Mächtigkeit dieser Kombination in Ermangelung entsprechender Anforderungen brach. Sie führt nur zu deutlich höherer Komplexität.

Fürst identifiziert in seiner Ausarbeitung selbst einige Nachteile seiner Lösung. Er stellt fest, dass die indirekte Abbildung der Datenmodellontologie auf das Wiki sehr komplex ist [4]. Zusätzlich ist die Installation des Systems auf

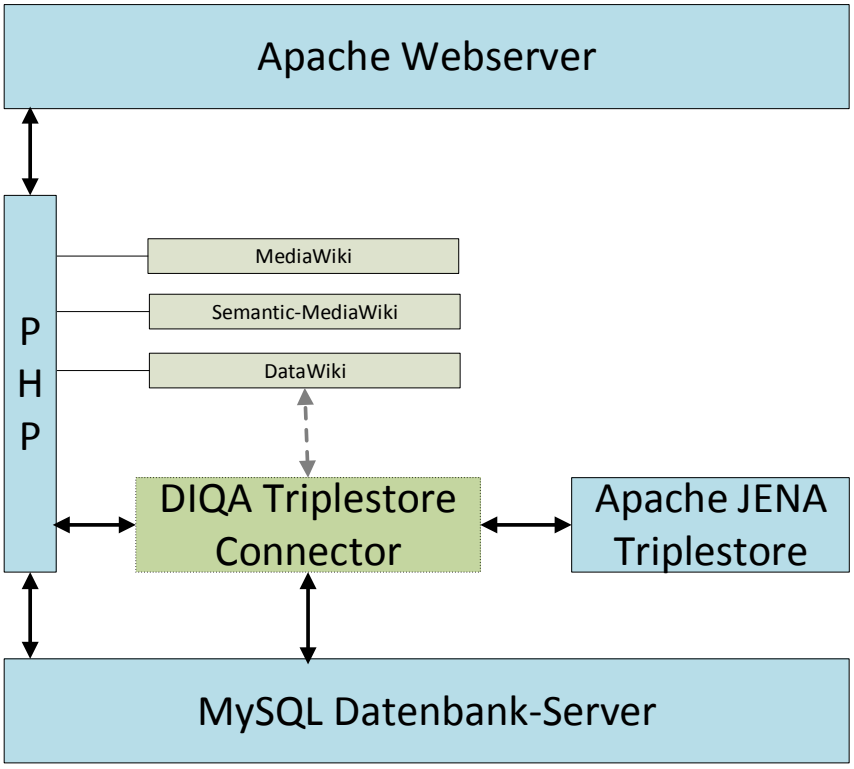

Abbildung 4.1: Architekturüberblick basierend auf der Lösung von Fürst [4]

einem Nicht-Windows-Server sehr aufwändig, kompliziert und fehleranfällig. Ein weiteres Problem ist, dass der Administrator ein sehr breit gefächertes Wissen benötigt: Er muss sich mit der Ontologiemodellierung in OWL auskennen, Programmierkenntnisse in JAVA vorweisen (für das Import-/Konfigurations-Tool) sowie in der Linux-Server-Administration bewandt sein [4]. An dieser Stelle sei erwähnt, dass es dem Autor auch nach mehreren Versuchen nicht gelungen ist, dass System auf einem Linux-Server fehlerfrei zu installieren bzw. zu betreiben. Fürst nennt als weiteren Nachteil die Unsicherheit über die Zukunft der DataWiki-Erweiterung. Wie sich zu Beginn dieser Diplomarbeit gezeigt hat, ist diese Befürchtung leider zur Realität geworden, da die Wartung der Open-Source-Version eingestellt wurde. Das DataWiki ist nur noch in (sehr) veralteter

Version verfügbar. Zusätzlich wurde von offizieller Seite jegliche Dokumentation entfernt.

Vom Autor wurde zunächst versucht, die Komponenten des Wikis auf aktuelle Versionen zu aktualisieren. Dies scheiterte jedoch vor allem an zwei Faktoren: Erstens ist die DataWiki-Erweiterung auf ganz spezielle und teilweise extrem veraltete Versionen der Wiki-Komponenten angewiesen, um überhaupt lauffähig zu sein. Zweitens müssen zur Installation des DataWikis zahlreiche Quellcode-Dateien aus dem MediaWiki-Kern von Hand angepasst werden. Diese Anweisungen erfolgen in der Form *Suche X in Zeile Y und ersetze es mit Z*. Es wird schnell ersichtlich, dass diese Problematik zum Fehlschlagen etwaiger Upgrade-Versuche führen kann.

