Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Nutzung von Standards und Technologien des Semantischen Webs zur Repräsentation von Mustersprachen

Manuela Weigold

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Michael Falkenthal
Beginn am:	15. April 2019
Beendet am:	15. Oktober 2019

Kurzfassung

Ein Entwurfsmuster (kurz Muster genannt) ist ein Textdokument, das eine abstrakte Lösung für ein Problem aufzeigt. In verschiedenen Fachgebieten ist die Veröffentlichung eines Entwurfsmusters eine etablierte Methode, um Expertenwissen für die Lösung von wiederkehrenden Probleme weiterzugeben. Mehrere Muster können zu Mustersprachen zusammengefasst werden, sodass zusammengehörige Entwurfsmuster Verbindungen (Links) zueinander aufweisen. Autoren von Mustersprachen veröffentlichen diese in unterschiedlichen Formaten, wie Büchern, Konferenzbeiträgen und Webseiten. Fachwissen ist hierdurch auf viele verschiedene Dokumente verteilt. Klassische Ansätze um dieses Wissen zu bündeln sehen einen zentralen Speicherort vor, beispielsweise in einem Repository von Mustern oder einem Wiki. Keins dieser Tools hat sich jedoch etabliert, sodass es kein zentrales Register über die stetig wachsende Anzahl von Dokumenten zu Entwurfsmustern gibt. Würden die Informationen zu Entwurfsmustern in einem maschinenlesbaren Format zur Verfügung stehen, könnten Programme auf die dezentral verteilten Informationen über Mustersprachen einfacher zugreifen und diese gezielter durchsuchen und verwenden. Im Zuge der vorliegenden Arbeit soll diese Idee mithilfe von Standards des *semantischen Webs* umgesetzt werden. Die maschinenlesbaren Daten können dann als wertvolle Wissensbasis über Mustersprachen genutzt werden, z.B. für ein Musterrepository. Mittels der Standarddatenformate und dem Prototypen wird es ermöglicht, die Wissensbasis zu bearbeiten, erweitern, referenzieren oder als Grundlage für weitere Anwendungen zu verwenden.

Inhaltsverzeichnis

Akronyme

HTTP Hypertext Transfer Protocol. 47

IRI Internationalized Resource Identifier. 24

OWL Webontology Language. 5, 26

PURL Persistent URL. 47

PURLs Persistent URLs. 48

RDF Resource Description Framework. 24, 25

RDFS RDF Schema. 5, 26

RIF Rule Interchange Format. 27

SPARQL SPARQL Protocol and RDF Query Language. 27, 29

SQL Structured Language. 27

Turtle Terse RDF Triple Language. 24

URI Uniform Resource Identifier. 23

URL Uniform Resource Locator. 24

W3C World Wide Web Consortium. 23

WWW World Wide Web. 22

XML Extensible Markup Language. 24

1 Einleitung

In diesem Kapitel wird ein Überblick über die Inhalte und Ziele dieser Masterthesis gegeben. Zunächst wird der Kontext der Arbeit dargestellt und hieraus die Fragestellung der Arbeit abgeleitet. Zuletzt wird aufgezeigt, was für Beiträge die Arbeit leistet und der inhaltliche Aufbau der Arbeit vorgestellt.

1.1 Motivation und Problemstellung

Im Jahr 2001 veröffentlichen die Autoren Berners-Lee et al. [BHL+01] ihre Vision einer neuen Ära des Webs, des semantischen Webs. Hierin beschrieben sie in einem Szenario, was diese Technologie in Zukunft ermöglichen könnte: Ein Softwareagent plant Lucys und Petes Fahrdienste für die Arzttermine ihrer kranken Mutter. Hierfür werden die Termine von Lucy und Pete mit freien Terminen in gut bewerteten Praxen in der Nähe abgeglichen. Für diese Aufgabe müssen verschiedenste Datenquellen integriert werden: Bewertungen der Praxen unter Berücksichtigung der Entfernung, freie Zeitslots der Praxen und die Kompatibilität mit dem Terminkalender eines Fahrers (Lucy oder Pete). Das Szenario der Autoren spiegelt das Potential der Verarbeitung von Daten im semantischen Web wider: Daten aus verschiedenen Webseiten können viel leichter integriert und von Softwareagenten interpretiert werden. Besonders in den Jahren um die 2000er Wende wurden die Grundlagen des semantischen Webs entwickelt und technische Standards dazu veröffentlicht [Hen08]. Immer mehr Wissen wurde hierdurch in strukturierter Form Maschinen zugänglich gemacht, beispielsweise veröffentlichten Bikakis et al. [BTG+13] mit ihrem Projekt *DBpedia* strukturierte Informationen zu Wikipedia-Inhalten im semantischen Web. Diese Datenquelle wurde dann von IBM Watson™ genutzt [BLK+09], um die führenden menschlichen Weltmeister in Jeopardy!® zu übertreffen [GBPM13]. Die Autoren führen den erfolgreichen Sieg unter anderem auf die große Wissensbasis zurück. Damit zeigt das Watson™ Projekt, dass Agenten wie sie von Berners-Lee et al. [BHL+01] beschrieben wurden, durchaus in Reichweite liegen.

Eine weitere Wissensbasis, die für Softwareagenten wie Watson™ interessant ist, sind Entwurfsmuster. In vielen Fachgebieten ist die Veröffentlichung eines Entwurfsmusters eine etablierte Methode um Expertenwissen weiterzugeben [FBB+16]. Sie beschreiben für ein wiederkehrendes Problem dessen Kontext und eine abstrakte Lösung [AIS+77]. Oft ist das zugrundeliegende wiederkehrende Problem in einer Fragestellung zusammengefasst: “How does one application communicate with another using messaging?” beschreibt beispielsweise den Zweck des *Message Channel* Musters von Hohpe und Woolf [HW04]. Die restlichen Abschnitte des Musters beschreiben die Lösung der Ausgangsfrage, daher wie auf Messaging basierte Kommunikation funktioniert. Schon allein diese Art von Frage-Antwort Daten ist für Softwareagenten wie Watson™ von Interesse. Darüber hinaus existieren zu Muster aber noch weitere interessante Kontextinformationen: Verwandte Muster können zu Mustersprachen gruppiert werden, die aus Mustern und Verbindungen (Links)

zwischen ihnen bestehen [AIS+77]. Ein Link zwischen zwei Mustern kann beispielsweise ausdrücken, ob zwei Muster miteinander verwendet werden können [FBL18]. In den meisten Fällen werden Mustersprachen getrennt voneinander in Büchern, Fachartikeln oder Autorenwebseiten veröffentlicht, und beinhalten nur Links innerhalb einer Mustersprache [FBL18]. Um Referenzen zu anderen Mustersprachen zu folgen, muss der Leser das Dokument wechseln und meist auch noch die entsprechende Seite heraussuchen (in Abb. 1.1 links dargestellt). Die Verwendung von Webtechnologien für die Repräsentation von Mustersprachen und die Veröffentlichung im semantischen Web ermöglicht es hingegen, Muster direkt zu referenzieren und somit auch Verlinkungen zwischen Mustersprachen erleichtern. In Abb. 1.1 rechts ist gezeigt, wie Muster über Dokumente hinweg eindeutig referenziert werden können. Das so beschriebene Wissen über Entwurfsmuster liegt als maschinenlesbare Aussagen vor, weshalb die Informationen leicht integriert werden können. Anwendungen wie ein Musterrepository können auf dieser Wissensbasis aufbauen, die Informationen zu Mustern vorverarbeiten, Schlussfolgerungen (Interferenzen) ziehen und für menschliche Leser aufbereiten. Anknüpfend an die am Anfang beschriebene Vision von Berners-Lee et al. [BHL+01] soll nun ebenfalls ein Szenario beschrieben werden, in dem die Möglichkeiten des semantischen Web für Mustersprachen aufgezeigt werden sollen. Hierin verwendet ein Musterrepository (im Szenario *PatternPedia* genannt), semantische Daten zu Mustersprachen aus dem Bereich der Informationstechnologie um den Benutzer in seiner Arbeit als Softwareentwickler besser zu unterstützen. Ein tieferes Verständnis der einzelnen Muster ist nicht notwendig, um die Funktionalitäten der Anwendung zu verstehen:

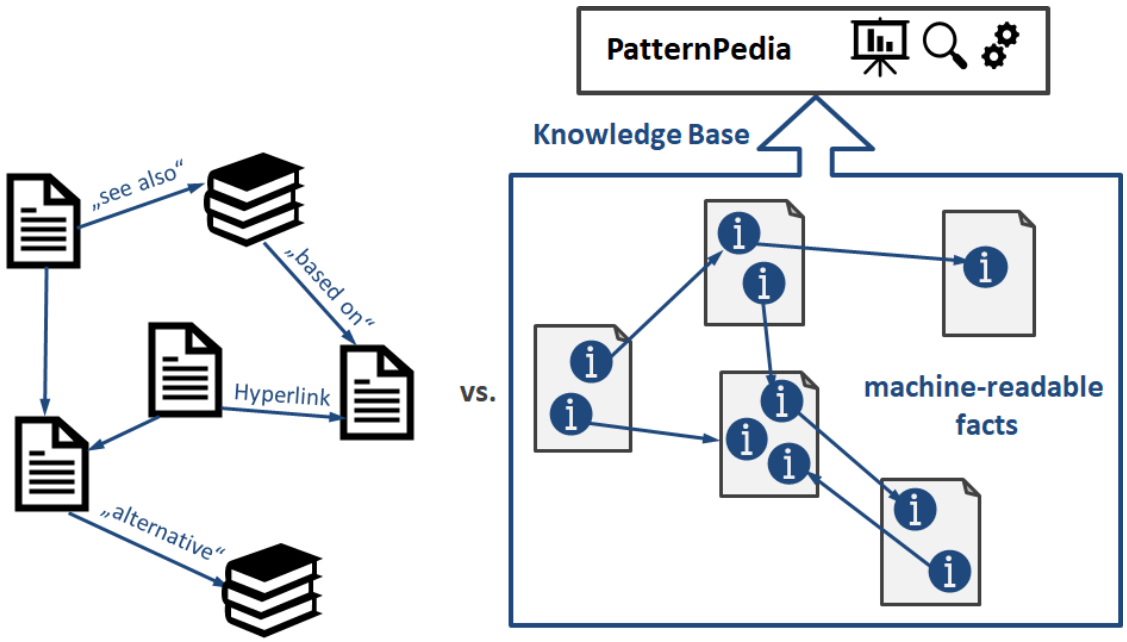


Abbildung 1.1: Vergleich von herkömmliches Wissen über Muster zu semantischen Daten. Links ist dargestellt, dass Wissen zu Mustersprachen in vielen verschiedene Quellen beschrieben wird. Verweise auf andere Muster erfordern oft einen Wechsel des Dokuments. Rechts ist dargestellt, wie Wissen zu Mustersprachen ebenfalls verteilt vorliegt, jedoch als maschinenlesbare Aussagen. Hierbei ist es auch möglich, direkt auf andere Muster zu verweisen. Die Informationen rechts können leicht integriert und als Wissensbasis verwendet werden.

Lucy erhält im Zuge ihrer Arbeit als Softwareentwicklerin die Aufgabe, die bestehende Software eines Webshops abzulösen. Die derzeitige Anwendung ist trotz hoher Serverkapazität bei vielen Benutzern schnell ausgelastet. Die neue Lösung soll in der Cloud bereitgestellt werden und hoch skalierbar sein. Um einen Überblick über den Aufbau von Anwendungen in der Cloud zu bekommen, beginnt Lucy sich in *PatternPedia* über die für sie relevanten Cloud Computing Entwurfsmuster zu informieren. Sie beginnt mit dem *Distributed Application* Muster (siehe Abb. 1.2 links), da die Anwendung je nach Arbeitslast skaliert werden soll. Zu diesem Entwurfsmuster schlägt ihr *PatternPedia* weitere Muster vor, wie z.B. das *User Interface Component* Muster, welches eine Schicht der skizzierten Lösungsarchitektur ist. Dies wiederum wird laut *PatternPedia* häufig in Verwendung mit einem *Elastic Load Balancer* verwendet, weshalb dieses Muster ihr ebenfalls als Vorschlag angezeigt wird. Lucy fügt die drei genannten Entwurfsmuster ihrer persönlichen Musterliste hinzu. In der Anzeige ihrer ausgewählten Muster wird ihr angezeigt, dass das *User Interface Component* Muster zur Verwendung des *Stateless Component* Muster führt (siehe Abb. 1.2 rechts). Als Begründungstext ist angegeben: Die Eigenschaft zustandslos zu sein, erleichtert das Skalieren der Komponente. Lucy wählt noch ein paar weitere vorgeschlagene Entwurfsmuster aus, wie das *Message-oriented Middleware* Muster und das *Key-value Storage* Muster für die Speicherung der Daten. Sie speichert dann ihre Musterliste und betätigt den Button “Anwendungsgerüst erstellen”. Ein Dialog fragt Lucy, welche Technologien sie verwenden möchte und ob sie einen bestimmten Cloudanbieter bevorzugt. Nach diesen Angaben stellt die Anwendung ihr Code bereit, der das Grundgerüst für die verschiedenen Komponenten enthält und nur noch um die Businesslogik ergänzt werden muss.

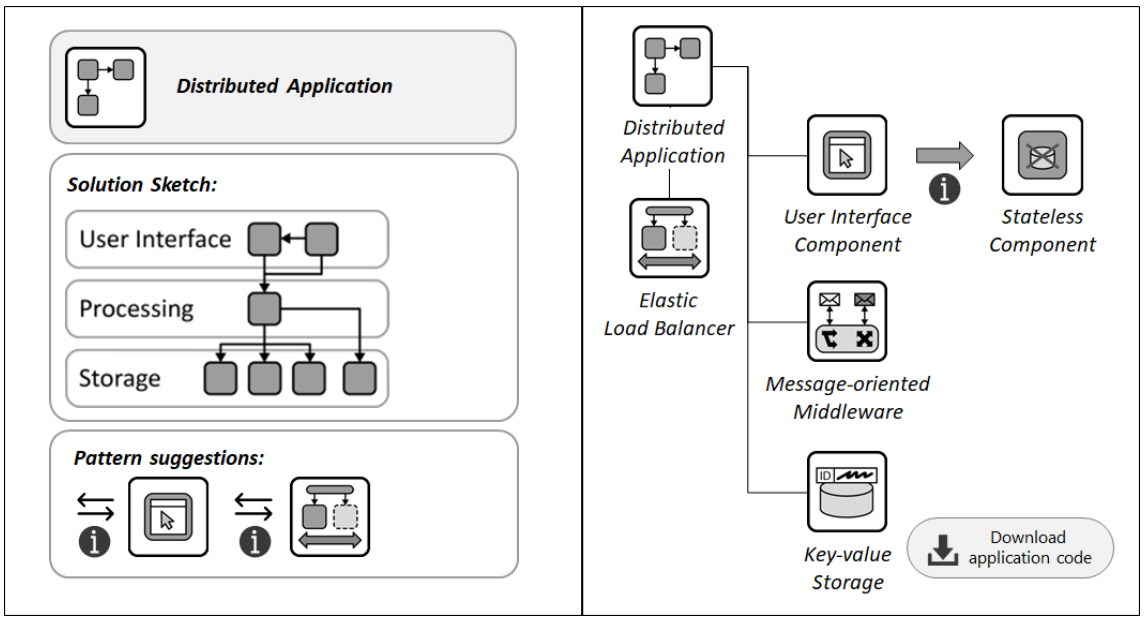


Abbildung 1.2: Ausschnitt der Musterrepository-Ansicht des *Cloud Computing* Musters *Distributed Application* von Fehling et al. [FLR+14] (links) sowie Ansicht der vom Benutzer gewählten Muster (rechts), inklusive weiterer Muster die von der Anwendung als erforderlich erkannt wurden (*Stateless Component*).

Auf den ersten Blick wirkt das *PatternPedia*-Szenario simpler als das von Berners-Lee et al. [BHL+01] beschriebene. Im Folgenden werden die Anforderungen aus unserem Szenario für ein Repository von Mustern zusammengefasst und dadurch herausgestellt, warum es sich ebenfalls um eine komplexe Aufgabe handelt:

1. **Mustersprachen übergreifendes Format**: Eine zentrale, dokumentübergreifende Übersicht über alle veröffentlichten Mustersprachen und Entwurfsmuster existiert bislang nicht. Eine weitere Schwierigkeit hierbei ist, dass Mustersprachen sehr unterschiedlich aufgebaut sein können. Köppe et al. [KISV16] analysierten 19 Repositorien und stellten fest, dass über 50% nur ein Format für Muster unterstützen und diesen Aspekt daher nicht berücksichtigen. Im zweiten Kapitel 2 (Abschnitt 2.1) wird nochmals herausgestellt, dass nur ein Format für Mustersprachen diese nicht gut widerspiegeln kann. Für *PatternPedia* soll daher der Empfehlung von Köppe et al. [KISV16] gefolgt werden, dass ein Musterrepository den gesammelten Mustersprachen kein bestimmtes Format aufzuzwingen soll.
2. **Erweiterbarkeit**: Um Lucy möglichst viel Information und weiterführende Muster anzeigen zu können, soll *PatternPedia* um weitere Inhalte (Muster, Mustersprachen und Verbindungen zwischen Mustern) ergänzt werden können. Der Name *PatternPedia* soll bewusst an Wikipedia erinnern, welches die größte gemeinschaftliche Sammlung von enzyklopädischem Wissen enthält [KV09]. Gemäß dem nutzerbasierten Ansatz von Wikipedia darf und kann jeder sich am Aufbau der Wissensbasis beteiligen. Insbesondere neues Wissen kann so einfach integriert werden. Wie im Kapitel zu Mustersprachen nochmals herausgearbeitet wird, sind Erweiterbarkeit und Flexibilität essentielle Eigenschaften von Mustersprachen. Eine Veröffentlichung einer Mustersprache in *PatternPedia*, spiegelt so die charakteristischen Eigenschaften und Dynamik von Mustersprachen wider: Neue Muster oder Verlinkungen entstehen und verändern die von Alexander et al. [AIS+77] als *lebendiges Netzwerk* bezeichnete Struktur.
3. **Verlinkungen**: In *PatternPedia* werden dem Benutzer passende, verlinkte Entwurfsmuster vorgeschlagen. Köppe et al. [KISV16] bewerteten die Funktionalität von 19 Repositories unter anderem auf die Möglichkeiten, Muster zu verlinken. Nur zwei der 19 Repositorien boten dem Nutzer an, zu den Links zwischen Mustern einen Typ anzugeben und diesen nicht nur aus ein paar wenigen Typen auszuwählen. Stattdessen bietet die Mehrzahl der Repositorien keine Zusatzinformationen zu Links an, obwohl diese so wertvolle Informationen wie z.B. die Kompatibilität von Mustern enthalten. Typisierte (semantische) Links stellen außerdem eine wichtige Grundlage für die Integration von Lösungen für ausgewählte Muster dar, wie Falkenthal et al. [FBB+14] für Mustersprachen aus verschiedensten Domänen zeigen. Um eine Wissensbasis über Mustersprachen aufzubauen, welche die Basis für Codegenerierung sein kann, ist also fundiertes Wissen über die Mustersprachen, Muster und Beziehungen zwischen ihnen notwendig. Da die meisten Probleme Lösungen aus unterschiedlichen Mustersprachen erfordern (in Lucys Businesslogik werden beispielsweise Muster aus der objektorientierten Programmierung benötigt) sollen auch mustersprachenübergreifende Links hinzugefügt und verarbeitet werden können.

Um passende Vorschläge und Codegenerierung für Muster anbieten zu können, benötigt *PatternPedia* im ersten Schritt eine breite Wissensbasis über Mustersprachen, wobei Beziehungen der Muster eine entscheidende Rolle spielen. Die Anzeige von Beziehungen zwischen Mustern unterstützt den Leser außerdem beim Navigieren durch Mustersprachen: Er wird auf weitere, in diesem Kontext interessante Muster hingewiesen und kann zur Ansicht von diesen wechseln. Mit der vorliegenden

Arbeit soll untersucht werden, wie durch den Einsatz von grundlegenden semantischen Webtechnologien eine solche Wissensbasis über Mustersprachen aufgebaut und mit dieser interagiert werden kann. Eine Dokumentation von konkreten Lösungen (wie Code im Szenario) ist nicht Teil dieser Arbeit, daher auch nicht die angesprochene Funktion zur Generierung von Code. Die Wissensbasis soll stattdessen sämtliche grundlegende Eigenschaften der Domäne (Mustersprachen Und Muster) repräsentieren, sodass insbesondere die Linkstruktur der Muster abgebildet wird. Ein Prototyp einer ersten Version von *PatternPedia* soll das gesammelte Wissen anzeigen und erweitern können. Hierfür soll eine Webanwendung erstellt werden, die es Endnutzern ermöglicht, mit der Wissensbasis zu interagieren und diese zu erweitern. Der Prototyp soll Endnutzern die Basisfunktionen eines Musterrepositories anbieten, daher sollen beispielsweise neue Mustersprachen und Muster angelegt werden können. Aufgabe dieser Arbeit ist den Einsatz grundlegender semantische Webtechnologien und Standards für diesen Anwendungsfall (eine Wissensbasis als Grundlage eines Musterrepository) zu evaluieren. Eine Vorgabe hierbei ist, dass kein Applikationsserver verwendet werden soll. Mit dem Einsatz der semantischen Technologien wird dabei die Integrierbarkeit und Wiederverwendbarkeit der Daten sichergestellt und somit Grundvoraussetzungen für weitere schlussfolgernde Softwareagenten wie Watson™ geschaffen. Für den Prototyp ist nicht gefordert, dass er Funktionen wie *intelligente Mustervorschläge* anbietet, die auf Schlussfolgerungen basieren. Er soll jedoch die Daten der Wissensbasis für den Nutzer verständlich anzeigen. Für die Wissensbasis ist gefordert, dass diese insbesondere bezüglich der Links alle notwendigen Informationen bereitstellt, die ein schlussfolgernder Agent für das Vorschlagen passender, kompatibler Muster benötigt.

1.2 Struktur der Arbeit

Im Grundlagenkapitel (Kapitel 1) wird zuerst die Anwendungsdomäne vorgestellt und daher das Konzept von Entwurfsmuster und Mustersprachen eingeführt. Dann folgt die Einführung in Ontologien, die dazu dienen Konzepte und Beziehungen einer Anwendungsdomäne zu beschreiben. Im Einführungsteil für das semantische Web wird dann darauf eingegangen, welche Technologien die Basis für das semantische Webs bilden und wie Ontologien mit diesen konkret formuliert werden können. Kapitel 3 verwendet anschließend die Grundlagen des vorangegangenen Kapitels um eine Ontologie für Mustersprachen zu erstellen. Formal beschrieben wird diese mittels der OWL, einem Standard des semantischen Webs (in Abschnitt 2.3.2 genauer beschrieben). Durch die Ontologie wird eine Wissensbasis aufgebaut, mit der mittels semantischer Webtechnologien interagiert werden kann. Kapitel 4 beschreibt, wie die Daten im semantischen Web veröffentlicht werden und wie ein implementierter Prototyp mit diesen interagiert. Dieser agiert als ein Musterrepository, welches die im dritten Kapitel aufgebaute Wissensbasis als Datengrundlage nutzt und erweitert. Abschließend werden die Ergebnisse in Kapitel 5 zusammengefasst und in einem Ausblick reflektiert.

2 Grundlagen

In diesem Kapitel werden die Grundlagen, auf denen die vorliegende Arbeit aufbaut vorgestellt. Zuerst wird in Abschnitt 2.1 die zu modellierende Domäne beschrieben und daher die grundlegenden Eigenschaften von Entwurfsmustern und Mustersprachen dargelegt. Als Basis wird die Formalisierung von Mustersprachen als Graph von Falkenthal et al. [FBL18] eingeführt. Außerdem werden drei Arten von Links identifiziert, die für bzw. in Mustersprachen verwendet werden. Als nächstes wird in Abschnitt 2.2 definiert, was unter Ontologien zu verstehen ist und wie diese aufgebaut werden. Im Abschnitt 2.3 wird zuerst die Grundidee des semantischen Webs erläutert und dann die wichtigsten Technologien und Standards vorgestellt. Der Fokus wird hierbei darauf gelegt, welche Möglichkeiten diese zur Modellierung von Ontologien bieten.

