Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Einsatz von semantischen Verfahren zur Unterstützung der SOA Governance

Alexander Blehm

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing habil. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf. Jan Königsberger
Beginn am:	1. Dezember 2016
Beendet am:	1. Juni 2017
CR-Nummer:	I.2.4, H.3.5

Kurzfassung

Die SOA Governance hat zum Ziel alle möglichen Aktivitäten serviceorientierter Architekturen zu unterstützen. Dies beinhaltet die Kontrolle und Regulation des Serviceangebots, der Servicenutzung und der Wiederverwendung von Services. Damit Services in einer serviceorientierten Architektur besser gefunden und ihre Funktionen besser verstanden werden, muss eine Semantik für diese Services definiert werden. Das Semantic Web bietet hierfür die nötigen Sprachen und Technologien. Diese Arbeit stellt ein Konzept vor, das serviceorientierte Architekturen mit einer SOA Governance semantisch unterstützen kann. Es werden semantische Technologien untersucht und Lösungen entwickelt, wie diese die Aktivitäten im Rahmen der SOA Governance unterstützen können. Dazu werden insbesondere eine Ontologie und semantische Regeln definiert, die von Software-Tools als Teil einer SOA adoptiert werden können. Dieses Konzept wird im Anschluss als Erweiterung eines SOA Governance Repositories implementiert, und es zeigt, wie die Ontologie und semantischen Regeln effektiv genutzt werden können.

Inhaltsverzeichnis

1. Einleitung

Das World Wide Web (WWW) wird durch den Einsatz von semantischen Verfahren und serviceorientierten Architekturen (SOA) weiterentwickelt. Informationen aus dem WWW sind nicht mehr nur explizit definiert, sondern können nun auch implizit durch die Bedeutung bestimmter Informationen erweitert werden. Das bedeutet, dass aus der Bedeutung (Semantik) einer zugrundeliegenden Informationsmenge neues Wissen hergeleitet werden kann, das zuvor nicht explizit in dieser Informationsmenge gegeben war. Der Prozess aus so einer Informationsmenge neues Wissen mit einer Semantik herzuleiten wird „Inferieren“ genannt. Somit hat sich das WWW zu einem Semantic WWW weiterentwickelt. Das Semantic WWW ist auch im Kontext von Unternehmen und serviceorientierten Architekturmustern interessant. Ein solches Architekturmuster wurde dabei in den letzten Jahren in Unternehmen etabliert, um die eigene IT-Abteilung flexibler zu machen und schneller auf sich ständig ändernde Geschäftsprozesse zu reagieren. Die daraus resultierenden Softwarelösungen und Datenbankstrukturen bilden damit ein zentrales Element im Unternehmen. Besonders auf Datenebene können Datensätze nun durch sogenannte Ontologien erweitert werden und bekommen dadurch neben den explizit vorliegenden Daten eine zusätzliche Bedeutung. Eine Ontologie ist eine Art Datenmodell, das es erlaubt, nicht nur die faktischen Daten zu beschreiben sondern auch eine Semantik für diese Daten zu definieren. Dies bringt sowohl innerhalb eines Unternehmens als auch unternehmensübergreifend Vorteile. Innerhalb eines Unternehmens können auf diese Weise neue Erkenntnisse aus einem Datensatz gewonnen werden. Die Software-Kommunikation zu anderen Unternehmen ist dadurch ebenfalls vereinfacht, weil Software nun „versteht“ welche Bedeutung fremde Daten haben und wie sie mit den eigenen Unternehmensdaten in Verbindung stehen. Diese Weiterentwicklung des WWW wird auch Semantic Web genannt [Hit08]. Das Semantic Web soll auch unabhängig von Unternehmen Vorteile für dessen Benutzer bringen. In der Literatur liegen dabei die Schwerpunkte für solche Vorteile bei (Web-)Services und der Informationssuche. Durch die semantische Erweiterung der Servicebeschreibungen sollen Services und deren Funktionen von potenziellen Service-Konsumenten besser verstanden werden und auch allgemeine Informationen bei der Nutzung von Suchmaschinen besser auffindbar sein [KYA16; ML16; VDR16]. Es ist also naheliegend zu untersuchen, wie semantische Technologien auch im Umfeld serviceorientierter Architekturen eingesetzt werden können und welche Aspekte einer SOA Governance dadurch unterstützt werden können.

1.1. Motivation

Das Ziel einer SOA Governance ist es, alle Aktivitäten serviceorientierter Architekturen durch Kontrolle und Regulation zu unterstützen. Dies beinhaltet unter anderem die Unterstützung des Serviceangebots, der Servicenutzung, die Wiederverwendung von Services so wie das flexible Anpassen von Services an die Geschäftsprozesse und umgekehrt. Im Zentrum einer SOA steht ein System, das Services mit allen dazugehörigen Inhalten registrieren und für Servicekonsumenten bereitstellen kann. In der Literatur wird ein solches System Service Registry oder Service Repository genannt [JBW11]. Am Institut für Parallele und Verteilte Systeme der Universität Stuttgart wurde ein Service Repository implementiert, das sich am Konzept der SOA Governance orientiert. Dieses Service Repository wird deshalb auch SOA Governance Repository genannt. Um die von der SOA Governance geforderte Flexibilität zu ermöglichen, wurde die Datenbank des SOA Governance Repositories von einer relationalen Datenbank zu einer NoSQL-Datenbank umgestellt [Miy15]. Die klassischen, relationalen Datenbanken konnten nur begrenzt zur Flexibilität im Datenmodell beitragen. Beispielsweise führt eine Änderung in Geschäftsprozessen oft auch zu einer Änderung im Datenmodell, welches bei relationalen Datenbanken nur mit einer aufwendigen Strukturänderung der Daten und Datenbanktabellen zu realisieren ist. Mit NoSQL-Datenbanken lässt sich die Datenhaltung flexibler gestalten, weil die Daten keiner strengen Strukturdefinition unterliegen. Graph-Datenbanken sind eine Form der NoSQL-Datenbanken [Zaf+16]. Eine Form von Graph-Datenbanken sind Triple-Stores, die besonders gut dafür geeignet sind, um Objekte und vor allem Beziehungen zwischen Objekten zu beschreiben. Zusätzlich erlaubt das Triple-Format durch semantische Annotationen erweitert zu werden und gewinnt dadurch an Bedeutung. In anderen Worten wird die klassische, faktische Datenhaltung durch eine Ontologie erweitert. Mit Hilfe einer Ontologie lassen sich dann auch zusätzliche (implizite) Informationen herleiten, die nicht explizit im Datensatz vorliegen [Hit08]. Beispielsweise lässt sich über eine längere Beziehungskette zwischen zwei Objekten eine neue Beziehung herleiten, die abkürzend beschreibt, wie diese zwei Objekte indirekt verbunden sind. Diese indirekte Beziehung ist im Zusammenhang mit ihrer Bedeutung eine semantische Beziehung, die für Menschen problemlos verständlich ist. Damit auch eine Software diese Beziehungen herleiten und „verstehen“ kann, müssen Regeln definiert werden, die dieses implizite Wissen herleiten können. Der Prozess, eine solche Ontologie und Regeln für eine Datenhaltung zu definieren und damit eine zusätzliche Bedeutung aus dem vorliegenden Datensatz herzuleiten, wird im Kontext dieser Arbeit „semantische Unterstützung“ genannt. Es gibt viele Möglichkeiten semantisches Wissen zu nutzen. Dies fängt beispielsweise bei der Erkennung von Synonymen oder Domänen-übergreifend ähnlichen Daten an und setzt sich über das Herleiten von neuen, semantisch sinnvollen Objekten und Beziehungen fort.

1.2. Ziele der Arbeit

Ziel dieser Arbeit ist es vorhandene Ansätze und Technologien im Bereich der semantischen Unterstützung serviceorientierter Architekturen zu betrachten, damit ein klares Verständnis

für die Möglichkeiten einer semantischen Unterstützung besteht. Im Anschluss soll ein Konzept erarbeitet werden, das zunächst die Merkmale einer SOA Governance identifiziert, die gegebenenfalls semantisch unterstützt werden können. In diesem Konzept werden dann eine Ontologie und semantische Regeln beschrieben, die den Datensatz eines Service Repositories um eine Bedeutung erweitern und daraus implizites Wissen herleiten. Eine Implementierung in Form einer Erweiterung des SOA Governance Repositoriy soll schließlich zeigen, wie diese semantische Unterstützung realisiert wird und effektiv genutzt werden kann.

1.3. Aufbau der Arbeit

Der Aufbau der Arbeit ist wie folgt:

Kapitel 2 – Grundlagen beschreibt häufig verwendete und grundlegende Begriffe im Kontext dieser Arbeit. Hier werden auch Technologien beschrieben, die zu einer semantischen Unterstützung beitragen können.

Kapitel 3 – Verwandte Arbeiten betrachtet verwandte Ansätze einer semantischen Unterstützung im Bereich von Services und Service Repositories.

Kapitel 4 – Konzept ist der Kern der Arbeit. Hier werden Merkmale einer SOA Governance ermittelt und im Anschluss eine Ontologie und semantische Regeln beschrieben, die bestimmte Aspekte der zuvor identifizierten Merkmale unterstützen.

Kapitel 5 – Implementierung beschreibt, wie die Ontologie und semantischen Regeln umgesetzt werden können. Dies wird in Form einer Erweiterung des SOA Governance Repositories implementiert.

Kapitel 6 – Zusammenfassung und Ausblick fasst die wichtigsten Erkenntnisse dieser Arbeit zusammen und gibt einen kurzen Ausblick zu möglichen fortführenden Themen und der Entwicklung der semantischen Unterstützung.

2. Grundlagen

In diesem Kapitel werden grundlegende Begriffe dieser Arbeit erklärt. Ebenso werden Technologien beschrieben, die für die semantische Unterstützung relevant sind, denn das Erstellen einer Ontologie und den dazugehörigen semantischen Regeln ist je nach Triple Store-Technologie unterschiedlich und nicht immer einheitlich gelöst. Erkenntnisse zu semantischen Reasonern und verschiedenen Reasoner-Typen sind im Unterkapitel 2.8 zu finden.

2.1. Services

Ein Service ist ein geläufiger Anglizismus eines Dienstes aus der Informatik. Er beschreibt eine Software oder allgemeiner eine technische Einheit, die eine bestimmte Funktionalität erfüllt. Zudem ist die Schnittstelle eines Services klar definiert und der Service selbst ist über einen Endpunkt ansprechbar. Die interne Programmlogik ist für potenzielle Service-Nutzer (man sagt auch Service-Konsumenten) versteckt, es sind nur bestimmte Funktionen eines Services nutzbar, die über die Schnittstellenbeschreibung freigegeben sind. Häufig ist in der Literatur hinter dem Begriff Service ein Webservice gemeint, der seine Schnittstelle über ein Netzwerk verfügbar macht.

2.2. Serviceorientierte Architektur

Eine serviceorientierte Architektur (SOA) ist ein Architekturmuster, das im Bereich von verteilten Systemen und Services Anwendung findet. Die Implementierungen von Services in einer SOA sind an Geschäftsprozessen orientiert. Dabei soll eine SOA helfen, Services so zu koordinieren, dass sich Services flexibel an Änderungen in Geschäftsprozessen anpassen können. Die Funktionen von Services sind in so einer Architektur idealerweise so feingranular, dass sie für das Ausführen verschiedener Geschäftsprozesse aggregiert werden können (als Service-Kompositionen).

2.3. SOA Governance

Die Menge an Services, Service-Providern, Service-Konsumenten, Verträgen und anderen relevanten Dokumenten und Objekten in einer SOA können dazu führen, dass eine SOA sehr unübersichtlich wird. Dies erschwert auch das Erzwingen von Sicherheits- und Qualitätseigenschaften, so wie das Monitoring verschiedener Services und Service-Kompositionen. Damit es eine Kontrolle und Regulation für nicht-funktionale Eigenschaften in einer SOA gibt, wurde das Konzept der SOA Governance eingeführt. Die Aufgaben einer SOA Governance lassen sich wie im folgenden beschreiben:

- SOA-Ressourcenüberwachung und Konfiguration
- Verwaltung und Erfüllung nicht-funktionaler Anforderungen
- Verbessertes Life Cycle Management
- Verwaltung und Überwachung von Geschäftsprozessen

Mit Hilfe der SOA Governance können alle SOA-Ressourcen besser überwacht und konfiguriert werden, sowie viele nicht-funktionale Anforderungen an eine SOA erfüllt werden [LDB09].

2.4. SOA Governance Repository

Das SOA Governance Repository (SGR) ist ein Service Repository, das die Aufgaben der SOA Governance unterstützt. Es basiert auf dem SOA-Governance-Meta-Model (SOA-GovMM), das 11 Kernanforderungen an die SOA Governance definiert [KSM14]:

- Service Life Cycle Management
- Verwaltung von Konsumenten
- Verwaltung von Meta-Daten
- Eine Struktur für Organisationen
- Verwaltung von Portfolios
- Einhalten von Architekturstandards
- Eine Governance-Hierarchie
- Ein Funding-Model (Preismodell)
- Service-Überwachung
- Messbarkeit einer (Service-)Reife
- Verwalten von Geschäftsobjekten (Business Objects)

Das SOA Governance Repository wurde im Rahmen mehrerer studentischer Arbeiten implementiert, v.a. [Bil16; Kra16a; Kra16b; Miy15]. Durch diese Arbeiten können nun die Vorteile einer RDF-Datenbank genutzt werden. Beispielsweise kann das Datenmodell dank des Triple-Formats sehr flexibel erweitert und wiederverwendet werden. Zuletzt ist es nun auch möglich das Governance Repository semantisch zu unterstützen, da die Datenhaltung im RDF-Format einfach durch semantische Annotationen erweitert werden kann und ein Reasoning über den Datensatz möglich ist [KM16]. Eine detaillierte Vorstellung der Architektur des SGR ist in Kapitel 5 zu finden.

2.5. Semantische Technologien

Es gibt verschiedene semantische Technologien, die im Kontext dieser Arbeit relevant sind. Für die semantische Datenhaltung gibt es Modelle zur Beschreibung von Daten mit semantischer Annotation. Zusätzlich existieren für diese Art der Datenhaltung eigene Abfragesprachen, um gezielt Datensätze auszulesen oder Daten zu manipulieren. In den folgenden Unterkapiteln wird genauer auf die Beschreibungsmodelle für semantische Datenhaltung und auf die dazugehörigen semantischen Abfrage- und Manipulationssprachen eingegangen, die für diese Arbeit relevant sind.

2.5.1. Semantische Datenhaltung

Ein Modell zur semantischen Datenhaltung wird auch **Ontologie** genannt. Sofern es möglich ist, den Datensatz mit zusätzlichen Informationen für eine Bedeutung der Daten zu erweitern (semantisch zu annotieren), so spricht man von einer Ontologie. Durch diese semantische Annotation wird es ermöglicht, dass zusätzliches Wissen aus einem Datensatz gewonnen werden kann, das zuvor nicht explizit im Datensatz gegeben war. Der Prozess implizites Wissen explizit zu machen wird auch „Inferieren“ genannt [Jep09]. Das Inferieren solcher semantischen Informationen ist die Hauptaufgabe eines Reasoners und wird im Unterkapitel 2.8 genauer vorgestellt.

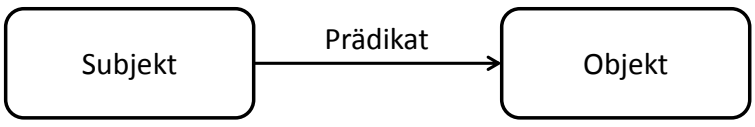

Abbildung 2.1.: Aufbau eines Triples in RDF

Das **Resource Description Framework (RDF)** ist eines der bekanntesten Modelle für eine semantische Datenhaltung [W3C02]. Die Kernstruktur jedes Datensatzes im RDF ist ein Triple dessen Struktur in Abbildung 2.1 gezeigt wird. Es besteht aus einem Subjekt, einem

Prädikat und einem Objekt. Das Objekt kann wieder über ein Prädikat mit einem weiteren Objekt verbunden sein und auf diese Weise entsteht eine Graph-Struktur. Die einzelnen Objekte und Beziehungen eines Triples in RDF werden Ressourcen genannt und sind durch einen Internationalized Resource Identifier (IRI) beschrieben. Mit Hilfe dieser IRIs kann die Herkunft, Bedeutung und der übliche Nutzen einer Ressource definiert werden. RDF erlaubt außerdem die Wiederverwendung von Vokabularen anderer Domänen, wobei ein Vokabular eine Sammlung von Bezeichnern ist, die innerhalb einer Domäne verwendet wird. IRIs dienen auch hier als Identifikator für diese Vokabulare. Durch diese Wiederverwendung kann die Interoperabilität zwischen zwei Systemen gestärkt werden, weil die Datensätze durch ein ähnliches Vokabular beschrieben sind. Mit dieser Grundstruktur bildet jeder RDF Datensatz einen Graphen mit Objekten als Knoten und Prädikaten als Kanten dieses Graphen. Zu der Art der Graph-Datenbanken, die RDF unterstützen, wird im Unterkapitel 2.7 weiteres erklärt.

Die **Web Ontology Language (OWL)** ist eine Ontologiesprache, die auf RDF aufbaut und primär in Form von RDF-Dokumenten definiert und geteilt wird. Im Kontext dieser Arbeit ist mit OWL die zweite, spezifizierte Version OWL2 gemeint. OWL unterscheidet im Vergleich zu RDF strenger in Klassen, Eigenschaften und Individuen. In RDF können Ressourcen sowohl Klassen als auch Instanzen derselben Klasse sein. OWL würde solche Fälle nicht zulassen: in OWL sind Klassen strikt von Instanzen (auch Individuen genannt) getrennt. Das Ziel von OWL ist es, eine strengere Überprüfung der Datenkonsistenz zu ermöglichen und möglichst einfach implizites Wissen mit Hilfe von Reasonern (siehe Kapitel 2.8) zu inferieren [W3C12]. Neben OWL gibt es noch eine Sprache zur semantischen Spezifikation von Webservices namens OWL-S. Die Ziele von OWL-S sind es, für Web Services eine automatische Service Discovery und Service Invocation zu ermöglichen. Auch Service-Kompositionen sollen mit Hilfe von OWL-S automatisiert werden können [W3C11].

2.5.2. Semantische Sprachen

Damit die Daten einer Ontologie auch ausgelesen und manipuliert werden können, gibt es Sprachen, die mit diesen Daten arbeiten. Das rekursive Akronym **SPARQL Protocol and RDF Query Language (SPARQL)** beschreibt eine graphenbasierte Abfragesprache für RDF. SPARQL arbeitet auf der Grundlage von Schlüsselwörtern und Triplen, die entweder explizite Daten auslesen können, oder Triple-Beschreibungen mit Variablen nutzen, um bei der Abfrage mehrere Datensätze gleichzeitig auszulesen. Ursprünglich war die erste SPARQL Version nur in der Lage RDF-Datensätze auszulesen, aber nicht zu manipulieren. Mit der Einführung von SPARQL-Update und der daraus resultierenden neuen SPARQL Version (SPARQL 1.1) ist nun auch die Manipulation von Daten möglich. Das Vokabular ist weitestgehend gleich geblieben, es gibt allerdings einige neue Operatoren zur Datenmanipulation [W3C03]. Im Kontext dieser Arbeit wird die spezifizierte Version SPARQL 1.1 zum Auslesen und Manipulieren von Daten mit „SPARQL“ abgekürzt.

Die **Semantic Web Rule Language (SWRL)** ist eine Sprache zum Beschreiben von semantischen Regeln und Inferieren neuer Datensätze. SWRL arbeitet auf Basis von OWL und RuleML,

was eine XML-Sprache zur Beschreibung von Datentransformationen (semantischer Daten) ist. Der grundlegende Aufbau jeder Regel in SWRL ist eine logische Verknüpfung mit einer Implikation. Diese Sprache soll es in Kombination mit kompatiblen Reasonern ermöglichen, komplexere semantische Zusammenhänge in OWL zu beschreiben und damit neues Wissen aus einem OWL-Datensatz herzuleiten [W3C05].

Eine weitere Sprache zum Herleiten von semantischen Beziehungen und Objekten ist die **SPARQL Inferencing Notation (SPIN)**. Auf Basis der SPARQL-Sprache erlaubt es SPIN bestimmte semantische Regeln (Spin-Rules) und Datenbeschränkungen (Spin-Constraints) zu definieren, die semantisches Reasoning auf RDF-Datensätzen ermöglichen. Jede SPIN-Regel wird dabei grundsätzlich an eine Ressourcen-Klasse gebunden und vor dem Ausführen in entsprechende SPARQL-Update-Befehle übersetzt. Ein kompatibler Reasoner führt diese Regeln dann aus, oder prüft im Falle eines Spin-Constraints ob die Beschränkungsregel (beispielsweise Beschränkung zu einem Datentyp) gebrochen wurde und unternimmt die definierten Maßnahmen [W3C09].