Vom Autor wurde versucht, die Beispielontologie, die Fürst mitliefert, auf eine andere Domäne zu portieren. Der anschließende Importversuch ins Wiki konnte nicht erfolgreich abgeschlossen werden. Die Gründe für das Scheitern liegen sicherlich zum Teil beim Autoren selbst. Auf der anderen Seite ist die Konfiguration des Wikis mit Hilfe einer Ontologie sehr aufwändig und für Laien kaum zu bewältigen.

Ein weitere Kritikpunkt ist die Abhängigkeit von externen Tools. Die Anwendung von Fürst lässt sich ohne den Java-Importer nicht konfigurieren. Zusätzlich muss die Ontologie mit Hilfe eines externen Modellierungstools erstellt und für den Import vorbereitet werden. Hier zeigt sich auch der große Nachteil dieses Ansatzes. Es ist nicht möglich, das Datenmodell zu ändern, ohne dass vorher die alten Daten gelöscht wurden. Jede Änderung gleicht somit einer Neuinstallation des gesamten Systems.

Zusammenfassend sind die folgenden Probleme bei Fürst zu erkennen:

- Schwierige Installation und Konfiguration
- Mangelnde Upgrade-Fähigkeit

- Abhängigkeit von externen Tools
- Für die tatsächlichen Anforderungen viel zu mächtige und komplexe Architektur (z.B. Ontologie als Konfiguration)

4.2 Entwurf und Implementierung

Dieser Abschnitt beschäftigt sich mit einer Architekturanalyse und den anschließenden Systemdesign-Entscheidungen. Im Anschluss wird die Architektur der konkreten Software genauer beschrieben.

4.2.1 Betrachtung alternativer Grundsysteme

Für die Implementierung des PCT wurden verschiedene Grundsysteme analysiert und in ihrer Nutzbarkeit bewertet. Im folgenden Abschnitt werden die drei relevantesten Ansätze kurz diskutiert.

Joomla

Joomla ist ein Open-Source-Content-Management-Framework, mit dessen Hilfe sowohl einfache Websites als auch mächtige Web-Anwendungen erstellt werden können. Die Vision des Joomla-Teams ist es, eine Plattform zu schaffen, die es Menschen auf der ganzen Welt erlaubt, mit Hilfe von freier, sicherer und qualitativ hochwertiger Software Inhalte zu veröffentlichen und sich in ihren jeweiligen Communities einzubringen. Den technischen Kern von Joomla bildet ein MVC-Application-Framework, welches sich unabhängig vom Content-Management-System verwenden lässt. Dies ist die größte Stärke von Joomla. Anders als bei alternativen Produkten, wird größter Wert auf Flexibilität und Erweiterbarkeit gelegt. Durch die strikte Trennung in Framework und CMS-Teil, lassen sich sehr einfach maßgeschneiderte Applikationen erstellen. Joomla ist in der Programmiersprach PHP geschrieben und verwendet Techniken

der objektorientierten Programmierung sowie verschiedene Software-Design-Patterns. Die Daten lassen sich in verschiedenen SQL-Datenbanken abspeichern (z.B. MSSQL, MySQL und PostgreSQL). Dies bedeutet wiederum, dass die Software auf einem normalen LAMP-Stack lauffähig ist. Bereits in der Standardinstallation bietet das Joomla-Paket zahlreiche Features wie etwa User-Management, Content-Management, Template-Management sowie Webservice-Management. Weitere Features lassen sich einfach in Form von der Community erstellten Erweiterungen nachinstallieren. Ein großer Vorteil des Joomla-Frameworks ist die sehr gute Code-Qualität, die ausgezeichnete Dokumentation sowie eine sehr engagierte und hilfsbereite Community. Diese Punkte machen das Entwickeln von Erweiterungen gerade auch für Einsteiger sehr gut möglich.

Für unsere Anwendungsfälle hat eine auf Joomla basierte Lösung allerdings auch einige Nachteile. Wir möchten für unser PCT ein System, das es auf einfach Art und Weise erlaubt an der Patternerstellung teilzuhaben. Dies bedeutet, dass mehrere Autoren über viele Iterationen hinweg an einem Patterndokument arbeiten können müssen. Der Standard-Joomla-Workflow sieht nicht vor, dass es mehrere Autoren gibt. Zusätzlich fehlt bisher eine Versionsverwaltung für Seiteninhalte (Änderungshistorie). Ein weiterer Nachteil ist das Fehlen von semantischen Erweiterungen. Es gibt zwar einige Ansätze, aber in der Summe sind die vorhandenen Erweiterungen bestenfalls als Prototypen zu bezeichnen. Da die semantischen Komponenten ein Kern-Feature des PCT sein sollen, müsste dieser Teil also von Grund auf neu entwickelt werden.