2.1 Entwurfsmuster und Mustersprachen

Der Begriff "Mustersprache" wurde von Christoph Alexander in seinem Werk *A Pattern Language* eingeführt [FBBL17]. Eine Mustersprache besteht aus Entwurfsmustern und Relationen (Links) zwischen diesen. Jedes Entwurfsmuster beschreibt eine Lösung und den Kontext eines wiederkehrenden Problems [AIS+77]. Wie der Begriff Mustersprache andeuten soll, existieren Entwurfsmuster nicht isoliert voneinander, sondern stehen wie Wörter einer Sprache in Beziehung zueinander und können miteinander kombiniert werden [FBBL17]. Falkenthal et al. [FBL18] beschreibt Alexanders Konzept einer Mustersprache formal als einen gerichteten Graphen:

$$\mathcal{G} = (\mathcal{N}, \mathcal{E}) \tag{2.1}$$

Hierbei entspricht $\mathcal{N}$ einer Menge von Entwurfsmustern und $\mathcal{E}$ einer Menge von Kanten zwischen ihnen: $\mathcal{E} \subseteq \mathcal{N}x\mathcal{N}$ [FBL18]. In Abb. 2.1 ist dies nochmal veranschaulicht, wobei P_i jeweils ein Entwurfsmuster und e_i eine Relation als eine Kante zwischen Entwurfsmustern repräsentiert. Eine Relation beschreibt, wie die Muster zusammenhängen. Oft geben Autoren Referenzen zu anderen Mustern im Inhalt des Entwurfsmusters an, z.B. im Fließtext [FBL18]. Diese Information können aber auch an anderer Stelle dargestellt werden: Fehling et al. [FLR+11] geben beispielsweise eine Matrix an, welche zeigt, ob Muster kohärent sind. Die Einträge in der Matrix der Autoren können daher Relationen zwischen den einzelnen Entwurfsmustern betrachtet werden und drücken aus, ob diese in Kombination benutzt werden können. Gamma et al. [GHJV94] stellen ihre objektorientierten Softwaredesignmuster in einer Übersichtsgraphik dar, in der Musterrelationen als beschriftete Pfeile zwischen den Mustern enthalten sind. Falkenthal et al. [FBBL17] erweitern die oben visualisierte Formalisierung von Mustersprachen dahingehend, dass diese Relationen genauer beschrieben werden und spezifizieren, dass Relationen einen Typ besitzen. Abhängig von diesem konkreten Relationstyp können noch weitere, strukturierte Informationen angehängt werden, die die Relation weiter beschreiben [FBL18]. Beispielsweise kann für eine Relation vom Typ "can be combined with" weitere Informationen über die konkrete Umsetzung der Kombination der Muster

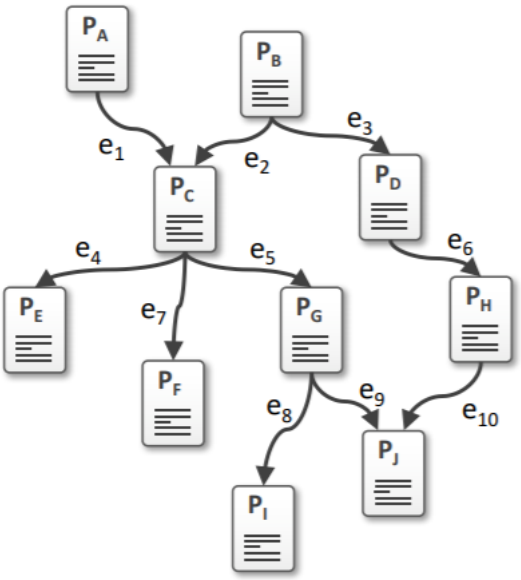


Abbildung 2.1: Graph einer Mustersprache mit gerichteten Kanten e_i als Beziehungen zwischen den einzelnen Mustern P_j. Quelle: [FBL18].

angegeben werden [FBL18]. Relationen, die die Kompatibilität von Mustern spezifizieren, können dazu verwendet werden, Muster zu Musterpfaden zu kombinieren [FBBL17]. Sie können ebenso auf ähnliche Muster verweisen und so weitere Lösungen aufzeigen. Links zwischen Mustern werden daher auf unterschiedliche Weise ausgedrückt (Matrix, Graphik, Fließtext, etc.), bieten dem Leser aber eine wichtige Orientierungshilfe beim Navigieren durch eine Mustersprache.

Links zwischen Mustern lassen sich in drei verschiedene Arten einteilen (in Abb. 2.2 dargestellt). Für die erste Art, Links innerhalb einer Mustersprache, wurden im letzten Abschnitt bereits einige Beispiele gegeben - der Graph der Mustersprache in Abb. 2.1 enthält nur solche Kanten. Neben

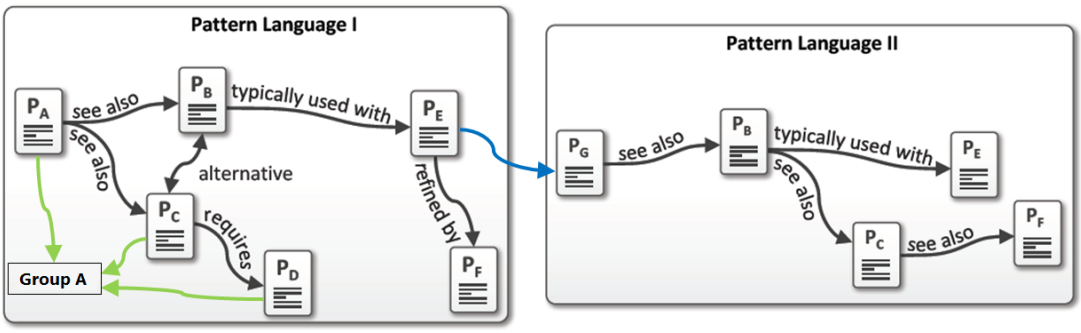


Abbildung 2.2: Alle drei beschriebenen Linkarten im Überblick. Die beschrifteten, schwarzen Kanten stellen Links innerhalb der Mustersprachen dar. Die blaue Kante zeigt einen Mustersprachen übergreifenden Link. Die grünen Kanten ordnen ein Muster der Gruppe A zu. Bearbeitete Abbildung nach Falkenthal et al. [FBL18].

Links innerhalb einer Mustersprache sind außerdem mustersprachenübergreifende Links möglich. Fehling et al. [FLR+14] geben beispielsweise im Fließtext ihres *Message-oriented Middleware* Muster an, dass dieses auf den Messaging Mustern von Hohpe und Woolf [HW04] aufbaut. Für eine Gesamtlösung eines komplexen Problems sind diese Referenzen sehr hilfreich, weil meist die Anwendung von Muster aus mehreren Mustersprachen benötigt wird: Beispielsweise kann die Architektur der Anwendung einem Cloud Computing Muster folgen und die Kommunikation der Komponenten mittels der *Messaging* Muster realisiert werden. Als letzter Typ von Links in Mustersprachen kann identifiziert werden, dass Muster von Autoren oft gruppiert oder Kategorien zugeordnet werden. In Abb. 2.3 ist die Gruppierung der *Messaging Muster* von Hohpe und Woolf [HW04] gezeigt, bei der jedes Muster einer Komponente zugeordnet wird. Die Anzahl von Gruppen variiert und teilweise ist es für Muster auch möglich, mehreren Gruppen bzw. Kategorien zugeordnet zu werden [KISV16]. Andere Beispiele für die Zuweisung von Kategorien oder Gruppen finden sich in den Mustersprachen von Fehling et al. [FLR+14] oder Gamma et al. [GHJV94]. Mithilfe der drei angesprochenen Linkarten lässt sich für eine Mustersprache die Basiseigenschaften ihrer Muster ausdrücken: Es lässt sich darstellen, welche Beziehungen die Muster untereinander aufweisen (Links innerhalb der Sprache), wie man diese inhaltlich zu Gruppen zusammenfassen kann (gruppierende Links) und in welchem Zusammenhang die Muster zu anderen Mustersprachen stehen (mustersprachenübergreifende Links).

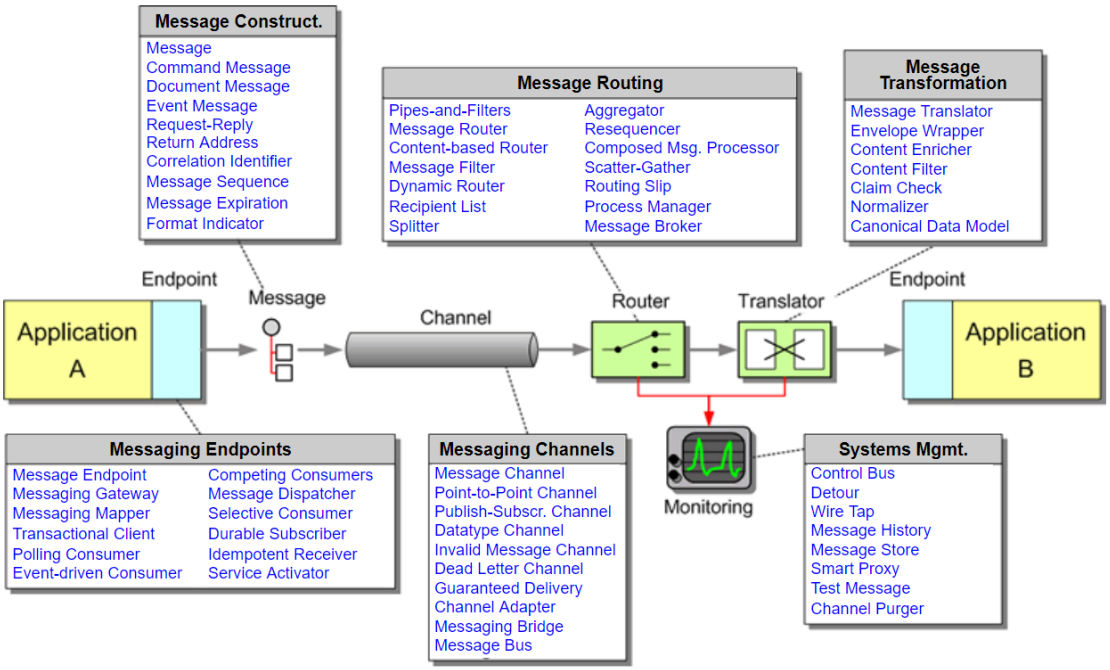


Abbildung 2.3: Mustersprache von Hohpe und Woolf [HW04], die Muster zur Integration von Unternehmensanwendungen auf Basis von Messaging enthält. In der Abbildung wird jedes Muster (in blau aufgelistet) einer Komponente in einem typischem Architekturaufbau zugeordnet (grau gestrichelte Linie). Für die dadurch entstehenden Gruppen ist jeweils ein Titel angegeben (durch grauen Hintergrund hervorgehoben).

Je nach Themenbereich, den eine Mustersprache abdeckt, bietet sich ein anderer Aufbau der Muster an. In Abb. 2.4 werden diesbezüglich zwei Mustersprachen aus verschiedenen Bereichen verglichen. Links ist der Aufbau von Mustern der *Cloud Computing* Mustersprache von Fehling et al. [FLR+14]

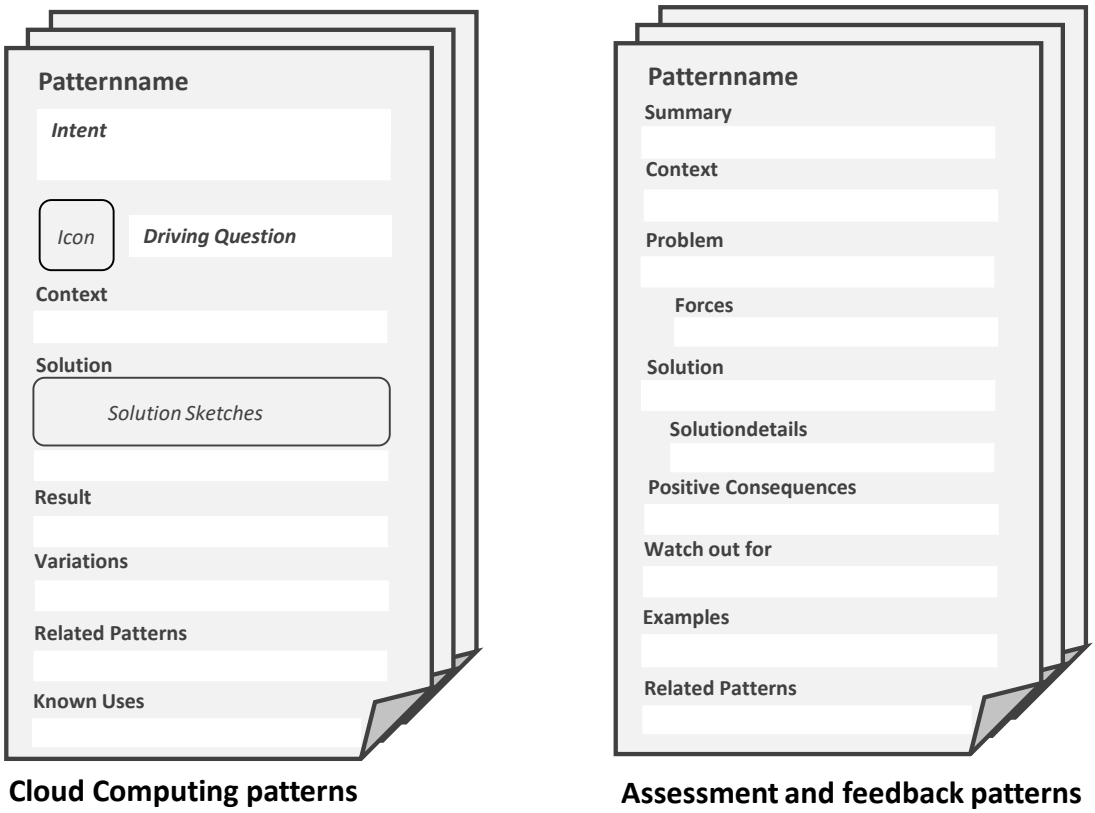


Abbildung 2.4: Aufbau von zwei Mustersprachen im Vergleich. Links ist der Aufbau der *Cloud Computing* Muster von Fehling et al. [FLR+14] dargestellt, rechts der Aufbau der *Assessment and feedback* Muster von Warburton et al. [WBK+16]

dargestellt, die aus dem Bereich der Informationstechnologie stammen. Im Vergleich zu den rechts dargestellten *Assessment and feedback* Mustersprache von Warburton et al. [WBK+16] finden viele Sektionen ein entsprechendes Pendant, z.B. enthalten beide Sektionen mit den Titeln *Context*, *Solution* und *Related Patterns*. Der Aufbau von Fehling et al. [FLR+14]'s Mustern sieht zusätzlich *Solution Sketches* (Lösungsentwurfsskizzen) vor. Da diese Mustersprache beschreibt, wie verschiedene Komponenten miteinander agieren, bietet es sich oft an die Beziehungen der Komponenten zueinander mit einer Skizze zu verdeutlichen (in Kapitel 1 Abb. 1.2 gezeigt). Eine graphische Visualisierung der menschlichen Interaktion in *Assessment and feedback* Muster von Warburton et al. [WBK+16] ist nicht zielführend, da menschliche Interaktion (inklusive Mimik, Gestik, Tonfall, etc.) in seiner Komplexität nicht darstellbar ist. Autoren neuer Mustersprachen sollen daher die Freiheit besitzen, für ihre Mustersprache Sektionen wegzulassen oder neue einführen.

2.1.1 Mustersichten

Betrachtet man Mustersprachen als Datenquellen, kann man für die Links zwischen ihnen eine weitere Datenquelle erstellen, die Sprachen verknüpft. Eine Mustersprache beschreibt demnach allein ihre Domäne und kann ihren eigenen Aufbau der Muster beibehalten. Ein weiterer Vorteil dieser Trennung ist, dass Links zwischen Sprachen nicht in beide eingepflegt werden müssen. Stattdessen fügen wir sie in einer sogenannten Mustersicht hinzu. Das Wort Mustersicht ist an traditionelle Datenbanksichten angelehnt, wie sie in einer relationalen Datenbank mit dem Schlüsselwort *view* erstellt werden können [UM12]. Hiermit wird keine physikalische Tabelle erzeugt, sondern eine abgeleitete, virtuelle Tabelle [UM12]. Mittels einer Datenbanksicht können Datenquellen integriert, und falls benötigt mit weiteren Daten angereichert werden. Dies bietet sich vor allem an, falls die Ursprungsdatenquellen beibehalten werden sollen [BKLW99]. Auf Mustersprachen übertragen, wird mit der Einführung einer Sicht für mustersprachenübergreifende Links eine konzeptionelle Trennung der Daten der separat gewachsenen Mustersprachen von diesen erreicht. Im Folgenden wird anhand von zwei Fallbeispielen verdeutlicht, warum die Einführung von Mustersichten vorteilhaft ist.

1. **Fallbeispiel 1 (Selektion einer Untermenge)**: Eine Datenbanksicht kann bestimmte Datensätze aus einer referenzierten Datenquelle selektieren. Für eine Mustersprache als Datenquelle entspricht dies der Selektion einer Menge von Mustern und Verbindungen. Eine Mustersicht kann z.B. alle Muster einer bestimmten Gruppe beinhalten (womit diese selektiert werden) und somit diese als separaten Mustergraph adressieren. Insbesondere für ein Teilgebiet einer Mustersprache, das immer stärker wächst (daher immer neue Muster entstehen), kann dieses somit als eigene Mustersprache behandelt werden ohne sämtliche Muster zu duplizieren.

2. **Fallbeispiel 2 (Aggregation von Datenquellen)**: Falls zwischen Mustersprachen Links entstehen, kann es sich anbieten, diese Sprachen zu einer größeren Mustersprache zusammenzufassen. Schon Alexander et al. [AIS+77] beschreibt diesen Fall und folgert, dass eine neue, größere Mustersprache entsteht, die die beiden Ausgangssprachen vereint. Hieran anknüpfend, erweitern Falkenthal et al. [FBL18] mit der Definition des *Pattern Language Aggregator* ihre Formalisierung einer Mustersprache dahingehend, auch Beziehungen zwischen Mustersprachen auszudrücken: Aus den Kanten $\mathcal{E}$ zwischen den Mustersprache, sowie den Kanten und Knoten der Mustersprachen $\mathcal{G}_1$ und $\mathcal{G}_2$ kann eine neue Sprache definiert werden [FBL18]:

$$\mathcal{G}_3 = \bigodot (\mathcal{G}_1, \mathcal{G}_2, \mathcal{E}) = (\mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}) \tag{2.2}$$

   Die neue Mustersprache enthält somit alle Muster und Links der Ursprungsmustersprachen $\mathcal{G}_1$ und $\mathcal{G}_2$, sowie die Links, die die Kanten $\mathcal{E}$ repräsentieren. Eine Mustersicht kann diese Aggregation zweier Mustersprachen durchführen, indem sie, der Definition von Falkenthal et al. [FBL18] folgend, alle Muster und Links der Ursprungsmustersprachen referenziert und zusätzliche Links zwischen ihnen einpflegt. Die daraus resultierende Mustersicht kann dadurch mehr ausdrücken als die beiden zugrundeliegenden, einzelnen Mustersprachen.

Die Fallbeispiele betonen eine weitere Charakteristik, die Alexander et al. [AIS+77] beschreibt: Mustersprachen sind nicht statisch, sondern *lebende Netzwerke*. Somit können über die Zeit hinweg neue Muster, Verbindungen oder Mustersprachen entstehen sowie bestehende Strukturen sich verändern. Mustersichten erlauben, diese Veränderungen flexibel abzubilden.

2.2 Ontologien

Für den Austausch von Informationen wird als Grundlage für die Kommunikation Wissen über Konzepte und Zusammenhänge benötigt [Den12]. Formal kann dieses Wissen als Ontologie beschrieben werden, die Dengel [Den12] folgendermaßen definiert:

> ”Eine Ontologie ist eine formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung.“ ([Den12], S.65)

Folgende Eigenschaften bzw. Aspekte über Ontologien sind mit der obigen Definition kurz zusammengefasst [DFH11]:

- *Formalität*: Ontologien werden in einer auf formaler Semantik basierender Sprache ausgedrückt, die maschinenlesbar ist [DFH11]. Zu den bekanntesten Sprachen um Ontologien ausdrücken, gehören das RDFS und OWL [DFH11], auf die in Abschnitt 2.3.2 und 2.3.2 nochmals speziell eingegangen wird.
- *Explizität*: Wissen wird explizit ausdrückt, sodass es von Maschinen verarbeitet werden kann [DFH11].
- *Konsens*: Ontologien spiegeln die Übereinstimmung einer Gruppe bezüglich Konzepten eines Gebiets wider [DFH11].
- *Konzeptualisierung*: Ontologien beschreiben ein Konzept möglichst allgemein [DFH11].
- *Domänenspezifisch*: Innerhalb einer Ontologie wird nur Wissen über eine Domäne repräsentiert [DFH11].

Im Folgenden werden die Grundelemente einer Ontologie (*Klassen, Eigenschaften, Instanzen* und *Axiome*) [DFH11] anhand der Beispielontologie in Abb. 2.5 erläutert. Unternehmen, Person und Säugetier sind *Klassen* dieser Ontologie und im oberen Bereich der Abbildung zu finden. Klassen können alternativ auch als Konzepte bezeichnet werden [Den12]. Mit *Eigenschaften* wie ”ist eine Unterklasse von“ oder ”arbeitet für“ können Beziehungen zwischen Klassen, Instanzen oder Attributen ausgedrückt werden. Im unteren Teil der Abbildung finden sich *Instanzen* dieser Klassen wie z.B. das Unternehmen ”Slate Rock Company“. Ebenfalls sind Datenwerte Teil der *Instanzen* der Ontologie, wie z.B. der Wert einer Zeichenkette. Die konzeptuellen Einheiten *Klassen, Eigenschaften* und *Instanzen* zur Wissensrepräsentation lassen sich von den Aussagen über die Domäne trennen [DFH11]. In *Axiomen* (von Dengel [Den12] auch Regeln genannt) werden diese konzeptionellen Einheiten dann genutzt, um mit ihnen Aussagen über Wissen der Domäne zu formulieren [DFH11]. Im Beispiel werden verschiedene Arten von Aussagen über die Domäne getroffen, die sich mit den meisten Sprachen für Ontologien formulieren lassen:

1. Die Klasse Person ist eine Unterklasse von Säugetier. (Subsumtion [DFH11])
2. Die ”Slate Rock Company“ ist eine Instanz der Klasse Unternehmen. (Instanziierung [DFH11])
3. Die linke Person arbeitet für die ”Slate Rock Company“. (Zwei Instanzen werden über eine Eigenschaft miteinander verbunden [DFH11])

4. Die Eigenschaft ”arbeitet für“ kann nur dazu verwendet werden, eine Instanz der Klasse Person mit einer Instanz der Klasse Unternehmen zu verbinden. (Einschränkung der Definitions- und Zielmenge der Eigenschaft [DFH11])

Außerdem nennen Domingue et al. [DFH11] noch zwei weitere Typen von Aussagen, für die unsere Beispielontologie leicht erweitert werden kann:

5. Eine Instanz der Klasse Unternehmen kann nicht ebenfalls eine Instanz der Klasse Person sein. (Aussagen über disjunkte Klassen)
6. Für die rechte Person ist als Vorname “Lucy” angegeben. (Zuweisung einer Eigenschaft)

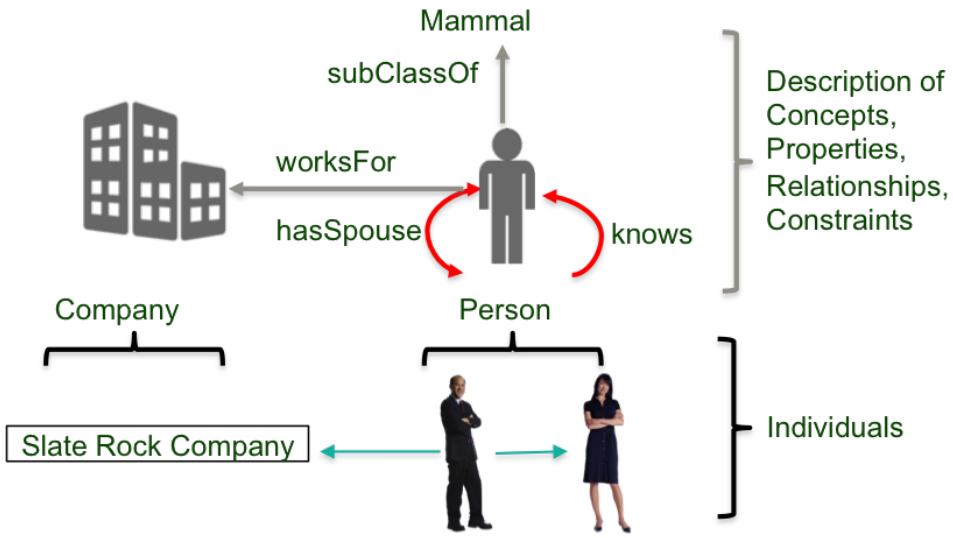


Abbildung 2.5: Darstellung einer Beispielontologie, für die im oberen Teil der Abbildung Konzepte, Eigenschaften, Relationen und Einschränkungen festgelegt und im unteren Teil der Abbildung konkrete Instanzen erstellt werden. Quelle: http://graphdb.ontotext.com/free/devhub/ontologies.html

Wie mit den oberen Aussagen gezeigt, repräsentieren Ontologien Domänenwissen, sodass dieses einfach wiederverwendet werden kann [Den12]. Gleichzeitig stellen sie hierfür ein gemeinsames Begriffslexikon bereit [Den12]. Anhand der Anzahl der Einschränkungen für die Verwendung von Klassen, Instanzen und Eigenschaften lassen sie sich in leichtgewichtig (wenig Einschränkungen) und schwergewichtig (viele Einschränkungen) einteilen [Den12].