2.6. Taxonomien

Eine Taxonomie ist eine hierarchische Klassifikation von Dingen oder Konzepten [DUD17]. Im Kontext eines Informationssystems definiert sie eine baumartige Struktur mit Kategorien, um eine Ordnung für bestimmte Ressourcen zu beschreiben. Taxonomien werden oft mit Ontologien zusammen verwendet oder sogar verwechselt. Jedoch ist eine Ontologie komplexer und beschränkt sich nicht nur auf eine baumartige Kategorisierung von Ressourcen. Der Unterschied von Taxonomien zu Ontologien kann so beschrieben werden: Eine Taxonomie kann eine hierarchische Klassifikation von Daten beschreiben. Eine Ontologie kann dazu noch mit Hilfe der Bedeutungen in dieser Klassifikation (wie z.B. Objekte innerhalb einer Kategorie haben eine gewisse Ähnlichkeit) zusätzliche semantische Annotationen machen. Mit dieser zusätzlichen Annotation können neue Beziehungen inferiert werden und Objekte beispielsweise in weitere Kategorien eingeordnet werden, in die sie semantisch passen würden. Weil eine Taxonomie mit der Klassifikation von Objekten auch eine bestimmte Semantik impliziert (Objekte sind semantisch gruppiert), kann eine Taxonomie auch als eine mögliche Teilform einer Ontologie gesehen werden.

2.7. Triple Stores

Semantische Technologien sind zusammen mit der Einführung von NoSQL-Datenbanken entstanden. NoSQL-Datenbanken bieten eine alternative Datenhaltung zu den klassischen relationalen Datenbanken. Es gibt viele Typen von NoSQL-Datenbanken, eine davon sind Graph-Datenbanken. Triple Stores sind wiederum ein besonderer Typ von Graph-Datenbanken. Ein Triple ist das grundlegende Format dieser Datenbanken. Ein vollständiges Triple wird

innerhalb einer Datenbank auch als Aussage (Statement) bezeichnet. Bekannte Triple Stores sind RDF4j (Sesame)[1], Apache Jena[2], Stardog[3] AllegroGraph[4] und einige andere, die im Rahmen dieser Arbeit nicht alle aufgezählt werden können.

2.8. Reasoner

Reasoner (oder semantische Reasoner) automatisieren das Herleiten von implizitem Wissen aus einem expliziten Datensatz. Damit ein Reasoner solche Informationen herleiten kann, muss der entsprechende Datensatz entweder semantisch annotiert sein, oder es müssen Regeln definiert werden, die dem Reasoner erklären, unter welchen Bedingungen neue Objekte und Beziehungen hergeleitet werden können. Es gibt keine offizielle Klassifikation für die Arten von Reasonern, weil jede Reasoner-Funktionalität stark von den definierten Annotationen und Regeln abhängig ist. Es gibt jedoch Ansätze diese in „Forward-Chaining"-Inferenz-Maschinen und probabilistic Reasoner zu unterteilen. Der Fokus der Funktionalität von „Forward-Chaining"-Inferenz-Maschinen liegt im Herleiten von Wissen auf Basis von semantischen Regeln. Dabei gibt es eine Bedingung, die definiert wird, und eine Folgerung, die bei Erfüllung dieser Bedingung neue Inferenzen herleitet. Probabilistic Reasoner versuchen in einem Datensatz auf die Frage: „Wie hoch ist die Wahrscheinlichkeit, dass ein gegebenes Dokument/eine gegebene Ressource bei einer Anfrage (query) als relevant beurteilt wird?" einen präzisen Wahrscheinlichkeitswert zu errechnen. Von beiden Reasoner-Typen gibt es Abwandlungen und Sonderformen, die eventuell auch in keiner der beiden Gruppen wiederzufinden sind. Eine ausführliche Liste von Reasonern bietet die University of Manchester an [SM01].

[1] RDF4j: http://rdf4j.org/
[2] Jena: http://jena.apache.org/
[3] Stardog: http://www.stardog.com/
[4] AllegroGraph: https://franz.com/agraph/allegrograph/

3. Verwandte Arbeiten

In diesem Kapitel werden Arbeiten vorgestellt, die eine Relevanz für das Thema dieser Arbeit haben. Dazu gehören Forschungsarbeiten, die semantische Technologien im Bereich von serviceorientierten Architekturen einsetzen. Die gewonnenen Erkenntnisse, die aus diesen thematisch relevanten Arbeiten entstehen, dienen als Orientierung und Inspiration für den erzielten Nutzen des Konzepts aus Kapitel 4.

3.1. Strategien zur Implementierung einer semantischen SOA

Die Autoren aus [SBB13] beschreiben beispielsweise allgemeine Ansätze, um aus einer Implementierung einer SOA eine semantische SOA zu implementieren. Dabei gehen sie zuerst darauf ein, welche Abstufungen von Ontologien es gibt:

Generelle Ontologien beziehen sich auf generelle Konzepte, wie mathematische Logik oder das Raum- und Zeit-Verständnis. Diese Ontologien wären auch außerhalb der genutzten Domäne oder des Unternehmens verständlich und man kann diese Ontologien unabhängig von der zeitlichen Reihenfolge der Implementierung erstellen.

Domänenspezifische Ontologien beschreiben im Rahmen einer Domäne oder eines Unternehmens Konzepte, die sich nahe an Geschäftsprozessen orientieren. Setzt man die Implementierung einer domänenspezifischen Ontologie an den Anfang, so wird ein Ansatz verfolgt, der „oben“ bei der Geschäftslogik beginnt und bei den konkreten Services endet (Top-Down).

Applikationsspezifische Ontologien sind die spezifischsten Ontologien, die nur im direkten Kontext einer Applikation verstanden werden. Wird hier mit der Implementierung begonnen, so werden zuerst alle Services in einer Service-Map identifiziert und dann „von unten her“ die Domänen-Ontologien gestaltet (Bottom-Up).

Zusätzlich gibt es noch Mischformen der zuvor genannten Ansätze, beispielsweise ist die Oustide-In-Strategie dazu gedacht, sowohl von der Geschäftsebene als auch bei den Services zu beginnen, endet dann jedoch in einer komplexen Phase, wenn bereits implementierte Services und definierte Geschäftsprozesse einander zugeteilt werden müssen. Die Middle-Out-Strategie beginnt mit einer Beschreibung der Ontologie im Zentrum und der darauf

folgenden Implementierung („middle-up“ und „middle-down“). Bei diesem Ansatz ist die Implementierung aufwendiger, weil die semantischen Strukturen, die in Geschäftsprozessen und Services verwendet werden, aufeinander abgestimmt sein müssen.

Das erarbeitete Konzept in dieser Arbeit soll domänenunabhängig sein und für alle Systeme angewendet werden können, die Services registrieren und verwalten. Da fast jede beschriebene Strategie von [SBB13] von einer Domäne abhängig ist und Auswirkungen auf die Entwicklung einer semantischen SOA hat, sind diese beschriebenen Strategien nicht direkt anwendbar. Der verfolgte Ansatz dieses Konzepts kann jedoch in einer Weise als Middle-Out-Strategie gesehen werden, da die semantische Unterstützung in der „unteren Hälfte“ bei den Services zuerst implementiert wird (middle-down). In jeder Domäne, in der das vorgestellte Konzept dieser Arbeit dann in einer Anwendung umgesetzt wird, könnte die semantische Unterstützung auf Seiten der jeweiligen Geschäftsprozesse nachgezogen werden (middle-up).

3.2. Einsatz semantischer Technologien bei Service Repositories

Im Bereich von Service Repositories beschreiben die Autoren aus [LX10] ein Umfeld, in dem ein Workflow Services über einen Enterprise Service Bus (ESB) aufruft. Der ESB ruft dann eine Matchmaking-Komponente auf, die für einzelne Workflow-Schritte Services in einem Service Repository finden soll. Wenn ein geeigneter Service gefunden wurde, ist die Aufgabe des ESB für den jeweiligen Workflow-Schritt beendet. Sollte kein geeigneter Service gefunden sein, so wird eine weitere Komponente (Semantic Web Service Compositor) aufgerufen, um mit Hilfe von semantisch annotierten Services eine geeignete Service-Komposition zu generieren, die die gewünschte Funktion erfüllen soll. Diese Service-Komposition (aus potenziell mehreren Services) wird dann auch als neuer Service in dem Service Repository registriert. Die Servicebeschreibungen für die Suche und für die Annotation der semantischen Services in dem Service Repository werden mit OWL-S definiert. Eine andere Arbeit von [XLPf08] nutzt ebenfalls OWL-S, um Services in einer Service Registry mit einer Semantik zu annotieren und dadurch automatisches Service-Matchmaking zu ermöglichen. Zusätzlich wird dort auch beschrieben, wie die manuelle Service-Suche für Benutzer der Service Registry vereinfacht wird.

Das Ziel in dieser Arbeit ist es nicht, ein automatisches Matchmaking für Services zu ermöglichen, jedoch wird es mit einer steigenden Anzahl von Services schwieriger, Services wiederzufinden. Da in einer Anwendung mit Services eine sehr hohe Anzahl an Services unterstützt werden sollten, geben diese Paper für das Konzept dieser Arbeit die Inspiration, eine Service-Suche umzusetzen. Eine semantische Such- und Filterfunktion ist der erste Schritt zu einer Service Discovery.

Neben der Automatisierung der Service Discovery wird in [ZS09] ein weiterer Nutzen einer semantischen SOA deutlich gemacht. Mit Hilfe der semantischen Beschreibung können Services

und ihre Funktionen auch über verteilte Systeme hinweg erkannt werden. Fremde Service Repositories sollen trotz anderer Terminologien und Servicebeschreibungen die eigenen Services mit Hilfe von Vermittlern verstehen können. Dabei werden verschiedene Ansätze für solche semantischen Vermittler verfolgt. Es gibt Vermittler die zwischen zwei Systemen lediglich die Ziele zweier Services vergleichen, andere Vermittler vergleichen auf Datenebene genutzte Ontologien. Es gibt Vermittler, die die Beschreibung von Webservices direkt vergleichen und damit die Funktionsbeschreibungen dieser Services. Schließlich gibt es auch noch Mischformen aller zuvor genannten Vermittler.

Vermittler zwischen verteilten Systemen sind kein Bestandteil des Beitrags dieser Arbeit, jedoch kann durch die Benutzung eines standardisierten Vokabulars eine Kommunikation zwischen mehreren semantischen Systemen vereinfacht werden. Das bedeutet, dass die Ontologie und alle semantischen Erweiterungen und Annotationen vor der Implementierung auf ein passendes, standardisiertes Vokabular geprüft werden müssen. Es sollen Vokabulare wiederverwendet werden, die sich im Semantic Web etabliert haben und gegebenenfalls bereits von anderen Anwendungen als Teil einer SOA verwendet werden. Dadurch wird die modellierte Ontologie im Konzept und jede Umsetzung davon in solch einer Anwendung mit sprachlichen Standards ausgestattet und kann vereinfacht mit anderen Server- und Klient-Anwendungen, die ein Service-Repository nutzen, interagieren.

3.3. Weitere Einsatzmöglichkeiten semantischer Technologien

Semantische Technologien finden auch außerhalb von Service Repositories viele Anwendungsfälle. Im Bereich der SOA-Lebenszyklus-Verwaltung beschreiben Seedorf et al. [SNK09] wie ein OWL-Reasoner eingesetzt werden kann, um „Semantic Traceability“, also das Verfolgen von Abhängigkeiten zwischen beispielsweise Geschäftsprozessen und Services, zu ermöglichen. Es gibt dabei vier Abhängigkeitstypen zu unterscheiden:

Geschäftsmodelle zur Quelle und deren Beziehungen festzuhalten, wobei eine Quelle in dem Sinne Stakeholder sind, die an einem bestimmten Prozess oder in einem Vertrag im Sinne einer SOA hinterlegt sind.

Zwischen Geschäftsobjekten sind es meistens Geschäftsprozesse und Services. Dies findet am häufigsten Gebrauch, um bei Änderungen schnell zu reagieren und relevante Services zu identifizieren.

Den Service-Lebenszyklus und damit alle relevanten Objekte entlang der Serviceentwicklung zu verfolgen. Dazu gehören beispielsweise frühe Dokumente, wie die Anforderungsspezifikation, die ersten Entwürfe, die Implementierung und Testergebnisse und vieles mehr.

Abhängigkeiten zwischen Services zu verfolgen, um zu wissen, welche Services andere nutzen, oder welche Services komplett aus einer Komposition anderer Services bestehen.

Dafür wird in der Architektur eine semantische Schicht mit einer Ontologie in OWL angelegt, um diese Abhängigkeiten festzuhalten und bei Bedarf zu aktualisieren. Bei Änderungen im Geschäftsmodell können auf diese Weise sofort die verantwortlichen oder relevanten Gegenstücke (wie Services oder Verträge) identifiziert werden, die gegebenenfalls geändert oder ausgetauscht werden müssen. Ein Semantic Traceability ist im Grunde eine Abkürzung einer längeren Beziehungskette zwischen zwei Objekten. Diese Abkürzung ist sinnvoll, weil sie mit einer dahinterliegenden Semantik begründet wird (im Fall dieses Papers einer Abhängigkeit zwischen einem Prozess und einem Objekt, die Teil einer SOA sind, wie einem Service).

Auch hier ist eine direkte Übertragung der Semantic Traceability auf das Konzept dieser Arbeit nicht möglich, da wieder der Bezug zu einer Domäne und den dazugehörigen Geschäftsprozessen fehlt. Es ist jedoch möglich, dass einige der vorgestellten Abhängigkeitstypen auch in dem SOA Governance Repository bestehen können. Beispielsweise können auf diese Weise alle Nutzer der Governance Repository, die irgendeine „sinnvolle“ Verbindung zu einem Service oder einem Business-Objekt haben, mit einer abkürzenden Beziehung verbunden werden. Wenn nun alle Service-Konsumenten ausgehend von einem Service identifiziert werden müssen, so ist dies dank der kurzen, inferierten Beziehung mit einfachen Abfragen möglich. Die Beschreibung dessen, was eine sinnvolle Verbindung ist, kann durch eine Ontologie und semantische Regeln ermöglicht werden.

In [HB07] wird „Gen/Tax“ erklärt, dabei handelt es sich um ein generisches Filterkonzept mit einer Taxonomie, das mit Hilfe der Sprache OWL verlinkt wird. Dabei wird ein Objekt in der Datenbank immer gleichzeitig mit einer generischen Kategorie verbunden, die keine Hierarchie mit anderen Kategorien bildet, und mit einer taxonomischen Kategorie verbunden, die in einem Hierarchiebaum von taxonomischen Kategorien ist. Der Autor will damit zwei Arten von Kategorien in einem Datenmodell unterstützen für zwei bestimmte Suchszenarien:

Die Suche nach einem bestimmten Objekt (beispielsweise ein Service mit bestimmter Funktion) soll mit Hilfe der generischen Kategorie erfolgen, denn da ist sich der Nutzer bewusst, welche genaue Funktion er braucht.

Die Suche nach einem unbestimmten Objekt (beispielsweise eine Art von Service mit unbestimmter Funktion) soll mit Hilfe der taxonomischen Kategorie erfolgen. Hier ist sich der Nutzer unsicher, was er genau braucht, kann aber in einem Hierarchiebaum die grobe Kategorie angeben und bekommt dann eine Auflistung möglicher Suchtreffer in der Kategorie und allen untergeordneten Kategorien.

Für das Konzept in dieser Arbeit ist es nicht nötig, eine Kategorisierung in eine generische Hälfte und eine Taxonomie-Hälfte im Datenmodell aufzuteilen. Stattdessen können im Konzept dieser Arbeit für die Suche nach bestimmten Services Schlüsselwörter verwendet werden, die relevante Eigenschaften dieser Services beschreiben. Unabhängig zu diesen Schlüsselwörtern kann dann eine Taxonomie definiert werden, um die Services in Kategorien zu gruppieren.

Diese Gruppierung wäre dann ein zusätzlicher Suchfilter und würde jeweils in Kombination mit Schlüsselwörtern mehrere Suchszenarien ermöglichen.

Die Autoren in [Ahm+15] stellen einen Ansatz zur semantischen Unterstützung vor, der im Bereich von Publish-Subscribe Systemen eingesetzt wird. Dabei werden Komponenten in einer Architektur vorgestellt, die mit Hilfe von semantischen Annotationen erweitert worden sind.

Managed Information Objects (MIO) sind die Informationsdaten, die im Verlauf des Publish-Subscribe Systems sind. Dabei sollen diese Daten neben den üblichen Kerndaten relevante Informationen für die semantischen Reasoner der anderen Komponenten tragen.

Der Data Management Service (DMS) soll lediglich zum Extrahieren von relevanten Informationen dienen und wird vom Characterization Service (CS) genutzt.

Der Characterization Service übernimmt – nachdem die Daten im richtigen Format vorliegen – die semantische Annotation. Anschließend werden die Daten in dem Semantic Repositoriy gesichert.

Das Semantic Repository übernimmt die Datenhaltung und den Großteil der Reasoning Funktionen.

Der Dissemination Service verteilt bei bestimmten Events Nachrichten an alle für den Event-Typ eingetragenen Subscriber.

Mit Hilfe dieser Architektur soll besser entschieden werden können, wie und welche Informationen bei Events schließlich bei den Subscribern ankommen. Zusätzlich wird eine Sicherheitskomponente vorgestellt, die mit Hilfe der semantischen Unterstützung im System auch bestimmte Sicherheitsvorschriften einhält und keine sicherheitskritischen Informationen an unbefugte Subscriber verschickt.

Publish-Subscribe Systeme sind nicht primär die Systeme, die vom Konzept dieser Arbeit unterstützt werden sollen. In dieser Arbeit geht es um Anwendungen, die sich an einer SOA orientieren und damit Services registrieren und verwalten können. Jede solche Anwendung verwaltet nicht nur Services sondern auch die Service-Konsumenten, die diese Services nutzen. Ähnlich wie Subscriber in einem Publish-Subscribe System sind auch Service-Konsumenten an den Änderungen und Neuigkeiten eines Services interessiert, den sie nutzen. Auch hier ist es wieder wichtig, diese Verbindung zu Service-Konsumenten zuerst zu erstellen, damit anschließend in der Logik einer Anwendung Nachrichten direkt an relevante Nutzer verschickt werden können. Eine Service-Nutzung kann entweder direkt sein, oder durch einen Vertrag geregelt sein. In beiden Fällen könnte im Konzept dieser Arbeit eine semantische Unterstützung helfen, solche abkürzenden Verbindungen zwischen Services und Service-Konsumenten zu erstellen und aktuell zu halten.

4. Konzept

In diesem Kapitel wird ein Konzept zur semantischen Unterstützung der SOA Governance vorgestellt. Damit das Konzept gezielt die SOA Governance unterstützen kann, muss zunächst geklärt werden, welche Merkmale eine SOA Governance hat. Marks [Mar08] definiert SOA Governance folgendermaßen:

> *„SOA governance refers to the organization, processes, policies, and metrics required to manage an SOA successfully. (...) In addition, an SOA governance model establishes the behavioural rules and guidelines of the organization and the participants in the SOA (...) These behavioural rules and guidelines are established via a body of defined SOA policies. (...) SOA governance defines and enforces the Web service policies that are needed to manage a SOA for business success. “*
> - E.A.Marks

Die SOA Governance bezieht sich demnach also auf eine Organisation, Prozesse, Regelungen und Metriken, die nötig sind, um eine SOA erfolgreich zu verwalten. Letztendlich ist es ein Kontrollmechanismus für alle Bausteine einer SOA. Im ersten Satz der Definition wird schon deutlich, dass sich die Merkmale einer SOA Governance nicht ohne die Merkmale einer SOA beschreiben lassen. Eine SOA selbst hat nach Marks[Mar08] die folgenden Merkmale:

- nutzt Services,
- verwaltet Service-Anbieter und Service-Konsumenten,
- implementiert eine Lebenszyklus-Verwaltung (von Services und anderen Geschäftsobjekten),
- unterstützt die Wiederverwendung und Komposition von Services (durch einfache Service Discovery) und
- hat eine flexible IT für sich ständig ändernde Geschäftsprozesse.

Das Nutzen von Services und die Beziehungen zwischen Service-Anbietern und Service-Konsumenten können direkt definiert sein oder lassen sich alternativ in Form von einem Zwischenelement, wie einem Vertrag beschreiben. Ein Vertrag könnte beispielsweise ein Service-Level-Agreement, finanzielle Vereinbarungen und andere Eigenschaften regeln, die für die Vertragspartner (Anbieter und Konsumenten) relevant sind. Eine semantische Unterstützung könnte in diesem Fall helfen, die relevanten Teilnehmer eines Vertrages zu identifizieren

und aktuell zu halten, wenn es beispielsweise einen Wechsel bei den Service-Verantwortlichen gibt.

Eine Lebenszyklus-Verwaltung ist im Kontext einer Servicenutzung ebenfalls wichtig, weil Services verschiedene Phasen einer Entwicklung und Benutzung durchlaufen können. So könnte eine semantische Unterstützung helfen, Service-Konsumenten darauf hinzuweisen, wenn eine neue Serviceversion erschienen (released) ist, oder zu warnen, wenn ein Service (im Status seines Lebenszyklus) veraltet ist, aber noch verwendet wird.

Damit die Wiederverwendung von Services unterstützt wird und eine flexible IT-Struktur ermöglicht werden kann, muss es eine gute Möglichkeit geben, Services in einem Service Repository zu finden. Es muss also eine Service Discovery ermöglicht werden, die mit Hilfe von verschiedenen Suchkriterien erweitert werden kann, um bestimmte Services aus einer Auswahl von Vielen zu filtern. Hierfür muss eine Struktur geschaffen werden, um neben der expliziten Service-Beschreibung (Service-Name und Schnittstellenbeschreibung) auch eine semantische Beschreibung zu haben. Diese Struktur kann mit einer Ontologie beschrieben werden, also einer semantischen Erweiterung des Datensatzes, die im Anwendungsfall der Service Discovery eine zusätzliche Bedeutung für Services gibt. Services und ihre Funktionen sollen mit Meta-Informationen wie Schlüsselwörtern und einer zugeordneten Kategorie erweitert werden können, damit die Suche nach Services vereinfacht wird.