MediaWiki

Die Verwendung einer wikiartigen Lösung bietet sich im Hinblick auf unsere Anforderungen an das PCT an. Am Beispiel der MediaWiki-Software in Kombination mit den SMW-Erweiterungen zeigt sich, dass viele der Anforderungen schon in der Basisinstallation erfüllbar sind. Genannt seien hier das ausgereifte Templatesystem, die semantischen Erweiterungen sowie die Versionsverwal-

tung. Dies bedeutet schlussendlich, dass Entwicklungszeit für die Erstellung tatsächlich nützliche Funktionen verwendete werden kann. Für das MediaWiki bzw. das SMW existieren sehr viele Erweiterungen. Leider ist deren Qualität oftmals nicht zufriedenstellend und man muss nachbessern. Dieses manuelle Patchen hat einen sehr starken negativen Einfluss auf die Wartbarkeit des Gesamtsystems. Auch die Dokumentation ist eher mangelhaft. Oftmals hilft nur das Studium des spärlich dokumentierten Quellcodes um die (vermutlich) korrekte Verwendung einer API herauszufinden. Vorhandene Beispiele sind meist trivial und helfen bei komplexeren Anforderungen kaum. Bei der Entwicklung von Erweiterungen steht dem Programmierer kein MVC-Framework zur Verfügung. Dies bedeutet, dass tatsächliche View-Elemente zwischen regulärem PHP-Code zusammengebaut werden müssen. Diese Vermischung von View- und Controller-Logik hat nicht nur Auswirkungen auf die Lesbarkeit und Verständlichkeit des Quellcodes, sondern sorgt auch direkt dafür, dass selbst kleinste View-Änderungen zum fehleranfälligen Suchen-und-Ersetzen-Abenteuer werden. Im Kapitel 2 werden die einzelnen Komponenten und Erweiterungen des MediaWikis genauer erläutert.

Eigene Lösung

Es wurde diskutiert ob eine eigens entwickelte und somit hochgradig spezifische Lösung implementiert werden soll. Dieser zunächst verlockende Ansatz stellt sich aber bei genauerer Betrachtung (auch im Hinblick der zeitlichen Begrenzung einer Diplomarbeit) als wenig erfolgsversprechend heraus. Elementare Features wie die Benutzerverwaltung oder ein Template-System müssten von Grund auf entwickelt werden und würden somit die vorhandene Zeit für tatsächlich nützliche Features minimieren. Zusätzlich haben bereits etablierte Lösungen viele Vorteile gegenüber einer Neuentwicklung. So hatten die meisten Alternativen Lösungen oft schon mehrere Jahre Zeit um eine ausgereifte, wartbare und erweiterbare Code-Basis zu schaffen. Es ist utopisch, anzuneh-

men, dass im Rahmen einer Diplomarbeit eine gleichwertig ausgereifte Lösung erstellt werden kann. Das Fazit ist somit, dass eine Neuentwicklung nicht ratsam ist und lieber auf etablierte Lösungen gesetzt werden sollte.

4.2.2 Komponenten und Implementierung

Der PCT-Prototyp wurde auf Basis des MediaWikis implementiert. Wie sich bei der Analyse verschiedener Grundsysteme gezeigt hat, deckt das MediaWiki bereits einen großen Teil unserer Anforderungen ab. Für die meisten noch offenen Anforderungen ließen sich passende Erweiterungen finden, die einfach integriert werden konnten. Abbildung 4.2 zeigt die grobe Architektur unserer Lösung.

Zunächst wurde das Standard-MediaWiki um die semantischen Erweiterungen, die unter dem Begriff *Semantic Bundle* zusammengefasst sind, ergänzt. Das *Semantic Bundle* beinhaltet bereits die meisten Erweiterungen die für den Betrieb eines semantischen Wikis benötigt werden. Die einzelnen Komponenten können je nach Bedarf aktiviert bzw. deaktiviert werden. Für unseren Prototypen verwendeten wir die in Abbildung 4.2 unter *Semantic Bundle* gruppierten Erweiterungen. An dieser Stelle sei erwähnt, dass im Rahmen dieser Arbeit nicht auf eventuelle Voraussetzungen (Erweiterungen) der einzelnen Komponenten eingegangen wird. Nachdem das *Semantic Bundle* integriert war, wurden im nächsten Schritt Unzulänglichkeiten der SMW-Komponenten bezüglich der Konfiguration des Domänenmodells identifiziert. Es wurde festgestellt, dass es keine zentrale Möglichkeit zum Anlegen bzw. Editieren der nötigen semantischen Properties gab. Stattdessen mussten die Properties mühsam einzeln angelegt werden. Da dies nicht besonders benutzerfreundlich ist, wurde beschlossen, zu diesem Zweck eine eigene Komponente zu implementieren:

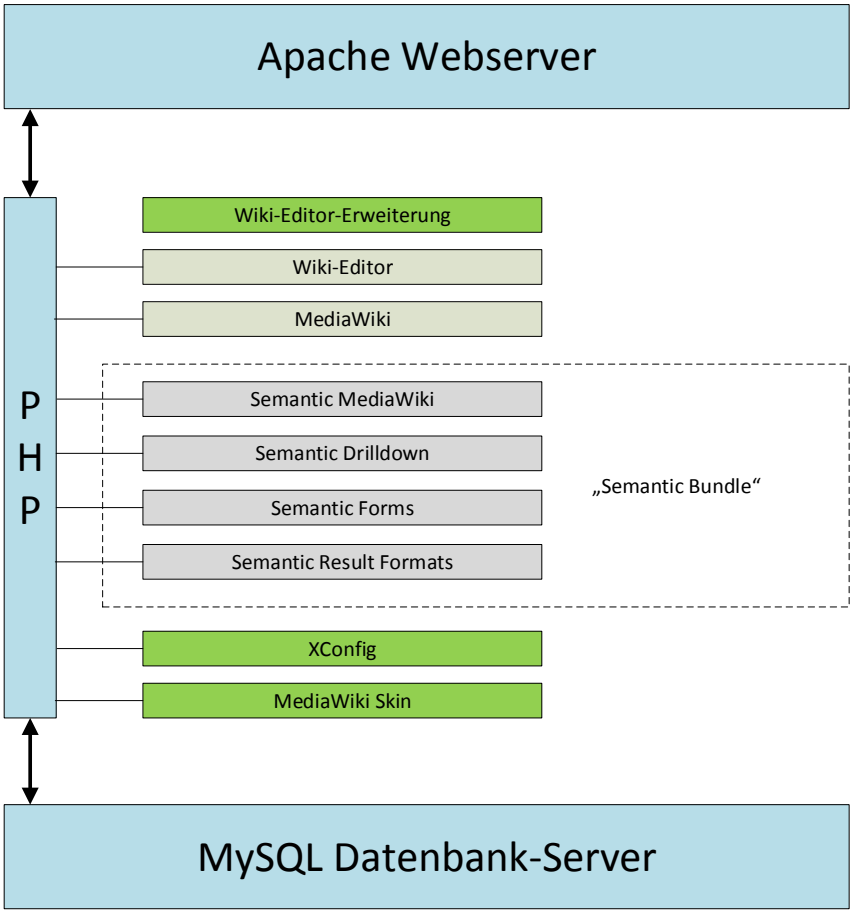

Abbildung 4.2: Architektur des PCT-Prototyps

XConfig-Erweiterung

Die XConfig-Erweiterung wurde als Wiki-Special-Page implementiert und hält sich an die Standard-Struktur dieser Erweiterungsart (siehe Abschnitt 2.5.14). XConfig bietet eine Benutzeroberfläche, die alle vorhandenen Properties des Wikis auflistet. Durch anklicken eines bestimmten Properties, kann dieses editiert werden. Zusätzlich existiert eine Eingabemaske, um neue Properties anzulegen. Der Benutzer kann mittels eigens implementiertem Javascript-Code mehrere Properties auf einmal anlegen.

Da die Standardansicht des Wikis nicht unseren Anforderungen bezüglich Layout und Ästhetik genügte, wurde ein eigener Wiki-Skin (siehe Abschnitt 2.5.14) implementiert:

MediaWiki-Skin

Die Basis für unseren eigenen Skin lieferte der *Bootstrap-MediaWiki*-Skin [1]. Dieser wurde ausgewählt, da er auf dem exzellenten *Bootstrap Framework* [2] basiert und somit eine solide Grundlage für unser Vorhaben darstellte. Die Anpassungen umfassten umfangreiche Änderungen an der HTML-Generierung sowie zahlreiche CSS-Anpassungen. Das Ziel war es, eine klar strukturierte und ansprechende Benutzeroberfläche zu erstellen.