2.3 Das semantische Web

Berners-Lee und Fischetti [BF01] beschreiben in ihrem Werk "*Weaving the Web*" ihre Vision für die Zukunft des Web: Mit der ersten Version des Webs wird die Kommunikation zwischen Menschen ermöglicht. Computer und Netzwerke stellen die Infrastruktur für den Informationsraum bereit. Darauf folgt eine Version des Webs, in der Maschinen stärker in Aktion treten - das semantische Web. Um genauer zu erläutern wie die Version ermöglicht werden soll, wird in diesem Unterkapitel zunächst die historische Entwicklung des Webs dargestellt. Damit wird verdeutlicht, worauf das semantische Web aufbaut und wieweit es sich von bisherigen Schwerpunkten des Webs unterscheidet. Anschließend werden die Technologien und Standards, auf denen das semantische Web aufgebaut ist, erläutert.

2.3.1 Geschichte und Evolution des Webs

Durch die offene, dezentrale Infrastruktur des World Wide Web (WWW) ist dieses geradezu prädestiniert für Wandel durch Innovation [Hen08]. Deshalb ist es nicht verwunderlich, dass das Web, das Tim Berners-Lee im Jahr 1989 erfand [BF01] einige Veränderungen durchlief. Anhand der wesentlichen Neuerungen lässt sich das Web in Generationen einteilen [AAM15]. Die Bezeichnungen für die einzelnen Generationen (Web 1.0 - 3.0) lassen zwar vermuten, dass diese einander vollständig ablösen, dies ist jedoch nicht der Fall. Durch technische Innovationen entstehen im Laufe der Zeit immer mehr Webseiten mit Funktionalitäten einer neuen Generation, sodass sich die einzelnen Generationen auch zeitlich überlappen. In den folgenden Abschnitten werden die Schwerpunkte und wichtigsten Neuerungen der Generationen kurz zusammengefasst.

Web 1.0

In der ersten Generation bestand das Web aus einem statischen Netzwerk, mit dem Inhalt einer Webseite konnte daher nicht interagiert werden [AAM15]. Eine der wichtigsten Neuerungen des dezentralen Netzwerks bestand darin, dass auf andere Dokumente im Web verlinkt werden konnte [AAM15; BF01]. Hierdurch entstand ein neuer Informationsraum, der es Endnutzern erlaubte nach Informationen zu suchen und diese zu lesen [Cho14].

Web 2.0

Technische Entwicklungen ermöglichten Endnutzern in der zweiten Generation des Webs auch aktiv Inhalte beizutragen [Cho14]. Zu den wichtigsten Applikationen im Web 2.0 gehören Flickr, Wikipedia, Facebook, MySpace und YouTube [Hen09]. Durch die Technologien und Anwendungen dieser Webgeneration wurden erstmalig dynamische Inhalte und Interaktion ermöglicht [AAM15]. Diese technologischen Neuerungen bezeichnete Tim O'Reilly als *Web 2.0* [Cho14], die den Übergang von statischer Informationsseiten des *Web 1.0* zu plattform-getrieben Inhalten einleiteten.

Web 3.0

Technologien des semantischen Webs ermöglichen es, dass Daten von Maschinen direkt oder indirekt verarbeitet werden können [BF01]. Der Ausdruck *semantisch* soll hierbei verdeutlichen, dass die Bedeutung von Daten im Web nicht nur von Menschen, sondern auch von Maschinen erfasst bzw. verstanden werden kann [AAM15; Pas04]. Alternativ wird diese Generation von John Markoff auch als *Web 3.0* bezeichnet [Hen08]. Der stärkste Anstieg von Aktivität zum semantischen Web erfolgte um die Jahrtausendwende [Hen08]. Wie Berners-Lee et al. [BHL+01] betont, ist das semantische Web kein neues, separates Web, sondern ergänzt das bisherige Web. Während im *Web 2.0* für Menschen geschriebenen Dokumente und Hyperlinks im Vordergrund stehen, erlaubt das semantische Web zusätzlich typisierte Links [AAM15]. Die maschinenlesbaren Daten einer durch eine URI referenzierte Ressource können in verschiedenen Dokumenten gespeichert werden und durch die Standardisierung des Datenformats leicht integriert werden [Pas04]. Durch die Verbreitung dieser Daten kann mit den Inhalten des semantischen Web wie mit einer großen, interoperable Wissensdatenbank interagiert werden [Pas04].

2.3.2 Semantic Web Layer Cake - Aufbau und Technologien des semantischen Webs

Das World Wide Web Consortium (W3C), welches eine zentrale Rolle in der Entwicklung von semantischen Webtechnologien einnimmt, veröffentlichte den sogenannte *Semantic Web Layer Cake*[Pas04]. Hiermit geben die Autoren einen Überblick über die wichtigsten Bausteine der Architektur, das dem semantischen Web zugrunde liegt [Pas04]. Es gibt einige Versionen dieses Dokuments [Pas04]; in Abb. 2.6 ist eine Version von Hogan [Hog13] dargestellt, die widerspiegelt, ob es für die jeweilige Schicht schon Standards oder Technologien der W3C gibt. Bei der Architektur handelt es sich um eine Schichtenarchitektur [Cho14]. Einzelne Schichten bauen dabei aufeinander auf, z.B. werden die in der untersten Schicht eingeführten Identifikatoren in allen Schichten darüber verwendet. Anhand der Schichtenarchitektur werden in den folgenden Unterabschnitten die einzelnen Bausteine des Diagramms (Abb. 2.6) kurz vorgestellt, angefangen mit den untersten Schichten.

Schriftzeichen (Characters)

Das semantische Web verwendet den Unicode-Zeichensatz [Hog13], ein internationaler Standard für die Kodierung von Schriftzeichen. Binär kodierte Zeichenketten können somit als Text angezeigt werden [Hog13].

Identifikatoren (Identifiers)

Um Ressourcen zu beschreiben, müssen diese klar referenziert werden können. Eine URI, welche im semantischen Web genau zu diesem Zweck verwendet wird, kann folgendermaßen definiert werden:

> ”A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource“ ([BFM05])

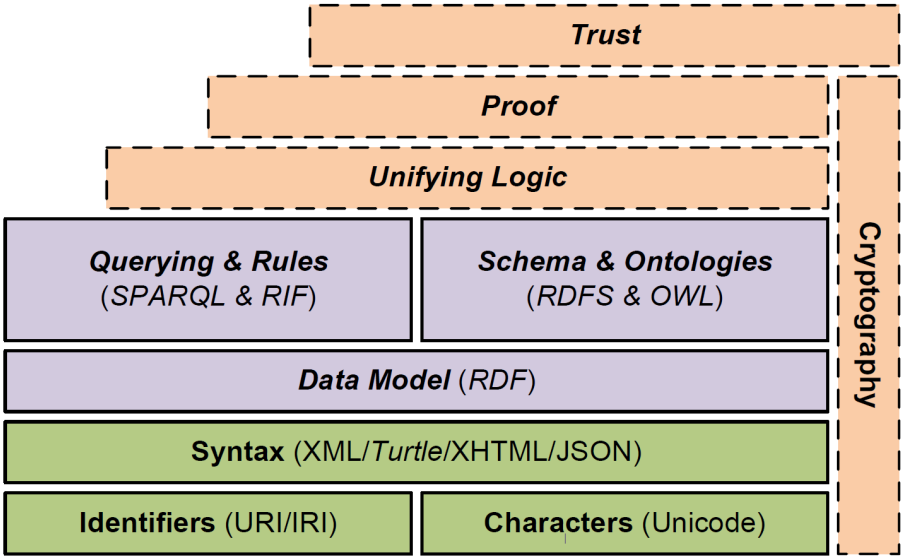


Abbildung 2.6: Wichtigste Technologien und Standards, auf denen das semantischen Web aufgebaut ist. Hierbei handelt sich um eine Schichtenarchitektur, bei der obere Schichten auf den unteren aufbauen. Schichten, für die noch keine Technologien oder Standards von der W3C festgelegt wurden, sind durch eine gestrichelte Umrandung gekennzeichnet. Quelle: [Hog13].

Ein Uniform Resource Locator (URL) ist ebenfalls eine URI und identifiziert somit eine Ressource [Pas04]. Zusätzlich gibt sie die Adresse (location) einer Ressource an [Pas04]. Internationalized Resource Identifier (IRI) sind eine Verallgemeinerung von URIs, die für den Identifikator Zeichen des Unicode Standard erlaubt [Hog13].

Eine Referenz zu einer URI besteht aus einer URI und optionalen Zeichen, z.B. der Teil einer URI, die dem #-Zeichen als sogenannter *fragment identifier* folgt [Pas04].

Syntax

Eine maschinenverständliche Syntax wird benötigt, um Inhalte zu formulieren, deren Daten von Maschinen in die wesentlichen Bestandteile zerlegt und analysiert werden können [Hog13]. Mit der Veröffentlichung des Resource Description Framework (RDF) Datenmodells (im nächsten Abschnitt beschrieben) publizierte das Komitee ebenfalls ein Extensible Markup Language (XML) Vokabular, die sogenannte RDF/XML Syntax Spezifikation [Pas04]. Mit einer solchen Syntax für RDF können Informationen über Ressourcen mithilfe des Vokabulars ausgedrückt werden. Als Alternative zur XML/RDF Syntax kann zum Beispiel die Terse RDF Triple Language (Turtle) Syntax verwendet werden [Pas04]. Im nächsten Abschnitt ist für die zwei Syntaxen XML/RDF und Turtle jeweils ein Beispiel angegeben.

Datenmodell (Data Model)

Resource Description Framework (RDF) ist ein Datenmodell des W3C Webkomitees, das dafür entwickelt wurde, Informationen über Ressourcen zu in maschinenlesbarer Form zu repräsentieren [Pas04]. Diese Informationen werden als dreiteilige Aussagen (*statements*) formuliert, die auch als *Triple* bezeichnet werden. Aussagen bestehen wie in 2.7 dargestellt aus Subjekt, Prädikat und Objekt. Der Wert eines Objektes kann wie im Beispiel gezeigt eine Zeichenkette sein. Solche primitiven Datentypen werden auch als Literale bezeichnet [Pas04]. Statt eines Literals kann aber auch eine andere Ressource als Objekt angegeben werden [Pas04].

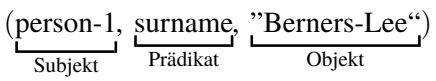


Abbildung 2.7: Aufbau einer dreiteiligen RDF Aussage nach Passin [Pas04]. Die Darstellung benutzt keine gültige Syntax für RDF, sondern stellt die Bestandteile anhand einer Bespielaussage dar (Subjekt, Prädikat, Objekt).

Die Aussage aus Abb.2.7 kann in einer Syntax für RDF formuliert werden, im Folgenden wird hierfür die XML/RDF sowie die Turtle Syntax gezeigt. In der Turtle Notation kann die Aussage aus Abb.2.7 folgendermaßen formuliert werden:

```
@prefix : <http:example.org/#> .
:person-1 :surname "Berners-Lee" .
```

Um für einen Vergleich die entsprechende Aussage in der XML/RDF Notation zu erhalten, wurde die Turtle Notation mit `http://www.easyrdf.org/converter` zu XML/RDF konvertiert:

```
<?xml version="1.0" encoding="utf-8" ?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns0="http:example.org/#">

  <rdf:Description rdf:about="http:example.org/#person-1">
    <ns0:surname>Berners-Lee</ns0:surname>
  </rdf:Description>

</rdf:RDF>
```

Hierbei wird deutlich, dass die Turtle-Syntax deutlich kompakter ist, was sich auch in der Bezeichnung der Syntax *Terse RDF Triple Language*, widerspiegelt, was mit Turtle abgekürzt wird.

Es ist aber auch möglich, die Aussage als Graph darzustellen, wie Abb.2.8 zeigt.

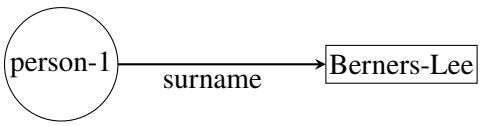


Abbildung 2.8: Graphdarstellung der Aussage aus Abb.2.7.

Im Gegensatz zu Daten einer relationalen Datenbank bieten Daten, die dem RDF Datenmodell folgen, folgende Vorteile [Pas04]:

- Sie ermöglichen den leichten Datenaustausch mit einer anderen Anwendung, die das RDF-Format ebenfalls verarbeiten kann.
- Das Datenformat ist standardisiert, so dass auch andere Software als die ursprüngliche Datenbank die Daten verarbeiten kann.
- Mit RDF Prozessoren können logische Schlussfolgerungen gezogen werden.

Durch benannte Eigenschaften und Datenwerten können mit RDF drei-teilige Aussagen über Ressourcen und Beziehungen zwischen Ressourcen formuliert werden [Den12]. Limitationen von RDF sind, dass RDF keine Funktionalität für Klassen zur Verfügung stellt und somit eine Eigenschaft auch nicht auf Ressourcen bestimmter Klassen beschränkt werden kann [Den12].

RDF Schema (RDFS) & Webontology Language (OWL)

Mit RDFS ist es möglich, einfache Ontologien zu erstellen [Den12]. Hierfür baut es auf RDF auf, indem alle RDFS Aussagen als RDF Aussagen ausgedrückt werden können [Pas04]. RDFS führt das Konzept von Klassen ein, jedoch muss für eine Repräsentation von Daten aus z.B. einer Datenbank mit RDFS zunächst keine Klassenstruktur für Instanzen definiert werden. Dieses Wissen kann bei Bedarf flexibel nachgepflegt werden [Den12]. Weiter ist es möglich, für Eigenschaften Werte- und Definitionsbereich einzuschränken [Den12]: In der vierten Aussage der Beispielontologie aus Abschnitt 2.2 wird der Wertebereich für die Eigenschaft ”arbeitet für“ auf die Klasse Person und den Definitionsbereich auf die Klasse Unternehmen eingeschränkt. Mit der beschriebenen Funktionalität lassen sich daher Ontologien aufzubauen und einzelne Instanzen für definierten Klassen erstellen [Den12].

OWL, ein weiterer W3C-Standard, bietet noch komplexere Ausdrucksmöglichkeiten als RDF und RDFS [Den12]. Genau wie RDFS benutzt OWL RDF, um Aussagen zu formulieren [Pas04]. Mit RDFS und RDF lassen sich zwar Ontologien definieren, aber wenige Einschränkungen für diese festlegen [Pas04]. Im Gegensatz hierzu erlaubt OWL, Existenz- und Kardinalitätseinschränkungen auszudrücken [Den12]. Beispielsweise könnte man zu der Ontologie von 2.5 ergänzen, dass eine Person höchstens einen Ehepartner besitzen kann und jede Person genau eine Mutter besitzt. Ebenfalls fehlt bei RDFS die Möglichkeit, die Definitions- und Zielmenge einer Eigenschaft lokal zu beschränken: Somit kann man nicht festlegen, dass die Eigenschaft “hasMother” für eine Instanz der Klasse Person auf eine Person verweist, während Tiere nur ein Tier zur Mutter haben dürfen [Den12]. Durch OWL ist es möglich, Eigenschaften genauer als mit RDFS zu beschreiben indem sich diese in zwei Klassen einteilen lassen: *Objekteigenschaften* (ObjectProperty) und *Datentypeigenschaften* (DatatypeProperty) [Yu14]. Mittels *Objekteigenschaften* werden Ressourcen miteinander verbunden, *Datentypeigenschaften* verbinden eine Ressource mit einem Literal [Yu14]. Zusätzlich können wie Yu [Yu14] beschreibt, Eigenschaften folgendermaßen charakterisiert werden:

- *symmetrisch*: Ist eine Ressource R_1 über eine symmetrische Eigenschaft mit einer zweiten Ressource R_2 verbunden, dann folgt hieraus dass R_2 ebenfalls über diese Eigenschaft mit R_1 verbunden ist. In der Beispielontologie von Abb. 2.5 kann man ”besitzt Ehepartner“ als symmetrische Eigenschaft modellieren. Anhand der Aussage ”Person A ist mit Person B verheiratet“, kann man die Aussage ableiten, dass B mit A verheiratet ist.

- *transitiv*: Ist eine Ressource R_1 über eine transitive Eigenschaft mit einer zweiten Ressource R_2 verbunden, und diese ist über die selbe Eigenschaft mit Ressource R_3 verbunden, dann folgt hieraus dass R_1 ebenfalls über die besagte Eigenschaft mit R_3 verbunden ist.
- *funktional*: Ist eine Eigenschaft funktional, so kann für eine Instanz nur ein Wert für diese Eigenschaft angegeben werden. Sie beschreibt somit eine n:1 Beziehung.
- *inverse funktional*: Eine invers funktionale Eigenschaft ist das Gegenteil einer funktionalen Eigenschaft und daher kann ein Wert nur einer Instanz zugeordnet werden. Dadurch wird eine 1:n Beziehung beschrieben.
- *invers*: Ist eine Ressource R_1 über eine Eigenschaften e_1 mit Ressource R_2 verbunden und e_2 die inverse Eigenschaft zu e_1, dann folgt hieraus, dass R_2 durch $e2$ mit R_1 verbunden ist.

Diese zusätzlichen Folgerungen, die sich anhand der fünf Charakteristiken ziehen lassen, sind implizite Aussagen. Die reichere Semantik um Eigenschaften zu beschreiben, resultiert daher in neuen Möglichkeiten für Interferenzen [Yu14].

Die Sprache ist bezüglich Entscheidbarkeit gut verstanden, sodass für jeden ihrer Dialekte ausgesagt werden kann, ob dieser entscheidbar ist. Entscheidbar bedeutet in diesem Kontext, dass für alle Schlussfolgerungen Folgendes gilt: Diese können von einer Interferenzmaschine berechnet werden, wobei die Berechnung ein Ergebnis liefert und daher terminiert - sie führen also immer zu einer Entscheidung [Yu14]. Das W3C-Kommitee veröffentlichte zwei Versionen von OWL, OWL und OWL2 [Yu14]. OWL2 unterstützt alle bisherigen Funktionen der ersten Version [Yu14]. Da keine spezielleren Ausdrucksmöglichkeiten von OWL2 im Verlauf der Arbeit benötigt wurden, wird an dieser Stelle nur auf OWL1 eingegangen. Die drei Dialekte von OWL 1 sind OWL Full, OWL DL und OWL Lite und unterscheiden sich bezüglich ihrer Entscheidbarkeit. Die letzteren genannten Dialekte schränken OWL Full ein, um logische Schlussfolgerungen entscheidbar zu halten [Pas04]. OWL Full ist der ausdrucksstärkste Dialekt, was dazu führt, dass dieser Dialekt nicht entscheidbar ist [Pas04]. OWL DL erlaubt nur ein- und zweistellige Prädikate [Den12], in anderen Worten *deskriptive Logik (descriptive logic)* [Pas04]. Hierdurch wird sichergestellt, dass ein OWL-Interferenzmaschine bezüglich einer Anfrage immer eine Entscheidung treffen kann [Den12]. OWL Lite ist noch restriktiver als OWL DL [Pas04], und dadurch ebenfalls entscheidbar [Den12]. Es erlaubt zum Beispiel nur die für die Kardinalität nur die Werte 0 oder 1 anzugeben [Pas04]. Der Hauptgrund für die Entstehung von OWL Lite bestand darin, in einem ersten Schritt Prozessoren für eine vereinfachte Sprache aufzubauen und diese dann für die mächtigeren Sprachdialekte zu erweitern [Pas04].

Anfragen und Regeln (Querying & Rules)

Um Daten des RDF-Schemas besser verarbeiten zu können, bietet es sich an hierfür eine Anfragesprache zu nutzen. SPARQL steht als rekursives Akronym für SPARQL Protocol and RDF Query Language. Als Anfragesprache für Daten des RDF-Schemas nimmt SPARQL für das semantische Web laut Tim Berners-Lee einen ähnlichen Stellenwert ein wie Structured Language (SQL) für relationale Datenbanken [DuC13]. Rule Interchange Format (RIF) ist ein von der W3C veröffentlichter Standard, um verschiedene Regelsprachen integrieren zu können [Hog13].

2.3.3 Unvollständige Schichten (*Unifying Logic*, *Proof*, *Trust* und *Cryptography*)

In der Abb. 2.6 ist durch gestrichelte Umrandungen markiert, dass für die Schichten *Cryptography*, *Unifying Logic*, *Proof* und *Trust* noch kein Standard von der W3C veröffentlicht wurde. An dieser Stelle wird deshalb kurz weiter auf die Idee hinter den bislang noch fehlenden Bausteinen eingegangen. Für die *Cryptography*-Schicht ist nicht vorgesehen, dass sie auf anderen Schichten aufbaut (vgl. Abb. 2.6). Typische kryptographische Anforderungen an Anwendungen, wie dass Identitäten verifiziert werden oder allgemeine Zugriffskontrolle benötigt wird, gelten ebenso für das semantische Web [Hog13]. Hierfür sollen geeignete Techniken wie digitale Signaturen deshalb ebenfalls eingesetzt werden [Hog13]. Die Schicht *Unifying Logic* bietet die Grundlage dafür, die unterhalb liegenden Schichten zu integrieren [Hog13]. Hogan [Hog13] führt hierfür aus, dass es bereits Ansätze gibt, welche tiefer liegende Bausteine wie Regeln und Anfragen, oder Regeln und Ontologien miteinander zu kombinieren. Mithilfe der Logik der *Unifying Logic*-Schicht sollen die Konsistenz und Korrektheit von Daten überprüft werden und implizite Schlussfolgerungen aus den Daten gezogen werden [Pas04]. Die darauf aufbauende Schicht (*Proof*) stellt dar, wie logische Schlüsse gezogen wurden [Pas04], daher kann beispielsweise dargestellt werden, welche externen Quellen für die Schlussfolgerungen benutzt wurden. Die Schicht *Trust* verwendet die unterliegenden Schichten, um Schlussfolgerungen darüber zu ziehen welche Daten aus einem vertrauenswürdigen Ursprung stammen [Pas04]. Auch wenn verschiedenste Schichten noch nicht vollständig umgesetzt wurden, betont Passin [Pas04], dass die komplette Umsetzung aller Schichten keine Voraussetzung dafür ist, bereits Teile des semantischen Webs erfolgreich zu nutzen und von ihnen zu profitieren.

2.3.4 Veröffentlichung von semantischen Daten

Da das semantische Web das bisherige Web ergänzen soll, wird im Folgenden noch darauf eingegangen, worauf bei der Veröffentlichung von semantischen Daten geachtet werden soll.