Zusätzlich wird in der Definition zur SOA Governance beschrieben, dass es Regelungen (Policies) gibt, die von der SOA Governance aufgestellt und durchgesetzt werden müssen. Auch solche Regelungen lassen sich mit einer semantischen Unterstützung beschreiben. Es können semantische Regeln definiert werden, die mit Ihrer Bedingung einen Regelbruch beschreiben. Wenn diese Bedingung erfüllt ist und damit eine Regelung verletzt ist, so führt ein Reasoner die nötigen Maßnahmen in der definierten Folgerung der Regel aus. Wichtige, fehlende Kontaktinformationen von Personen könnten beispielsweise nicht nur programmatisch sondern auch durch semantische Regeln festgestellt werden und der Eintrag dieser Kontaktinformationen erzwungen werden. Das Konzept in diesem Kapitel zielt darauf ab, von Software-Tools adaptiert werden zu können, die eine SOA Governance unterstützen. In den folgenden Unterkapiteln werden diese genannten Konzeptideen einer semantischen Unterstützung für eine SOA Governance vorgestellt und genauer erläutert.

4.1. Semantisches Modell

Das volle Potenzial einer semantischen Unterstützung kann erst effektiv ausgeschöpft werden, wenn die zentralen Elemente eines Systems als Teil einer SOA mit einer zusätzlichen Bedeutung erweitert werden können. Diese zentralen Elemente sind Services (oder Serviceversionen für Systeme mit einer Versionsverwaltung). Im folgenden Unterkapitel wird daher zuerst eine allgemeine Ontologie beschrieben, die in einer SOA Governance Anwendung findet. Im Anschluss werden für diese Ontologie zwei semantische Erweiterungen definiert: die Erweiterung

durch Schlüsselwörter (Englisch: Tags), die an Services gebunden werden können, und die Erweiterung durch eine Taxonomie, die Services an bestimmte Kategorien bindet und damit semantisch gruppieren kann. Der gewonnene Nutzen der Ontologie und der Erweiterungen ist eine semantische Suche, die schließlich im letzten Unterkapitel dieses Kapitels erläutert wird.

Für das Erstellen einer Ontologie gibt es keine offiziellen Vorgaben. Es gibt jedoch „Best Practices“, die dabei helfen können eine Ontologie schrittweise zu erstellen. In [NM01] wird so ein Vorgehen schrittweise anhand eines Beispiels beschrieben. Die Schritte umfassen:

(1) Domäne der Ontologie identifizieren: Hier muss die Domäne und der Nutzen der Ontologie identifiziert werden. Dies geht beispielsweise durch die Betrachtung eines existierenden Datenmodells oder durch das Formulieren von Fragen, die die Ontologie beantworten soll.

(2) Existierendes Vokabular wiederverwenden: In vielen Domänen wurden bereits gängige Probleme durch Ontologien und den damit definierten Vokabularen gelöst. Solche Vokabulare könnten wiederverwendet werden, um die Kommunikation des eigenen Systems mit anderen zu vereinfachen.

(3) Wichtige Begriffe aufzählen: Bevor konkrete Klassen und Klasseneigenschaften definiert werden können, sollen auf konzeptioneller Ebene die wichtigsten Elemente einer Ontologie aufgezählt werden, über die man Aussagen tätigen möchte. Es ist auch deshalb wichtig, weil auf diese Weise einfacher festgestellt werden kann, welche Beziehungen zwischen den Begriffen existieren können und welches neue Wissen inferiert werden könnte.

(4) Klassen und Klasseneigenschaften definieren: Schließlich werden in [NM01] in den nächsten Schritten die Klassen, Klasseneigenschaften und gegebenenfalls strikte Definitionen für die Eigenschaften dieser Klassen erstellt. In diesem Schritt können auch semantische Regeln definiert werden, die auf Basis von bestimmten Klasseneigenschaften neue Beziehungen inferieren können.

(5) Instanzen erstellen: Konkrete Instanzen der Klassen entstehen, wenn die Ontologie verwendet wird. Erst mit solchen Instanzen wird sichtbar, ob die Definition der Klasseneigenschaften korrekt ist und ob die semantischen Regeln funktionieren.

Die Schritte (1), (3) und (4) können hier bereits auf konzeptioneller Ebene durchgeführt werden. Schritt (2) ist im Konzept noch nicht erforderlich, weil auf konzeptioneller Ebene noch kein konkretes Vokabular erstellt werden muss. Schritt (2) kann bei der Implementierung nachgeholt werden, wenn für die Erweiterungen der Ontologie ein neues Vokabular erstellt werden muss.

4.1.1. SOA Governance Ontologie

Die SOA Governance Ontologie, die im folgenden vorgestellt wird, basiert auf dem SOA Governance Meta Model (SOA-GovMM) [KSM14] und formalisiert das aktuelle Datenmodell des SOA Governance Repositories [KM16]. Die Klassen dieser Ontologie wurden prototypisch mit Protege[1] erstellt und mit Hilfe des Plugins Ontograf[2] dargestellt. Das Listing 4.1 zeigt einen Ausschnitt der Ontologie als XML-Beschreibung. Ausgelassene Zeilen sind mit (...) gekennzeichnet, da das vollständige Dokument für den Rahmen dieser Arbeit zu lang wäre.

```
1  (...)
2  <!-- /// Data Properties /// -->
3  (...)
4     <owl:DatatypeProperty rdf:about="http://www.blehmar.uni-stuttgart.de/
          ontology/soa_gov/rdf_ontology.owl#description">
5        <rdfs:domain rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#Environment"/>
6        <rdfs:domain rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#Person"/>
7        <rdfs:domain rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#Responsibility"/>
8        <rdfs:domain rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#Role"/>
9        <rdfs:domain rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#ServiceArtifact"/>
10       <rdfs:domain rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#Skill"/>
11       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
12    </owl:DatatypeProperty>
13
14 (...)
15 <!-- /// Classes /// -->
16    <owl:Class rdf:about="http://www.blehmar.uni-stuttgart.de/
          ontology/soa_gov/rdf_ontology.owl#Assignable"/>
17
18    <owl:Class rdf:about="http://www.blehmar.uni-stuttgart.de/
          ontology/soa_gov/rdf_ontology.owl#Binding"/>
19
20    <owl:Class rdf:about="http://www.blehmar.uni-stuttgart.de/
          ontology/soa_gov/rdf_ontology.owl#BusinessObject">
21       <rdfs:subClassOf rdf:resource="http://www.blehmar.uni-stuttgart.de/
             ontology/soa_gov/rdf_ontology.owl#Assignable"/>
22    </owl:Class>
23 (...)
```

Listing 4.1: Ausschnitte aus der SOA Governance Ontologie

[1] Protege Tool: http://protege.stanford.edu/
[2] Ontograf Github: https://github.com/protegeproject/ontograf

Im Listing sind Ausschnitte von einem Feld (*Data Properties*) und einigen Klassen (*Classes*) zu sehen. Das Feld *description* (Zeile 4 - 13) wird von mehreren Klassen (als *Domain* spezifiziert) verwendet – es ist eine allgemeine Freitextbeschreibung. Die Klassen *Assignable*, *Binding* und *BusinessObject* sind ebenfalls im XML-Ausschnitt dargestellt (Zeilen 17 - 26). Das *Binding* ist ein Teil einer Serviceschnittstellen-Beschreibung mit WSDL. Die Klasse *BusinessObject* und ihre Funktion im SOA Governance Repository wird später im Abschnitt „Ontologie für Business-Objekte“ genauer erklärt. An dieser Stelle ist anzumerken, dass *BusinessObject* der Klasse *Assignable* untergeordnet ist, weil Business-Objekte im SGR zu bestimmten Personen gehören können sollen (z.B. als Eigner eines Business-Objekts). Die Abbildung 4.1 stellt die vollständige Klassenhierarchie der SOA Governance Ontologie dar.

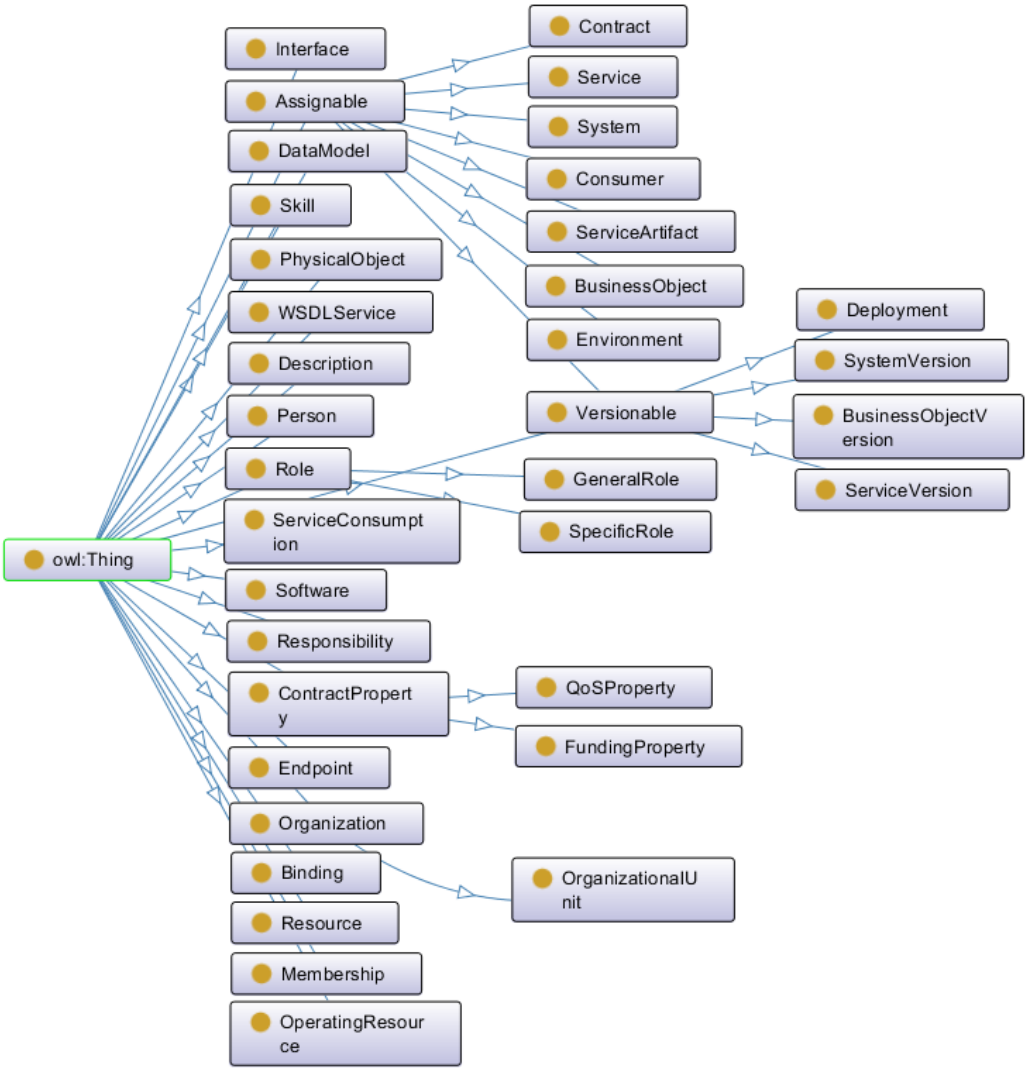

Abbildung 4.1.: Klassenhierarchie der SOA Governance Ontologie (Screenshot aus Protege)

In dieser Abbildung werden Klassen mit Rechtecken und die Verbindungen für untergeordnete Klassen mit blauen Pfeilen dargestellt – die konkreten Beziehungen zwischen den einzelnen Klassen werden später erläutert. Zentrale Elemente in einer SOA, wie Services und Serviceversionen, sowie die Konsumenten und Verträge sind in der Hierarchie wiederzufinden. Man achte besonders auf die Hierarchie *Assignable* → *Versionable*. Alle *Assignables* sind Klassen, die untereinander gegenseitig zugewiesen werden können. *Versionable*-Klassen sind ebenfalls *Assignables*, aber beschreiben zusätzlich Objekte, die einer Lebenszyklus-Verwaltung unterliegen. Diese *Assignables* sind als Klassen anzusehen, die flexible Beziehungen zulassen, die im Modell bewusst nicht konkreter definiert sind. Diese flexible Definition ermöglicht in der Ontologie beispielsweise folgende Beziehungen:

- Das Beschreiben von Service-Anbieter (*Service*) und Service-Konsumenten (*Consumer*) in einem Vertrag,
- Das Beschreiben von zusätzlichen Serviceversion-Ressourcen, wie Business-Objekte (z.B. weitere Dokumente zur Beschreibung des Services),
- Das Beschreiben, auf welcher Laufzeit-Umgebung (*Environment*) eine *Serviceversion* betrieben wird und
- sämtliche weitere Beziehungen, die im Kontext einer SOA Governance sinnvoll sind.

Für die genauere Beschreibung der Klassen und deren Beziehungen werden die Klassen im Folgenden in vier logische Gruppen unterteilt (analog der Unterteilung im SOA-GovMM). Jede der Abbildungen für diese Unterteilungen stellt Klassen in orange-umrandeten Rechtecken, einfache Beziehungen mit blauen Pfeilen und offenen Pfeilspitzen und hierarchische Beziehungen mit blauen Pfeilen und gefüllten Pfeilspitzen dar. Beidseitige Beziehungen sind in der Ontologie als inverse Beziehungen definiert. Das hat den Vorteil, dass es genügt, bei der Entwicklung mit diesem Modell nur eine Beziehungsrichtung explizit zu formen. Ein Reasoner kann die rückwirkende Beziehung dann automatisch inferieren.

Ontologie für Service-Anbieter

Die Abbildung 4.2 zeigt die Ansicht der Service-Anbieter, in der zentral die Services und Serviceversionen zu sehen sind. Eine Serviceversion kann dabei über ein Deployment in einer Laufzeitumgebung betrieben werden. Die Laufzeitumgebung hat dabei Hardware- und Software-Ressourcen, die für das Betreiben der Serviceversion nötig sind. Zusätzlich hat jede Serviceversion ein *ServiceArtifact*, das eine Ressource für die Service-Beschreibung ist. So eine Ressource kann auch eine standardisierte Schnittstellenbeschreibung sein, wie eine WSDL-Datei. Das Element *InterfaceDescription* fasst in dieser Abbildung aus Übersichtsgründen alle Elemente einer WSDL-Schnittstellenbeschreibung zusammen, die zuvor in der Klassenhierarchie zu sehen waren (*WSDLService*, *Description*, *Binding* und *Interface*). Schließlich hat jede Serviceversion einen Endpunkt, über den ein Service-Konsument diesen erreichen und nutzen kann.

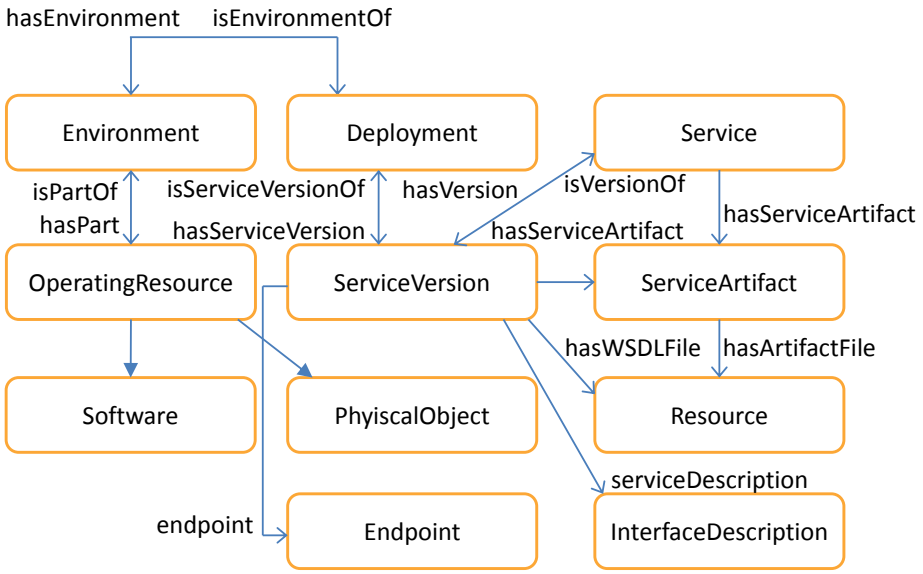

Abbildung 4.2.: Service-Anbieter-Ansicht der SOA Governance Ontologie

Ontologie für Service-Konsumenten

In der Abbildung 4.3 wird die Ansicht der Service-Konsumenten dargestellt. Auf der rechten Seite ist in der Mitte die Service-Nutzung (*ServiceConsumption*) zu sehen. Diese Objekte sind auf Seiten des Service-Anbieters mit dem Service-Endpunkt verbunden. Ein Vertrag reguliert diese Service-Nutzung und hält Vertragseigenschaften wie die Qualität der Service-Nutzung (*QoSProperty*) und Zahlungsvereinbarungen (*FundingProperty*) fest. Ein *Consumer* ist ein Klient-System, das einen Service nutzt. Es soll mit einer Systemversion verbunden sein, was in der Abbildung nicht direkt modelliert ist. Das liegt daran, dass beides *Assignables* sind und die Verbindung an dieser Stelle über eine *Assignable*-Beziehung abgebildet werden kann (wie zuvor in der Klassenhierarchie beschrieben). Auch Systemversionen gehören immer einem System an und unterliegen einer Lebenszyklus-Verwaltung.

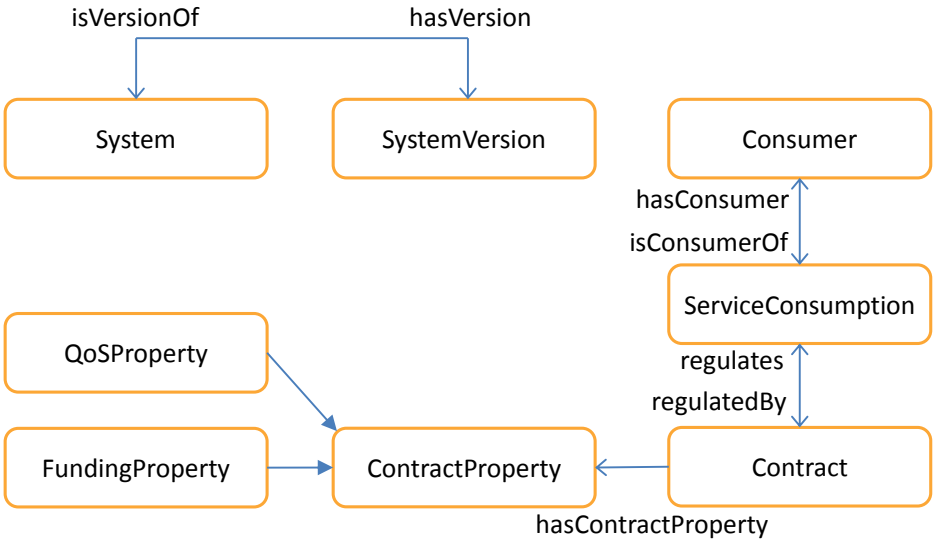

Abbildung 4.3.: Service-Konsumenten-Ansicht der SOA Governance Ontologie

Ontologie für Organisationen

Abbildung 4.4 zeigt die Organisationsansicht der SOA Governance Ontologie. Hier befindet sich die Benutzerverwaltung, in der die Benutzer des Systems links mit der Klasse *Person* definiert sind. Ein Benutzer hat bestimmte Fähigkeiten (*Skills*) und gehört einer Mitgliedschaft (*Membership*) an, das eine Rolle (*Role*) im Umfeld des SOA Governance-Systems hat. Diese Rolle hat bestimmte Verantwortungen (*Responsibility*) und benötigt bestimmte Fähigkeiten zur Erfüllung dieser Rollenverantwortungen. Zusätzlich ist eine Rolle zu unterscheiden in eine allgemeine Rolle (*GeneralRole*) und eine spezifische Rolle (*SpecificRole*). Eine *GeneralRole* beschreibt dabei allgemeine Rollen im System, wie Administratoren oder den Benutzer-Support. Spezifische Rollen sind *Assignable*, das heißt sie beziehen sich immer auf eine Instanz einer Klasse, die der *Assignable*-Klasse untergeordnet ist. Im Kontext eines Services kann dies bedeuten, dass eine Serviceversion (*Assignable*) mit einem Service-Verantwortlichen (*SpecificRole*) oder technischen Service-Kontakt (*SpecificRole*) verbunden wird. Auf Seiten von Service-Konsumenten kann dies einen Eigner eines Konsumentensystems (*SpecificRole*) mit einer Systemversion (*Assignable*) verbinden. Eine Person kann ebenfalls Mitglied einer *Organisation* sein. Die Organisation selbst kann aus Organisationseinheiten bestehen (*OrganizationalUnit*), in der jede Einheit wiederum eine eigene Organisation sein kann.