Beim Testbetrieb des Prototypen wurde festgestellt, dass das Hinzufügen von semantischen Annotation mit Hilfe von Standard-Wiki-Markup nicht nur sehr fehleranfällig ist, sondern es auch an Benutzerfreundlichkeit mangelt. Daher wurde eine Erweiterung für den Wiki-Editor implementiert:

Wiki-Editor-Erweiterung

Um das Hinzufügen von semantischen Annotationen zu vereinfachen und dem Benutzer nicht das Schreiben von Wiki-Markup zuzumuten, wurde eine Erweiterung für den Wiki-Editor geschrieben. Mit der Hilfe dieser Erweiterung kann der Benutzer ein beliebiges Textfragment in eine semantische Annotation umwandeln. Dabei wird er durch einen modalen Dialog bei der Auswahl des gewünschten Property-Typs unterstützt. Der Wiki-Editor bietet bereits eine Schnittstelle, um neue Funktionalität hinzuzufügen. Die Implementierung erfolgte daher in Form einer Javascript-Callback-Funktion.

[1] Matthew Batchelder - http://borkweb.com
[2] http://getbootstrap.com/

Im nächsten Schritt wurde die Anforderung des initialen Daten-Imports angegangen. Dazu wurde ein Programm implementiert, dass die Ur-Daten in das Wiki-Format konvertiert:

Initialer Datenimport

Das PCT soll am Beispiel von Cloud-Computing-Patterns (Fehling et al. [3]) mit Daten vorbefüllt werden. Da diese Aufgabe bei manueller Ausführung sehr zeitaufwändig, arbeitsintensiv und fehleranfällig ist, soll der Initialimport automatisiert geschehen. MediaWiki bietet bereits eine Funktion um Daten im XML-Format zu exportieren bzw. zu importieren. Dabei muss das XML-Markup in einer für MediaWiki verarbeitbaren Form vorliegen. Im Folgenden soll dieses Format grob beschrieben werden:

Wie in Abbildung 4.3 zu sehen, teilt sich die Datei in den Header-Bereich sowie den Content-Bereich. Im Header stehen allgemeine Informationen der Wiki-Installation wie etwa die URL des Wikis oder Angaben zu den vorhandenen Wiki-Namespaces. Die Tatsache, dass im MediaWiki sämtliche Inhalte in Seiten organisiert sind, sorgt dafür, dass das Export-Format relativ simpel gehalten werden kann. Im Content-Bereich befindet sich daher eine beliebige Anzahl von Page-XML-Knoten, welche jeweils eine Seite des Wikis repräsentieren. Wie in Abbildung 4.3 zu sehen, besteht eine Page aus mindestens dem Titel sowie einer einzelnen Revision des Seiteninhalts. Optional können hier auch mehrere Revisionen inklusiv der Autoreninformationen enthalten sein.

Die Cloud-Computing-Patterns selbst liegen in XML-Form vor. Es ist daher relativ einfach, diese in das Importformat des Wikis zu überführen. Nach Prüfung wurde hier beschlossen, statt einer XSL-Transformation auf ein Transformationsprogramm zu setzen. Es gibt für diese Entscheidung mehrere Gründe. Der Hauptgrund ist, dass die nötigen Umformungen bei genauer Betrachtung nicht trivial sind und sehr viel Programmlogik erfordern. Dazu kommt, dass

```
<?xml version="1.0" encoding="utf-8"?>                    HEADER
<mediawiki version="1.0" lang="DE-de">
 <siteinfo>
  <sitename>Seitenname</sitename>
  <base>http://www.example.com</base>
  <generator>Convert-XML 0.1</generator>
  <case>first-letter</case>
  <namespaces>
   <namespace key="1">Namespace1</namespace>
  </namespaces>
 </siteinfo>
 <page>                                                   CONTENT
  <title>Test-Seite</title>
  <id>1</id>
  <revision>
   <id>1</id>
   <timestamp>2012-12-13T12:12:12</timestamp>
   <contributor>
    <username>User 1</username>
    <id>1</id>
   </contributor>
   <text space="default">Inhalt Test-Seite</text>
  </revision>
 </page>
</mediawiki>
```

Abbildung 4.3: Wiki-Importformat

der Autor deutlich vertrauter mit der Erstellung von programmatischen Transformationen ist, als mit der Verwendung von XSLT.