Linked Open Data

Tim Berners-Lee betonte in seinem Artikel ”Design Issues: Linked Data note“ die Wichtigkeit, große Datensätze in maschinenlesbaren Formaten zur Verfügung zu stellen, um den Wert aller verknüpften Daten zu steigern [Den12]. Hieraus bildete sich die *Linked Open Data*-Bewegung, die sich zum Ziel gesetzt hat, die Veröffentlichung großer, semantischer Datensätze weiter vorantreiben [Den12]. Abbildung 2.9 zeigt eine Visualisierung der miteinander verknüpften, öffentlich zugänglich gemachten Datenmengen.

Linked Data Principles

Die *Linked Data Principles* von Tim Bernes-Lee [Ber09] beschreiben, wie Daten am besten als *Linked Open Data* veröffentlicht werden sollten:

1. Verwendung von URIs zur Identifikation
2. Verwendung von HTTP URIs, sodass die Namen zur Identifikation nachgeschlagen werden können

3. Eine URI sollte auf sinnvolle Informationen verweisen, die den Standards RDF oder SPARQL Protocol and RDF Query Language (SPARQL) folgen

4. Verweise auf andere URIs, damit weitere Daten gefunden werden können

Die Verwendung dieser Prinzipien hat sich als bewährte Vorgehen zur Veröffentlichung neuer Daten im *Linked Open Data* etabliert [Den12]. Die Prinzipien fordern dazu auf, das Finden und Verarbeitung der Daten möglichst einfach zu halten (Punkt 1-3) und existierende Ressourcen wiederzuverwenden (4. Punkt).

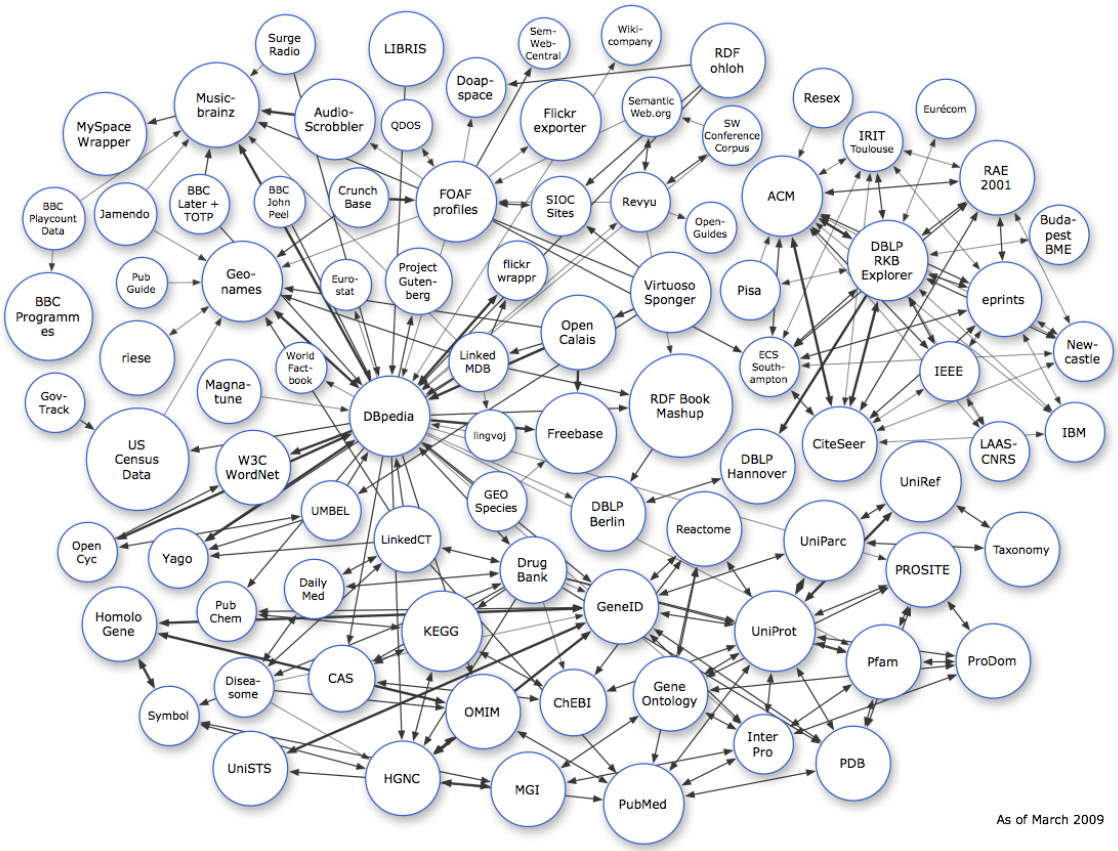


Abbildung 2.9: Visualisierung der *Linked Open Data Cloud* (Stand März 2009) von `http://lod-cloud.net/`. Hierbei sind die verschiedenen Namensräume jeweils als Knoten repräsentiert. Durch Kanten wird angegeben welche anderen Namensräume sie referenzieren. Es ist keine aktuelle Version der *Linked Open Data Cloud* gezeigt, da für diese die einzelnen Beschriftungen aufgrund der großen Menge an Knoten und Kanten an dieser Stelle nicht gut lesbar darzustellen ist.

3 Mustersprachen als Webontologien

In diesem Kapitel wird mit OWL eine Ontologie für Mustersprachen und Entwurfsmuster eingeführt. Dazu werden zuerst die speziellen Anforderungen der Domäne an die Ontologie erörtert. Daraufhin wird die erstellte Ontologie beschrieben und das dafür eingeführte Vokabular aufgeführt. Am Ende wird mittels ausgewählter Instanzen gezeigt, wie die Ontologie um weitere Mustersprachen, Relationen und Muster erweitert werden kann und so als Datengrundlage eines Musterrepositories genutzt werden kann.

3.1 Anforderungen an die Ontologie

Zuerst sollen in der folgenden Aufzählung die Anforderungen kurz aufgeführt werden, die sich für die zu erstellende Ontologie ergeben. Hierbei soll diese die im Grundlagenkapitel aufgeführten grundlegenden Eigenschaften von Mustersprachen und Entwurfsmustern widerspiegeln.

1. Es soll ein **Vokabular** für die Konzepte von Mustersprachen, -sichten, Muster sowie die Relationen zwischen diesen eingeführt werden. Dieses soll durch die Veröffentlichung in einem eigenen Namensraum von anderen, separat entwickelten Ontologien genutzt werden können. Das Vokabular soll auf RDF aufbauen und daher URIs für die darin beschriebenen Konzepte nutzen. Außerdem sollen für konkrete Mustersprachen, -sichten, Muster und Relationen ebenfalls URIs eingeführt werden, um diese referenzieren zu können. Hierdurch soll Anwendungen ermöglicht werden, die Struktur und Semantik der Daten über Muster und Mustersprachen zu verarbeiten. Autoren können dann auf diesem Vokabular aufbauend Mustersprachen veröffentlichen, beispielsweise ergänzend zur Autorenwebseite ihrer Mustersprache. Besonders zu betonen ist, dass auch Beziehungen zu Mustern aus anderen Mustersprachen beschrieben werden können, da die URI des Musters es erlaubt, diese über Dokumente hinweg zu referenzieren. Wissen über Mustersprachen kann somit dezentral erweitert werden und frei verfügbar als *Linked Open Data* zur Verfügung gestellt werden.
2. In Abschnitt 2.1 wurde beschrieben, dass Mustersprachen je nach Domäne andere Abschnitte und Struktur für ihre Entwurfsmuster definieren. Die **Abschnitte eines Musters sind daher nicht fest vorzugeben**. Eine Ontologie für Mustersprachen soll flexibel genug gehalten werden, sodass Autoren von Mustersprachen diese frei wählen können. Hiermit wird der Empfehlung von Köppe et al. [KISV16] gefolgt. Konkret bedeutet dies, dass Autoren von Mustersprachen die Reihenfolge und Titel der Abschnitte definieren. Zusätzlich soll es für Abschnitte möglich sein, einen Datentyp anzugeben. Aufbauend auf den Angaben zum Datentyp ist es dann möglich, den Nutzer bei der Erstellung eines Musters zu unterstützen und ein passendes Eingabeformat vorzuschlagen. Zusätzlich kann überprüft werden, ob es sich um den richtigen Datentyp handelt. Ebenfalls soll eingeschränkt werden können, dass

für Abschnitte nur begrenzt viele Werte möglich sind. Hierdurch kann bei der Eingabe eines Musters validiert werden, dass für ein Muster z.B. nur ein Icon angegeben werden kann, nicht mehrere.

3. Die zwei **Grundbestandteile von Mustersprachen sind** nach der Formalisierung von Falkenthal et al. [FBL18] **Entwurfsmuster und Relationen** zwischen diesen. Muster beschreiben den Kontext und Lösungen für wiederkehrende Probleme. Relationen beschreiben Zusammenhänge zwischen Mustern und sollen wie möglichst detailliert repräsentiert werden, da diese im ersten Kapitel als wertvolle Wissensbasis für semantische Agenten motiviert wurden.

4. Für eine **Relation** zwischen Mustern empfiehlt Köppe et al. [KISV16] die Angabe folgender Eigenschaften: ***relationType*** (Typ der Relation), ***direction*** (Richtung der Relation), ***additional information*** (zusätzliche Informationen). Falkenthal et al. [FBL18] beschreiben in ihrer Repräsentation als Graph ebenfalls die von Köppe et al. [KISV16] genannten Eigenschaften, führen aber die zusätzlichen Informationen genauer aus: Abhängig vom spezifizierten Typ können **weitere, strukturierte Informationen** für die Relation angegeben werden. Beispielsweise können für den Relationstyp "can be combined with" weitere Informationen über die konkrete Umsetzung der Kombination der Muster angegeben werden [FBL18]. Durch diese Informationen wird der Zusammenhang der Muster möglichst genau dargestellt. Zur Richtung der Relation soll außerdem die Möglichkeit bestehen, dafür einen kommutativen Relationstyp zu benutzen. Ein Beispiel für einen kommutativer Linktyp, ist *joint appearance* in der Kostümmustersprache von Barzen und Leymann [BL15]. Um eine möglichst große Wiederverwendung von Linktypen zu erreichen, sollen in diesem Kapitel häufig benutzte Relationstypen identifiziert und ebenfalls als Vokabular bereitgestellt werden.

5. Die **Gruppierung von Mustern beziehungsweise das Zuweisen zu Kategorien** wird von Köppe et al. [KISV16] als typische Funktionalität eines Musterrepository beschrieben. in Abschnitt 2.1 wurden bereits Beispiele dafür genannt, dass Mustersprachen Gruppen bzw. Kategorien zugewiesen werden. Dies soll deshalb in der Ontologie für Mustersprachen ebenfalls möglich sein. Um für Mustersprachen verwandter Themengebiete die Gemeinsamkeiten von verschiedenen Mustern hervorheben zu können, soll eine Gruppierung von Mustern auch über Mustersprachen hinweg durch eine Mustersicht ermöglicht werden. Dadurch kann ein Musterrepository das auf diesen Daten aufbaut, Mitglieder einer bestimmten Gruppe (z.B. farblich) in einer Übersicht hervorheben und Verbindungen zu Muster desselben Themengebiet aufzeigen.

6. Um Mustern aus verschiedenen Mustersprachen oder eine Teilmenge einer Mustersprache genauer zu beschreiben, soll es möglich sein Mustersichten zu nutzen. Innerhalb einer **Mustersicht** können Muster und Links aus Mustersprachen referenziert werden, ohne diese zu duplizieren. Wie in der Motivation für Mustersichten in Abschnitt 2.1.1 beschrieben, können hiermit zwei Mustersprachen zu einer größeren Sprache zusammengefasst werden und Links zwischen Mustern der ursprünglich getrennten Sprachen eingepflegt werden. An dieser Stelle soll das Konzept Mustersichten klar von der Verwendung des Begriffs *Pattern view* von Köppe et al. [KISV16] abgegrenzt werden: Im Gegensatz zur *Pattern view* von Köppe et al. [KISV16] beinhaltet eine Mustersicht keine Informationen zur Visualisierung von Mustern, sondern strukturierte Informationen zu den referenzierten Mustern und Relationen.

7. Die Modellierung der Ontologien soll mit **OWL DL** erfolgen, da dieser Dialekt entscheidbar ist [Den12]. Dies befähigt Anwendungen, die auf der Wissensbasis aufbauen, Schlussfolgerungen zu ziehen ohne auf unentscheidbare Fragestellungen Rücksicht zu nehmen. Für ein Muster kann beispielweise ermittelt werden, welche anderen Muster durch Relationen (direkt oder indirekt) mit diesem verbunden sind. Werden dabei nur Relationen betrachtet, die angeben dass Muster zusammen verwendet werden können, so lassen sich hierüber weitere Vorschläge für die Anwendung von Mustern ermitteln.

3.2 Identifikation typischer Relationstypen

Wie in der vierten Anforderung an die Ontologie aufgeführt, soll für Relationen zwischen Mustern auch ein Typ angegeben werden. Falkenthal et al. [FBL18] beschreiben, dass Mustersprachen abhängig von ihrer Domäne die Verbindungen zwischen ihren Mustern mit einer Menge von Relationstypen beschreiben (*domain-specific types*). Der Typ charakterisiert die Semantik des Links durch einen kurzen Beschreibungstext: Beispielsweise nennen die Autoren den Relationstypen ”contains“ der beschreibt, dass ein Muster in einem anderen Muster enthalten ist. Die Zuweisung eines Relationstypen ermöglicht Anwendungen, Mustergraphen anhand der Semantik der Relationen zu traversierten [FBL18] und z.B. von einem Muster ausgehend sinnvolle Verbindungen zu anderen Mustern zu finden. Um Musterautoren die Annotation von Relationstypen zu erleichtern, sollen Relationstypen, die von vielen Domänen verwendet werden, im Standardvokabular bereitgestellt werden. Diese können dem Endnutzer dann bei der Erstellung einer Relation vorgeschlagen werden. Im Folgenden werden daher von Musterautoren oft verwendete Linktypen beschrieben und Kategorien zugeordnet. Für jede Kategorie erfolgt dann abschließend die Empfehlung einer oder mehreren Relationstypen für das Standardvokabular. Um den Linktyp als *fragment identifier* in URIs verwenden zu können, werden Leerzeichen im Relationstyp vermieden. Ziel des Standardvokabular an Relationstypen ist, möglichst viele Anwendungsfälle abdecken und eine Basis für Relationstypen zu bilden, die spezifisch für Mustersprachen erweitert werden kann.

3.2.1 Undefinierter Relationstyp

Da Relationstypen in bisherigen Veröffentlichungen nicht standardisiert sind, finden sich oft auch Musterrelationen ohne Angabe eines konkreten Linktypen. Hohpe und Woolf [HW04] geben auf ihrer Webseite zu ihrer Mustersprache (`https://www.enterpriseintegrationpatterns.com/`) für ein Muster jeweils damit zusammenhängende Muster (*Related Patterns*) an. Weitere Ausführungen dazu, wie diese Muster mit dem aktuell dargestellten Muster in Zusammenhang stehen, fehlen oder müssen dem Fließtext entnommen werden. Eine weitere Formulierung, die ebenfalls keine Rückschlüsse über den Typ der Relation zulässt, ist ”siehe auch” (in Mustersprachdarstellung von Falkenthal et al. [FBL18] enthalten, siehe Abb. 2.2). Um schon bestehende Mustersprachen besser integrieren zu können, soll ein undefinierter Relationstyp ins Vokabular aufgenommen werden.

Relationstypen für das Standardvokabular: *isRelatedTo*

3.2.2 Reihenfolge und Abhängigkeiten

Für einige Kombinationen von Mustern ist es wichtig zu beachten, in welcher Reihenfolge diese angewandt werden [Zdu07]. Zdun [Zdu07] beschreibt wie eine formale Grammatik für eine Mustersprache erstellt werden kann, anhand der gültige Mustersequenzen abgeleitet werden können. Die Reihenfolge der Muster in der Mustersequenz geben eine zeitliche Einordnung darüber an, wann die Muster ausgewählt wurden [Zdu07]: Das erste Muster der Sequenz wird vor dem zweiten ausgewählt, das zweite vor dem dritten etc. Sie beschreiben weiter, dass diese Grammatiken zu Musterbeziehungen abgebildet werden können. In den *Assessment and feedback* Mustern von Warburton et al. [WBK+16] spezifizieren die Autoren der Mustersprache ebensolche Beziehungen mit den Relationstypen ”is used before“ und ”leads to“.

Relationstypen für das Standardvokabular: *isUsedBefore*, *isUsedAfter*, *dependsOn*

3.2.3 Kompatibilität

Eine Information darüber, ob Muster miteinander kompatibel sind, unterstützt die Auswahl von gültigen Musterkombinationen. Fehling et al. [FLR+11] geben für ihre Muster jeweils an, ob diese ”kohärent“ zueinander sind und daher zusammen verwendet werden können. Zimmer [Zim95] identifiziert ”X can be combined with Y“ als Oberkategorie für Beziehungen zwischen Mustern zu objektorientierten Softwaredesign von Gamma et al.

Relationstypen für das Standardvokabular: *canBeUsedWith*, *cannotBeUsedWith*

3.2.4 Verwendung des Musters

Schon Alexander et al. [AIS+77] beschreibt die Tatsache, dass Muster in anderen Mustern verwendet werden. Hirmer und Mitschang [HM16] beschreibt mit dem Relationstyp ”consistsOf“ wie Muster aus anderen Mustern zusammengesetzt werden können. Ein weiteres Beispiel findet sich in einem Übersichtsdiagramm von Fernandez-Buglioni [Fer13]: Dieses zeigt einen Link mit der Beschriftung ”contains“ zwischen zwei ihrer Muster. Zimmer [Zim95] findet in den Mustern von Gamma et al. mehrere Beispiele, sodass er ’X uses Y in its solution“ als Kategorie von Linktypen aufführt.

Relationstypen für Standardvokabular: *consistsOf*, *uses*, *usedIn*

3.2.5 Ähnlichkeit

Um das am besten geeignete Muster auf ein Problem anzuwenden, bietet es sich an ähnliche Muster ebenfalls in Betracht zu ziehen. [Zdu07] beschreiben wie die in Abschnitt 3.2.2 erwähnte Grammatik einer Mustersprachen zu Musterrelationen wie ”variants“ und ”alternative“ abgebildet werden können, womit sie diese implizit als Relationtypen aufführen. Zimmer [Zim95] weist einige Relationen der Muster von Gamma et al. der Kategorie ”X is similar to Y“ zu. Fernandez-Buglioni [Fer13] und Fehling et al. [FLR+14] heben die Relationen dieses Typ explizit hervor, indem sie für ihre Muster Varianten in einem eigenen Abschnitt auflisten.

Relationstypen für das Standardvokabular: *isAlternativeTo*, *isVariationOf*

Die aufgeführten Relationstypen, die ins Standardvokabular aufgenommen werden bilden die Basis für die genauere Beschreibung von Links und können bei Bedarf noch erweitert werden. Spezielle Linktypen, die in einem spezifischen Domänenkontext verwendet vorkommen wie z.B. *opponent* in der Kostümmustersprache von Barzen und Leymann [BL15] sollen innerhalb der Mustersprache definiert werden können.

3.3 Einführung eines Vokabulars für die Beschreibung von Mustern und Mustersprachen

Im folgenden Abschnitt wird die durch OWL erstellte Ontologie der Domäne (Muster und Mustersprachen) mit einer kurzen Beschreibung vorgestellt. Das damit definierte Vokabular ist wie in der ersten Anforderung verlangt unter einem eigenen Namensraum bereitgestellt, `https://purl.org/patternpedia`. Dort finden sich im Turtle-Format die genauen Einschränkungen und Spezifikationen für die unten gelisteten konzeptionellen Einheiten. In den folgenden drei Abschnitten folgt eine kurze Darstellung des Vokabulars, sowie falls benötigt eine kurze Begründung. Die einzelnen Terme sind dafür mit der Präfix-Notation von Turtle angegeben, die ein vorangehendes ":" durch das dafür definierte Präfix ersetzt. Für die folgenden Terme ist das Präfix ":" dementsprechend mit `https://purl.org/patternpedia#` zu ersetzen um die komplette URI zu erhalten. Die deutsche Übersetzung zum Fachbegriff ist jeweils in Klammern angegeben.

3.3.1 Klassen

Die in den Anforderungen beschriebenen Konzepte werden wie für OWL üblich als Klassen modelliert. Hiermit werden noch keine konkreten Instanzen erstellt da eine Klasse in OWL DL niemals eine Instanz sein kann [Yu14]). Es können jedoch pro Klasse Einschränkungen für Instanzen dieser Klasse festgelegt werden. Mittels Klassenhierarchien werden die Gemeinsamkeiten wie zum Beispiel die der drei Linkarten ausgedrückt. Eine Übersicht aller Klassen ist in Abbildung 3.1 dargestellt.

a) **:Pattern (Muster)**

   Mit dieser Klasse können wir eine Ressource den Typ Muster zuweisen. Die URI der Ressource identifiziert dann das durch weitere Eigenschaften beschriebene Muster. Für alle Musterinstanzen muss ein Name angegeben sein; weitere Eigenschaften sind nicht vorgegeben, da sich diese je nach Mustersprache unterscheiden.

b) **:PatternGraph (Mustergraph)**

   Ein Mustergraph besteht aus Mustern und Relationen zwischen diesen. Wie der Name schon andeutet, wird dies durch einen Graphen realisiert. Für einen Mustergraphen muss ein Name angegeben werden, optional kann dazu noch ein Logo angegeben werden. Die Klassen Mustersprache und -sicht erben von dieser Klasse und können daher auch Muster und Relationen beinhalten.

c) :**PatternLanguage (Mustersprache)**

Mustersprache ist eine Unterklasse von Mustergraph, die eine Mustersprache repräsentiert. Für ein Instanz vom Typ Mustergraph können URIs verwendet werden, um Zugehörigkeit von Mustern auszudrücken: Im Namenshashraum einer Mustersprache können Muster angelegt werden. Beispielsweise kann über die URI `URI_Mustersprachen_Instanz#PatternA` das Muster *PatternA* angelegt und referenziert werden.

d) **:PatternView (Mustersicht)**

Diese Klasse ist ebenfalls eine der Unterklassen von Mustergraph. Eine Mustersicht repräsentiert eine Sicht auf Mustersprachen, die Muster aus potentiell verschiedenen Mustersprachen und Relationen zwischen den referenzierten Mustern enthält. Wie in Abschnitt 2.1.1 ausgeführt, können mithilfe einer Mustersicht Mustersprachen flexibel beschrieben werden. Da Relationen zwischen allen beinhalteten Mustern Teil einer Mustersicht sein können, lassen hier insbesondere auch mustersprachenübergreifende Relationen erstellen.

e) **:PatternPedia (Musterrepository)**

Eine Instanz der Klasse PatternPedia enthält Mustergraphen, einen Namen und ein Logo. Sie repräsentiert somit eine Sammlung von Mustersprachen und Mustergraphen, ein Musterrepository.

f) **:PatternRelationDescriptor (Relationsbeschreibung)**

Mit dieser abstrakten Klasse werden Beschreibungen von Musterrelationen zusammengefasst. Relationen zwischen Mustern sind bewusst nicht nur als Eigenschaften modelliert, da mit der Modellierung als Klasse weitere Eigenschaften zu einer Relation angegeben werden können. Wie im Abschnitt 3.2 beschrieben, soll ein Relationstyp angegeben werden. Optional kann ein Beschreibungstext spezifiziert werden. Außerdem bietet die Modellierung als Klasse den Vorteil, dass wir weitere Unterklassen erstellen können.

g) **:DirectedPatternRelationDescriptor (Beschreibung einer gerichteten Musterrelation)**

Diese Klasse beschreibt eine gerichtete Relation zwischen zwei Mustern, die mit den Eigenschaften **:hasSource** und **:hasTarget** angegeben werden. Im Mustergraphen entspricht diese Relation einer gerichteten Kante zwischen zwei Knoten (Mustern).

h) **:UndirectedPatternRelationDescriptor (Beschreibung einer ungerichteten Musterrelation)**

Mithilfe dieser Klasse kann eine ungerichtete Relation zwischen genau zwei Mustern beschrieben werden. Die Angabe, welche Muster durch eine ungerichtete Relation verbunden sind, erfolgt durch die Eigenschaft **:hasPattern**. Im Mustergraphen entspricht diese Relation einer ungerichteten Kante.

i) **:PatternSetRelationDescriptor (Beschreibung einer Relation einer Menge von Mustern)**

Diese Klasse beschreibt eine Relation, mithilfe der einer Menge von Mustern eine Bezeichnung und eine Beschreibung zugewiesen werden. Diese Klasse wird dazu verwendet, Muster einer Gruppe zuzuweisen wie in Anforderung 5 gefordert. Die Art von Links die sie repräsentiert, in Abb. 2.2 als grüne Kanten in einem Mustergraphen dargestellt.