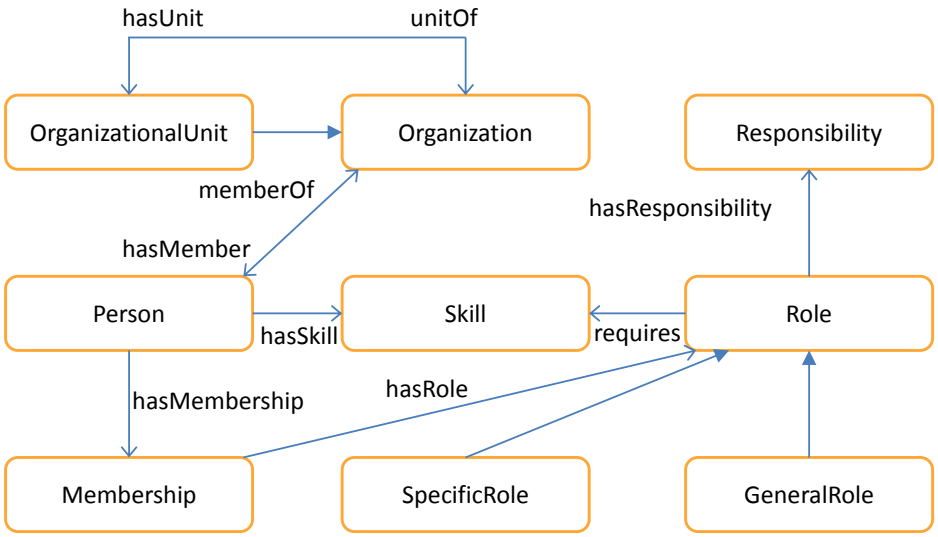

Abbildung 4.4.: Organisationsansicht der SOA Governance Ontologie

Ontologie für Business-Objekte

Zuletzt beschreibt die Abbildung 4.5 die Ansicht für Business-Objekte. Business-Objekte (Klasse *BusinessObject*) sollen die Struktur von jeglichen Objekten innerhalb eines Systems beschreiben, das eine SOA Governance unterstützt. Dies können Services/Serviceversionen sein, beschränken sich aber nicht nur darauf. Das Ziel einer zusätzlichen Beschreibung mit Business-Objekten ist die Möglichkeit, bestimmte Strukturen innerhalb einer Domäne und zwischen mehreren Domänen einheitlich zu beschreiben. Das Problem der Nutzung verschiedener Terminologien und (Service-)Schnittstellenbeschreibungen soll durch diese Vereinheitlichung gelöst werden. Die Modellbeschreibung von Business-Objekten ist an keine Implementierung gebunden und kann in Formaten, wie einem XML-Schema-Dokument beschrieben werden. Ähnlich wie bei Serviceversionen und Systemen, haben auch Business-Objekte eine Versionsverwaltung (Klasse *BusinessObjectVersion*).

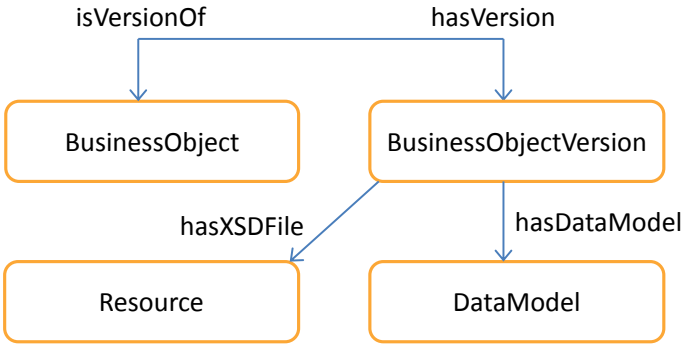

Abbildung 4.5.: Business-Objekt-Ansicht der SOA Governance Ontologie

Durch die Beschreibung der SOA Governance Ontologie ist es nun möglich, den Datensatz eines Software-Tools, das dieses Konzept adoptiert, vollständig semantisch zu unterstützen. Das heißt, dass der Datensatz semantisch annotiert werden kann und mit Hilfe von Reasonern die folgenden Vorteile entstehen:

- Syntaktische und semantische Regelungen können besser kontrolliert und eingehalten werden. Die SOA Governance Ontologie beschreibt genau, welche Klassen es gibt und welche Eigenschaften und Beziehungen diese haben müssen. Zusätzlich können semantische Regeln eingeführt werden, die im Kontext einer SOA Governance sinnvoll erscheinen. Ein Beispiel hierfür wäre eine Regel, die für Service-Verantwortliche vorschreibt, immer eine Mail-Adresse anzugeben, damit diese zu jeder Zeit bei Anliegen zu ihrem Service kontaktiert werden können.
- Zusätzliches Wissen kann dynamisch aus dem Datensatz zur Laufzeit inferiert werden. Durch die Definition von semantischen Regeln, die auf Basis einer Bedingung neue Beziehungen und Objekte erstellen können, ist es nun möglich neues Wissen zu generieren, ohne dafür Programmcode zu schreiben.
- Performance- und Komplexitätsersparnisse können aus der SOA Governance Ontologiebeschreibung resultieren. Viele Reasoner arbeiten auf der Ebene der Datenbank und können dort Operationen im Datensatz vornehmen, ohne dass die relevanten Datensätze dafür in den Arbeitsspeicher einer Anwendung geladen werden müssen. Auch können semantische Regeln in wenigen Zeilen zusammenfassen, was oft in mehreren Klassen Programmcode nötig wäre, um bestimmte Regeln im Datenmodell einzuhalten. Dies erlaubt es, den Programmcode einer Anwendung mit der SOA Governance Ontologie einfacher zu gestalten.

4.1.2. Erweiterung: Schlüsselwörter

Die erste Erweiterung für die SOA Governance Ontologie sind semantische Schlüsselwörter. Ein Schlüsselwort ist in diesem Kontext eine Kurzbeschreibung eines Service-Aspektes in Form eines Wortes. Schlüsselwörter können für Services von den rechtmäßigen Service-Anbietern frei vergeben werden. Es können beliebig viele Schlüsselwörter für einen Service definiert werden, die beispielsweise die Funktion eines Services, die Service-Qualität oder andere Aspekte eines Services beschreiben. Der Service-Anbieter entscheidet bei der Vergabe, welche Schlüsselwörter für seinen Service sinnvoll sind, damit Service-Konsumenten diesen besser finden können. Empfehlenswert sind Schlüsselwörter, die aussagekräftig sind und besondere oder wichtige Merkmale eines Services beschreiben, da generische Schlüsselwörter bei der Service-Suche in einem Software-Tool auch irreführend sein könnten.

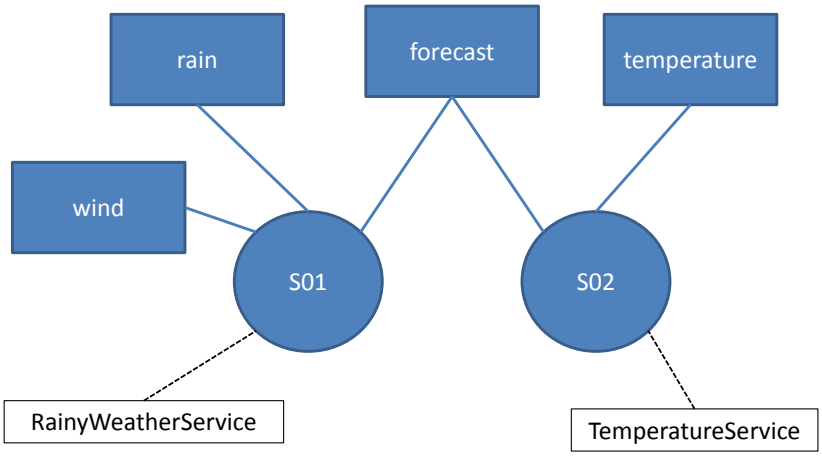

Abbildung 4.6.: Beispiel vom Einsatz der Schlüsselwörter

In Abbildung 4.6 wird gezeigt, wie die eingesetzten Schlüsselwörter aussehen könnten. Die blauen Rechtecke sind die Schlüsselwörter, die blauen Kreise stellen Services (oder Serviceversionen) dar, die weißen Kästchen sind die Namen der Services. Blaue Verbindungen stellen Prädikate dar, die Services mit Schlüsselwörtern verbinden; schwarze, gestrichelte Verbindungen sind Prädikate, die Services mit Namen verbinden. Beide Services sollen das Wetter vorhersagen, allerdings mit verschiedenen Funktionen. *S01* ist ein Wetterservice, der Niederschlagswerte und Windgeschwindigkeiten liefert, während *S02* nur Temperaturwerte für bestimmte Zeitabschnitte liefern soll (daher auch die Namen „*RainyWeatherService*“ und „*TemperatureService*“). Entsprechend der Funktion der beiden Services haben Service-Provider Schlüsselwörter angelegt, wie in Abbildung 4.6 zu sehen ist und es fällt auf, dass beide Services ein Schlüsselwort gemeinsam haben: das „*forecast*-“Schlüsselwort (Vorhersage). Da Schlüsselwörter in diesem Konzept eindeutig durch ihre Zeichenkette definiert sind, könnte auf diese Weise einfach eine Ähnlichkeit der semantischen Funktionalität beider Services festgestellt werden. Eine semantische Regel könnte an dieser Stelle definiert werden, die bei einem geteilten Schlüsselwort zwei Services mit einer „Ähnlichkeitsbeziehung“ verbindet (dazu im Kapitel 4.3 mehr). Ein weiterer Nutzen wäre eine semantische Suche, die nicht mehr nur nach den Servicenamen sondern auch nach Schlüsselwörtern sucht. *S01* würde in diesem Fall auch über das Schlüsselwort „*wind* “ gefunden werden, obwohl es im Servicenamen nicht vorkommt.

4.1.3. Erweiterung: Taxonomie

Die zweite Erweiterung der SOA Governance Ontologie ist die Taxonomie, die Services in semantisch sinnvollen Kategorien einordnen kann. Welche Kategorien sinnvoll sind, sollte durch den Administrator der jeweiligen Domäne des SOA-Systems definiert werden. Services sind dabei in einer oder mehreren Kategorien eingeordnet, damit sie bei einer Suche in einer Anwendung nach bestimmten Kategorien zusammen mit „semantisch ähnlichen “ Services

aufgelistet werden können. Kategorien sind in einer Hierarchie aufgebaut, die einen Baum von Kategorien darstellt. Das hat den Vorteil, dass auch höher angeordnete Kategorien gewählt werden können, die sämtliche Services der Kategorie und alle Unterkategorien auflisten können. Ein einfaches Beispiel in einer Domäne mit Services zur Wettervorhersage soll dies veranschaulichen:

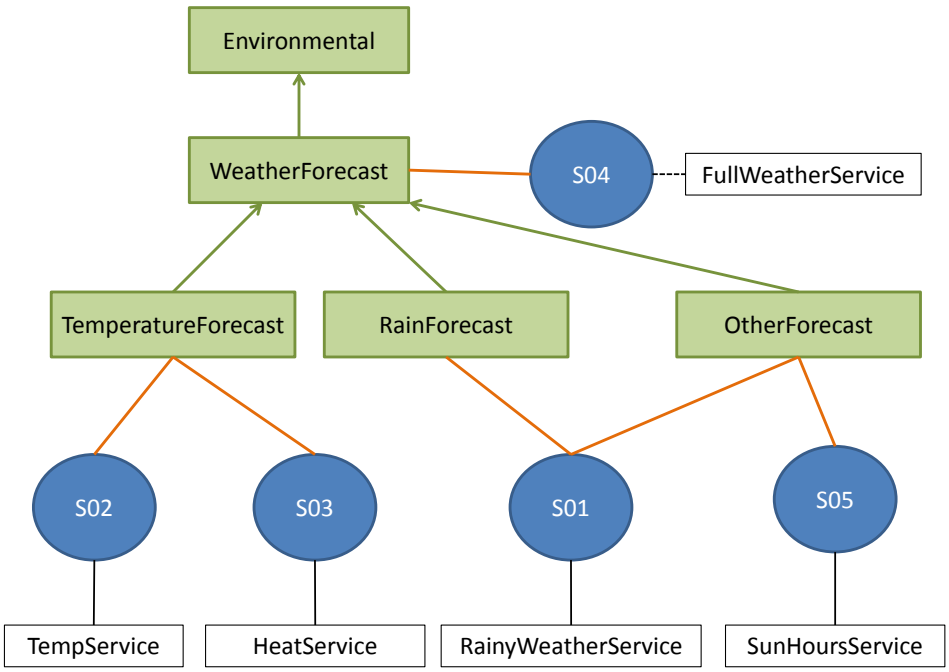

Abbildung 4.7.: Beispiel einer Taxonomie für Wettervorhersage-Services

In der Abbildung 4.7 werden die Kategorien in grünen Rechtecken dargestellt, die Services wieder in blauen Kreisen und die Servicenamen in weißen Kästchen. Die grünen Beziehungen sind Prädikate zwischen den Kategorien, um zu beschreiben welche Kategorien unter welchen untergeordnet sind. Diese sind gerichtet, weil sie eine Zugehörigkeit zu einer Elternkategorie beschreiben sollen, wobei die *Environmental*-Kategorie die oberste des Kategorie-Baums darstellt. Die Services *S02* und *S03* haben ähnliche Funktionen, weil sie Temperaturwerte zu bestimmten Zeiten wiedergeben können, daher sind sie beide in der *TemperatureForecast*-Kategorie eingeordnet. Der Service *S01* ist der gleiche wie im letzten Beispiel. Er hat die Funktion, Niederschlagswerte und Windgeschwindigkeiten in vorgegebenen Zeitabschnitten zu bestimmen. Daher ist dieser Service auch in zwei Kategorien gleichzeitig: der Niederschlagsvorhersage-Kategorie (*RainForecast*) und der Andere-Vorhersage-Kategorie (*OtherForecast*). Der Service *S04* ist ein Service, der mehrere Funktionen einer Wettervorhersage gleichzeitig implementiert, und damit an die allgemeinere, übergeordnete Wettervorhersage-Kategorie (*WeatherForecast*) gebunden ist. In der Kategorie *OtherForecast* werden Services eingeordnet, die nur eine bestimmte Funktion erfüllen, aber weder Temperaturwerte noch Niederschlagswerte bestimmen. *S05* ist ein Service, der lediglich die Sonnenstunden zu einem bestimmten Tag im Jahr wiedergibt

und damit Teil der Kategorie *OtherForecast* ist. Der Vorteil einer Taxonomie ist es, dass eine Suche nach unbestimmten Services in einer groben funktionalen oder semantischen Richtung ermöglicht wird. Dadurch können Service-Konsumenten sich einen passenden Service aus einer vor-gefilterten Liste aussuchen.

4.2. Suche

Eine Suche nach Services in einem Software-Tool ist nun durch die zuvor erstellte Ontologie mit mehreren Herangehensweisen möglich. Services können in erster Linie durch Suchbegriffe gefunden werden. Dafür kann eine einfache Suchleiste mit zusätzlichen Suchoptionen erstellt werden, die mehrere Modi unterstützt.

Einfache Suche – sucht nach ganzen Servicenamen und Teilwörtern davon. Das Leerlassen des Suchbegriffes listet alle Services auf. Diese Suche ist sinnvoll, wenn der Service-Konsument den Namen des gewünschten Services kennt und keine anderen Services sucht.

Semantische Suche – sucht nach Servicenamen und Schlüsselwörtern (und ebenfalls allen Teilwörtern). Leerlassen des Suchbegriffes listet alle Services auf. Diese Suche sollte dann eingesetzt werden, wenn ein Service-Konsument den Namen eines Services vergessen hat, aber dessen Funktionen oder Eigenschaften noch kennt. Allgemein könnte er auch einfach nach einer bestimmten Funktion oder Eigenschaft eines neuen Services suchen.

Suche in Kategorien – sucht nach Servicenamen (und Teilwörtern) innerhalb ausgewählter Kategorien (und ggf. allen Unterkategorien der ausgewählten Kategorie). Das Leerlassen des Suchbegriffes listet alle Services in der Kategorie und allen Unterkategorien auf. Dieser Suchmodus ist zu benutzen, wenn ein Service-Konsument noch nicht genau weiß, welchen Service er braucht. Bei der Suche in bestimmten Kategorien bekommt er eine Vorstellung davon welche Services es gibt, die für seinen Anwendungsfall zufriedenstellend sein könnten.

Semantische Suche in Kategorien – sucht nach Servicenamen und Schlüsselwörtern (und Teilwörtern) innerhalb ausgewählter Kategorien und Unterkategorien. Das Leerlassen des Suchbegriffes listet alle Services in der Kategorie und allen Unterkategorien auf. Dieser Suchmodus ist im gleichen Fall wie mit dem Modus zuvor zu nutzen, nimmt jedoch zusätzliche Ergebnisse auf, die mit dem Suchbegriff und semantischen Schlüsselwörtern übereinstimmen.

Die erstellte Ontologie kann auch in weiteren Herangehensweisen zur Service-Suche sinnvoll sein. Beispielsweise könnte man nun mit Hilfe von Abfragen zur Service-Ähnlichkeit alle Services auflisten, die sich ein oder mehrere Schlüsselwörter teilen. Diese Ergebnisliste könnte zusätzlich geordnet werden, da bei mehreren geteilten Schlüsselwörtern des Ausgangs-Services mit einem anderen Service die Gewichtung der Ähnlichkeit zunimmt. Ein Service S01 könnte

diese Ähnlichkeitsbeziehung mit mehreren Services und zwischen den Services mit mehreren Schlüsselwörtern aufweisen. Beispielsweise sei folgendes Szenario gegeben:

- S01 hat **drei** Schlüsselwörter mit S02 gemeinsam
- S01 hat **ein** Schlüsselwort mit S03 gemeinsam
- S01 hat **zwei** Schlüsselwörter mit S04 gemeinsam

Das Ergebnis einer geordneten Liste mit abnehmender Ähnlichkeit zum Service S01 wäre damit: **S02, S04 und S03**. Dies kann in einem Software-Tool entweder als Teil der Suche umgesetzt werden, oder direkt in einer detaillierten Service-Ansicht in einem semantischen Abschnitt angezeigt werden.

4.3. Reasoning

Damit ein Reasoner neue Beziehungen herleiten kann, muss ein Datensatz semantisch annotiert werden oder es müssen Regeln definiert werden, die neue Objekte und Beziehungen implizieren. Je nach Technologie und gewähltem Reasoner werden für das Schreiben von semantischen Regeln unterschiedliche Sprachen wie SWRL oder SPIN verwendet. Ontologiesprachen wie OWL unterstützen beispielsweise Reasoning auf Basis von OWL-Annotationen, die für einfache Zusammenhänge nicht zwingend Regeln erfordern, um neue Beziehungen und Objekte herzuleiten. OWL kann jedoch mit Regeln kombiniert werden, die in SWRL definiert sind, um komplexere Zusammenhänge zu erkennen und semantische Informationen zu inferieren. Reasoning bringt mehrere Vorteile für das Entwickeln eines Software-Tools mit semantischer Unterstützung. Das Definieren von Regeln für einen Reasoner ist einfacher, als dieselbe Logik auf Programmebene zu schreiben und umgeht damit komplexeren Anwendungscode. Zudem können wenige Zeilen einer semantischen Regel viele Zeilen Programmcode ersetzen. Ein weiterer Vorteil ist, dass nicht alle Beziehungen im Datenmodell explizit modelliert werden müssen. Auch das kann Programmcode ersparen, da ein Reasoner durch semantische Regeln implizit Beziehungen herleiten kann und auch aufbauend aus hergeleiteten Beziehungen rekursiv neue Beziehungen herleiten kann. Schließlich arbeiten einige Reasoner direkt auf Datenbankebene. Je nach Reasoner-Technologie kann das Inferieren von neuen Objekten und Beziehungen dadurch schneller sein und weniger Ressourcen verbrauchen, als ein Programmcode auf Seiten der Anwendung. Im Unterkapitel 4.3.1 werden zuerst Regeln beschrieben, die auf Basis der vorgestellten Ontologie arbeiten. Im Anschluss werden im Unterkapitel 4.3.2 allgemeinere Regeln beschrieben, die eine Anwendung semantisch unterstützen können.

4.3.1. Reasoning auf Basis der Ontologie

Aus der zuvor definierten Ontologie lassen sich im Bereich der Schlüsselwörter und der Taxonomie bereits erste neue Beziehungen durch einen Reasoner herleiten. Wenn sich zwei Services ein Schlüsselwort teilen, so haben sie in einem (Teil-)Aspekt eine semantische Ähnlichkeit.