Das Konvertierungsprogramm (im Folgenden *Konverter* genannt) wurde als Kommandozeilen-Tool umgesetzt und ist durch die Verwendung der Programmiersprache Ruby plattformunabhängig. Der Konverter besteht aus zwei Teilen. Der erste Teil ist ein Parser, der das Ursprungsformat einliest und daraus Ruby-Objekte erstellt. Als XML-Bibliothek wurde *Nokogiri* verwendet, mit deren Hilfe der nötige Programmieraufwand sehr klein gehalten werden konnte. *Nokogiri* abstrahiert die XML-Knoten so weit, dass mit einer CSS-ähnlichen Selektorenbeschreibung auf Inhalte zugegriffen werden kann. Abbildung 4.4 dient der Verdeutlichung.

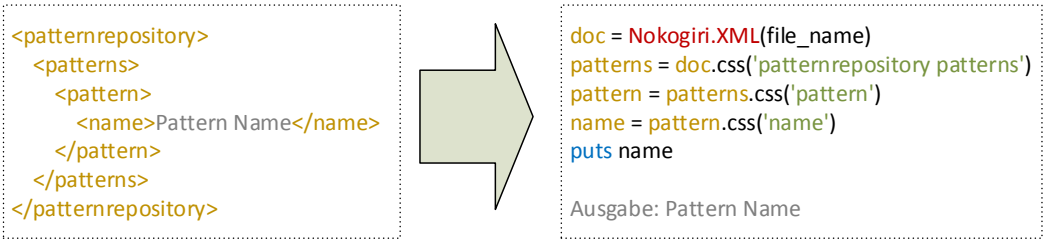

Abbildung 4.4: Nokogiri - XML nach Ruby

Der zweite Teil des Programms kümmert sich um die tatsächliche Ausgabe in das Zielformat. Zur sauberen Trennung von Programmlogik und XML-Ausgabe wurde eine Template-Engine (*Slim* [3]) verwendet. Es existieren daher für alle Ruby-Objekte entsprechende Templates, die die Serialisierung des Objekts in das MediaWiki-Format steuern.

Für diese Arbeit wurde der Konverter speziell für den Import von Cloud-Computing-Patterns geschrieben. Die Architektur ist jedoch so flexibel gehalten, dass sich durch einfaches Austauschen der Ruby-Objekte bzw. der Templates beliebige Ursprungsformate (in XML-Form) konvertieren lassen. Der Autor geht davon aus, dass sich die Daten ähnlicher Domänen mit minimalen Anpassungen importieren lassen.

[3] http://slim-lang.com/

5 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde ein Pattern-Collaboration-Tool (PCT) entwickelt. Dabei wurde zunächst ein Vorgängerprototyp analysiert und auf seine Eignung als wartbare und zunkunftsfähige Basis bewertet. Dabei wurden zahlreiche Problemfelder identifiziert. Durch die komplexe Architektur stellte sich sowohl die Installation als auch die Konfiguration des Systems als große Hürde heraus. Zusätzlich scheiterte der Versuch, die verwendeten Komponenten auf aktuelle Versionen zu aktualisieren. Dies lag vor allem an der zentralen proprietären DataWiki-Erweiterung, deren Wartung eingestellt worden war. Nach diesen Rückschlägen wurde die Situation analysiert und es wurde beschlossen, das Vorhaben, den alten Prototypen zu modernisieren, aufzugeben. Bei der ursprünglichen Betrachtung kamen bereits starke Zweifel auf, ob die verwendeten Technologien (z.B. OWL, Triplestore, etc.) überhaupt nötig sind, um die Anforderungen an ein PCT zu erfüllen. Diese Vermutung wurde schließlich mit einer Anforderungsanalyse formalisiert. Nachdem die zentralen Rollen und Anforderungen festgehalten worden waren, zeigte sich, dass diese sich mit einer deutlich einfacheren Architektur erfüllen lassen würden. Das Ziel der Neuentwicklung wurde festgelegt: Es sollten so viele wie möglich bereits existierende und ausgereifte Komponenten verwendet werden. Die Menge an Neuentwicklungen sollte also minimal gehalten werden. Nachdem das neue Ziel klar definiert war, wurde damit begonnen, verschiedene Grundsysteme auf ihre Tauglichkeit hin zu untersuchen. Durch das Vorgängerprojekt war klar, dass sich MediaWiki höchstwahrscheinlich als Basis eignen würde.

Nichtsdestotrotz wurden noch weitere Ansätze verfolgt. Zunächst wurde das Joomla-Framework in Betracht gezogen. Diesem mangelte es aber an ausgereiften, semantischen Komponenten. Es hätte also sehr viel Zeit in die Grundlagenentwicklung investiert werden müssen. Aus dem selben Grund wurde auch der Ansatz einer kompletten Neuentwicklung verworfen. Nach der weiteren Bertrachtung des MediaWikis, zeigte sich, dass ein Großteil unserer Anforderungen entweder schon nativ oder mit Hilfe ausgereifter Erweiterungen erfüllt werden konnte.