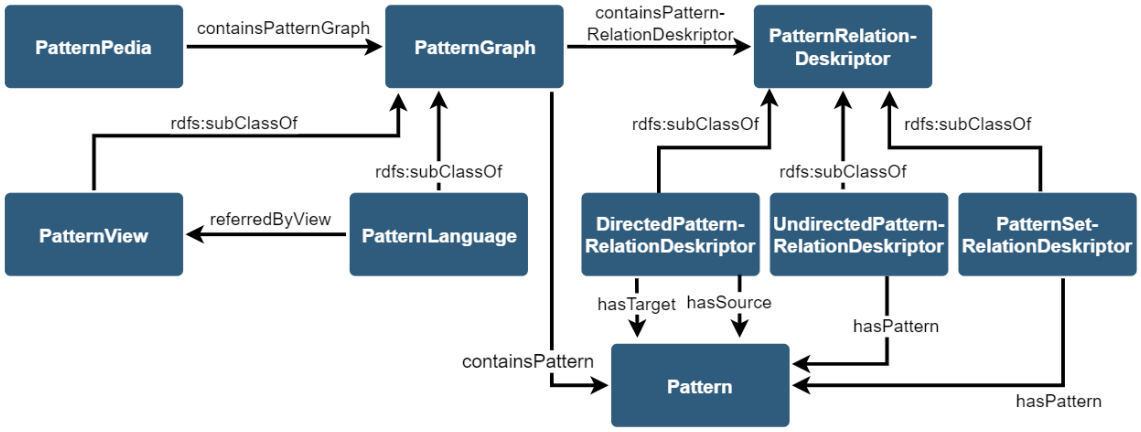


Abbildung 3.1: Darstellung der Klassen und Objekteigenschaften der modellierten Ontologie, die es ermöglichen Instanzen von Klassen über Eigenschaften zu verbinden. Zur Übersicht sind für die hier gezeigten Klassen zur Relationsbeschreibung keine Unterklassen angegeben.

j) Die in Abschnitt 3.2 erarbeiteten Linktypen stehen ebenfalls als Unterklassen der Beschreibung einer gerichteten Relation (**:DirectedPatternRelationDescriptor**) zur Verfügung. Für den Relationstyp ist jeweils der vorgeschlagene Name eingetragen.

k) **:DatatypePropertyListItem**

Diese Klasse erbt von der Klasse Datentypeigenschaft (*owl:DatatypeProperty*), für die einen Listenindex angegeben werden muss. Wird verwendet, um die Reihenfolge von Sektionen in einem Entwurfsmuster festzulegen. Hiermit wird auf zweite Anforderung für die Ontologie realisiert, dass der Aufbau und die Eigenschaften eines Musters individuell pro Mustersprache festgelegt werden sollen.

3.3.2 Eigenschaften für Objekte

Im Folgenden werden die Eigenschaften des bereitgestellten Vokabulars aufgezählt. Zur besseren Übersicht sind in Abb. 3.1 die Klassen und Objekteigenschaften angelehnt an die Darstellung der Ontologie in Kapitel 2, Abb.2.5 dargestellt. Durch die aufgezählten Objekteigenschaften kann man Instanzen verschiedener Klassen miteinander verbinden.

a) **:containsPattern (enthält Muster)**

Mit dieser Eigenschaft kann für einen Mustergraphen (realisiert durch eine Mustersprache oder Mustersicht) ausgedrückt werden, dass dieser ein Muster enthält. Die über diese Eigenschaft zugewiesenen Muster entsprechen den Knoten des Mustergraphs (siehe Abb. 2.1 für eine graphische Repräsentation).

b) **:containsPatternRelationDescriptor (enthält Musterrelation)**

Mit dieser Eigenschaft kann für eine Mustersprache ausgedrückt werden, dass diese eine Relation zwischen Mustern enthält. Die über diese Eigenschaft einer Mustersprache zugewiesenen Relationsbeschreibung entsprechen den Kanten des Mustergraphen (vgl. Abb. 2.1).

c) **:containsPatternGraph (enthält Mustergraphen)**

Hiermit kann ausgedrückt werden, dass eine Instanz der Klasse Musterrepository einen Mustergraphen enthält und somit der Mustergraph Teil dieses Musterrepositories ist.

d) **:hasPattern (besitzt Muster)**

Für eine ungerichtete Relation zwischen Mustern kann mit dieser Eigenschaft angegeben werden, welche beiden Mustern sie verbindet. Hiermit werden die Knoten des Mustergraphen angegeben, die über eine ungerichtete Kante verbunden sind.

e) **:hasTarget (besitzt Ziel)**

Hiermit kann für eine gerichtete Relation zwischen Mustern angegeben werden, auf welches Muster die Relation zeigt. Über diese Eigenschaft wird ausgedrückt, auf welches Muster die entsprechende gerichtete Kante im Mustergraphen zeigt (vgl. die Pfeilspitzen in Abb. 2.1).

f) **:hasSource (besitzt Start)**

Für eine gerichtete Relation zwischen Mustern kann mit dieser Eigenschaft angegeben werden, von welchem Muster die Relation ausgeht. Im Mustergraphen entspricht das Muster, das über diese Eigenschaft angegeben ist, dem Ausgangsknoten der gerichteten Kante.

g) **:referredByView (durch Sicht referenziert)**

Hiermit kann ausgedrückt werden, dass es eine Mustersicht gibt, die Muster oder Relationen dieser Mustersprache referenziert. Anhand dieser Information können mustersprachenübergreifende Links für die Mustersprache in Mustersichten gefunden werden.

3.3.3 Datentyp-Eigenschaft

Zusätzlich zu Objekteigenschaften wurden noch Eigenschaften definiert, mit denen man Instanzen mit Literalen verbinden kann. Die Eigenschaften werden über Mustersprachen/-sichten hinweg verwendet, und sind deswegen ebenfalls an zentraler Stelle bereitgestellt.

a) **:hasLogo (besitzt Logo)**

Gibt ein Logo an, das für Darstellung einer Instanz verwendet werden kann. Für Mustersprachen ist es hier beispielsweise denkbar, auf das Buchcover einer Veröffentlichung zu verweisen.

b) **:hasName (besitzt Namen)**

Gibt einen Namen an z.B. für Mustersprachen und Muster. Dieser kann im Gegensatz zu URIs auch Sonderzeichen (inklusive Leerzeichen und Umbrüchen) enthalten.

c) **:hasDescription (besitzt Beschreibung)**

Diese Eigenschaft kann verwendet werden, um eine zusätzliche Beschreibung hinzuzufügen. Sie kann für Musterrelationen verwendet werden, um weitere Informationen zur Relation anzugeben wie Köppe et al. [KISV16] und Falkenthal et al. [FBL18] es für Relationen vorschlagen.

d) **:hasLabel (besitzt Bezeichnung)**

Gibt eine Bezeichnung an, z.B. für eine Relation die Muster einer Gruppe zuweist. Mittels der Eigenschaft kann die Bezeichnung der Gruppe angegeben werden. Diese kann im Gegensatz zu URIs auch Sonderzeichen (inklusive Leerzeichen und Umbrüche) enthalten.

e) **:hasListIndex (besitzt Listenindex)**

Gibt für die Datentyp-Eigenschaft einen Index an. Dieser kann dazu benutzt werden, eine Reihenfolge festzulegen, beispielsweise für die Abschnitte eines Musters.

3.4 Erstellung von Instanzen zum Aufbau und Erweiterung eines Musterrepositories

In Abb. 3.2 wird eine Auswahl von Instanzen dargestellt, die die Struktur der Ontologie nochmals verdeutlicht. Anhand dieser Instanzen wird in den folgenden Abschnitten erklärt, wie ein Musterrepository mithilfe der Ontologie angelegt werden kann. Außerdem wird gezeigt, wie dieses um Mustersprachen, -sichten, Muster und Relationen ergänzt werden kann. Zur Übersicht wurden in der Abbildung die Ebenen, in der sich die Instanzen anordnen lassen farblich markiert: Ebene 1 besitzt daher einen weißen, Ebene 2 einen hellblauen Hintergrund, etc. Für die einzelnen Ebenen wird die Semantik der Instanzen kurz erklärt. Die Klassen der Instanzen sind jeweils am oberen Rand angegeben. In dem gezeigten Minimalbeispiel eines Musterrepositories wird die konkrete Verwendung aller drei Arten von Links, sowie der grundlegenden Klassen dargestellt.

3.4.1 Ebene 1: Musterrepository Instanz

Eine Instanz der Klasse Musterrepository (*PatternPedia*) enthält durch die Eigenschaft *containsPatternGraph* zwei verschiedene Arten von Mustergraphen: Mustersprachen und Mustersichten. Im vereinfachten Beispiel sind die Sprachen *CloudComputingPatterns* von Fehling et al. [FLR+14] und *EnterpriseIntegrationPatterns* von Hohpe und Woolf [HW04] enthalten. Beide Mustersprachen werden von einer Mustersicht referenziert, die ebenfalls in der Instanz von PatternPedia enthalten ist. Die Instanz der Klasse PatternPedia beinhaltet somit eine Sammlung von Mustersprachen und -sichten, und bildet so ein Musterrepository ab. Durch das Hinzufügen von weiteren Mustersichten und Mustergraphen durch die Eigenschaft *containsPatternGraph* kann das Musterrepository weiter um Inhalte ergänzt werden.

3.4.2 Ebene 2: Mustergraph Instanzen

Die Mustersprachen der zweiten Ebene beinhalten Entwurfsmuster der vierten, sowie Links der dritten Ebene. Im Beispiel sind drei Muster der *CloudComputingPatterns*, sowie ein Muster der *EnterpriseIntegrationPatterns* gezeigt. Die *CloudComputingPatterns* Mustersprache enthält außerdem noch eine gerichtete Relation zwischen den *ElasticPlatform* und *Message-orientedMiddleware* Muster. Die Muster stammen aus dem Hash-Namensraum der Mustersprache, was daran zu erkennen ist, dass sie mit der URI der Mustersprache vor dem *fragment identifier* # beginnen. Für Mustersprachen muss berücksichtigt werden, dass diese die Abschnitte ihrer Muster vorgeben.

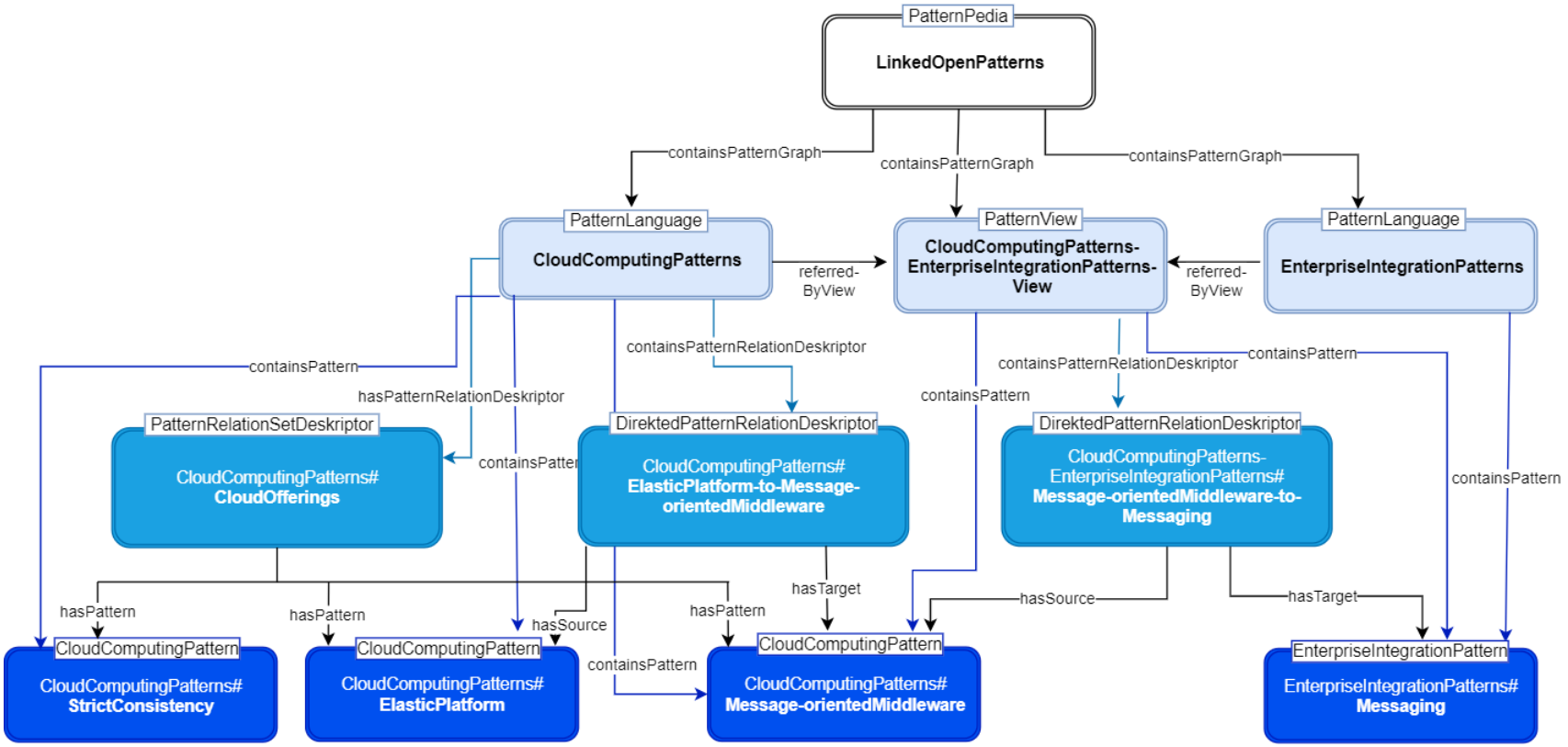


Abbildung 3.2: Auswahl von einigen erstellten Instanzen und Eigenschaften für die in den vorherigen Abschnitten beschriebenen Ontologie. Die Klasse der Instanz ist jeweils am oberen Rand angegeben. Die URIs der Instanzen sind verkürzt dargestellt, hier ist nur das letzte Segment aufgeführt. Zur besseren graphischen Darstellung sind die Instanzen der selben Ebene gleich eingefärbt. Alle drei Relationsarten sind zu erkennen: die Zuweisung zur Gruppe *CloudOfferings* mittels eines PatternRelationSetDeskriptor, eine mustersprachenübergreifenden Relation und eine Relation innerhalb einer Mustersprache.

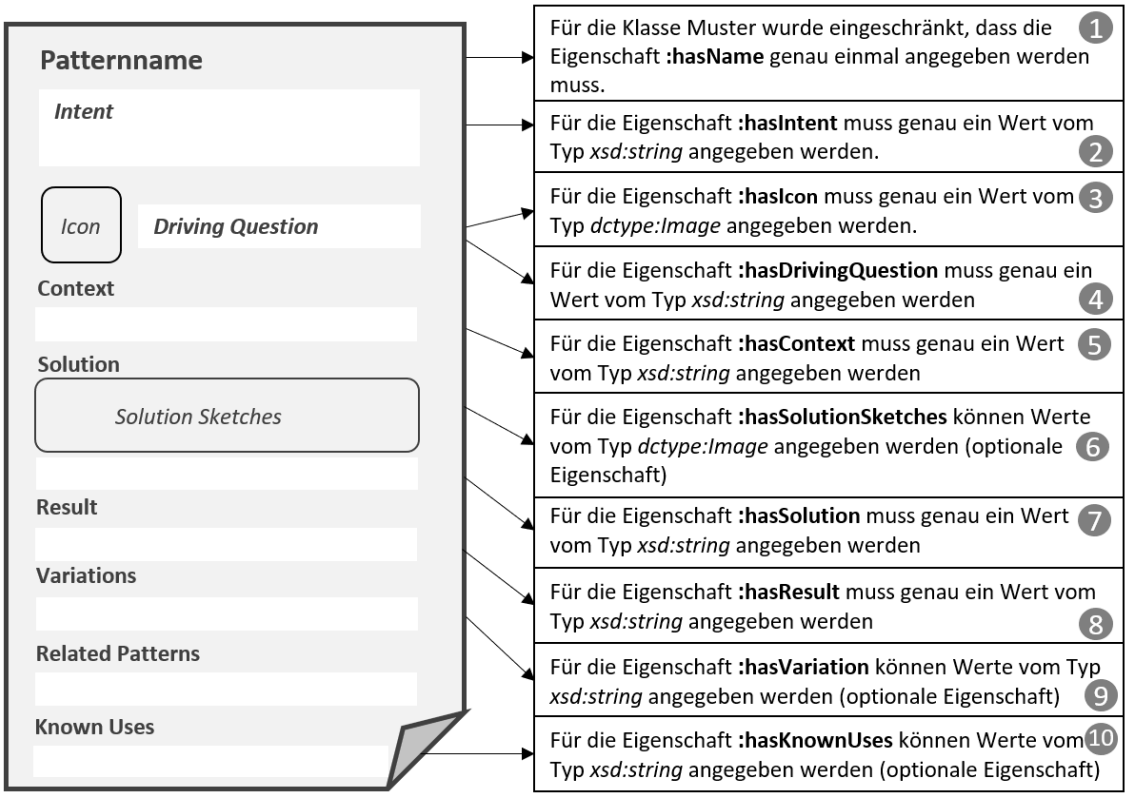


Abbildung 3.3: Aufbau eines *Cloud Computing* Musters von Fehling et al. [FLR+14] (links) und sich daraus ergebende Einschränkungen für ein Muster der Mustersprache (rechts).

Für die *CloudComputingPatterns* Mustersprache ist dies in Abb. 3.3 links gezeigt. Anhand der Abschnitte lassen sich für ein Muster der Sprache Eigenschaften und Einschränkungen ableiten. Diese Einschränkungen sind in Abb. 3.3 rechts als nummerierte Sätze angegeben, wobei abgeleitete Eigenschaften für Abschnitte fett hervorgehoben wurden. Anhand der *CloudComputing* Mustersprache wird beschrieben, wie diese Vorgaben aus dem rechten Teil der Abbildung als Einschränkungen in OWL umsetzt werden:

- Die Eigenschaften des Musters werden als Datentypeigenschaften definiert, womit auch eine URI für jede Eigenschaft eingerichtet wird. Der Name der Eigenschaft ist an den Namen des Abschnitts angelehnt, hierbei wurden lediglich Leerzeichen vermieden, um den Namen als *fragment identifier* verwenden zu können. Diese Eigenschaften können dann für ein Muster der Sprache verwendet werden, um die Werte für die Abschnitte, z.B. eine Zeichenkette für die Beschreibung des Abschnitts *Context* anzugeben.
- Eine neue Klasse *CloudComputingPattern* wird definiert, welche von der Klasse Muster erbt und diese um die Einschränkungen ergänzt, sodass das Musterformat der Mustersprache umgesetzt wird. Da für die Klasse Muster bereits eingeschränkt ist, dass die Eigenschaft **:hasName** genau einmal angegeben werden soll, ist hiermit der erste Satz der rechten Seite von Abb. 3.3 schon umgesetzt.

- Zuletzt werden die Einschränkungen für die Klasse *CloudComputingPattern* formuliert und somit die restlichen Sätze aus dem rechten Teil der Anwendung umgesetzt: Für jede Eigenschaft kann mittels OWL eine Einschränkung formuliert werden, die besagt:

  1) welcher Datentyp für Werte dieser Eigenschaft verwendet werden kann bzw. soll

     sowie

  2) wie oft diese gesetzt werden soll bzw. kann (Kardinalität).

  Satz 8 im rechten Teil der Abbildung kann somit durch die folgende Einschränkung umgesetzt werden: Für eine Instanz der Klasse *CloudComputingPattern* muss die Eigenschaft **:hasDrivingQuestion** genau einmal angegeben werden, der Wert hierfür soll dem Datentyp *xsd:string* entsprechen. Für diese Mustersprache wurden Datentypen verwendet, die unter `http://www.w3.org/2001/XMLSchema#` (Präfix *xsd:* in der Abbildung) bzw. `https://purl.org/dc/dcmitype/` (Präfix *dctype:* in der Abbildung) spezifiziert sind. *xsd:string* verweist somit in Turtle-Notation auf die URI `http://www.w3.org/2001/XMLSchema#string`, womit der Datentyp Zeichenkette angegeben wird. Unter `http://www.w3.org/2001/XMLSchema#` sind von der W3C vorgegebene Standarddatentypen für RDF bereitgestellt, die für solche Einschränkungen verwendet werden können. Der Datentyp *xsd:string* soll für mehrere Abschnitte verwendet werden (Sätze 1, 2, 4, 5, 7-10). Es ist jedoch auch möglich, weitere Datentypen zu definieren und zu verwenden; unter dem Namensraum `https://purl.org/dc/dcmitype/` wird dies beispielsweise realisiert. Hier werden Datentypen wie Text, Software und Image eingeführt. In der Abbildung wurde *dctype:Image* als Datentyp für die Abschnitte **:hasIcon** und **:hasSolutionSketches** spezifiziert (Sätze 3 und 6). Die Angabe der Kardinalität einer Einschränkung kann benutzt werden, um zu fordern, dass Werte für Abschnitte mindestens/maximal oder genau *n*-mal angegeben werden müssen. Es kann stattdessen auch lediglich gefordert werden, dass alle Werte einen bestimmten Datentyp aufweisen, womit diese Eigenschaft optional bleibt. Für die Eigenschaften **:hasVariation** und **:hasKnownUses** wurde eine solche Einschränkung verwendet um auszudrücken, dass nur der Datentyp *xsd:string* verwendet werden soll (Sätze 9 und 10). Beide Eigenschaften, sowie **:hasSolutionSketches** sind somit optionale Eigenschaften, da sie für ein *CloudComputing* Muster nicht angegeben werden müssen. Für alle anderen Eigenschaften wurde als Kardinalität 1 angegeben, somit müssen diese genau einmal angegeben werden, wie in den Sätzen 2-5 und 7 gefordert war.

Durch die Festlegung der Einschränkungen für die Klasse *CloudComputingPattern* wurde das Musterformat der Sprache festgelegt und somit vorgegeben, welche Werte für die Abschnitte eines Musters angegeben werden können und müssen. Für den Abschnitt *Related Patterns* wurden keine Eigenschaft definiert, da referenzierte Muster in den Relationsbeschreibungen der Mustersprache enthalten sind. Sie müssen dadurch nicht nochmals extra aufgeführt werden. Der Inhalt des Abschnitt *Variations* hätte ebenfalls auch durch die Angabe von Musterrelationen ersetzt werden können, hier wurde aber entschieden die Variationen des Musters explizit als Text aufzuführen. Welche Varianten es für ein Muster gibt, kann auch zusätzlich durch Relationen angegeben werden. Die *EnterpriseIntegration* Mustersprache gibt ein ähnliches Format mit den Abschnitten *Patternname, Icon, Context, Problem, Forces, Solution, Sketch, Results, Next, Sidebars, Examples* für ihre Muster vor. Hier kann wie für die *CloudComputing* Mustersprache vorgegangen werden, um Einschränkungen für die Klasse *EnterpriseIntegrationPatttern* zu formulieren. Das Musterformat und die sich hieraus ableitenden Vorgaben für Einschränkungen sind der Vollständigkeit halber im Anhang auf S. 65 aufgeführt.

Mustersichten beinhalten Muster, die aus verschiedenen Mustersprachen referenziert werden können. Genau wie Mustersprachen beinhalten Mustersichten ebenfalls Links zwischen allen Mustern die im Mustergraph beinhaltet sind. Dies können für Mustersichten daher auch mustersprachenübergreifende Links sein, wie die Relation von *Message-orientedMiddleware* zum *Messaging* Muster zeigt.