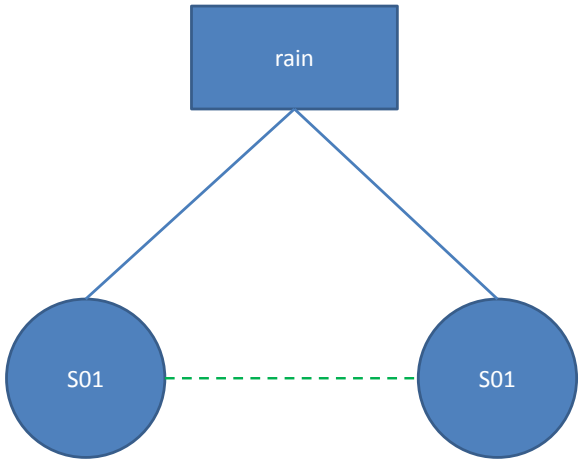

Abbildung 4.8.: Herleiten einer Ähnlichkeitsbeziehung

In der Abbildung 4.8 wird dies verdeutlicht. Die blauen Kreise stehen wieder für Services, das blaue Rechteck für ein Schlüsselwort, das in diesem Fall mit beiden Services *S01* und *S02* verbunden ist. Diese Ähnlichkeit kann nun durch einen Reasoner deutlich gemacht werden, indem zwischen den Services diese Ähnlichkeitsbeziehung erstellt wird (grüne, gestrichelte Linie). Es müsste also eine Regel geschrieben werden, die mit Hilfe einer Implikation arbeitet. Im Listing 4.2 wird diese Regel in einem Pseudocode formuliert und dient als Vorlage für die Regel in der Implementierung.

```
1 WENN
2       ?service http://ex.org#hasTag ?tag .
3       ?otherservice http://ex.org#hasTag ?tag .
4       (?service != ?otherservice)
5 DANN
6       ?service http://ex.org#similarTo ?otherservice .
7       ?otherservice http://ex.org#similarTo ?service
8 ENDE
```

Listing 4.2: Ähnlichkeitsbeziehung herleiten

Der Pseudocode nutzt das Format *?<Name>* um eine Variable zu beschreiben. Jeder Punkt in der WENN-Klausel kann als UND-Operator gesehen werden, der eine weitere Bedingung anfügt. Diese Schreibweise ist an SPARQL angelehnt und damit in der Arbeit mit Triples geläufig. *http://ex.org#hasTag* ist die Beziehung die einen Service mit einem Tag verbinden soll. *?service* und *?otherservice* sind zwei Services, die das gleiche Tag in der Variable *?tag*

haben. Zuletzt soll in Klammern sichergestellt werden, dass *?service* und *?otherservice* nicht den gleichen Service beinhalten. Wenn all dies gegeben ist, so wird in der DANN-Klausel die Beziehung *http://ex.org#similarTo* zwischen den Services beidseitig vom Reasoner hergeleitet. Diese Beziehung ist deswegen sinnvoll, weil mit Hilfe von einfachen Abfragen nun alle Services aufgelistet werden können, die eine Ähnlichkeit mit einem Ausgangs-Service haben.

Auch in der Taxonomie kann ein Reasoner helfen, um die transitive Beziehung „istUnterKategorie“ (*isSubCategory*) rekursiv für alle Kinder-Kategorien einer Eltern-Kategorie herzuleiten. Die Abbildung 4.9 verdeutlicht diese hergeleitete Beziehung.

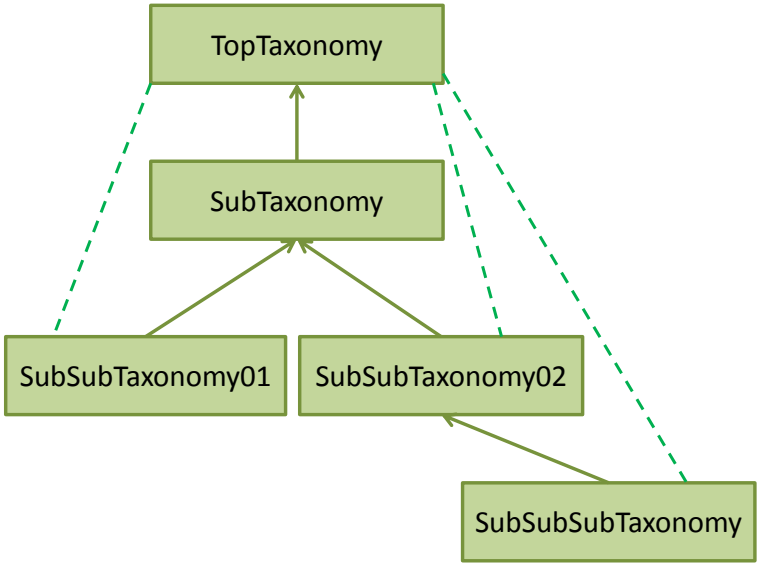

Abbildung 4.9.: Herleiten von transitiven Unterkategorien

In Ontologiesprachen wie OWL kann dies direkt definiert werden, indem die Beziehung selbst als transitive Beziehung annotiert wird und in Kombination mit einem OWL-Reasoner würden alle rekursiven Beziehungen hergeleitet werden. Jedoch unterstützt dies grundsätzlich nicht jeder Reasoner und nicht jede Art der Datenhaltung. Für andere Arten der semantischen Datenhaltung und Reasoner, die keine direkte semantische Annotation für transitive Beziehungen verstehen, muss wieder eine Regel in Form einer Implikation definiert werden, wie dies im Listing 4.3 gemacht wird.

```
1 WENN
2       ?category http://ex.org#hasSubCategory ?subCategory .
3       ?subCategory http://ex.org#hasSubCategory ?subSubCategory .
4 DANN
5       ?category http://ex.org#hasSubCategory ?subSubCategory .
6 ENDE
```

Listing 4.3: Unterkategorien in Taxonomie rekursiv herleiten

Im Grunde wird durch das Betrachten von Unterkategorien in zwei Tiefen-Ebenen eine *hasSubCategory*-Beziehung zwischen Großeltern-Kategorien und Enkel-Kategorien hergestellt. Es genügt die Regel auf zwei Tiefen-Ebenen zu definieren, da durch das Erstellen einer neuen Beziehung in der DANN-Klausel die Kategorie modifiziert wurde und die Regel durch einen Reasoner erneut evaluiert werden muss. So wird die Regel rekursiv immer wieder evaluiert (im Taxonomiebaum abwärts), bis keine neue Beziehung mehr hergeleitet werden kann. Das identifizieren von Unterkategorien könnte man auch auch in einer Implementierung auf der Serverseite einer Anwendung lösen, jedoch arbeiten Reasoner auf Datenbankebene und können diese Beziehungen schneller und mit weniger Ressourcenverbrauch herleiten. Alles was dafür benötigt wird sind drei Zeilen einer semantischen Regel und damit stellt diese Regel eine attraktive Alternative zum Identifizieren von Unterkategorien in einem potenziell größeren Taxonomiebaum von mehreren hundert oder tausend Kategorien dar.

4.3.2. Reasoning auf Basis einer SOA

Anwendungen im Umfeld einer SOA weisen die Eigenschaften und Merkmale auf, wie diese in der Einleitung im Kapitel 4 vorgestellt wurden. Sie haben Services, die von Service-Anbietern gestellt und von Service-Konsumenten genutzt werden, sie beschreiben Services mit einem Endpunkt und einer Schnittstellen-Beschreibung, sie verwalten Service-Lebenszyklen und haben für Service-Anbieter und Service-Konsumenten eine Art Benutzerverwaltung. Auf Basis dieser Merkmale können für solche Anwendungen allgemeinere Regeln definiert werden, die mit Hilfe eines Reasoners neue, semantisch sinnvolle Beziehungen und Objekte herleiten.

Beziehungen um einen Vertrag herum

In der Abbildung 4.10 werden mehrere Möglichkeiten dargestellt, um semantisch sinnvolle Beziehungen herzuleiten (grün, gestrichelte Linien). Zum einen werden Beziehungen zwischen einem Vertrag (*Contract*) und allen Teilnehmern des Vertrags hergestellt. Auf diese Weise können Vertragspartner schneller identifiziert werden und bei einem Wechsel von beispielsweise Service-Verantwortlichen aktuell gehalten werden. Zum anderen können allgemein im Bezug auf einen Service alle Nutzer identifiziert werden, die eine Relevanz für einen Service haben. So eine Relevanz könnte das Informieren von Benutzern sein, wenn der Lebenszyklus eines Services sich ändert, der Service außer Betrieb gerät oder wenn eine neue Version dieses Services verfügbar ist. Auch an dieser Stelle kann ein Reasoner bei einem Wechsel von Benutzern auf

Seiten des Konsumenten-Systems oder der Service-Verantwortlichen die jeweiligen Personen aktuell halten.

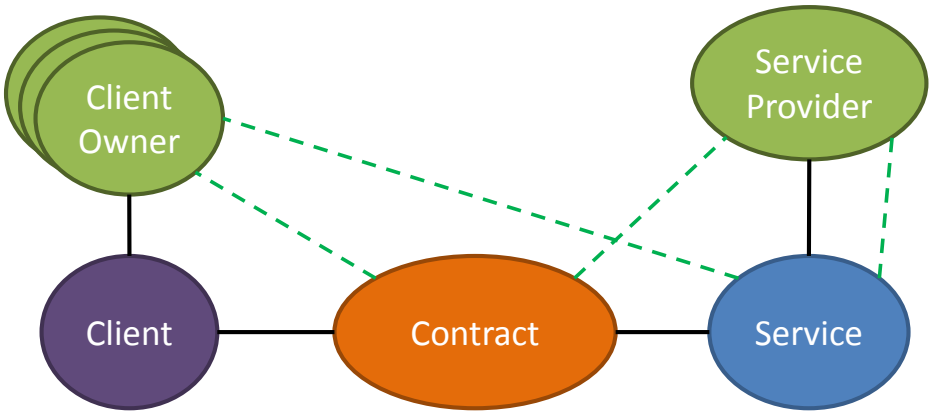

Abbildung 4.10.: Herleiten von semantischen Beziehungen eines Vertrags und eines Services

Die Regel zum Identifizieren aller Vertragspartner wird in Listing 4.4 definiert. Der Vertrag (*?contract*) hat auf Seite der Konsumenten das Client-System (*?consumerClient*) und die dazugehörigen Eigner des Clients (*?clientOwner*). Diese Beziehungskette wird mit *http://ex.org#consumerRelatedToContract* zum *Contract* abgekürzt. Auf Seite des Services (*?providerService*) und dem dazugehörigen Service-Anbieter (*?serviceProvider*) wird diese Beziehungskette mit *http://ex.org#providerRelatedToContract* abgekürzt.

```
1 WENN
2       ?contract http://ex.org#hasConsumer ?consumerClient .
3       ?contract http://ex.org#hasProvider ?providerService .
4       ?service http://ex.org#hasContact ?serviceProvider .
5       ?consumerClient http://ex.org#hasContact ?clientOwner
6 DANN
7       ?contract http://ex.org#consumerRelatedToContract ?clientOwner .
8       ?contract http://ex.org#providerRelatedToContract ?serviceProvider .
9 ENDE
```

Listing 4.4: Beziehungen eines Vertrags zu seinen Vertragspartnern

Beziehungen zu einem Service

Zum Identifizieren aller Personen, die irgendeine Beziehung zu einem *Service* haben (Service-Konsumenten und Service-Anbieter) wird die entsprechende Regel in Listing 4.5 definiert. Die Beziehungskette ist die gleiche wie im vorherigen Beispiel. An dieser Stelle wird jedoch vom Service (*?service*) ausgehend jeder Vertrag (*?contract*) nach Clients (*?consumerClient*) durchsucht, um alle Eigner des Konsumentensystems des Services zu identifizieren. Sowohl der Service-Anbieter (*?serviceProvider*) als auch die Service-Konsumenten (*?clientOwner*) werden dann beidseitig mit dem *Service* referenziert, damit auch eine einfache Abfrage ausgehend vom

Service möglich ist. Beispielsweise wäre es vor einer Änderung an einem Service auch von Seiten des Service-Anbieters interessant zu wissen, wie viele und welche Konsumenten durch eine Ausfallzeit des Services betroffen wären.

```
1  WENN
2        ?service http://ex.org#hasContact ?serviceProvider .
3        ?service http://ex.org#hasContract ?contract .
4        ?contract http://ex.org#hasConsumer ?consumerClient .
5        ?consumerClient http://ex.org#hasContact ?clientOwner
6
7  DANN
8        ?service http://ex.org#serviceRelatedToUser ?serviceProvider .
9        ?serviceProvider http://ex.org#userRelatedToService ?service .
10       ?service http://ex.org#serviceRelatedToUser ?clientOwner .
11       ?clientOwner http://ex.org#userRelatedToService ?service
12 ENDE
```

Listing 4.5: Relevante Beziehungen zwischen Personen und einem Service

Überprüfung des Service-Lebenszyklus

Im Bereich der Service-Lebenszyklus-Verwaltung kann die semantische Unterstützung helfen, auf bestimmte Phasen im Service-Lebenszyklus hinzuweisen. Dies ist auch hier wieder für Services relevant, die mit einem Vertrag gebunden sind und damit genutzt werden. Die Abbildung 4.11 stellt dar, welche Beziehungen hier hergeleitet werden können. Im Unterschied zu den zuvor definierten Regeln wird hier nicht nur eine Beziehung sondern ein neuer Knoten mit einem String-Inhalt erstellt, der eine Warnmeldung für den Vertrag beinhaltet.

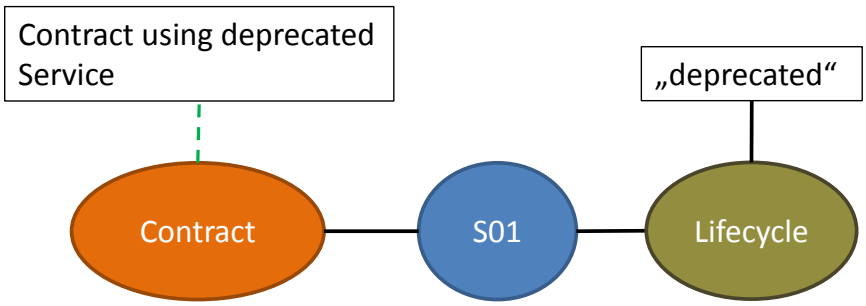

Abbildung 4.11.: Warnmeldung für Vertrag bei Nutzung eines veralteten Services

Es gibt dabei mehrere Szenarios, für welche die semantische Regel hier verfasst werden kann. Zum einen kann die Regel auf einen Service hinweisen, der veraltet oder abgeschaltet ist, zum anderen kann sie eine Warnung geben, wenn ein Service genutzt wird, der noch nicht die Testphase erfolgreich hinter sich hat (und damit potenziell größere Fehler aufweisen kann).

Die Listings 4.6 und 4.7 beschreiben genau diese Regeln.

```
1 WENN
2       ?contract http://ex.org#hasProvider ?service
3       ?service http://ex.org#hasLifecycle ?lifecycle .
4       ?lifecycle http://ex.org#hasState ("deprecated" || "retired") .
5 DANN
6       ?contract http://warn.ing#deprecatedUsage "Contract using deprecated Service"
7 ENDE
```

Listing 4.6: Vertrag in Verbindung mit einem veralteten Service identifizieren

```
1 WENN
2       ?contract http://ex.org#hasProvider ?service
3       ?service http://ex.org#hasLifecycle ?lifecycle .
4       ?lifecycle http://ex.org#hasState ("testing" || "implementation" || "design"
            || "specification") .
5 DANN
6       ?contract http://warn.ing#untestedUsage "Contract using untested Service"
7 ENDE
```

Listing 4.7: Vertrag in Verbindung mit einem ungetesteten Service

Die Regeln in den Listings 4.6 und 4.7 sind beide nur vom Vertrag ausgehend, weil die Relation zu den Service-Konsumenten schon durch die zuvor definierten Regeln gegeben ist. Der Vertrag wird in beiden Regeln (über den Service und den Lebenszyklus des Services) auf den Status des Lebenszyklus geprüft. Sollte in beiden Fällen ein Status festgestellt werden, zu dem eine Warnung ausgegeben werden könnte, so wird der Vertrag mit einer Warnung „markiert“. Es gibt Reasoner, die ebenfalls anbieten, Warnmeldungen in eine Log-Datei zu schreiben, indem ähnlich wie die definierte Regel eine Datenbeschränkung definiert wird. In Kombination mit den zuvor definierten Regeln können nun Abfragen gemacht werden, die alle Service-Konsumenten anzeigen, die in einem Vertrag sind, der einen veralteten oder noch ungetesteten Service nutzt. Eine entsprechende Warnmeldung könnte dann an die Beteiligten ausgegeben werden, wenn dies in einem SOA-System, wie einer Service-Repository, erwünscht wäre.

Überprüfung von fehlenden Daten

Mit Hilfe von semantischen Regeln könnte man auch einfachere Zusammenhänge im Datensatz vom Reasoner prüfen lassen. Beispielsweise könnte man auf fehlende Informationen zu bestimmten Objekten im Datensatz prüfen. Das Listing 4.8 zeigt eine solche Regel, die überprüft, ob ein Service, der freigegeben (released) ist, einen Endpunkt angegeben hat.

```
1 WENN
2       ?service http://ex.org#hasLifecycle ?lifecycle .
3       ?lifecycle http://ex.org#hasState "released" .
4       ?service http://ex.org#hasEndpoint <NULL>
5 DANN
6       ?service http://err.or#missingEndpoint "Released Service is missing and
            Endpoint"
7 ENDE
```

Listing 4.8: Prüfung ob Service-Endpunkt angegeben ist

Ähnlich wie in den zuvor definierten Regeln im Bereich des Service-Lebenszyklus, wird hier geprüft, ob der Service im „freigegeben-“Status (*released*) ist. Wenn dies der Fall ist und der Service im Endpunkt keinen Eintrag hat, so wird der Service hier beispielsweise mit einer Fehlermeldung markiert. Man beachte an dieser Stelle, dass das Überprüfen auf nicht-Existenz sehr stark von der semantischen Sprache abhängig ist, denn es gibt Sprachen, die mit vordefinierten Filtermethoden auf nicht-Existenz prüfen. Bestimmte Reasoner würden dann ebenfalls in der Lage sein, hier wieder eine Fehlermeldung zu in eine Log-Datei zu schreiben, wenn sich eine Datenbeschränkung auf nicht-Existenz definieren lässt. Das Definieren einer solchen Regel hat den Vorteil, dass solche Prüfungen nicht im Programmcode der Anwendung geschrieben werden müssen, die diese Datenbank nutzt.

5. Implementierung

Es ist naheliegend, die Implementierung in dieser Arbeit als Erweiterung des SOA Governance Repository umzusetzen, weil sowohl das Konzept als auch das Repository auf dem gleichen Datenmodell in [KSM14] aufbauen. Die Basisontologie des Konzepts (Unterkapitel 4.1.1) muss daher nicht neu implementiert werden und kann so, wie sie in dem SOA Governance Repository vorkommt, verwendet werden. Auch die Prüfung auf Wiederverwendung eines Vokabulars muss für die bereits vorhandene Basis des Datenmodells nicht mehr gemacht werden, weil dies zuletzt in [Miy15] gemacht wurde.

Bevor das Datenmodell erweitert werden soll, wird hier zunächst die Architektur des SOA Governance Repositories (SGR) vorgestellt. Das SGR lässt sich, wie in Abbildung 5.1 dargestellt, in vier Ebenen beschreiben: der Daten-Ebene, der Komponenten-Ebene, der Interoperabilitätsebene und der Präsentationsebene [KM16].

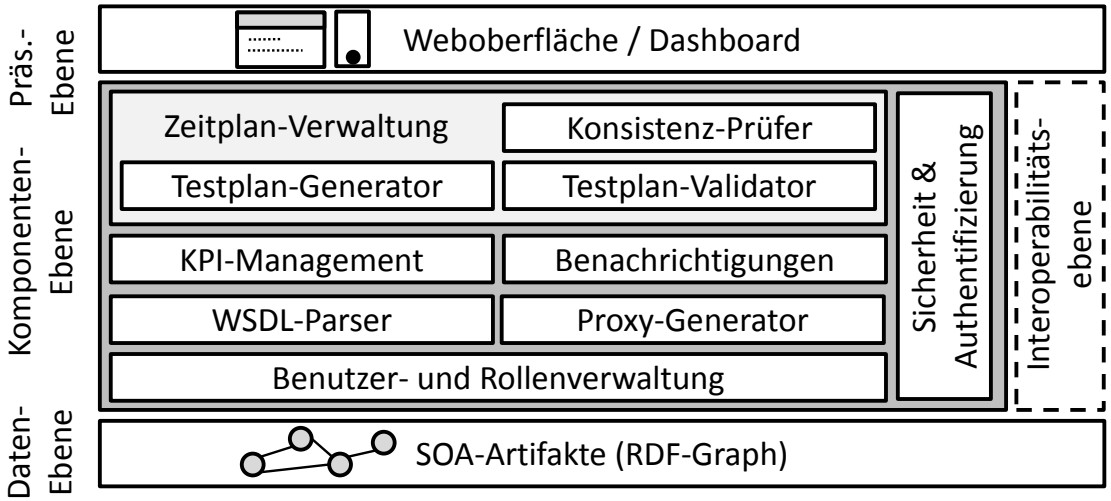

Abbildung 5.1.: Architektur des SGR (angelehnt an [KM16])

Das SGR ist komplett mit der Java Enterprise Edition (Java EE)[1] implementiert. Die **Daten-Ebene** gibt einen Zugriff auf alle (Meta-)Daten des SGR. Als Datenabstraktionsschicht wird

[1] Java EE: http://www.oracle.com/technetwork/java/javaee/overview/index.html

Empire[2] verwendet. Empire ist eine Implementierung der Java Persistence API [3] für RDF-Datensätze. Die Datenbank selbst ist ein Triple-Store von RDF4j [4]. Die **Komponenten-Ebene** beschreibt, wie sich das SGR in logische Komponenten unterteilen lässt. In dieser Ebene gibt es:

Benutzer- und Rollenverwaltung Verwaltet alle Stakeholder im SGR. Es gibt administrative Benutzer, Service-Anbieter, Service-Konsumenten und einige mehr. Durch die verschiedenen Rollen und Verantwortlichkeiten ist eine feingranulare Rollenverwaltung notwendig. Dies wird mit Hilfe der allgemeinen Rollen (GeneralRole) und der spezifischen Rollen (SpecificRole) gelöst, wie im Konzept bereits vorgestellt wurde.