Nachdem im nächsten Schritt die einzelnen Erweiterungen integriert worden waren, wurden die noch existierenden Unzulänglichkeiten angegangen. Das Resultat war eine Erweiterung zur einfachen Konfiguration der semantischen Eigenschaften, eine Usability-Erweiterung für das Frontend sowie ein eigens erstellter Skin. Bereits zu Beginn der Arbeit war klar, dass das PCT am Beispiel von Cloud-Computing-Patterns demonstriert werden sollte. Da die Patterns in XML-Form vorlagen, musste ein Konverter implementiert werden. Dieser sorgte dafür, dass das Ursprungsformat in eine vom Wiki verarbeitbare Form transformiert wurde.

Zusammenfassend kann festgestellt werden, dass ein Großteil der gestellten Anforderungen erfüllt werden konnte. Es wurde eine einfach zu installierende und leicht zu konfigurierende wartbare Basis geschaffen. Dabei wurde größter Wert auf den Erhalt der Zukunftsfähigkeit gelegt.

5.1 Einschränkung und Ausblick

In diesem Abschnitt wird kurz auf die noch offenen Anforderungen eingegangen. Zusätzlich werden mögliche Ansatzpunkte einer zukünftigen Arbeit betrachtet.

Zunächst muss ganz klar gesagt werden, dass das ursprüngliche Ziel der Arbeit, nämlich den existierenden Prototypen zu erweitern, nicht erreicht wurde.

Des Weiteren wurden Anforderungen nach persönlichen Leselisten und Favoriten nicht erfüllt. Auch fehlt die Möglichkeit, Patterns zu bewerten. Zu diesem Punkt gehört auch das fehlende System zur Beurteilung der Pattern Maturity. Kommen wir also zu den Aspekten, die verbessert bzw. implementiert werden könnten:

Usability

Es kann noch sehr viel Aufwand in die Verbesserung der Usability gesteckt werden. Dazu gehört z.B. die Verbesserung des geführten Durchsuchens des Wikis, aber auch die Umstellung des Wiki-Editors auf eine WYSIWYG-Lösung. Zusätzlich könnte eine Erweiterung zur zentralen Verwaltung der verwendeten Komponenten erstellt werden.

Community-Features

Im Moment mangelt es noch an den sogenannten Community-Features, wie etwa einer Patternbewertungs- und Kommentierfunktion. Es existieren hier bereits Wiki-Erweiterungen, die entweder integriert werden oder als Basis einer Neuimplementierung dienen könnten.

Rechtesystem

Aktuell wird die Standard-Wiki-Rechtefunktion verwendet. Für die Zukunft sollte hier in ein mächtigeres System mit fein definierbaren Rollen und Rechten investiert werden. Auch hier existieren bereits Lösungen.

Wartbarkeit

Um spätere Aktualisierungen sicherer zu gestalten, könnte in eine automatisierte Upgrade-Lösung investiert werden. Dies würde vermutlich auch die Erstellung einer Testplattform nach sich ziehen.

Algorithmen

Ein weiterer Ansatzpunkt wäre die Erstellung von Auswertungsalgorithmen, die z.B. sinnvolle alternative Patterns vorschlagen könnten. Hiermit ließe sich auch die Suchfunktion aufwerten.

Abbildungsverzeichnis

Abkürzungen

API	Application programming interface
CSS	Cascading Style Sheets
DOT	DOT (graph description language)
HCI	Human Computer Interaction
HTML	HyperText Markup Language
KISS	Keep It Stupid Simple
LAMP	Linux Apache MySQL PHP
MVC	Model–view–controller
MW	MediaWiki
PCT	Pattern-Collaboration-Tool
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
RDF	Resource Description Framework
SMD	Semantic-Drilldown
SMF	Semantic-Forms

SMW Semantic-MediaWiki

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

SRF Semantic-Result-Formats

SVG Scalable Vector Graphics

URL Uniform resource locator

W3C World Wide Web Consortium

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

XSL Extensible Stylesheet Language

Literatur

[1] Christopher Alexander. “A Pattern Language”. In: (1977).

[2] *Duden | Domäne*. URL: http://www.duden.de/rechtschreibung/Domaene#Bedeutung2.

[3] Christoph Fehling et al. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014. ISBN: 978-3709115671.