3.4.3 Ebene 3: Relationen zwischen Mustern

In dieser Ebene sind alle Relationen zwischen Mustern gezeigt, die in Mustersprachen oder Mustersichten enthalten sein können. Eine Relation kann eine der drei Arten von Links repräsentieren, wie in der Abbildung demonstriert: *CloudOfferings* weist allen drei Mustern der *CloudComputing-Pattern* Sprache eine Gruppe zu. *ElasticPlatform-to-Message-orientedMiddleware* verbindet zwei Muster innerhalb einer Mustersprache. *Message-orientedMiddleware-to-Messaging* stellt eine Verbindung zwischen Mustern unterschiedlicher Mustersprachen her. Jede Relation wird durch einen Relationstyp und einen optionalen Beschreibungstext weiter beschrieben. Referenzen zu anderen Mustern können so getrennt vom Inhalt des Musters gepflegt werden. Dies erleichtert es, Referenzen automatisiert zu erfassen (sie müssen nicht erst aus dem Fließtext des Musters extrahiert werden) und ermöglicht einer Anwendung anhand der Daten z.B. für ein Muster anzugeben, zu welchen anderen Mustern eine Relation besteht.

3.4.4 Ebene 4: Muster

In dieser Ebene sind alle Muster gezeigt, die in Mustersprachen und/oder Mustersichten enthalten sind. Je nach Mustersprache sind für ein Muster andere Abschnitte definiert, wie für zwei Mustersprachen exemplarisch aufgeführt wurde. Wie im Abschnitt für Mustersprachen beschrieben, wird das Musterformat durch Einschränkungen für die Musterklasse der Sprache umgesetzt. Für die drei abgebildeten Instanzen der *CloudComputingPattern*-Klasse ist daher zum Beispiel ein Wert für die Eigenschaft **:hasDrivingQuestion** angegeben, wohingegen für die Instanzen der *EnterpriseIntegrationPattern*-Klasse ein Wert für die Eigenschaft **:hasProblem** angegeben ist. Jedes Muster gibt Expertenwissen darüber wieder, wie ein wiederkehrendes Problem durch Anwendung eines Musters gelöst werden kann, an. Dadurch, dass je nach Mustersprache die Informationen in durch die Mustersprache definierten Eigenschaften gespeichert wird, ist das Wissen der Domäne in einem sinnvollen Format repräsentiert.

4 Umsetzung eines Prototyps

In diesem Kapitel wird die Umsetzung eines Prototyps beschrieben, der die in Kapitel 3 erstellte Ontologie als Basisdaten verwenden und erweitern kann. Dieser bietet dem Endnutzer grundlegende Funktionen eines Musterrepositories an. Zuerst wird auf die zugrundeliegende Architektur eingegangen und insbesondere erklärt, wie die Ontologie-Daten im *Linked Open Data* veröffentlicht werden. Im Anschluss werden noch die Funktionen beschrieben, welche die Anwendung für die Interaktion mit den Daten der Mustersprachen bietet. Durch die Veröffentlichung der Daten in Standardformaten des semantischen Webs können diese wiederum von anderen Anwendungen genutzt und integriert werden.

4.1 Architektur

Im Folgenden soll ein Überblick über die Architektur des Prototyps gegeben werden. Zuerst sollen die Anforderungen beschrieben werden, die sich für die Anwendung ergeben. Daraufhin werden die Architektur und insbesondere die dafür verwendeten semantischen Webtechnologien aufgezeigt. Im Wesentlichen bietet eine Webanwendung (*PatternPedia*) dem Endnutzer die folgende Funktionalität eines Musterrepositories an: Bestehende Mustersprachen werden angezeigt, können bearbeitet oder um neue Mustersprachen erweitert werden. Die Daten des Musterrepositories sind hierbei im *Open Linked Data* veröffentlicht.

4.1.1 Anforderungen an die Architektur

Im Folgenden werden die Anforderungen an die Architektur beschrieben, die sich anhand der Mustersprachen-Ontologie und der Zielsetzung der Arbeit ergeben.

1. Die Daten der Ontologie sollen als **Linked Open Data** veröffentlicht werden. Wie schon in der ersten Anforderung an die Ontologie (3.1) ausgeführt, soll hierdurch die Wiederverwendung des Vokabulars ermöglicht werden. Hierbei sollen die von Tim Berners-Lee veröffentlichten *Linked data principles* befolgt werden und insbesondere URIs für die Identifikation von Ressourcen verwendet werden. Dies erlaubt eine Referenzierung der Ressourcen für die URIs existieren, unter anderem sind dies Musterrepositories, Mustersichten, -sprachen, -relationen und Muster.
2. Da die Ontologie-Daten im Web veröffentlicht werden, soll mittels einer **Webanwendung** mit ihnen interagiert werden.

3. Die Webanwendung soll den Anforderungen dieser Arbeit folgend **grundlegenden semantischen Technologien und Standards verwenden und möglichst ohne zusätzliche Logik** auskommen. Es soll insbesondere kein Applikationsserver zur Implementierung einer Datenzugriffsschicht verwendet werden. Ziel der Arbeit ist zu untersuchen, inwieweit die Anforderungen an das Musterrepository mit grundlegenden semantischen Webtechnologien umgesetzt werden können. Daher soll für die Anwendung präferiert auf semantische Webtechnologien zurückgegriffen werden.
4. Der Empfehlung von Köppe et al. [KISV16] folgend, soll es möglich sein, Muster zu **versionieren**. Alte Versionen eines Musters sollen daher nicht verworfen werden, sondern zumindest für Administratoren zugänglich sein.
5. Um die Sicherheit der Daten zu gewährleisten und Änderungen nachvollziehen zu können müssen Nutzer für einen Schreibvorgang authentifiziert sein.

4.1.2 Grundaufbau der Architektur und darin verwendeten Webtechnologien

Die für diese Arbeit verwendete Architektur ist in Abb. 4.1 dargestellt. Im Folgenden wird diese kurz beschrieben und dann die Designentscheidungen erläutert. Links ist die Webanwendung abgebildet, die in Anforderung 2 gefordert wurde. Diese ermittelt über die URIs der Ontologieinstanzen die Daten für die jeweilige Instanz. Den *Open Data Principles* folgenden, werden HTTP URIs verwendet, sodass eine GET-Anfrage die benötigten Daten zurückliefert. Hierfür werden zwei Weiterleitungen durchgeführt und zuletzt auf den Inhalt einer Datei in einem Github-Repository verwiesen. Das Github-Repository stellt eine Schnittstelle zur Änderung und Erstellung der Dateien bereit, wodurch die Webanwendung Änderungen in den Daten persistieren kann.

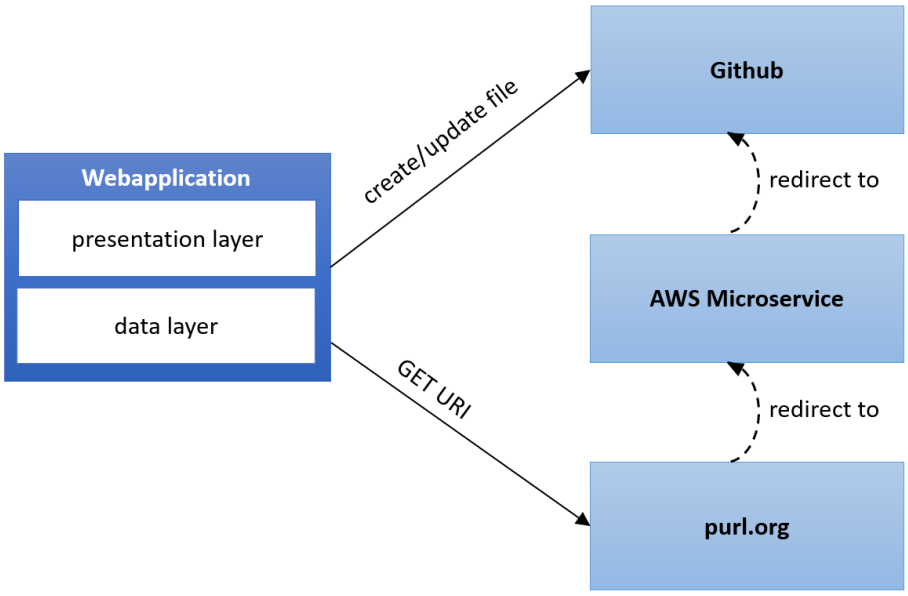


Abbildung 4.1: Architektur des Prototyps

Bezüglich der Architekur wurde entschieden, den Empfehlungen der W3C zur Veröffentlichung von Daten im *Linked Open Data* [BPM+08] zu folgen und wie in Anforderung 3 gefordert, möglichst wenig eigene Logik zu implementieren. Die Empfehlungen [BPM+08] beschreiben als einfachste Möglichkeit die Speicherung der Daten auf Dateibasis und zeigen auf, wie Persistent URLs (PURLs) zur Einrichtung von URIs verwendet werden können. Eine PURL ist eine URL, die jedoch nicht direkt auf eine Ressource zeigt, sondern für diese einen Weiterleitungsservice hinterlegt [HK06]. Hierdurch wird die Bezeichnung der Ressource von der physikalischen Adresse entkoppelt und Hypertext Transfer Protocol (HTTP)-Standardfunktionen zur Weiterleitung genutzt [HK06]. Da sich für eine Speicherung der Daten auf Dateibasis entschieden wurde und möglichst wenig eigene Logik verwendet werden soll, wurde auf die Implementierung einer eigenen Datenzugriffsschicht bzw. Einrichtung einer Datenbank verzichtet. Stattdessen wurden die Dateien in einem Github-Repository hinterlegt, da Github Funktionalität zur Änderung und Erstellung von Dateien bereitstellt. Wie in Abschnitt 4.1.3 dargestellt wird, kann der PURL-Weiterleitungsservice nicht direkt auf die entsprechende Githubdatei weiterleiten, weshalb ein zusätzlicher AWS Microservice eingerichtet wurde. Mithilfe von diesem ist es möglich, auf die Anfrage einer URI durch zwei Weiterleitungen mit dem Inhalt der Datei mit den dazugehörigen RDF-Daten zu antworten. Die Webanwendung kann daher durch eine GET-Anfrage für die entsprechende URI die nötigen Dateien für die Ontologieinstanz abrufen (unterer Pfeil). Alternativ kann sie direkt die Github bereitgestellte Schnittstelle nutzen (dokumentiert unter `https://developer.github.com/v3/`), um Dateien abzurufen, zu ändern oder anzulegen (oberer Pfeil). Hierdurch erfolgt ebenfalls eine Versionierung der Dateiänderungen, wie in Anforderung 4 gefordert wurde. Für die Auswertung der Daten wurde in der Webanwendung eine Datenschicht implementiert, die die JavaScript Bibliothek *rdfstore-js* verwendet, um die RDF-Daten einzulesen und gezielt Informationen durch SPARQL anzufragen. Mit dieser kann beispielsweise die Muster einer Mustersprachen abgefragt werden und das Ergebnis der Präsentationsschicht als JavaScript-Objekte bereitgestellt werden.

Wie in den Anforderungen formuliert, sollen Standards der Schichten eingesetzt, für die das W3C bereits Standards veröffentlichte (*Identifiers, Characters, Syntax, Data Model, Querying & Rules* sowie *Schema & Ontologies*, siehe Abb. 2.6). Für die Kodierung von Schriftzeichen wurde der Unicode-Zeichensatz benutzt (*Characters* Schicht). Wie in den *Open Data Principles* empfohlen, wurden HTTP URIs zur Identifikation von Ressourcen eingesetzt (*Identifiers* Schicht). Für konkrete Muster, Mustersprachen und -sichten sowie die abstrakten Konzepte dahinter wurden daher eigene URIs eingeführt, um diese klar identifizieren zu können. Als konkrete Syntax für RDF wurde Turtle benutzt, da diese Syntax erlaubt die Aussagen kompakt darzustellen (*Syntax* Schicht). Für die Modellierung der Domäne (*Schema & Ontologies* Schicht) wurden OWL DL und RDFS eingesetzt. Mit RDFS lassen sich wie in Abschnitt 2.3.2 erläutert einfache Hierarchien ausdrücken und Vererbung wie ”Mustersprache ist eine Unterklasse von Mustergraph“ ausdrücken. Weiter wurde OWL DL verwendet, da Funktionen von OWL benötigt wurden, um beispielsweise zu spezifizieren, dass eine ungerichtete Musterrelation genau zwei Muster verbindet, was mit RDFS nicht möglich wäre. Die fünf Schichten (*Identifiers, Characters, Syntax, Data Model* und *Schema & Ontologies*) wurden damit schon im letzten Kapitel für die Modellierung der Domäne verwendet. Insgesamt wurden alle sechs standardisierten Schichten des *Semantic Web Layer Cake* in der Anwendung verwendet.

4.1.3 Bereitstellung der Mustersprachen als Linked Open Data

Den Linked Data Principles (siehe Abschnitt 2.3.4) folgend, soll das Vokabular der Ontologie und alle Instanzen als *Linked Open Data* veröffentlicht werden. Wie im vorherigen Abschnitt bereits beschrieben, wurden *Persistent URLs (PURLs)* als URIs verwendet. Die Einrichtung einer PURLs Domäne (für unsere Ontologie `https://purl.org/patternpedia`) erlaubt dann für diese URI und URIs unterhalb der PURL Domäne (z.B. `https://purl.org/patternpedia/patternlanguages`) eine Weiterleitung einzutragen. Durch die Wahl des Domainnamen *patternpedia* veranschaulicht die URI den Zweck der darunter gespeicherten Daten. Wird eine URIs angefragt, die mit dem eingerichteten Namensraum beginnt (`https://purl.org/patternpedia`) leitet der Weiterleitungsservice von `purl.org` an die dafür spezifizierte Adresse weiter (in Abb. 4.2 gezeigt).

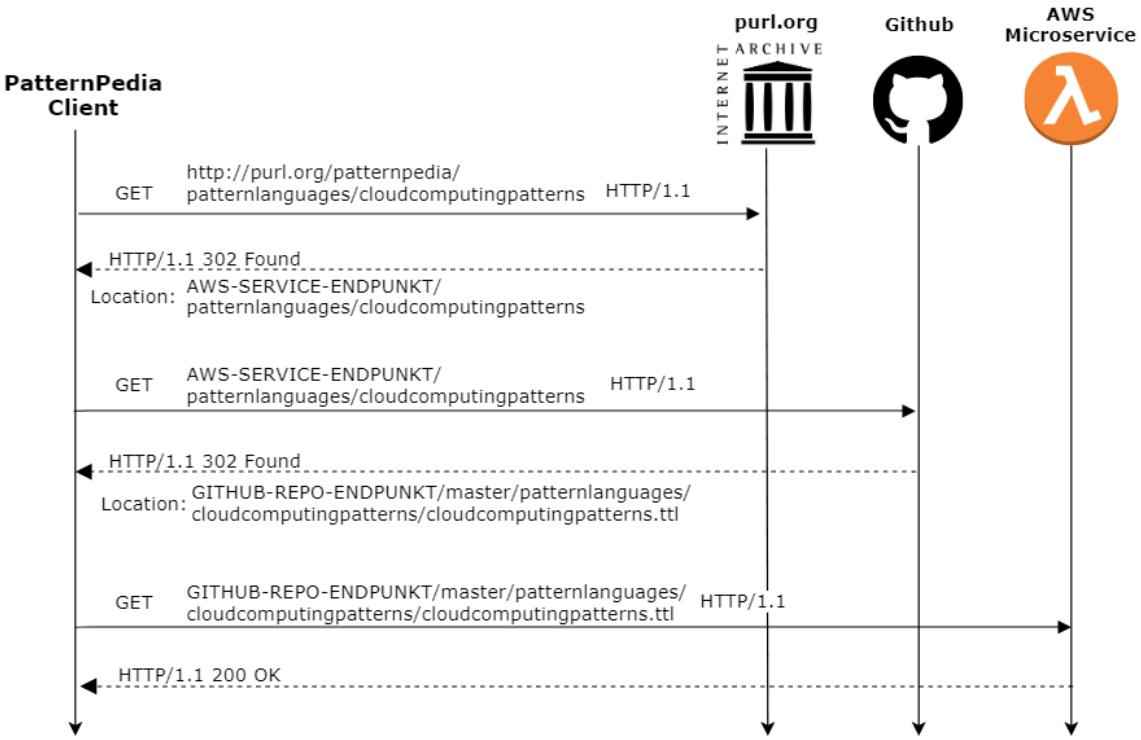


Abbildung 4.2: Bereitstellung im Linked Open Data

Hierfür wird der erste Teil der angefragten URL (beispielsweise `purl.org/patternpedia/patterlanguages/cloudcomputingpatterns/cloudcomputingpatterns#PublicCloud`) durch die Adresse eines AWS Lambda Services ersetzt und somit eine partielle Weiterleitung durchgeführt. Der zusätzliche Service wird benötigt, da weder der Weiterleitungsservice von `purl.org` noch das Github-Repository, in dem die Daten gespeichert sind, die entsprechende Dateiendung (.ttl) ergänzen kann. Außerdem sollen unter der URI einer Musterspache (`purl.org/patternpedia/patternlanguages/NAME_MUSTERSPRACHE`) die Basisinformationen der Sprache und unter `purl.org/patternpedia` das Basisvokabular abgerufen werden können. Für alle diese Fälle wird die korrekte Weiterleitung mit einem eigens dafür eingerichteten AWS Services ermöglicht: dieser löst die Anfrage zu der URL auf, unter der die Datei bei Github erreicht werden kann. Eine Weiterleitung zu der Basisdatei einer Mustersprache ist beispielhaft in Abb. 4.2 dargestellt. Die Speicherung der Daten im Github Repository stellt für jede Datei eine URL (`https://raw.githubusercontent.com/REPOSITORY_PFAD`) bereit,

unter der sich der Dateiinhalt abrufen lässt. Alternativ zu der gerade beschriebenen Lösungsvariante wäre es auch möglich gewesen, eine eigene Infrastruktur für das Hosting der Dateien einzurichten und anstatt der zweiten Weiterleitung den Inhalt der Datei für die angefragte URL durch HTTP-Funktionalitäten zur Inhaltsvereinbarung (Content-Negotation) zu ermitteln. Da aber eine der Anforderungen an die Arbeit ist, keinen Applikationsserver zu verwenden wurde von dieser Lösung abgesehen.

Die Organisation der Dateien im Github Repository ist in Abb. 4.3 dargestellt. Um für jede Muster-

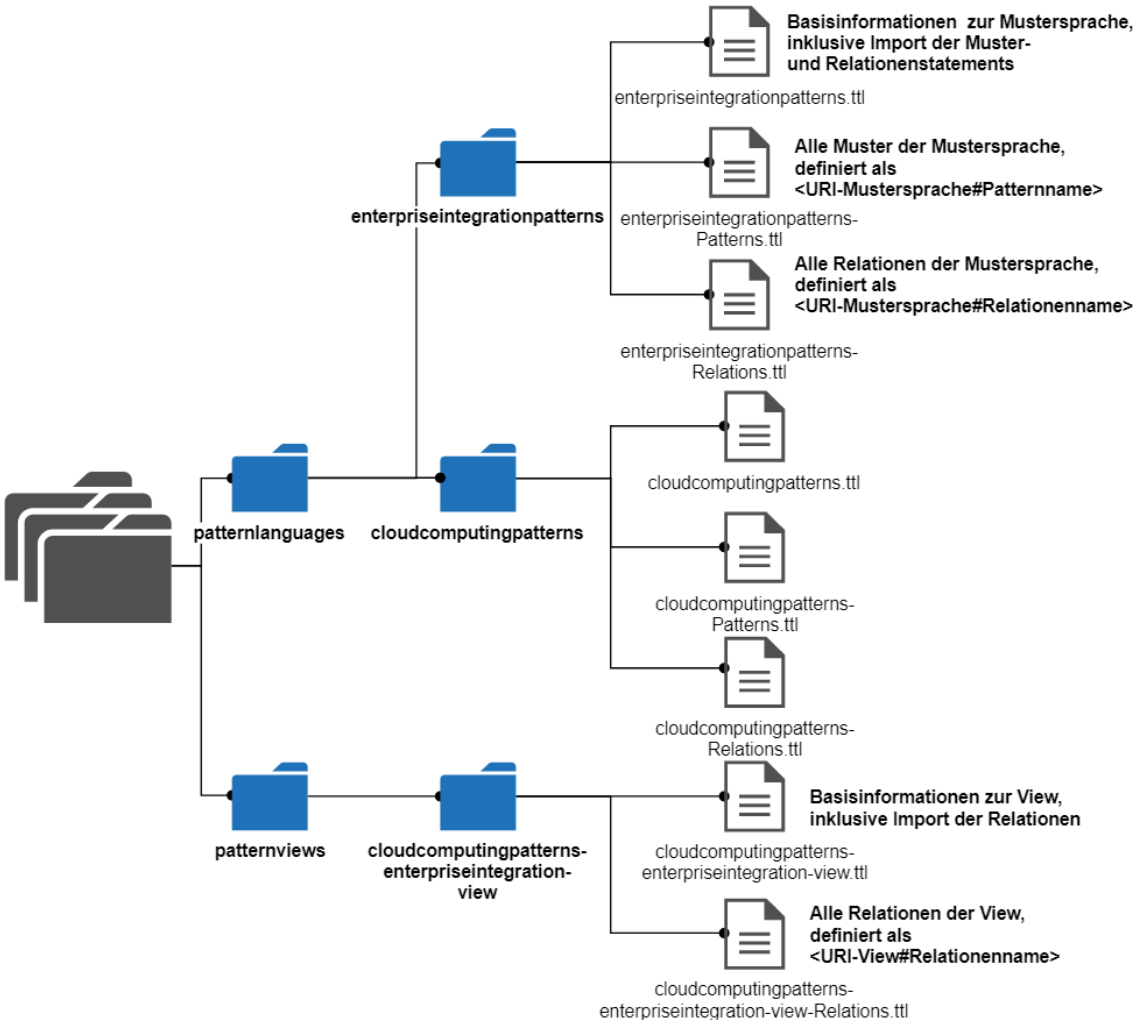


Abbildung 4.3: Organisation der Dateien im Github Repository, die die Aussagen zur Ontologie enthalten. Für die erste der dargestellten Mustersprachen, sowie für die Mustersicht ist eine kurze Beschreibung der Dateien gegeben. Die Aufteilung in die einzelnen Dateien ermöglicht es weitere Informationen zu Mustersprachen oder -sichten bei Bedarf nachzuladen.

sprache und Mustersicht einfacher festzuhalten, wer an der Veröffentlichung beteiligt war, werden die Daten der einzelnen Mustergraphen in separaten Ordnern abgelegt. Für ein effizientes Laden der Daten sind die Basisdaten zur Mustersprache, die Muster und die Relationen auf drei Dateien verteilt (siehe z.B. den Ordner *enterpriseintegrationpattern* für die *Enterprise Integration* Mustersprache).

Somit kann beispielsweise für eine Anzeige aller Mustersprachen nur die URIs von diesen angefragt werden, womit die Basisinformationen der Sprachen geladen werden (in der Abbildung entspräche dies den Dateien *enterpriseintegrationpatterns.ttl* und *cloudcomputingpatterns.ttl*). Diese enthalten Informationen zur Mustersprache wie das Format der Muster sowie die Aussagen darüber, welche Muster und Relationen in der Sprache enthalten sind. Bei Navigation zur Übersicht der Mustersprache können dann die Informationen über die Muster abgerufen werden (diese sind in *cloudcomputingpatterns-Patterns.ttl* und *enterpriseintegrationpatterns-Patterns.ttl* enthalten). Mit dieser Aufteilung der Daten nach diesem Schema wird ermöglicht, weitere Informationen bezüglich der Mustersprachen oder Mustersichten bei Bedarf nachzuladen. Es wäre auch möglich gewesen, die Daten zu den Mustern jeweils in eigene Dateien auszulagern, jedoch würde dies den Ladeaufwand für eine Übersichtsanzeige der Muster deutlich erhöhen da dann für jedes Muster eine Datei angefragt und geladen werden muss.