KPI-Management Stellt in einer übersichtlichen Ansicht (dem SGR Dashboard [Bil16]) aktuelle Informationen, wie Key Performance Indikatoren (KPI), dar.

WSDL-Parser und Proxy-Generator Diese logische Komponente hat zwei Funktionen: Zum einen können WSDL-Schnittstellenbeschreibungen von Services vollständig ins Datenmodell importiert werden können, zum anderen kann der Prozess der Serviceerstellung vereinfacht werden, indem WSDL-Dateien beim Ausfüllen von Feldern einer Service-Maske automatisch durch das Parsen der WSDL-Datei gefüllt werden [Kra16a]. Der Proxy-Generator erstellt automatisch REST-to-SOAP Schnittstellen, für Services, die in ihrer Schnittstellenbeschreibung nur das SOAP-Protokoll unterstützen.

Benachrichtigungssystem Im SGR sind einige Benachrichtigung, wie das Erscheinen neuer Serviceversionen, von großer Bedeutung für Stakeholder. Daher ist es auch nötig nur relevante Nachrichten an jeweils betroffene Benutzer des SGR zu versenden. Mit Hilfe der Rollenverwaltung wird dies ermöglicht.

Zeitplan-Verwaltungssystem Für Elemente im SGR, wie Services, Serviceversionen und Business-Objekte werden Zeitpläne verwaltet. Es gibt beispielsweise Zeitpläne für geplante Lebenszyklus-Änderungen von Serviceversionen oder Business-Objekten. Der Testplan-Generator hilft bei der (semi-)automatischen Erstellung von Testzeitplänen im SGR. Der Testplan-Validator arbeitet eng mit dem Konsistenz-Prüfer zusammen, um zu überprüfen, ob bestimmte Testzeitpläne valide sind (z.B. ob sie zeitliche Regeln einhalten) [Kra16b].

Sicherheit & Authentifizierung Für alle Zugriffe im System muss eine Authentifizierung eines Benutzers erfolgen. Dafür ist die Komponente der Sicherheit & Authentifizierung verantwortlich. Mit der Authentifizierung kann ein Benutzer auch korrekt in das System der Rollenverwaltung eingeordnet werden.

[2]Empire: https://github.com/mhgrove/Empire
[3] Java Persistence API: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[4]RDF4j: http://rdf4j.org/

Das SGR bietet mit der **Interoperabilitätsebene** eine Möglichkeit über eine Schnittstelle Service Endpunkt-Informationen zu erfragen. Externe Tools können auf diese Weise das SGR wie eine Service Registry nutzen. Diese Schnittstelle könnte beispielsweise von einem Service-Bus oder anderen Klienten-Tools verwendet werden. Die **Präsentationsebene** des SGR ist eine responsive Weboberfläche, die es Benutzern von jedem Endgerät ermöglicht, die Plattform zu benutzen. Als UI-Framework wird für die Benutzeroberfläche Primefaces[5] verwendet. Diese Ebene beinhaltet ein Dashboard, das als erste Seite nach dem Login jedes Benutzers wichtige KPI-Werte für die jeweilige Rolle des Benutzers anzeigt. Service-Anbieter sollen beispielsweise sehen, wie viele Anfragen Ihre eigenen Services erhalten haben. Zusätzlich bietet die Präsentationsebene eine Übersicht aller Services, Serviceversionen, Systeme (von Konsumenten) und Business-Objekte, wobei jede Ansicht dank der Rollenverwaltung auch nur erlaubte Informationen gemäß den Benutzerrechten anzeigt. Es gibt auch viele ausführliche Dialoge zur Erstellung sämtlicher Objekte, die Teil einer SOA sind. Diese Dialoge dienen als Leitfaden für Benutzer, die neu im Bereich einer SOA sind. In den folgenden Kapiteln wird die Implementierung des im Kapitel 4 vorgestellten Konzepts umgesetzt.

5.1. Erweiterung des Datenmodells

Für jede Erweiterung im Datenmodell muss vor der Implementierung jeder Erweiterung geprüft werden, welches Vokabular verwendet werden soll und ob es nicht bereits ein wiederverwendbares Standard-Vokabular für den Anwendungsfall gibt. Dies wird in den folgenden Unterkapiteln gemacht: Es wird zuerst eine Prüfung auf ein wiederverwendbares Vokabular durchgeführt und dann wird das Datenmodell des SOA Governance Repositories erweitert.

5.1.1. Schlüsselwörter

Die erste Erweiterung des Basismodells der SOA Governance Repository sind Schlüsselwörter, die bestimmte Eigenschaften von Services beschreiben sollen (Konzept-Kapitel 4.1.2). Es gibt einige Ontologien für Schlüsselwörter, die aber auf bestimmte Anwendungsfälle oder Domänen zugeschnitten ist. Beispielsweise gibt es von der British Broadcasting Corporation eine Ontologie: die „Creative Work Ontology“ [BBC12]. Diese Ontologie hat zwar Schlüsselwörter (Tags), diese sind aber primär für die Beschreibung von News-Artikeln und Blog-Posts auf der BBC-Webseite und damit nicht anwendbar auf Services. Dann gibt es die „structure and semantics of a collection of tags“ (SCOT) Ontologie, die für soziale Netzwerke zugeschnitten ist und mit den Schlüsselwörtern primär Online-Nutzer beschreibt [Col05]. Diese ist etwas flexibler definiert, weil Schlüsselwörter neben Benutzern auch Ressourcen oder Anwendungen beschreiben können sollen, ist aber trotzdem hauptsächlich für Online-Nutzer sozialer Netzwerke definiert und bringt viel Vokabular mit, das nicht wiederverwendet werden könnte

[5] Primefaces: https://www.primefaces.org/

im Kontext des SGR. Die Modular Unified Tagging Ontology (MUTO) ist ein Standard, der solche Schlüsselwörter flexibel genug beschreibt, um sie im Kontext von Services zu verwenden [Loh11]. Im MUTO-Vokabular wird ein Schlüsselwort „Tag“ genannt. Tags werden im MUTO-Vokabular aber nicht direkt an eine Ressource gebunden, sondern sind über ein Zwischenelement namens „Tagging“ angebunden. Ein Tagging ist eine Sammlung von Tags und ist an genau eine Ressource gebunden (diese Ressource wird im Falle des SOA Governance Repositories eine Serviceversion sein).

Im Datenmodell des SOA Governance Repositories müssen dafür zwei neue Klassen angelegt werden: eine Klasse für *Taggings* und eine für *Tags*. In der Abbildung 5.2 ist der relevante Ausschnitt für diese Erweiterung in Form eines Klassendiagramms zu sehen. Eine *ServiceVersion* hat neben den bereits existierenden Feldern (die mit (...) abgekürzt sind) eine Liste von *Taggings*. Die Klasse *Tagging* hat wiederum eine Liste von *Tags* und genau eine *ServiceVersion*, der es angehört. *Tags* sind im SOA Governance Repository mit ihrem *tagLabel* (der *Tag*-Bezeichnung als String) einzigartig und haben eine Liste von *Taggings*, an die sie gebunden sein können. Ein *Tag* soll im Kontext der SOA Governance Repository von mehreren *Taggings* verwendet werden können, um einfacher Ähnlichkeiten von Serviceversionen festzustellen.

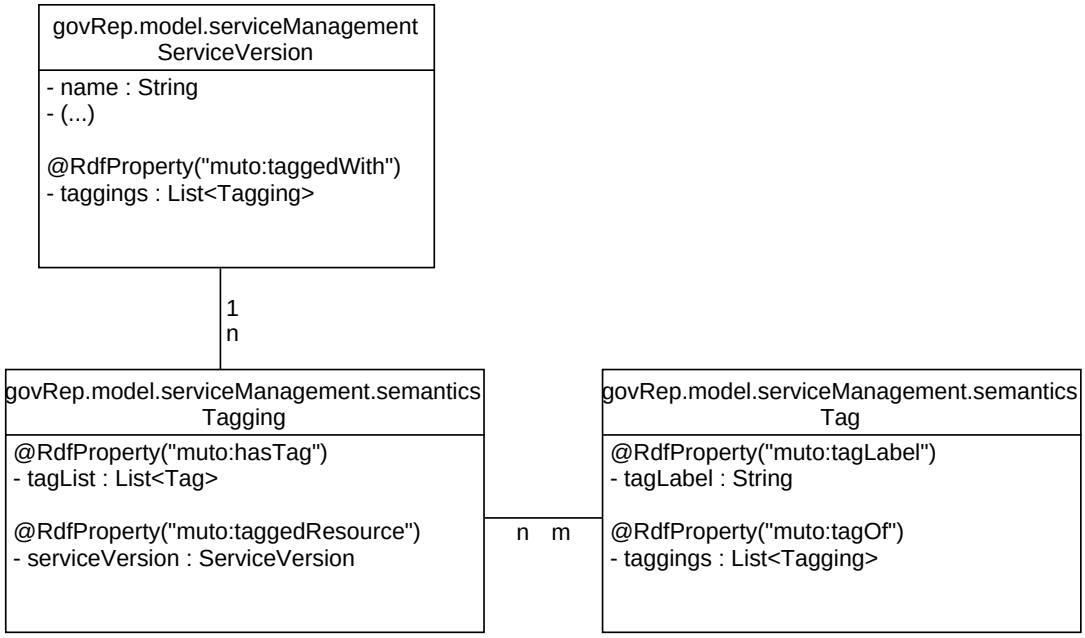

Abbildung 5.2.: ServiceVersion-, Tagging- und Tag-Klasse

5.1.2. Taxonomie

Taxonomien sind die zweite Erweiterung des Datenmodells (Konzept-Kapitel 4.1.3) und werden im SOA Governance Repository als Kategorisierung für Serviceversionen eingesetzt. Dabei soll

eine Serviceversion an mehrere Kategorien angebunden werden können, um diese mit anderen Serviceversionen zu gruppieren, die eine Ähnlichkeit haben. Für Kategorien oder Taxonomien gibt es viele Vokabulare, die allerdings für Domänen der Pflanzen- oder Tierartenforschung erstellt wurden. Beispielsweise gibt es die Biological Taxonomy Vocabulary [Buz02] und deren spezifischere Untervokabulare: für Pflanzenarten [Buz03a] und für Tierarten [Buz03b]. Alle Taxonomien in diesen Vokabularen sind sehr speziell für Pflanzenkategorien oder Tierkategorien und Hierarchien. Diese eignen sich nicht für die Kategorisierung von Serviceversionen. Aus diesem Grund werden in dieser Arbeit für die Serviceversionskategorien selbst und die hierarchischen Beziehungen zwischen Kategorien eigene Bezeichner gewählt. Es gibt allerdings für die Verbindung zwischen Services und Kategorien ein standardisiertes Vokabular, das diese Beziehung für den Kontext des SOA Governance Repositories passend beschreibt: Good Relations [Goo10]. Dieses Vokabular definiert mit der Beziehung „gr:category“, dass Ressourcen (wie Services) mit einer Kategorie verbunden werden können.

Folglich ist in der Abbildung 5.3 diese Erweiterung in Form eines Klassendiagrammes zu sehen. Es ist wieder die Klasse für Serviceversionen (*ServiceVersion*) abgebildet, deren Felder aus Zwecken der Übersichtlichkeit mit (...) abgekürzt sind. Eine *ServiceVersion* wird nun mit der Beziehung *gr:category* mit einer Liste von Kategorien verbunden. Die *Category*-Klasse definiert diese Kategorien und die Beziehungen zwischen den Kategorien. Es gibt einen Kategorie-Namen (*categoryName*) für die Bezeichnung der Kategorie und Beziehungen für eine Eltern-Kategorie (*parentCategory*) und mehrere Kinder-Kategorien (*subCategories*), damit eine Hierarchie von Kategorien erstellt werden kann.

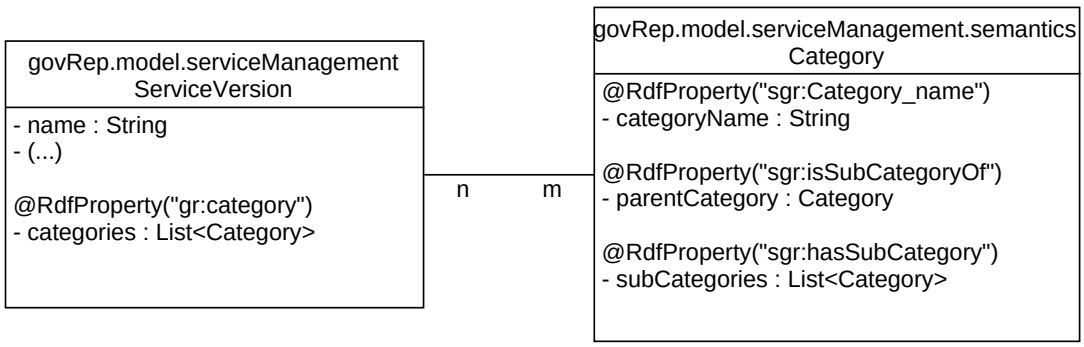

Abbildung 5.3.: ServiceVersion- und Category-Klasse

5.2. Erweiterung der Benutzeroberfläche

Mit der Erweiterung des Datenmodells des SOA Governance Repository kann nun die Benutzeroberfläche angepasst werden, um die Erweiterungen zu nutzen. Die nächsten Unterkapitel beschreiben unter anderem Erweiterungen der Benutzeroberfläche, die aufgrund von

Schlüsselwörtern und der Taxonomie nötig sind. Das beinhaltet zum einen die Änderung der Serviceversionsansicht, denn Serviceversionen können nun mit Schlüsselwörtern angelegt und bearbeitet werden. Zum anderen kann eine Serviceversion aber auch einer oder mehrerer Kategorien einer Taxonomie angehören. Auch dies muss die Ansicht sowohl beim Anlegen als auch beim Bearbeiten einer Serviceversion unterstützen. Zusätzlich können ähnliche Serviceversionen anderer Services nun auch angezeigt werden, sofern Serviceversionen dieselben Schlüsselwörter haben. Auch dies soll in der detaillierten Ansicht einer Serviceversion angezeigt werden. Das SOA Governance Repository nutzt die RDF4j-Datenbank und bringt damit einen Reasoner mit, der mit Hilfe von SPIN-Regeln neue Beziehungen und Objekte inferieren kann, oder Datensätze auf bestimmte Regeln überprüfen kann. Solche SPIN-Regeln können nur über die Workbench eingepflegt werden, einer separaten Anwendung die das Manipulieren der Datensätze in der Datenbank ermöglicht. Damit SPIN-Regeln auch benutzerfreundlicher in die Datenbank gelangen können, wird das SOA Governance Repository erweitert, um Regeln auf der Benutzeroberfläche anzulegen oder über eine Import-Funktion beim Login einzutragen. Auf einer beispielhaften Datenbank werden diese Erweiterungen in den folgenden Unterkapiteln erläutert und in Screenshots des SOA Governance Repositories dargestellt.

5.2.1. Ansicht der Serviceversionen

Serviceversionen können nun zusätzlich mit Schlüsselwörtern beschrieben werden und einer oder mehrerer Kategorien einer Taxonomie angehören. In der Ansicht zum Anlegen von Serviceversionen wurden entsprechend Eingabefelder hinzugefügt, die es ermöglichen dies für Serviceversionen zu definieren. Die Abbildung 5.4 zeigt einen Screenshot des Dialogs zum Anlegen einer Serviceversion.

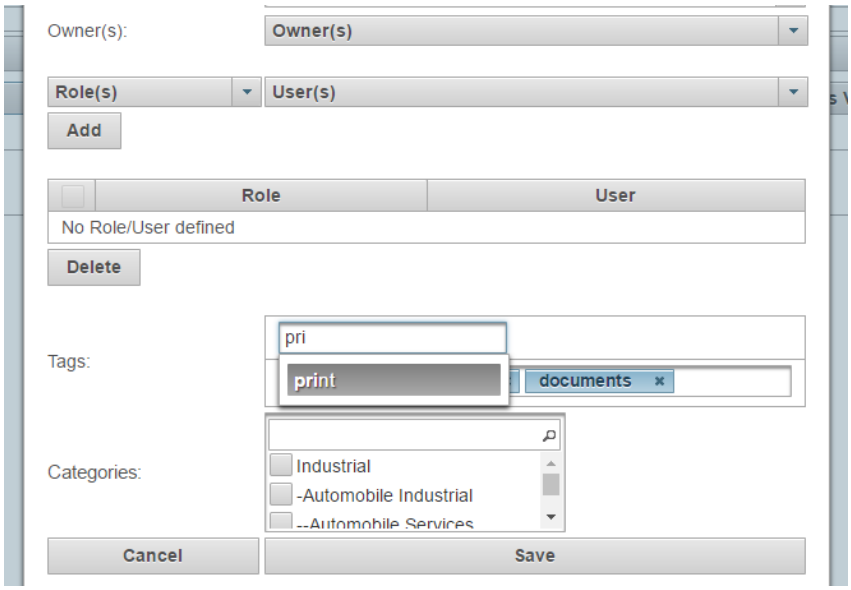

Abbildung 5.4.: Hinzufügen einer Serviceversion mit Schlüsselwörtern und Kategorie

Für die Schlüsselwörter (*Tags*) gibt es zwei Eingabefelder. Das obere Eingabefeld unterstützt eine asynchrone, serverseitige Suche nach bereits existierenden Schlüsselwörtern, sofern der Service-Anbieter Schlüsselwörter im Repository wiederverwenden will. Wenn der Service-Anbieter ein Schlüsselwort wiederverwenden will, so wählt er aus den Vorschlägen im Dropdown ein passendes Schlüsselwort aus. Nach der Auswahl kann er mit der ENTER-Taste bestätigen und das Schlüsselwort wird analog dem bereits hinzugefügten Schlüsselwort „*documents*“ hinzugefügt. Will er ein neues Schlüsselwort anlegen, das noch nicht existiert, so tippt er es einfach ein und bestätigt ebenfalls mit der ENTER-Taste. Für die Kategoriezugehörigkeit gibt es eine Auswahl-Liste von Kategorien, in der ein Service-Anbieter eine oder mehrere Kategorien angeben kann, denen die Serviceversion angehören soll (durch Auswahl der *Checkboxen*). Die Auswahl-Liste der Kategorien beginnt bei Kategorien auf der höchsten Ebene (ohne Einrückung) und jede untergeordnete Kategorie ist zunehmend mit „-“-Symbolen eingerückt, falls sie der zuvor angezeigten Kategorie in der Liste untergeordnet ist. Das Einpflegen und Aktualisieren von Kategorien und den Taxonomien bestimmter Domänen, in denen das SGR Einsatz finden soll, gehört zum Aufgabenbereich eines Administrators. Die Kategorien gelangen momentan nur über die Workbench in die Plattform: Die Importfunktionen lassen beispielsweise den direkten Upload von RDF-Datensätzen zu, mit denen man solche Kategorien beschreiben könnte. Nachdem alle nötigen Felder eingegeben sind, wird die Serviceversion durch Klick auf *Save* angelegt.

In der detaillierten Ansicht zum Bearbeiten einer Serviceversion und den zugehörigen Schlüsselwörtern und Kategorien verfährt ein Service-Anbieter ähnlich, wie bei dem Dialog zum Anlegen einer Serviceversion. Die Abbildung 5.5 zeigt diese Ansicht.

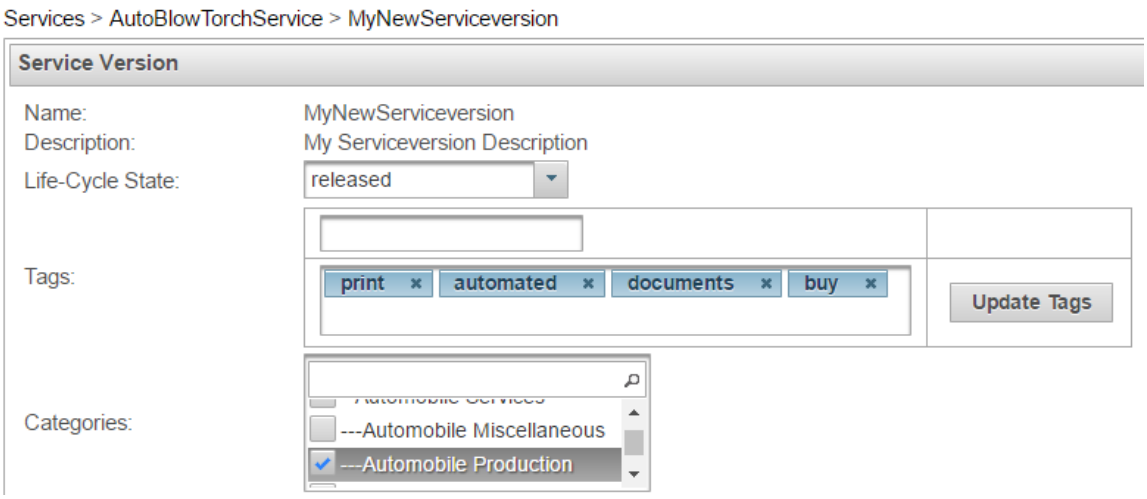

Abbildung 5.5.: Bearbeiten einer Serviceversion mit Schlüsselwörtern und Kategorie

Das Feld zum Eintippen eines Schlüsselworts (*Tags*), das eine asynchrone Suche nach Schlüsselwörtern ermöglicht, ist hier wiederzuerkennen und darunter das Feld aller Schlüsselwörter, die aktuell die Serviceversion beschreiben. Die Auswahl-Liste für Kategorien ist ebenfalls

im Screenshot zu sehen. Die Funktion dieser Eingabefelder und des Dropdowns ist identisch mit der Funktion im Dialog zum Anlegen der Serviceversion bis auf das Speichern der neuen Sammlung von Schlüsselwörtern. Sofern Schlüsselwörter hinzugefügt, gelöscht oder bearbeitet wurden, muss die Änderung über den *Update Tags*-Button gesichert werden.