[4] Norbert Fürst. “Semantisches Wiki zur Erfassung von Design-Patterns”. Diplomarbeit. Universität Stuttgart, 2013.

[5] Erich Gamma et al. *Entwurfsmuster*. 5. Auflage. Addison-Wesley, 1996. ISBN: 3-8273-1862-9.

[6] *Graphviz*. URL: http://www.graphviz.org/Documentation.php.

[7] Robert Hanmer. “Pattern Mining Patterns”. In: (2012), S. 1–16.

[8] Robert Hanmer. *Pattern-oriented software architecture for dummies*. John Wiley & Sons, Ltd., 2012. ISBN: 9781119963998.

[9] Neil B Harrison. “The language of shepherding - A Pattern Language for Shepherds and Sheep”. In: *Proc. PLoP* (1999).

[10] *Help:Categories*. URL: http://www.mediawiki.org/wiki/Help:Categories.

[11] *Hook*. URL: http://www.mediawiki.org/wiki/Manual:Hooks.

[12] *Infobox*. URL: http://de.wikipedia.org/wiki/Hilfe:Infoboxen.

[13] *Kiss Principle*. URL: http://people.apache.org/~fhanik/kiss.html.

[14] Helmut Leitner. *Mustertheorie*. 1. Auflage. Verlag Nausner & Nausner, 2007. ISBN: 978-3-901402-50-0.

[15] Gerard Meszaros und Jim Doble. "MetaPatterns: A Pattern Language for Pattern Writing". In: (), S. 1–39.

[16] *Namespaces*. URL: http://meta.wikimedia.org/wiki/Namespaces.

[17] *Parser Function*. URL: http://www.mediawiki.org/wiki/Manual:Parser_functions.

[18] Luka Pavlic, Marjan Hericko und Vili Podgorelec. "Improving design pattern adoption with Ontology-Based Design Pattern Repository". In: *ITI 2008 - 30th Int. Conf. Inf. Technol. Interfaces*. IEEE, Juni 2008, S. 649–654. ISBN: 978-953-7138-12-7. DOI: 10.1109/ITI.2008.4588487.

[19] *Properties + Property-Types*. URL: http://semantic-mediawiki.org/wiki/Help:User-defined_properties.

[20] René Reiners et al. "Requirements for a Collaborative Formulation Process of Evolutionary Patterns". In: (2013), S. 1–10.

[21] *Semantic Drilldown*. URL: http://semantic-mediawiki.org/wiki/Semantic_Drilldown.

[22] *Semantic Forms*. URL: http://semantic-mediawiki.org/wiki/Semantic_Forms.

[23] *Semantic MediaWiki*. URL: http://semantic-mediawiki.org/wiki/Semantic_MediaWiki.

[24] *Semantic Result Formats*. URL: http://semantic-mediawiki.org/wiki/Semantic_Result_Formats.

[25] *Skin*. URL: http://www.mediawiki.org/wiki/Manual:Skins.

[26] *SPARQL Specification*. URL: http://www.w3.org/TR/rdf-sparql-query/.

[27] *Special Page*. URL: http://www.mediawiki.org/wiki/Manual:Special_pages.

[28] *Special-Pages*. URL: http://meta.wikimedia.org/wiki/Special_pages.

[29] *Tag Extension*. URL: http://www.mediawiki.org/wiki/Manual:Tag_extensions.

[30] *Templates*. URL: http://meta.wikimedia.org/wiki/Help:Template.

[31] Noam Tractinsky. "Aesthetics and apparent usability: empirically assessing cultural and methodological issues". In: *CHI'97 Proc. ACM SIGCHI Conf. Hum. factors Comput. Syst.* 1997, S. 115–122. ISBN: 0897918029. DOI: 10.1145/258549.258626.

[32] *Triple Store*. URL: http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html.

[33] *W3C SEMANTIC WEB ACTIVITY*. URL: http://www.w3.org/2001/sw/.

[34] Tim Wellhausen und Andreas Fießer. "How to write a pattern ? A guideline for first-time pattern authors". In: (2011).

[35] *Wiki-Editor*. URL: http://www.mediawiki.org/wiki/WikiEditor.

[36] Ralf Wirdemann. *Scrum mit User Stories*. 2., erweit. Carl Hanser Verlag Müchen Wien, 2011. ISBN: 978-3-446-42660-3.

[37] *XML 1.0 Specification*. URL: http://www.w3.org/TR/REC-xml.

Alle Links wurden zuletzt am 28. Januar 2014 überprüft.

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*