4.1.4 Autorisierung der Anwendung und Versionierung der Änderungen

Für Änderungen an den Daten des Musterrepository über die Github-API (`https://api.github.com` wird ein Authentifizierungstokens benötigt. Nutzer sollen deshalb die Anwendung autorisieren können, für dessen Github-Konto Änderungen an den Daten des Musterrepositories vorzunehmen. Die hierfür benötigten wesentlichen Schritte werden in diesem Abschnitt erläutert. Um dies zu ermöglichen, wurde der *OAuth Web App Flow* implementiert (dargestellt in Abb. 4.4) und die Anwendung als OAuth-App bei Github registriert (erster gelb hinterlegter Schritt). Auf der Startseite wird dem Nutzer ein Button zur Autorisierung angeboten, der ihn zu einer Github-Seite weiterleitet (erster Pfeil der Abbildung). Dort kann er die Autorisierung vornehmen: Hierfür meldet er sich mit seinem Benutzerkonto an und bestätigt, dass er die Anwendung *PatternPedia* dazu autorisieren möchte, für das Github-Repository der Ontologie-Daten in seinem Namen Änderungen vorzunehmen (in der Box "First time user" dargestellt). Daraufhin wird er auf die dafür eingetragene `callback_url` der Anwendung zurück geleitet. Falls der Nutzer die Anwendung schon einmal autorisiert hat, wird der letzte Schritt übersprungen und der Nutzer direkt wieder zur Anwendung zurückgeleitet. Mit den Angaben in den URL-Parametern fragt die Webanwendung dann einen Service an, der mit Konfigurationsdaten der registrierten OAuth-App das Zugriffstoken anfragt (Pfeil von Service zu Github). Dieser zusätzliche Service wird benötigt, damit in der Webanwendung die Werte des `client_secret` nicht ausgelesen werden, was in der Anfrage des Zugriffstokens enthalten ist. Der Service antwortet der Webanwendung dann mit dem erhaltenen Zugriffstoken (letzter Pfeil), womit diese dann Änderungen der Daten über die Github-API durchführen kann.

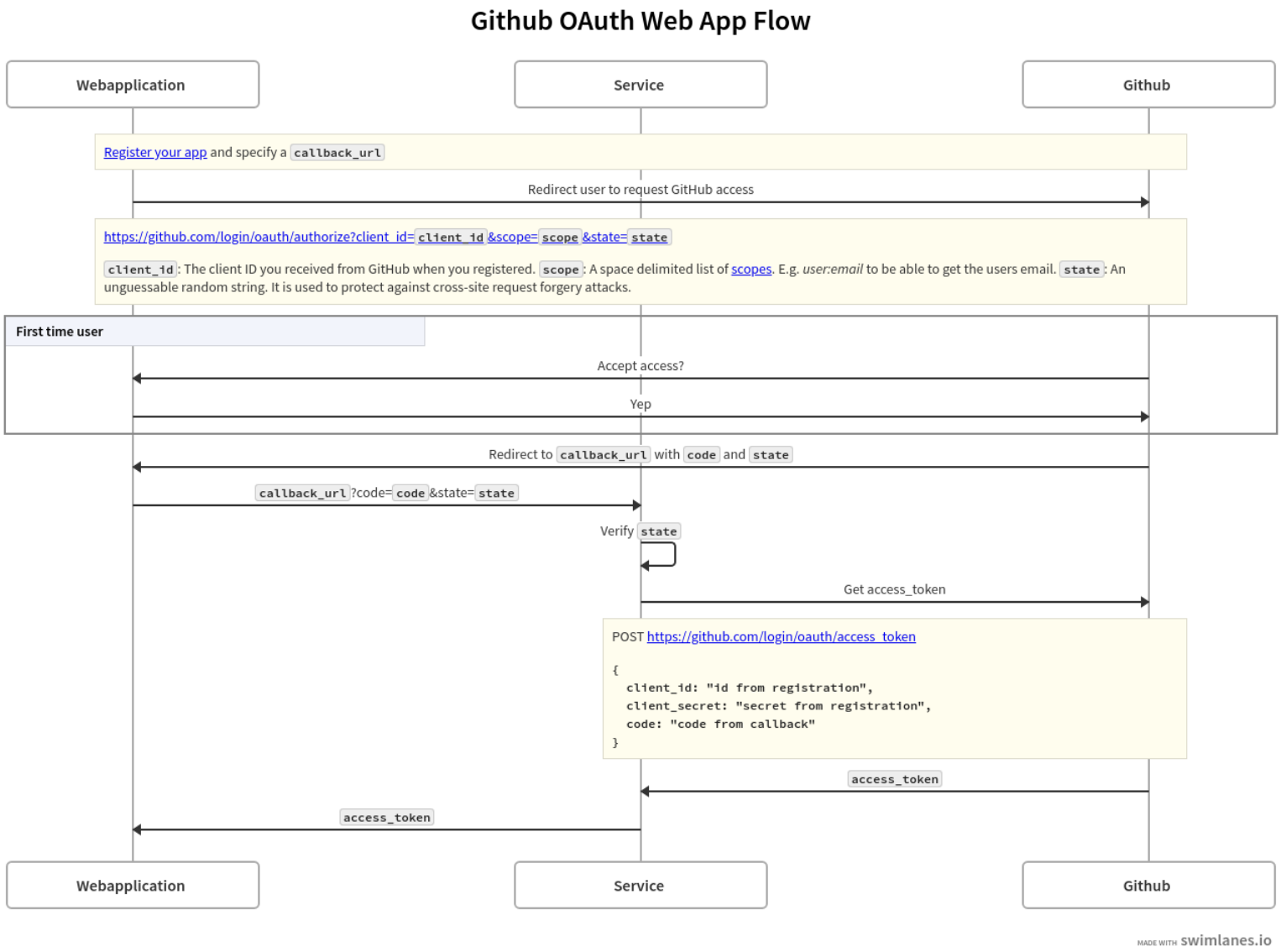


Abbildung 4.4: OAuth-Flow für die Autorisierung von Github OAuth-Apps. Dieser Flow wurde für die Anwendung umgesetzt, sodass ein Nutzer durch einen Link auf Github umgeleitet wird und dort die Anwendung dazu autorisieren kann, für sein Konto Änderungen am Repository der Daten vorzunehmen. Abbildung nach https://swimlanes.io/gallery/github-oauth.

4.2 Funktionen des Prototyps

In den folgenden Abschnitten folgt eine Darstellung der Funktionen eines Musterrepositories für den Prototypen: Anlegen von Mustersprachen, Mustern, Links sowie eine graphische Darstellung der Daten. Muster können außerdem in der Detailansicht bearbeitet werden.

4.2.1 Anlegen einer neuen Mustersprache

Der Prototyp bietet dem Nutzer die Funktionalität, eigene Mustersprachen hinzuzufügen. Zu Anzeigezwecken muss hierfür ein Name vergeben und die URL eines Icons hinterlegt werden (siehe Abb. 4.5). Da sich diese in ihrer Struktur je nach Domäne stark unterscheiden können, sind die

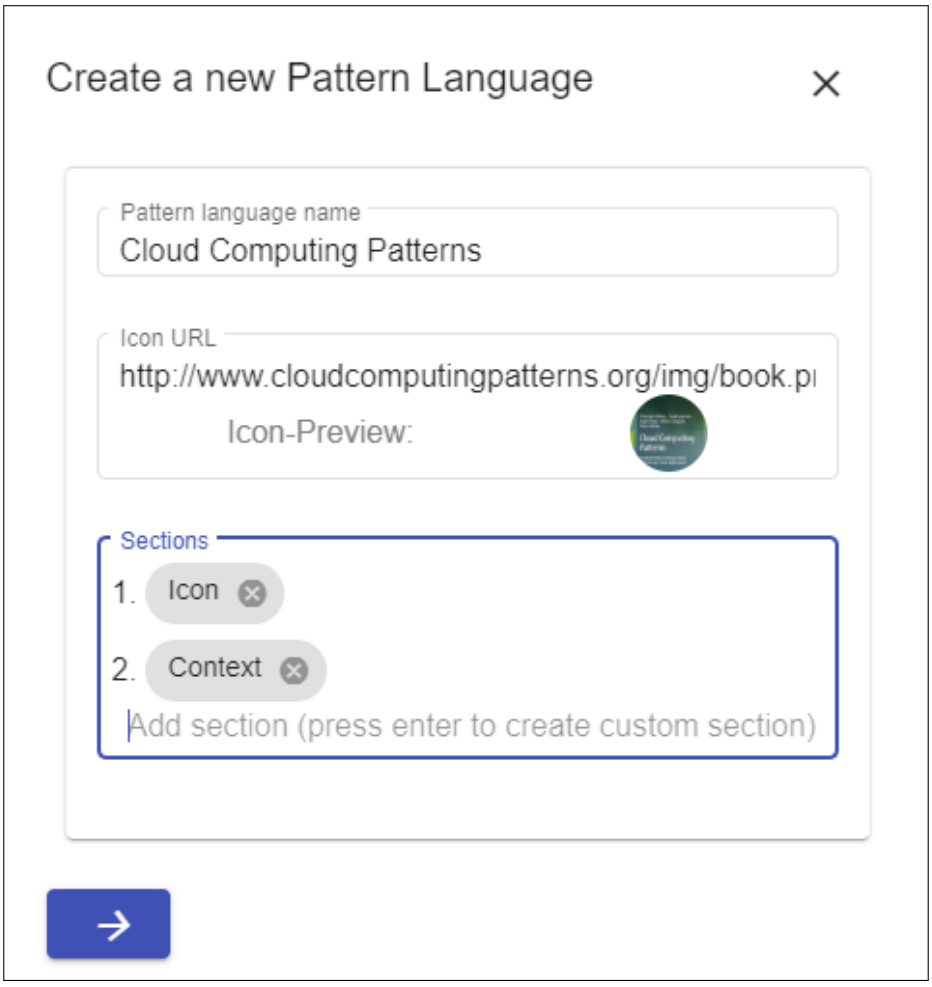


Abbildung 4.5: Erster Schritt bei der Erzeugung einer neuen Mustersprache. Es müssen Name, Icon und die geordnete Liste von Sektionen festgelegt werden.

Sektionstitel ebenfalls selbst auszuwählen. Der Benutzer kann hierfür aus einer Liste von vorgegebenen Sektionen auswählen oder neue Sektion eingeben. Hierbei wird, wie durch die Nummerierung visualisiert die Reihenfolge der Sektionen ebenfalls spezifiziert. Im nächsten Schritt (siehe Abb. 4.6)

können dann Einschränkungen für die einzelnen Sektionen festgelegt werden. Die Benutzeroberfläche für die Festlegung von Einschränkungen wurde an Protegé angelehnt, einem gängigen Programm zur Modellierung von OWL-Ontologien. Daher ist es möglich pro Sektion ein Minimum/Maximum oder eine genaue Anzahl an Werten eines bestimmten Datentyps festzulegen. Ebenso kann der Datentyp der erlaubten Werte einer Sektion eingeschränkt werden: Wie in der Abb. 4.6 gezeigt, ist für die Sektion Context nur Werte einer Zeichenkette (xsd:string) erlaubt. Durch die Buttons neben einer Einschränkung kann eine weitere Einschränkung für die jeweilige Sektion hinzugefügt werden, sodass z.B. sowohl Minimum als auch Maximum an erlaubten Werten gesetzt werden können. Die Integration von selbst definierten Typen wird durch Präfixe ermöglicht, die gesetzt werden können. Standardmäßig werden Typdefinitionen der Präfixe xsd und dctype (ausführliche Liste der spezifizierten Typen unter `https://www.w3.org/2011/rdf-wg/wiki/XSD_Datatypes` bzw. `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#section-7`) vorgeschlagen, die weit verbreitet sind. Nach dem Speichern wird die zunächst leere Mustersprache bereits als Teil der im *Linked Open Data* veröffentlichten Mustersprachen angezeigt. Insgesamt ist es durch die Führung des Nutzers möglich, ohne viel Hintergrundwissen zu OWL eine Ontologie für eine Mustersprache zu modellieren.

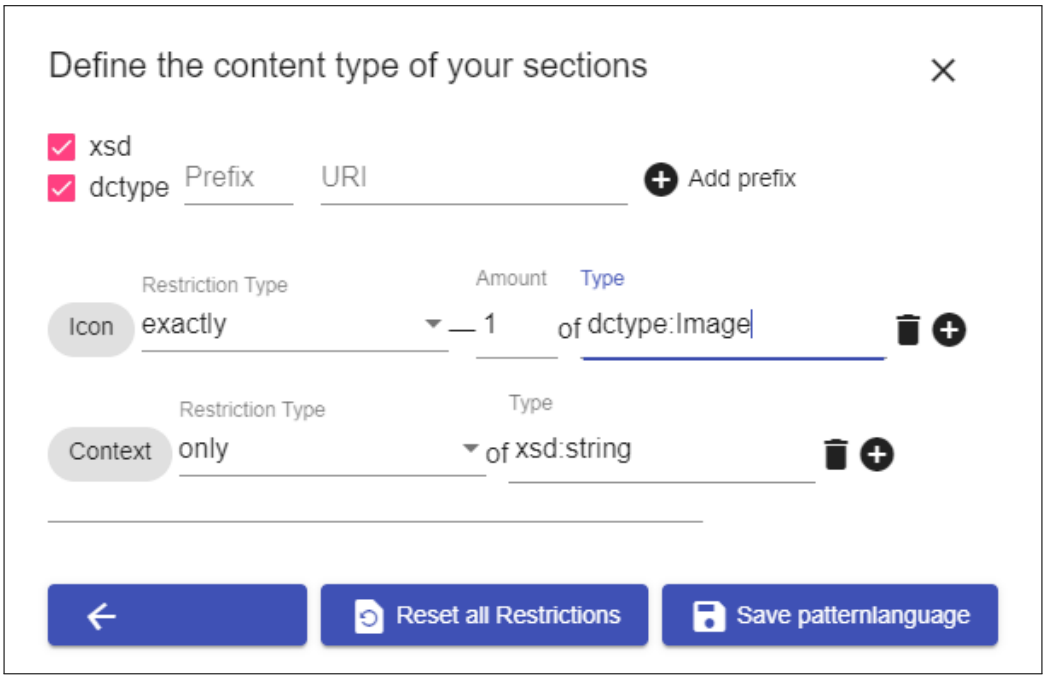


Abbildung 4.6: Zweiter Schritt bei der Erzeugung einer neuen Mustersprache, hierbei sind für die bessere Darstellung nur zwei Sektionen gezeigt. Es können Einschränkungen für die einzelnen Sektionen angelegt werden: Für die Sektion Icon ist bei der obigen Einstellung nur genau ein Wert vom Typ dctype:Image erlaubt.

4.2.2 Anlegen eines neuen Musters

Für bestehende Mustersprachen können Muster hinzugefügt werden. Hierzu bietet die Anwendung wie in Abb. 4.7 gezeigt, an den Namen des Musters (Überschrift erster Ordnung) und den Inhalt der Sektionen (jeweils der Überschrift zweiter Ordnung der Sektion) für das Muster zu spezifizieren. Die

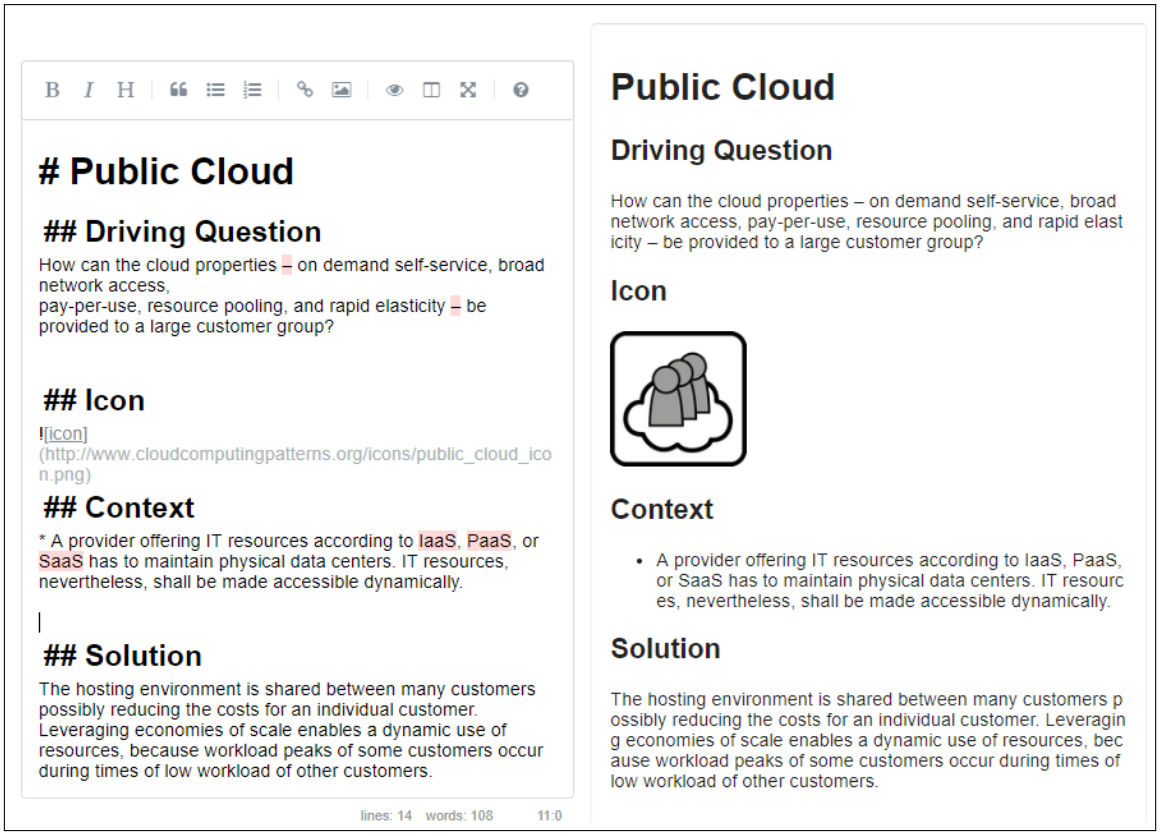


Abbildung 4.7: Eingabefeld und Vorschau zum Anlegen eines Musters, hier für die Mustersprache "CloudComputingPatterns" von Fehling et al. [FLR+14] gezeigt. Zur besseren Darstellung sind nur ausgewählte Sektionen dargestellt. Links kann der Name des Musters festgelegt werden (Überschrift erster Ordnung) und der Inhalt für die einzelnen Sektionen nach der jeweiligen Überschrift der Sektion (zweiter Ordnung) eingefügt werden. Die Eingabe erfolgt mithilfe des Editors in Markdown, wofür rechts eine Vorschau angezeigt wird. Falls mehrere Werte für eine Sektion erlaubt sind, können diese Aufzählung eingegeben werden (siehe Sektion *Context*).

Eingabe erfolgt mit einem Markdown-Editor und einer entsprechenden Vorschau der eingetragenen Daten. Markdown ist eine einfache Auszeichnungssprache, die gut zu lesen ist und in Webinhalte (HTML) übersetzt werden kann [Mai19]. Hierdurch wird ermöglicht, Bilder auf deren URL in der Markdown-Syntax verwiesen wird bereits in der Vorschau anzuzeigen (siehe *Icon*-Sektion der Abbildung). Da die gespeicherten Daten später wieder genauso angezeigt werden wie in der Vorschau, kann die volle Funktionalität des Markdown-Editors genutzt werden. Durch den Editor können in den Text einer Sektion flexibel Links, Bilder, kursive/fette Schrift, etc. eingebettet werden. Für die Eingabe von mehreren Werten wurde die Konvention gewählt, dass diese in einer Aufzählung gelistet werden (in der Abbildung für die Sektion *Context* begonnen). Nach dem Hinzufügen eines Musters taucht dieses in der Anzeige der Mustersprache auf, von der auch auf die Detailansicht des Musters navigiert werden kann.

4.2.3 Erzeugen eines neuen Links zwischen Mustern einer Mustersprache

Bei der Anzeige eines Musters kann dieses um Links ergänzt werden. Dies ist in Abb. 4.8 dargestellt. Hierzu ist die Richtung der Relation, das Muster zu dem die Verbindung hergestellt werden soll und ein optionaler Beschreibungstext anzugeben. Somit kann jede Mustersprache um Links zwischen den Mustern erweitert werden, die wie gefordert durch die Richtung, dem Typ der Relation und weiteren Informationen beschrieben werden.

Abbildung 4.8: Eingabefeld und Vorschau zum Anlegen eines Links zwischen Mustern. Hierzu kann die Richtung des Links ausgewählt werden: Der Nutzer kann zwischen einer gerichteten Verbindung von diesem Muster zum zweiten Muster (⟶), der gegensätzliche Richtung (⟵) oder einer ungerichteten Verbindung (⇆) auswählen. Im rechten Eingabefeld ist ein zweites Muster aus der Mustersprache auszuwählen.

4.2.4 Graphdarstellung der Links

Im Zuge der Arbeit von Esin [Esi19] wurde ebenfalls eine Graphdarstellung für Mustersprachen erstellt, die Muster als Knoten und Relationen zwischen den Mustern als Kanten realisiert. Beim Klick auf ein spezifisches Muster wird eine Liste der Muster anzeigt, die Verbindungen zu dem Muster aufweisen. Hierbei werden für ein Muster auch mustersprachenübergreifende Verbindungen angezeigt.

5 Fazit und Ausblick

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und bewertet. Anschließend wird in einem Ausblick diskutiert, welche Schritte und Implementierungen noch fehlen, um die Vision aus dem Grundlagenkapitel zu verwirklichen.

Fazit der Arbeit

Im Zuge dieser Arbeit wurden die wesentlichen Aspekte von Mustersprachen charakterisiert und eine formale Ontologie aufgebaut, die diese widerspiegelt. Zur Erstellung der Ontologie wurden semantischen Webtechnologien benutzt und hiermit ein Vokabular zur Repräsentation von Mustersprachen aufgebaut. Zur besseren Wiederverwendung wurden diese Daten als Wissensbasis im *Linked Open Data* unter dem Namensraum `https://purl.org/patternpedia` veröffentlicht. Mithilfe eines Prototyps wurde gezeigt, dass Anwendungen auf dieser Wissensbasis aufbauen und diese erweitern können. Der Prototyp implementiert die Grundfunktionen eines Musterrepositories: Benutzer können Mustersprachen anlegen und das Format der Muster festlegen, Mustern anlegen und bearbeiten, sowie Links zu anderen Mustern der Mustersprache hinzufügen. Im Zuge der Arbeit von Esin [Esi19] wurde von diesem außerdem eine Graphdarstellung für Mustersprachen erstellt die auch mustersprachenübergreifende Links darstellt. Einige Funktionen eines Musterrepositories fehlen noch, wie mehr Bearbeitungs- und Löschfunktionalität, das Anlegen von Links zur Gruppierung von Mustern oder Links über Mustersprachen hinweg. Für die beiden Linkarten, die sich somit nicht über die Anwendung anlegen lassen, wurde im der Ontologie jedoch bereits die notwendigen Konzepte erarbeitet und das notwendige Vokabular einführt. Insbesondere für mustersprachenübergreifende Links wurde in Analogie zu klassischen Datenbanksichten die Einführung von Mustersichten motiviert. Diese bieten den Vorteil, dass mustersprachenübergreifende Links nicht in beiden Mustersprachen gespeichert werden müssen und die Duplizierung von Mustern vermieden wird, da die Muster referenziert werden können. Insgesamt wurde durch die Implementierung der angesprochenen Basisfunktionen die Machbarkeit eines Musterrepositories für semantische Daten gezeigt.