Zusätzlich gibt es in der detaillierten Ansicht der Serviceversion (unten) die Ansicht der ähnlichen Serviceversionen. Abbildung 5.6 stellt eine Tabelle dar, in der ähnliche Serviceversionen mit ihrem Namen und der Anzahl gleicher Schlüsselwörter in der detaillierten Ansicht einer Serviceversion angezeigt werden. Die erste Spalte beschreibt den Namen der ähnlichen Serviceversionen und die zweite Spalte zählt die Anzahl gleicher Schlüsselwörter (sortierbar). Das Kriterium für die Anzahl gleicher Schlüsselwörter, ab der eine Serviceversion als „ähnlich“ bewertet wird, sollte je nach Gesamtanzahl aller Serviceversionen im System angepasst werden. Da für demonstrative Zwecke eine kleine Beispieldatenbank verwendet wurde (unter 100 Services), gibt es hier ähnliche Serviceversionen schon ab einem gleichen Schlüsselwort. Über den Button mit dem Lupen-Symbol in der letzten Spalte gelangt man zu einer detaillierten Ansicht der jeweils ähnlichen Serviceversion.

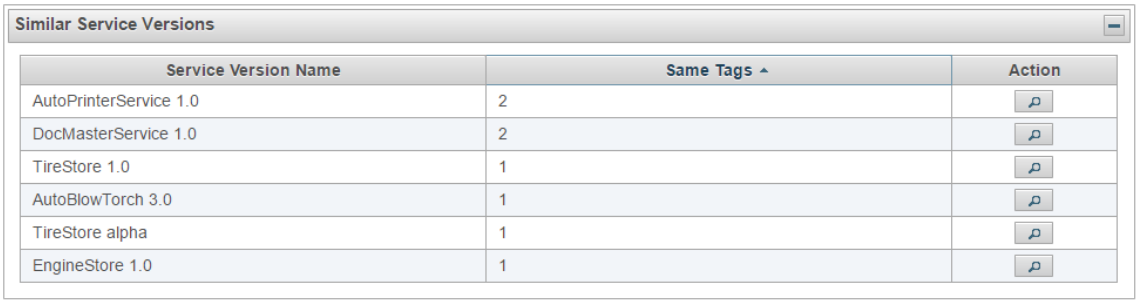

Abbildung 5.6.: Tabelle für ähnliche Serviceversionen

5.2.2. Semantische Suche

Mit der Einführung der semantischen Unterstützung und der Erweiterung durch Schlüsselwörter und einer Taxonomie für Serviceversionen kann nun die Ansicht der Services um eine Suche erweitert werden (Konzept-Kapitel 4.2). Abhängig von den gewählten Suchkriterien kann jede Suche nicht nur nach dem Namen eines Services ausgeführt werden, sondern auch nach vergebenen Schlüsselwörtern in den Serviceversionen. Die Suchergebnisse können durch Kategorien als Filter eingeschränkt werden. In der Abbildung 5.7 wird dieser neue Abschnitt für die Servicesuche gezeigt.

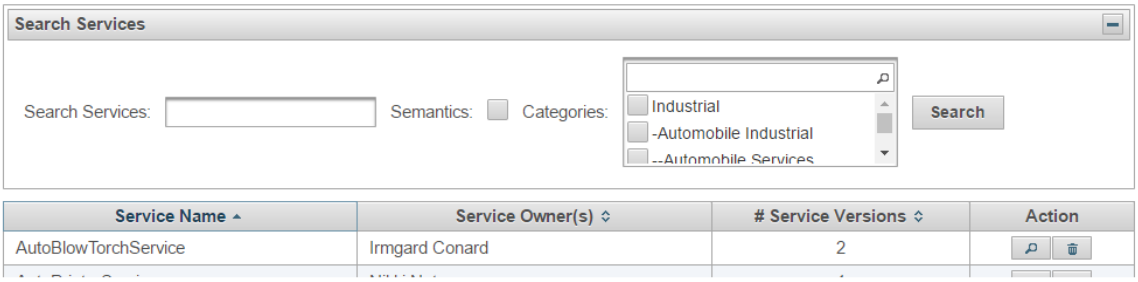

Abbildung 5.7.: Semantische Suche nach Services und Serviceversionen

Die Suche unterstützt die im Konzept definierten Suchszenarien (Kapitel 4.2). Man kann nur nach dem Namen eines Services suchen, kann aber weitere Suchtreffer erlangen, wenn man die semantische Suche auswählt (Checkbox: *Semantics*). Das Auswählen der semantischen Suche erweitert die Suchergebnisse durch Serviceversionen, die in ihren Schlüsselwörtern eine (Teilwort-)Übereinstimmung mit dem Suchbegriff haben. Die Suche kann wiederum weiter eingeschränkt werden, wenn anstatt der gesamten Menge der Services und Serviceversionen nur in bestimmten Kategorien gesucht werden soll (Auswahl-Liste: *Categories*). Jede Suche betrachtet dabei unabhängig von den gewählten Suchfiltern die Namen der Services und Serviceversionen. Bei einem leeren Suchfeld ohne gewählte Filter werden alle Services des SOA Governance Repositories aufgelistet.

5.2.3. SPIN-Regeln: Import und Editor

SPIN-Regeln können in RDF4j nur über die RDF4j-Workbench (im folgenden nur „Workbench“ genannt) eingepflegt werden. Die Workbench ist eine Anwendung, die Tools zur Verfügung stellt, um Datensätze in der Datenbank zu manipulieren. Nicht jeder Benutzer des SOA Governance Repositories wird den Zugriff zur Workbench haben oder mit deren Umgang vertraut sein. Zusätzlich ist die Grundfunktion vom SPIN-Reasoner in RDF4j so implementiert, dass SPIN-Regeln, die im Nachhinein auf eine bereits existierende Datenbank mit vorhandenen Daten geschrieben werden, nicht automatisch ausgeführt werden. Erst das Bearbeiten bestimmter Objekte führt zum Ausführen einer SPIN-Regel, weil diese immer an bestimmte RDF-Klassen gebunden sind. Dies ist problematisch, weil das SGR auch die Ausführung von Regeln auf existierende Datensätze unterstützen soll. Die Erweiterung für SPIN-Regeln soll diese Probleme lösen und Regeln einfacher ins System bringen können. Abbildung 5.8 zeigt den Login, der nun erweitert ist durch eine weitere Checkbox zum Auswählen.

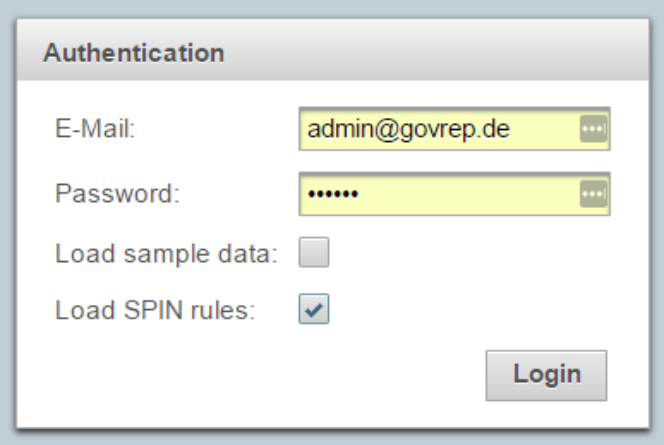

Abbildung 5.8.: Importieren von SPIN-Regeln beim Login

Wenn diese Checkbox (*Load SPIN rules*) beim Login ausgewählt wird, werden im lokalen Ordner des SOA Governance Repository unter „C:/GovRepDataFolder/spinrules“ sämtliche SPIN-Regeln gelesen und in die Datenbank eingetragen. Ein Administrator kann dort Regeln in Form von Turtle-Dateien ablegen und den Vorgang zum Importieren durch das Auswählen der Checkbox beim Login starten. Jede Regel, die in diesem Ordner gelöscht wird, wurde erfolgreich vom System eingetragen. Regeln, die nicht gelöscht wurden, haben möglicherweise ein Problem verursacht und konnten nicht eingetragen werden. Jede dieser Aktionen wird zusätzlich auch im System in die Log-Datei geschrieben, um beidseitig zu überprüfen, ob und welche Regeln eingetragen wurden und welche nicht (und aus welchen Gründen dies nicht funktioniert hat). SPIN-Regeln, die auf diese Weise ins System eingetragen wurden, werden jedoch ähnlich wie über die RDF4j-Workbench nicht automatisch auf existierenden Datensätzen ausgeführt.

Damit auch das Problem des automatischen Ausführens einer SPIN-Regel gelöst wird, wurde eine weitere Ansicht im SOA Governance Repository eingebaut: Der Reiter „SPIN Rules“ im Navigationsmenü ist diese neue Ansicht. Das Formular dieser Ansicht orientiert sich am Aufbau einer allgemeinen SPIN-Regel oder eines SPIN-Constraints (SPIN-„Einschränkung“). Zur Veranschaulichung wird zuvor ein kurzer SPIN-Constraint in Listing 5.1 gezeigt und danach, wie dieser Analog in die Ansicht eingepflegt werden kann. In diesem Listing sind zuerst Prefix-Deklarationen (@prefix) zu sehen, die nötig sind, um alle Ressourcen der Regel eindeutig zu identifizieren. Im Anschluss wird mit *sgr:user* die Klasse angesprochen, die auf diesen SPIN-Constraint vom SPIN-Reasoner geprüft werden muss. In diesem Fall sollen mit der Regel alle Benutzer des Systems auf fehlende E-Mail Adressen geprüft werden. Im Falle einer Regelverletzung soll eine Warnung in die Log-Datei geschrieben werden, mit der Meldung „Missing Mail Adress for Repository User“. Der *SPIN-Constraint* beschreibt zuerst, dass der Inhalt über eine *sp:Construct*-Struktur definiert wird (dies ist das gängige Konstrukt für komplexere Zusammenhänge und ist für SPIN-Regeln und SPIN-Constraints gleich). Innerhalb des *sp:text*-Abschnitts, der mit dreifachen Anführungszeichen beginnt und endet, wird der

Inhalt der Regel in SPARQL-Syntax definiert. Innerhalb der geschweiften Klammern nach CONSTRUCT, wird die „Konsequenz“ des SPIN-Constraints definiert. Die Zeile 14 ist für alle SPIN-Constraints gleich, es definiert den Typ der Regelverletzung. Über *spin:violationRoot* wird das Kernobjekt der Regelverletzung markiert (in diesem Fall der Benutzer selbst mit *?this*). Der Pfad zu einer Beziehung, die vom Kernobjekt ausgeht, ist mit *spin:violationPath* markiert und das Zielobjekt mit dem Wert, der die Regel verletzt, wird mit *spin:violationValue* angezeigt (in diesem Fall *?mailAdress* die fehlende E-Mail-Adresse). Die Bedingung zur Prüfung der Regelverletzung und dem Ausführen der Konsequenz (CONSTRUCT) wird in der WHERE-Klausel innerhalb der geschweiften Klammern definiert. In diesem Fall ist diese Bedingung in einer Zeile zusammengefasst: Es ist ein Filter, der auf die Nicht-Existenz der E-Mail-Adresse prüft. Ist diese Bedingung für irgendeinen Benutzer erfüllt, so wird die Konsequenz der Regel für diesen Benutzer ausgeführt.

```
1  @prefix spin: <http://spinrdf.org/spin#> .
2  @prefix sp: <http://spinrdf.org/sp#> .
3  @prefix sgr: <http://sgr#> .
4  @prefix sgrspin: <http://sgrspin#> .
5  sgr:user
6        spin:constraint
7        [     a sp:Construct;
8              sp:text """
9                    PREFIX sgr: <http://sgr#>
10                   PREFIX sgrspin: <http://sgrspin#>
11                   PREFIX spin: <http://spinrdf.org/spin#>
12                   PREFIX sp: <http://spinrdf.org/sp#>
13                   CONSTRUCT {
14                         _:violation a spin:ConstraintViolation ;
15                         spin:violationRoot ?this ;
16                         spin:violationPath sgr:mailAdress ;
17                         spin:violationValue ?mailAdress ;
18                         spin:violationLevel spin:Warning ;
19                         rdfs:label "Missing Mail Adress for Repository User"
20                   }
21                   WHERE {
22                         FILTER NOT EXISTS {?this sgr:mailAdress ?mailAdress} .
23                   }
24                   """
25       ] .
```

Listing 5.1: Beispiel eines SPIN-Constraints für fehlende Mail-Adressen

Diese Regel lässt sich in der Ansicht für SPIN-Rules nachbilden und wird in Abbildung 5.9 gezeigt. Das Formular nimmt alle nötigen variablen Daten für die SPIN-Rule oder den SPIN-Constraint auf und fügt sie in eine Vorlage ein, um die Regel in die Datenbank einzutragen. Zunächst muss im Formular die Klasse gewählt werden, an der *SPIN-Rule* oder *SPIN-Constraint* geprüft werden sollen. Im Beispielfall war dies die Klasse der *sgr:user* – der Benutzer des SOA Governance Repository. Anschließend wird der Typ gewählt, der im Beispielfall ein *SPIN-Constraint* war. Der Regelinhalt ist der Teil, der zuvor im *sp:text*-Teil zu sehen war, mit dem

Unterschied, dass weitere Präfix-Definitionen bereits in der internen Vorlage für diese Regel definiert sind (Präfixe müssen und sollen an dieser Stelle also nicht mehr definiert werden).

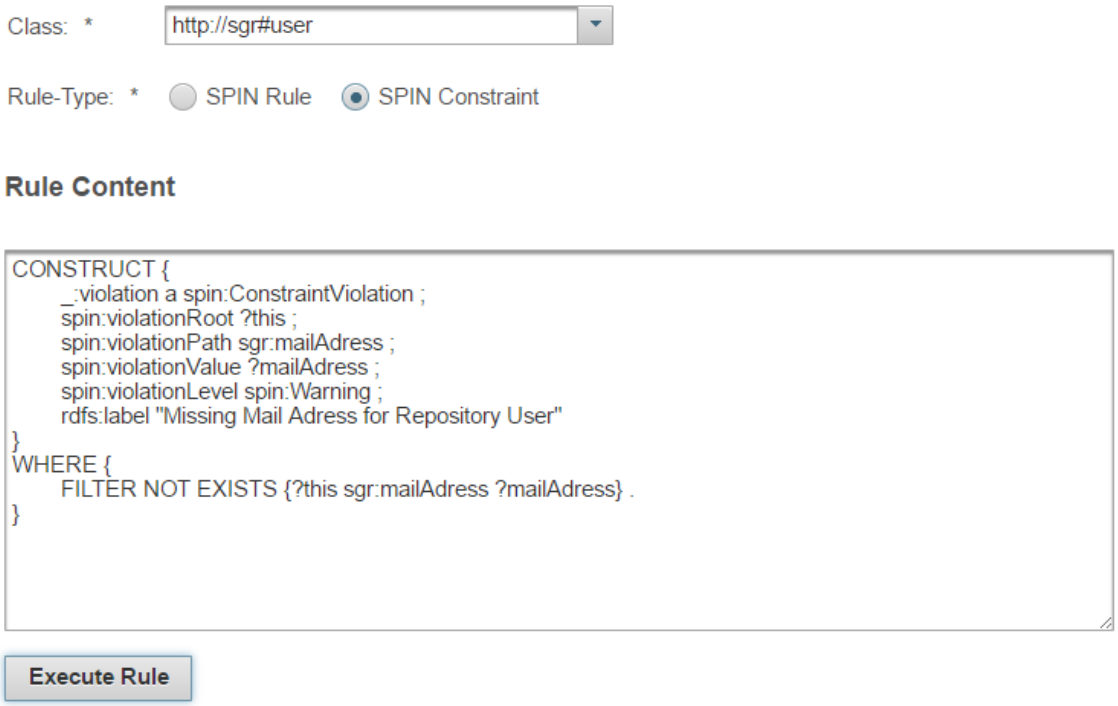

Abbildung 5.9.: SPIN-Constraint in SPIN Rules-Ansicht nachgestellt

Wird diese Regel nun über den Button *Execute Rule* abgesendet, passiert folgendes:

1. Für jede Instanz der im Formular ausgewählten Klasse wird ein temporäres Platzhalterobjekt eingefügt. Das Platzhalterobjekt hat keine inhaltliche Bedeutung und wird im dritten Schritt wieder gelöscht. Es ist lediglich ein Triple, das über das Prädikat „<http://sgrspin#dummy>“ einen String mit dem Inhalt „check“ an die Instanzen der ausgewählten Klasse anfügt.
2. Die Regel wird in die Datenbank geschrieben.
3. Das zuvor eingefügte Platzhalterobjekt wird wieder in jeder Instanz der ausgewählten Klasse gelöscht. Das Löschen dieses Platzhalterobjektes führt zum Ausführen der SPIN-Regel oder des SPIN-Constraints.

5.3. Initiale SPIN-Regeln für SOA Governance Repository

Neben den Erweiterungen im Programmcode des SOA Governance Repository sind noch einige SPIN-Regeln und SPIN-Constraints erstellt worden, die als initiale Regeln im SOA Go-

vernance Repository eingesetzt werden. Das heißt, dass diese Regeln grundsätzlich bei jeder Instanziierung des SGR initial in der Datenbank vorliegen werden, um das SGR mit der SOA Governance zu unterstützen. Diese Regeln helfen dem SOA Governance Repository semantisch in vielen Bereichen. Sie können entweder in Form von SPIN-Regeln neue Beziehungen inferieren oder als Kombination von SPIN-Regeln und SPIN-Constraints bestimmte Regeln im Datenmodell des SOA Governance Repositories überprüfen und einhalten. Da diese Regeln auf der Datenbank-Ebene ausgeführt werden, bringen sie einige Vorteile mit sich:

- Die Regeln fassen komplexere Zusammenhänge in wenigen Zeilen zusammen, für die viele Zeilen an Programmcode nötig wären. So kann die Komplexität des Programmcodes im SGR niedrig gehalten werden.
- Die Regeln arbeiten performanter als Programmcode im SGR. Ein Programmcode müsste zum Inferieren von Daten ganze Teile der Datenbank in den Arbeitsspeicher laden, ein Reasoning ausführen und diese Daten dann wieder in die Datenbank schreiben. Die SPIN-Regeln und SPIN-Constraints in der RDF4j-Datenbank arbeiten direkt auf der Datenbank und führen ihre Operationen so schneller aus.

Die Regeln werden in folgenden Unterkapiteln mit den textuellen Regelinhalten dargestellt. Regeln, die über längere Beziehungsketten inferieren, werden zusätzlich grafisch dargestellt, um den Nutzen der Regel zu verdeutlichen. Blaue Ovale stellen Objekte mit entsprechender Beschriftung dar, wie sie auch den Variablen-Namen der Regel zugeordnet werden können (z.B. ein blaues Oval mit Beschriftung „Contract“ soll für die Variable ?contract in den Regeln stehen). Blaue, durchgezogene Linien sind bereits existierende Beziehungen in der Datenbank, gestrichelte Linien sind die neu inferierten Beziehungen aus der CONSTRUCT-Klausel einer aktuell erklärten Regel.

5.3.1. SPIN-Regeln für Verträge und Serviceversionen

Im Bereich der Service-Verträge ist es wichtig alle Vertragspartner und vor allem Personen (Systembenutzer) des Vertrags zu markieren. Für diesen Fall können SPIN-Regeln eingesetzt werden, die über längere Beziehungsketten eine neue Beziehung als Abkürzung von Benutzer zu Vertrag und umgekehrt inferieren.