Dadurch, dass Mustersprachen um Muster und Links ergänzt werden können, werden Mustersprachen als *lebendige Netzwerke* im Sinne von Christoph Alexander repräsentiert. Besonders herkömmliche Veröffentlichungen in Büchern, Fachzeitschriften oder -artikeln bieten im Gegensatz zu dem entwickelten Musterrepository nur begrenzte Möglichkeit zur Weiterentwicklung der Mustersprache. Die im Grundlagenkapitel aufgezeigte Graphstruktur von Mustersprachen kann insbesondere durch die Daten im RDF-Format direkt widergespiegelt werden, da diese ebenfalls einen Graphen aufspannen. Der Einsatz von semantische Webtechnologien bietet den Vorteil, dass andere Anwendungen auf den Daten aufbauen und das ebenfalls im *Open Data* bereitgestellte Vokabular wiederverwenden können. Globale URIs der Muster und Mustersprachen erlauben es, diese in anderen Dokumenten zu referenzieren. Diese Repräsentation von Mustersprachen und den

drei verschiedenen Linkarten mithilfe semantischer Webtechnologien und Standards ist in bisherigen Musterrepositories nicht zu finden. Im Vergleich hierzu nutzten nur zwei andere Veröffentlichungen semantische Webtechnologien. Fehling et al. [FBFL15] implementierten ein Repository auf Wiki-Basis, das den Endnutzern erlaubte semantische Annotationen einzupflegen. Links zu anderen Musterdokumenten sind möglich, jedoch sind die wichtigsten Konzepte (Muster, Links, etc.) nicht explizit als Ontologie formalisiert. Pavlič et al. [PHP08] führen eine Ontologie für Mustersprachen ein, die jedoch nur Links innerhalb von Mustersprachen erlauben. Semantische Agenten, die auf der Wissensbasis der vorliegenden Arbeit aufbauen, können deshalb auch gezielt Informationen zu Beziehungen zwischen Mustern verschiedener Mustersprachen verarbeiten.

Um die Arbeit zu reflektieren, werden einige Architekturentscheidungen nochmals kritisch hinterfragt. Die Speicherung der Daten soll als erstes diskutiert werden: Die Daten der Wissensbasis wurden in Daten in einem Github-Repository angelegt. Da die Daten zu keinem Zeitpunkt in einer Datenbank persistiert werden, lädt die Client-Anwendung die benötigten Dateien, stellt die RDF-Daten der Dateien intern in einer Graphstruktur und erstellt anhand dieser die zu visualisierenden Objekte. Bei großer Anzahl der zu ladenden Dateien und benötigten Anfragen, resultiert dies in einer schlechten Performance. Bei einer Änderung muss jeweils die gesamte Datei neu geschrieben werden. Um Dateikonflikte der Daten vorzubeugen und Änderungen auf Objektebene vornehmen zu können, bietet sich die Persistierung der Daten in einer Datenbank an. Hierfür könnte man beispielsweise Amazon Neptune (`https://docs.aws.amazon.com/neptune`) als Graphdatenbank einsetzen, das SPARQL unterstützt. Die Businesslogik, die dann das Einfügen bzw. Bearbeiten von neuen Objekten (Mustersprachen, -sichten, Mustern, Links) übernimmt, kann dadurch in einen Applikationsserver ausgelagert werden. Zusätzlich wäre es auch möglich, einen SPARQL-Endpunkt bereitzustellen, wie Berners-Lee in den *Linked Data Principles* empfiehlt.

Vor allem bei der Anfrage einer großen Anzahl von URI beeinflusst die Durchführung von zwei HTTP-Weiterleitungen die Performance negativ. Die Einrichtung der PURL Namensdomäne ermöglicht zwar die Trennung der genauen Bezeichnung der Ressource und deren physikalischer Adresse, die Infrastruktur zum Speichern der Daten muss jedoch selbst bereitgestellt werden. Insgesamt wurde für die Veröffentlichung der Daten als *Linked Open Data* im Zuge dieser Arbeit viel Fachwissen benötigt, um die Logik zur Weiterleitung und Identitätsmangement zu ergänzen. Für die Veröffentlichung von kleineren Datenmengen wäre es von Vorteil, wenn die Betreiber `purl.org` oder eine andere, unabhängige Organisation einen Hostingservice für *Linked Open Data* anbieten würde. Ein kostenloses Hosting, das sich ohne weiteren Aufwand mit dem Weiterleitungsservice von PURL benutzen lässt, würde die Veröffentlichung von weiteren Daten fördern. Hierdurch können auch Nutzer ohne detailliertes Wissen zur Bereitstellung von Infrastruktur Daten beitragen. Im nächsten Schritt könnte dieser Hosting-Service auch Identitätsmanagement und Versionierung der Datenänderungen zur Verfügung stellen, damit andere Nutzer die Daten gegenfalls korrigieren und erweitern können. Die Hürde, die Daten z.B. auf einen eigenen Server selbst bereitzustellen, verhindert das Hinzukommen von weiteren verlinkten Daten - was dem eigentlichen Ziel der *Linked Open Data*-Bewegung widerspricht.

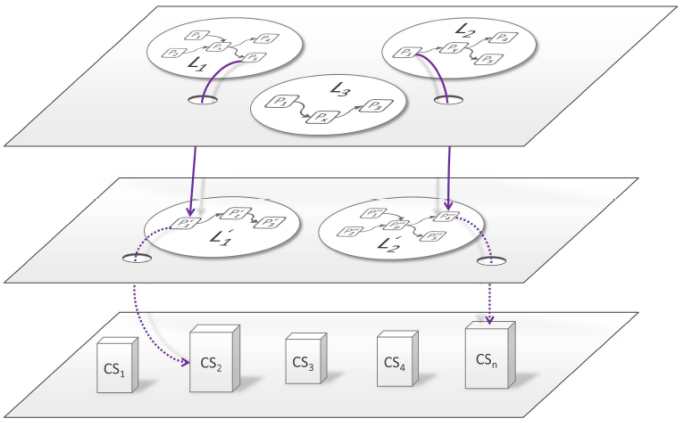


Abbildung 5.1: Darstellung von Mustern auf verschiedenen Abstraktionsebenen von Falkenthal et al. [FBB+16], die mittels *refinement*-Links verbunden sind. Auf der mittleren Ebene sind die Muster P_i mit *konkreten Lösungen* (CS_i) verbunden. Lösungen wurden in der erstellten Ontologie nicht abgebildet.

Ausblick

In der Vision wird beschrieben, dass *PatternPedia* Codegenerierung für eine Auswahl von Mustern aus dem IT Bereich unterstützt. Der Endnutzer muss dann nur noch die Businesslogik des heruntergeladenen Codegerüsts ergänzen. Um Muster mit passendem Code assoziieren zu können, kann in einem nächsten Schritt das Hinzufügen von *konkreten Lösungen* (beispielsweise Code) zu einem konkreten Muster ermöglicht werden (dargestellt in Abb. 5.1). Die Dokumentation von *konkreten Lösungen* für ein Muster erleichtert die Wiederverwendung dieser *konkreten Lösungen* [FBB+14]. Leider bieten bisherige Musterrepositories diese Funktionalität bislang nicht, allein in der ersten Version von PatternPedia [FBFL15] wurde dies für die Domäne von Kostümmustern umgesetzt [FBB+14]. Sind die *konkreten Lösungen* auch untereinander verlinkt, kann dadurch ausgedrückt werden wie diese miteinander verwendet werden können [FBB+14]. Die Links der Muster zu den Lösungsimplementierungen können auch als Auswahlkriterien verwendet werden. Für den Falle von Mustern aus dem Bereich IT kann ein Link die in der Lösung verwendete Programmiersprache beschreiben [FBB+14]. Anhand der Präferenzen des Endnutzer kann dann eine Lösung in der gewählten Programmiersprache ausgewählt werden. In der Praxis wird oft die Anwendung von mehreren Mustern benötigt und somit müssen die möglichen konkreten Lösungen aller verwendeten Muster aggregiert werden [FBB+14]. Die Autoren zeigen für einige ausgewählte Mustersprachen, wie durch die beschriebenen Links festgehaltenes Wissen über konkrete Lösungen die Basis dafür bietet. Für Muster aus der IT resultiert eine solche Aggregation in der Codegenerierung von ausgewählten Mustern [FBB+14], was der *PatternPedia* Vision der Einleitung entspricht.

Ein Vorteil der Verwendung der Standardformate des semantischen Webs ist der Umgang mit Inkonsistenzen, welcher in herkömmlichen Datenbanksystemen oft nicht vorgesehen ist. Im anfangs erwähnten Szenario kann z.B. noch ein Forum zur Diskussion über Muster hilfreiche Hinweise zu Links beitragen:

> Nach der Codegeneration stellt Lucy im Zuge der Implementierung dieser Businesslogik in einem Forum die Frage, welche Cloudservices sich für die Implementierung des *Server Session State* Entwurfsmuster von Fowler [Fow02] eignen. Dies wurde in der bisherigen Anwendung verwendet. Außerdem nutzt sie die Möglichkeit des Forums die jeweiligen in *PatternPedia* gelisteten Muster zu referenzieren, um möglichst viel Kontextinformationen zu geben. Ein anderes Forumsmitglied antwortet ihr, dass das Muster von Fowler [Fow02] dem gleichnamigen *Server Session State* der Cloud Computing Autoren Fehling et al. [FLR+14] entspricht. Er empfiehlt außerdem, für ihre Anwendung eher das *Database Session State* zu verwenden. Dadurch könne der *Load Balancer* jeden beliebigen Server auswählen, da dieser dann zustandslos gehalten werden kann. Lucy bedankt sich und verlinkt in ihrer Musterliste die Muster *Database Session State* und *Load Balancer* als ”häufig zusammen verwendet“ und die *Database Session State* Muster von Fehling et al. [FLR+14] und Fowler [Fow02] als ”übereinstimmend“. Nachdem weitere *PatternPedia*-Benutzer die Verlinkungen ihrer Musterliste als sinnvoll markiert haben, taucht das *Database Session State* in den Vorschlägen für das *Load Balancer* Muster auf. Als Lucy später noch das *Database Session State* Muster von Fehling et al. [FLR+14] in ihre Musterliste aufnimmt, zeigt *PatternPedia* bereits an, dass das Muster mit dem von Fowler [Fow02] übereinstimmt. Dies hat außerdem zur Folge, dass ihr weitere passende Entwurfsmuster zu beiden *Database Session State* Mustern vorgeschlagen werden.

In unserer erweiterten Vision lernt *PatternPedia*, dass Lucys Verlinkung mit hoher Wahrscheinlichkeit richtig ist. Hierbei ist es erlaubt, dass Aussagen wie “Muster X und Y stimmen überein” diskutabel bleiben. Dass ein Nutzer anderer Meinung ist und einen gegensätzlichen Link einfügt ist daher erlaubt. Angenommen, einer von 1 von 100 Nutzern ist dieser Meinung. Trotz dieser Dateninkonsistenz soll *PatternPedia* schlussfolgern, dass Lucy Vorschläge für beide Mustern angezeigt werden sollen. Inkonsistenzen im semantischen Web treten zum Beispiel auch durch die Integration von weiteren Datenquellen auf, die unabhängig voneinander entstanden sind [HVT05]. Schlussfolgerungen sind mit der richtigen Vorgehensweise trotz inkonsistenten Daten möglich [HVT05]. Dass Wissen subjektiv ist und Aussagen diskutabel bleiben sollten, sind einige der Herausforderungen für semantische Web, unter dessen Gesichtspunkten das Datenformat RDF entworfen wurde [Den12]. Im Hinblick auf Inkonsistenzen sind semantische Webtechnologien besonders gut geeignet, komplexe Wissensbasis über Thematiken wie Muster und Mustersprachen aufzubauen.

Eine Grundvoraussetzung für die Umsetzung der Vision sind ausreichend vorhandene Daten. Auch wenn andere semantische Wissensbasen wie DBPedia leicht integriert werden können, sind semantische Daten über Mustersprachen und konkreten Lösungen essentiell. Leider unterliegen viele Veröffentlichungen zu Mustersprachen dem Urheberrecht, sodass die Inhalte geschützt sind. Hier muss zusammen mit den Verlagen noch eine Lösung gefunden werden, sodass zumindest gekürzte Inhalte in die Wissensbasis mit aufgenommen werden können.

Literaturverzeichnis

[AAM15] A. A. Algosaibi, S. Albahli, A. Melton. „World Wide Web: A Survey of its Development and Possible Future Trends“. In: *The 16th International Conference on Internet Computing and Big Data-ICOMP*. Bd. 15. 2015, S. 79–84 (zitiert auf S. 22, 23).

[AIS+77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel. *A pattern language: Towns, buildings, construction*. Oxford University Press New York, 1977 (zitiert auf S. 9, 10, 12, 15, 19, 34).

[Ber09] T. Berners-Lee. *Linked Data - Design Issues*. https://www.w3.org/DesignIssues/LinkedData.html. Juni 2009. (Besucht am 28. 08. 2019) (zitiert auf S. 28).

[BF01] T. Berners-Lee, M. Fischetti. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. DIANE Publishing Company, 2001, S. 246 (zitiert auf S. 22, 23).

[BFM05] T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. http://www.rfc-editor.org/info/rfc3986. Jan. 2005. (Besucht am 09. 04. 2019) (zitiert auf S. 23).

[BHL+01] T. Berners-Lee, J. Hendler, O. Lassila et al. „The semantic web“. In: *Scientific american* 284.5 (2001), S. 28–37 (zitiert auf S. 9, 10, 12, 23).

[BKLW99] S. Busse, R.-D. Kutsche, U. Leser, H. Weber. „Federated information systems: Concepts, terminology and architectures“. In: *Forschungsberichte des Fachbereichs Informatik* 99.9 (1999), S. 1–38 (zitiert auf S. 19).

[BL15] J. Barzen, F. Leymann. „Costume Languages as Pattern Languages“. In: *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014*. Hrsg. von P. Baumgartner, R. Sickinger. PURPLSOC 2015, Juni 2015, S. 88–117 (zitiert auf S. 32, 35).

[BLK+09] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann. „DBpedia - A crystallization point for the Web of Data“. In: *Journal of Web Semantics*. The Web of Data 7.3 (Sep. 2009), S. 154–165. (Besucht am 27. 05. 2019) (zitiert auf S. 9).

[BPM+08] D. Beerrueta, J. Phipps, A. Miles, T. Baker, R. Swick. *Best Practice Recipes for Publishing RDF Vocabularies*. https://www.w3.org/TR/swbp-vocab-pub/. Aug. 2008. (Besucht am 29. 08. 2019) (zitiert auf S. 47).

[BTG+13] N. Bikakis, C. Tsinaraki, N. Gioldasis, I. Stavrakantonakis, S. Christodoulakis. „The XML and semantic web worlds: technologies, interoperability and integration: a survey of the state of the art“. In: *Semantic hyper/multimedia adaptation*. Springer, 2013, S. 319–360 (zitiert auf S. 9).

[Cho14] N. Choudhury. „World wide web and its journey from web 1.0 to web 4.0“. In: *International Journal of Computer Science and Information Technologies* 5.6 (2014), S. 8096–8100 (zitiert auf S. 22, 23).

[Den12] A. Dengel. *Semantische Technologien: Grundlagen. Konzepte. Anwendungen.* Springer-Verlag, 2012, S. 460 (zitiert auf S. 20, 21, 26–29, 33, 60).

[DFH11] J. Domingue, D. Fensel, J. A. Hendler. „Handbook of semantic web technologies“. In: Springer Science & Business Media, 2011 (zitiert auf S. 20, 21).

[DuC13] B. DuCharme. *Learning SPARQL: querying and updating with SPARQL 1.1*. O'Reilly Media, Inc., 2013 (zitiert auf S. 27).

[Esi19] R. Esin. „Konzept und Implementierung einer Patternlandkarte zur Navigation durch Patternsprachen“. Magisterarb. Universität Stuttgart, Aug. 2019 (zitiert auf S. 55, 57).

[FBB+14] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann. „Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains“. In: Bd. 7. 3&4. Xpert Publishing Services (XPS), 2014, S. 710–726 (zitiert auf S. 12, 59).

[FBB+16] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann, A. Hadjakos, F. Hentschel, H. Schulze. „Leveraging Pattern Application via Pattern Refinement“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*. epubli GmbH, 2016, S. 38–61 (zitiert auf S. 9, 59).

[FBBL17] M. Falkenthal, J. Barzen, U. Breitenbücher, F. Leymann. „Solution Languages: Easing Pattern Composition in Different Domains.“ In: *International Journal on Advances in Software* (2017), S. 263–274 (zitiert auf S. 15, 16).

[FBFL15] C. Fehling, J. Barzen, M. Falkenthal, F. Leymann. „PatternPedia – Collabroative Pattern Identification and Authoring“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*. epubli GmbH, Juni 2015, S. 252–284 (zitiert auf S. 58, 59).

[FBL18] M. Falkenthal, U. Breitenbücher, F. Leymann. „The Nature of Pattern Languages“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*. Okt. 2018, S. 130–150 (zitiert auf S. 10, 15, 16, 19, 32, 33, 38).

[Fer13] E. Fernandez-Buglioni. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. 1st. Wiley Publishing, 2013 (zitiert auf S. 34).

[FLR+11] C. Fehling, F. Leymann, R. Retter, D. Schumm, W. Schupeck. „An Architectural Pattern Language of Cloud-based Applications“. In: *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP)*. ACM, 2011, S. 2 (zitiert auf S. 15, 34).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014 (zitiert auf S. 11, 17, 18, 34, 39, 41, 54, 60).

[Fow02] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002 (zitiert auf S. 60).

[GBPM13] A. Gliozzo, O. Biran, S. Patwardhan, K. McKeown. „Semantic Technologies in IBM Watson“. In: *Proceedings of the Fourth Workshop on Teaching NLP and CL*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, S. 85–92 (zitiert auf S. 9).

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994 (zitiert auf S. 15, 17).

[Hen08] J. Hendler. „Web 3.0: Chicken Farms on the Semantic Web“. In: *Computer* 41.1 (Jan. 2008), S. 106–108 (zitiert auf S. 9, 22, 23).

[Hen09] J. Hendler. „Web 3.0 Emerging“. In: *Computer* 42.1 (Jan. 2009), S. 111–113 (zitiert auf S. 22).

[HK06] H.-W. Hilse, J. Kothe. *Implementing Persistent Identifiers: Overview of concepts, guidelines and recommendations*. Consortium of European Libraries, European Commission on Preservation und Access, Jan. 2006, S. 57 (zitiert auf S. 47).

[HM16] P. Hirmer, B. Mitschang. „FlexMash – Flexible Data Mashups Based on Pattern-Based Model Transformation“. en. In: *Rapid Mashup Development Tools*. Communications in Computer and Information Science. Springer International Publishing, 2016, S. 12–30 (zitiert auf S. 34).

[Hog13] A. Hogan. „Linked Data and the Semantic Web Standards“. In: *Linked Data and the Semantic Web Standards*. Chapman und Hall / CRC Press, 2013, S. 3–53 (zitiert auf S. 23, 24, 27, 28).

[HVT05] Z. Huang, F. Van Harmelen, A. T. Teije. „Reasoning with Inconsistent Ontologies“. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, S. 454–459 (zitiert auf S. 60).

[HW04] G. Hohpe, B. Woolf. *Enterprise integration patterns: designing, building, and deploying messaging solutions*. Boston: Addison-Wesley, 2004 (zitiert auf S. 9, 17, 33, 39, 65).

[KISV16] C. Köppe, P. S. Inventado, P. Scupelli, U. Van Heesch. „Towards Extending Online Pattern Repositories: Supporting the Design Pattern Lifecycle“. In: *Proceedings of the 23rd Conference on Pattern Languages of Programs*. PLoP ’16. The Hillside Group, 2016, S. 14–27 (zitiert auf S. 12, 17, 31, 32, 38, 46).

[KV09] M. Krötzsch, D. Vrandecic. „Semantic Wikipedia“. In: *Social Semantic Web: Web 2.0 – Was nun?* Hrsg. von A. Blumauer, T. Pellegrini. Springer Berlin Heidelberg, 2009, S. 393–421 (zitiert auf S. 12).

[Mai19] T. Mailund. „Why use markdown and pandoc“. In: *Introducing markdown and pandoc: using markup language and document converter*. Apress, 2019, S. 5–12 (zitiert auf S. 54).

[Pas04] T. B. Passin. *Explorer’s guide to the semantic web*. Bd. 18. Manning Greenwich, 2004, S. 281 (zitiert auf S. 23–28).

[PHP08] L. Pavlič, M. Hericko, V. Podgorelec. „Improving design pattern adoption with Ontology-Based Design Pattern Repository“. In: Juli 2008, S. 649–654 (zitiert auf S. 58).

[UM12] M. Unterstein, G. Matthiessen. „Datensichten in SQL“. In: *Relationale Datenbanken und SQL in Theorie und Praxis*. Springer Berlin Heidelberg, 2012, S. 195–206 (zitiert auf S. 19).

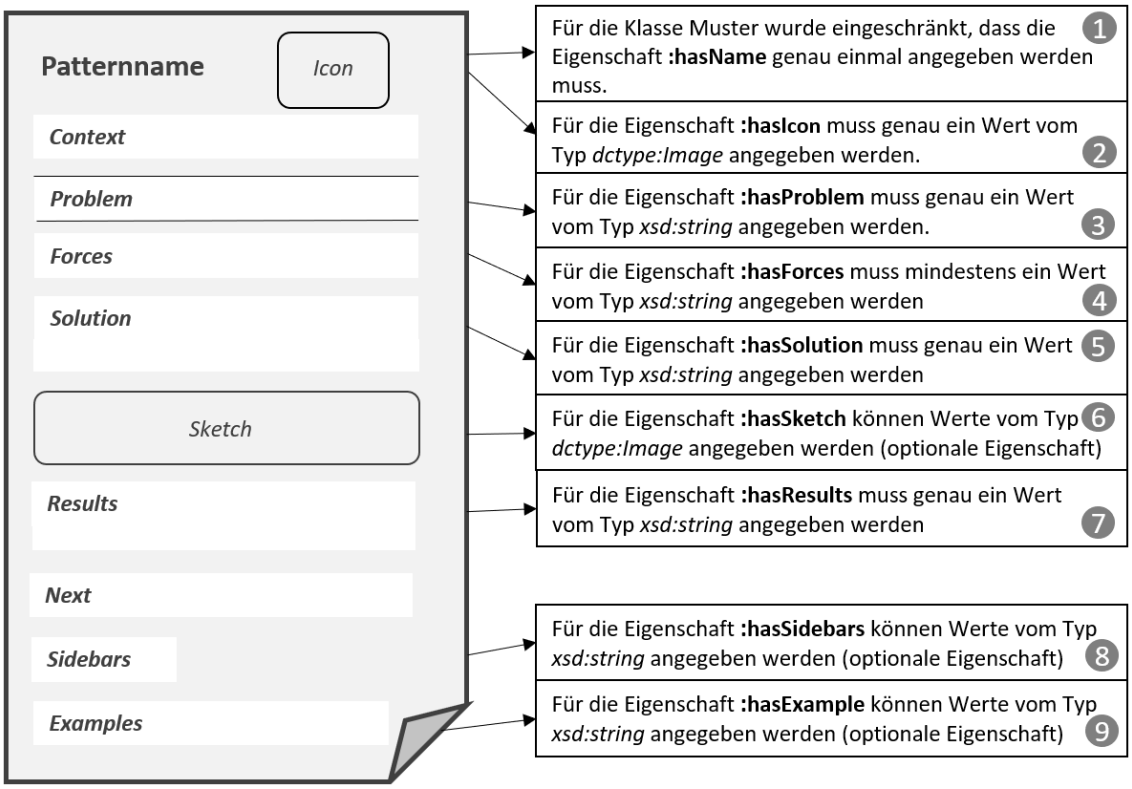
[WBK+16] S. Warburton, J. Bergin, C. Kohls, C. Köppe, Y. Mor. „Dialogical assessment patterns for learning from others“. In: *Proceedings of VikingPLoP 2016 - 10th Northern Conference on Pattern Languages of Programmes*. Nov. 2016 (zitiert auf S. 18, 34).

[Yu14] L. Yu. *A Developer's Guide to the Semantic Web*. 2. Aufl. Springer Berlin Heidelberg, 2014, S. 829 (zitiert auf S. 26, 27, 35).

[Zdu07] U. Zdun. „Systematic pattern selection using pattern language grammars and design space analysis“. In: *Software: Practice & Experience* 37 (Juli 2007), S. 983–1016 (zitiert auf S. 34).

[Zim95] W. Zimmer. „Relationships Between Design Patterns“. In: *Pattern Languages of Program Design*. Addison-Wesley Publishing Co., 1995, S. 345–364 (zitiert auf S. 34).

Anhang



Ergänzende Abbildung A1: Aufbau eines *Enterprise Integration* Musters von Hohpe und Woolf [HW04] (links) und sich daraus ergebende Einschränkungen für ein Muster der Mustersprache (rechts). Für den Abschnitt *Next* indem weiterführende Muster beschrieben werden wurden keine Einschränkungen abgeleitet, da die Referenzen zu diesem Muster durch Musterrelationen ausgedrückt werden können.

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