Beziehungen von Seiten der Service-Konsumenten

Das Listing 5.2 zeigt, wie diese Beziehungen von Vertragspartnern von Seiten der Konsumenten inferiert werden.

```
1  sgr:user rdf:type rdfs:Class;
2        spin:rule [
3              rdf:type sp:Construct;
4              rdfs:comment "Infer relations from side of Consumer (User)"^^xsd:string
                   ;
5              sp:text """
6                    PREFIX sgr: <http://sgr#>
7                    PREFIX sgrspin: <http://sgrspin#>
8                    CONSTRUCT {
9                          ?this sgrspin:userRelatedToServiceVersion ?serviceversion .
10                         ?serviceversion sgrspin:serviceVersionRelatedToUser ?this .
11                         ?this sgrspin:userRelatedToService ?service .
12                         ?service sgrspin:serviceRelatedToUser ?this .
13                         ?this sgrspin:consumerRelatedToContract ?contract .
14                         ?contract sgrspin:contractRelatedToConsumer ?this .
15                   }
16                   WHERE {
17                         ?userrolesystem sgr:user ?this .
18                         ?userrolesystem rdf:type sgr:UserRoleSystem .
19                         ?userrolesystem sgr:relationObject ?system .
20                         ?system rdf:type sgr:System .
21                         ?contract sgr:system ?system .
22                         ?contract rdf:type sgr:Contract .
23                         ?contract sgr:serviceVersion ?serviceversion .
24                         ?serviceversion rdf:type sgr:ServiceVersion .
25                         ?serviceversion sgr:service ?service .
26                         ?service rdf:type sgr:Service .
27                   }
28                   """ ;
29       ] .
```

Listing 5.2: Beziehungen von Service-Konsumenten zum Vertrag

Mit dieser Regel werden alle Beziehungen von Benutzern (Konsumenten) zu Verträgen, Services und Serviceversionen inferiert. Zunächst definiert die WHERE-Klausel über das Datenmodell des SOA Governance Repositories, dass ein Benutzer einem System angehört (Zeilen 17-19). Dieses System kann direkt als Konsumentensystem zu einem Vertrag gehören (Zeile 21). Auf Seiten der Service-Anbieter im Vertrag werden dann die betroffene Serviceversion und der übergeordnete Service identifiziert (Zeilen 23-26). Die CONSTRUCT-Klausel inferiert jetzt im Falle solch einer Beziehung (der WHERE-Klausel) Beziehungen vom „Konsument-Benutzer“ zu Verträgen, Services und Serviceversionen (und jeweils umgekehrt). Durch diese Abkürzungen können nun kurze SPARQL-Abfragen geschrieben werden, die sofort alle Konsumenten ausgehend von einem Vertrag, eines Services oder einer Serviceversion identifizieren können. Die gleiche Regel wurde auch aus der Sicht eines Vertrags geschrieben (in der ersten Zeile

annotiert mit *sgr:Contract*, statt *sgr:user*) damit auch Änderungen im Vertrag diese Regeln neu evaluieren. Die Abbildung 5.10 stellt die Regel grafisch dar.

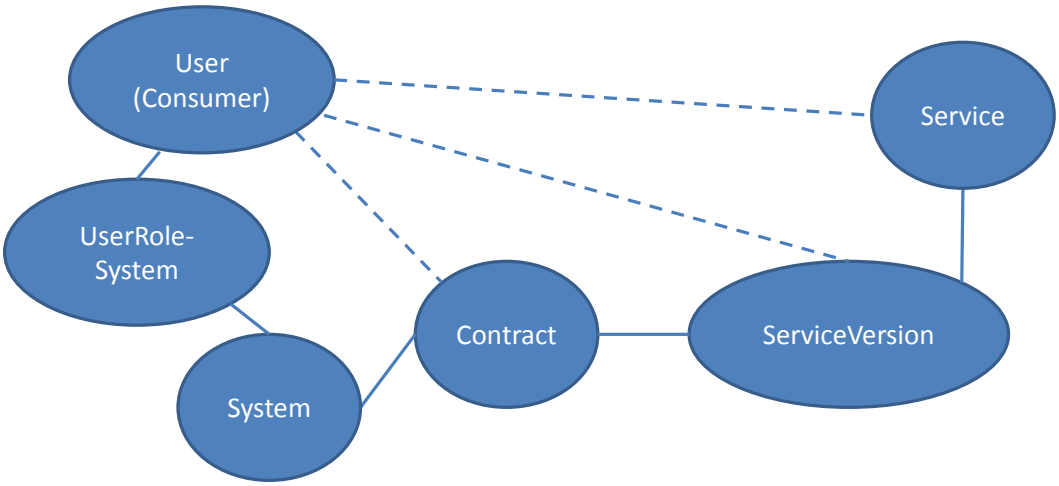

Abbildung 5.10.: Beziehungen von Service-Konsumenten zum Vertrag

Diese Regeln bewirken, dass man mit einfachen Abfragen im SGR nun ausgehend von einem Vertrag sofort eine Liste aller Service-Konsumenten findet. Beispielweise könnte man auf diese Weise nun einfacher Service-Konsumenten identifizieren, um mit ihnen die Service-Level-Agreements zur Service-Nutzung zu vereinbaren („Welche finanzielle Grundlage hat der Vertrag?“, „Wie oft darf der Service aufgerufen werden im Tag/Monat/Jahr?“,...).

Beziehungen von Seiten der Service-Anbieter

Diese Art der Beziehungen lässt sich nun auch von Seiten der Service-Anbieter beschreiben. Das Listing 5.3 zeigt eine entsprechende Regel von Seiten der Service-Anbieter.

```
1  sgr:user rdf:type rdfs:Class;
2        spin:rule [
3              rdf:type sp:Construct;
4              rdfs:comment "Infer relations from side of ServiceProvider
                   (User)"^^xsd:string ;
5              sp:text """
6                    PREFIX sgr: <http://sgr#>
7                    PREFIX sgrspin: <http://sgrspin#>
8                    CONSTRUCT {
9                          ?this sgrspin:providerRelatedToContract ?contract .
10                         ?contract sgrspin:contractRelatedToProvider ?this .
11                   }
12                   WHERE {
13                         ?userroleservice sgr:user ?this .
14                         ?userroleservice rdf:type sgr:UserRoleService .
```

```
15                      ?userroleservice sgr:relationObject ?service .
16                      ?service rdf:type sgr:Service .
17                      ?service sgr:version ?serviceversion .
18                      ?serviceversion rdf:type sgr:ServiceVersion .
19                      ?contract sgr:serviceVersion ?serviceversion .
20                      ?contract rdf:type sgr:Contract .
21                  }
22                  """ ;
23      ] .
```

Listing 5.3: Beziehungen von Service-Anbietern zum Vertrag

Erneut werden ausgehend von einem Benutzer (Zeile 1) über die bereits bekannte Beziehungskette in der WHERE-Klausel Beziehungen vom Vertrag zu den verantwortlichen Personen eines Services inferiert. Im Kontext des SOA Governance Repositories sind das beispielsweise technische Kontakte eines Services oder die Hauptverantwortlichen eines Services. Diese Regel wurde ebenfalls ausgehend von einem Vertrag geschrieben (in der ersten Zeile *sgr:Contract* statt *sgr:user*), damit auch Änderungen eines Vertrags die Regel neu evaluieren. Zusätzlich wurde diese Regel für den Fall, dass Serviceversionskontakte nicht dieselben sind wie die übergeordneten Service-Kontakte, auch für Serviceversionen erstellt (wiederum in zweifacher Ausführung für *sgr:Contract* und *sgr:user* in Zeile 1). Die Abbildung 5.11 stellt die definierte Regel für Beziehungen von Service-Anbietern zum Vertrag grafisch dar.

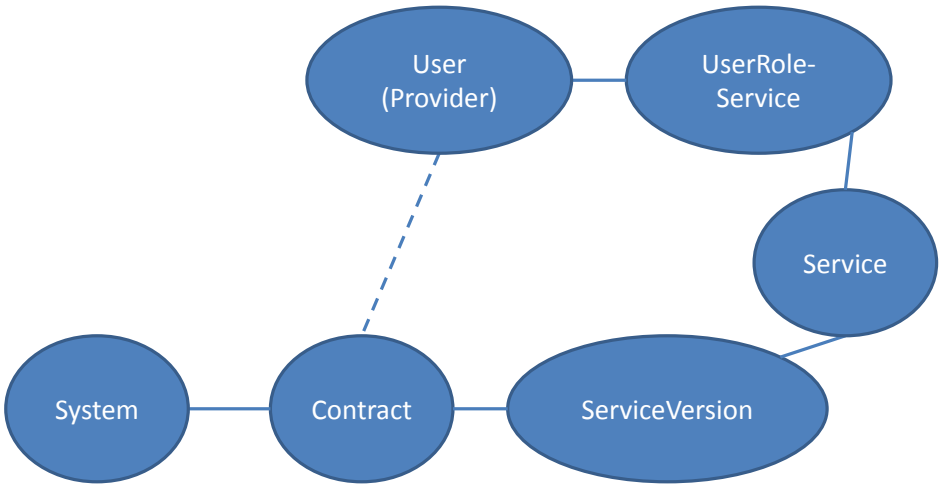

Abbildung 5.11.: Beziehungen von Service-Anbietern zum Vertrag

Der Kontakt zu Service-Anbietern lässt sich nun ausgehend vom Vertrag einfach und direkt herstellen. Dazu sind nun auch kürzere SPARQL-Abfragen möglich, die nicht mehr über mehrere Zwischen-Objekte in der Datenbank formuliert werden müssen. Die Service-Anbieter in einem Vertrag zu identifizieren ist besonders wichtig, weil Service-Konsumenten in Fragen zum Service, zum Status einer Serviceversion (beispielsweise bei Ausfällen) oder zum Erschei-

nungsdatum einer neuen Serviceversion direkt wissen, wer die Ansprechpartner in diesem Vertrag sind.

Aufbauende und ergänzende Regeln bei Verträgen

Ausgehend von einem Service wurde noch eine zusätzliche Regel erstellt, die auf den zuvor definierten Regeln aufbaut. Mit dem Hintergrund, abschätzen zu können wie kritisch ein Ausfall einer Serviceversion oder ein Update einer Serviceversion ist, sollen alle „kritischen" Beziehungen ausgehend von einer Serviceversion inferiert werden. Das sind im Modell des SOA Governance Repositories Beziehungen von einer Serviceversion zu einem Vertrag und von diesem Vertrag aus zu allen Benutzern, die an diesen Vertrag gebunden sind. Listing 5.4 zeigt diese Regel.

```
1  sgr:ServiceVersion rdf:type rdfs:Class;
2        spin:rule [
3              rdf:type sp:Construct;
4              rdfs:comment "Infer relations to ServiceVersion that can later be used
                   as critical dependencies"^^xsd:string ;
5              sp:text """
6                    PREFIX sgr: <http://sgr#>
7                    PREFIX sgrspin: <http://sgrspin#>
8                    CONSTRUCT {
9                          ?this sgrspin:serviceVersionRelatedToUser ?user .
10                         ?user sgrspin:userRelatedToServiceVersion ?this .
11                         ?this sgrspin:serviceVersionRelatedToUser ?providerUser .
12                         ?providerUser sgrspin:userRelatedToServiceVersion ?this .
13                         ?this sgrspin:serviceVersionRelatedToUser ?consumerUser .
14                         ?consumerUser sgrspin:userRelatedToServiceVersion ?this .
15                   }
16                   WHERE {
17                         ?contract sgr:serviceVersion ?this .
18                         ?contract sgrspin:contractRelatedToUser ?user .
19                         ?contract sgrspin:contractRelatedToProvider ?providerUser .
20                         ?contract sgrspin:contractRelatedToConsumer ?consumerUser .
21                         ?contract rdf:type sgr:Contract .
22                         ?user rdf:type sgr:user .
23                         ?providerUser rdf:type sgr:user .
24                         ?consumerUser rdf:type sgr:user .
25                   }
26                   """ ;
27       ] .
```

Listing 5.4: Beziehungen relevanter Benutzer zu einer Serviceversion

Die Regel baut in der WHERE-Klausel nun auf den zuvor inferierten Beziehungen der anderen Regeln von einem Vertrag aus auf. In der CONSTRUCT-Klausel werden diese Beziehungen schließlich beidseitig von einer Serviceversion zu einem Benutzer und umgekehrt abgekürzt. Die Abbildung 5.12 stellt diese Regel grafisch dar.

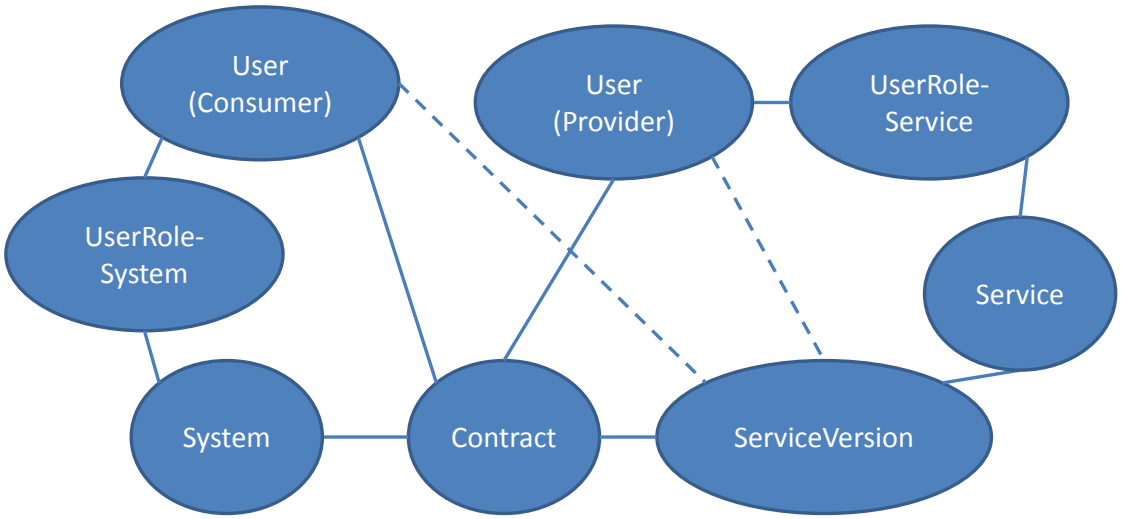

Abbildung 5.12.: Beziehungen relevanter Benutzer zu einer Serviceversion

Die Beziehung von Verträgen zu einer Serviceversion existiert bereits, jedoch sollte auch die inverse Beziehung einer Serviceversion zu einem Vertrag bestehen, weil dies auch als „kritische“ Beziehung gesehen werden kann. Im Listing 5.5 wird diese inverse Beziehung in einer kurzen Regel inferiert.

```
sgr:Contract rdf:type rdfs:Class;
      spin:rule [
            rdf:type sp:Construct;
            rdfs:comment "Infer relation from SV back to Contract"^^xsd:string ;
            sp:text """
                  PREFIX sgr: <http://sgr#>
                  PREFIX sgrspin: <http://sgrspin#>
                  CONSTRUCT {
                        ?sv sgrspin:contract ?this .
                  }
                  WHERE {
                        ?this sgr:serviceVersion ?sv .
                        ?this rdf:type sgr:Contract .
                        ?sv rdf:type sgr:ServiceVersion .
                  }
                  """ ;
      ] .
```

Listing 5.5: Inverse Beziehung von Serviceversion zum Vertrag

Damit diese Beziehungen zusammen auch als „kritische“ Beziehungen erkannt und abgefragt werden können, sollten diese mit einer Klasse markiert werden. Dafür ist keine SPIN-Regel nötig, dies kann einfach mit einem SPARQL-Update-Befehl als Standard-Datensatz initial in die Datenbank des SOA Governance Repositories geschrieben werden. Listing 5.6 zeigt

diesen kurzen SPARQL-Update-Befehl. Mit diesem Zusatz können alle kritischen Beziehungen wieder über kurze SPARQL-Queries ausgehend von einer Serviceversion gefunden werden und Service-Anbieter können so schnell und einfach entscheiden, wie kritisch ein Ausfall oder ein Update der eigenen Serviceversion ist.

```
1 INSERT DATA {
2       sgrspin:contract rdf:type sgrspin:dependencyRelation .
3       sgrspin:serviceVersionRelatedToUser rdf:type sgrspin:dependencyRelation .
4 }
```

Listing 5.6: Beziehungen als kritisch markieren

5.3.2. Weitere SPIN-Regeln und SPIN-Constraints

Es gibt einige weitere SPIN-Regeln, die sehr verschiedene Aufgaben im Datenmodell des SGR übernehmen. Diese werden in diesem Kapitel beschrieben. Es sind Regeln, die Service-Ähnlichkeiten unterstützen, die Service-Lebenszyklusphasen überprüfen, oder allgemeine Regeln, die in einem hierarchischen Kategoriebaum einer Service-Taxonomie transitive Beziehungen zwischen den Kategorien herleiten.

Ähnlichkeitsbeziehungen zwischen Serviceversionen

Durch die Erweiterung der Schlüsselwörter können nun auch mit Hilfe des SPIN-Reasoners ähnliche Services identifiziert werden. Die SPIN-Regel in Listing 5.7 beschreibt wie ein SPIN-Reasoner abkürzend ähnliche Services über dieselben Tags identifiziert und mit einer neu-inferierten Beziehung *gr:isSimilarTo* abkürzt.

```
1 sgr:ServiceVersion rdf:type rdfs:Class;
2       spin:rule [
3             rdf:type sp:Construct;
4             rdfs:comment "Infer IsSimilarTo-Relations for ServiceVersions with same
                  Tags"^^xsd:string ;
5             sp:text """
6                   PREFIX sgr: <http://sgr#>
7                   PREFIX sgrspin: <http://sgrspin#>
8                   PREFIX gr: <http://purl.org/goodrelations/v1#>
9                   PREFIX muto: <http://purl.org/muto/core#>
10                  CONSTRUCT {
11                        ?this gr:isSimilarTo ?otherserviceversion .
12                        ?otherserviceversion gr:isSimilarTo ?this .
13                  }
14                  WHERE {
15                        ?this muto:taggedWith ?tagging .
16                        ?tagging rdf:type muto:Tagging .
17                        ?tagging muto:hasTag ?tag .
18                        ?tag rdf:type muto:Tag .
19                        ?otherserviceversion muto:taggedWith ?svtagging .
```

```
20                      ?otherserviceversion rdf:type sgr:ServiceVersion .
21                      ?svtagging rdf:type muto:Tagging .
22                      ?svtagging muto:hasTag ?tag .
23                      FILTER (?otherserviceversion != ?this) .
24                  }
25                  """ ;
26      ] .
```

Listing 5.7: Ähnlichkeitsbeziehungen vom Reasoner inferieren lassen

Diese Ähnlichkeitsbeziehung (*gr:isSimilarTo*) beschreibt ebenfalls ähnliche Ressourcen (wie zum Beispiel Services) und wird hier vom „Good Relations"-Vokabular wiederverwendet [Goo10]. Die gleiche Regel wurde auch von Seiten der Schlüsselwörter erstellt, für den Fall, dass die Regel bei der Änderung an einem Schlüsselwort neu evaluiert werden muss. Diese Ähnlichkeitsbeziehung kann dem SGR helfen, ähnliche Services über kurze SPARQL-Queries zu identifizieren. Auch hier ist dank der Regel der Umweg über dieselben Tags zwischen mehreren Serviceversionen im SPARQL-Query nicht mehr nötig. Die Abbildung 5.13 stellt diese Regel grafisch dar. Es sind zwei beliebige Serviceversionen (*SV_1* und *SV_2*), die über ihr *Tagging* ein gleiches *Tag* (Schlüsselwort) referenzieren. Die Ähnlichkeitsbeziehung wird folglich zwischen den beiden Serviceversionen inferiert.

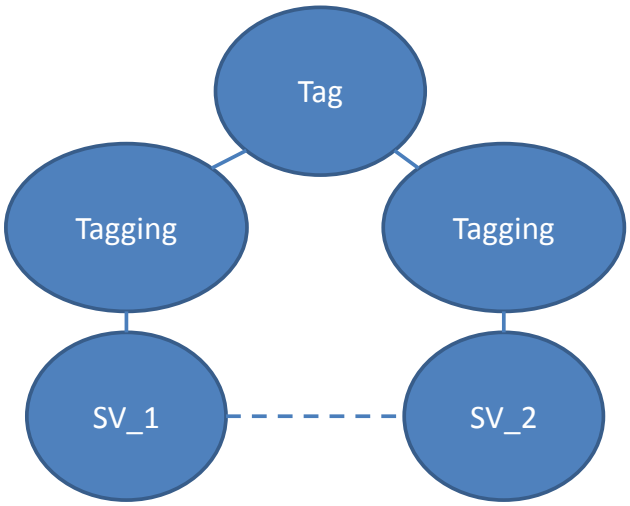

Abbildung 5.13.: Ähnlichkeitsbeziehung zwischen zwei Serviceversionen mit gleichem Tag

SPIN-Constraint für fehlende Endpunktadresse

SPIN-Constraints wurden ebenfalls als Standard in das Service Governance Repository eingebaut. Sie helfen, bestimmte Regeln im Datenmodell zu überprüfen, ohne dies im Programmcode beschreiben zu müssen. Beispielsweise können Benutzer auf fehlende aber nötige Kontaktinformationen geprüft werden, oder die Benutzung von Serviceversionen untersucht werden, die einen ungeeigneten Status für die Benutzung aufweisen. Das Listing 5.1 wurde bereits als Beispiel vorgestellt für die Überprüfung von fehlenden Mail-Adressen bei Benutzern des SOA

Governance Repositories und ist auch ein Standard-Constraint. Ein weiterer SPIN-Constraint überprüft, ob eine Serviceversion, die mindestens freigegeben ist, auch einen Endpunkt angegeben hat. Dieser SPIN-Constraint wird in Listing 5.8 beschrieben.

```
1  sgr:ServiceVersion
2        spin:constraint
3        [     a sp:Construct;
4              sp:text """
5                    PREFIX sgr: <http://sgr#>
6                    PREFIX sgrspin: <http://sgrspin#>
7                    PREFIX spin: <http://spinrdf.org/spin#>
8                    PREFIX sp: <http://spinrdf.org/sp#>
9                    CONSTRUCT {
10                         _:violation a spin:ConstraintViolation ;
11                         spin:violationRoot ?this ;
12                         spin:violationPath sgr:lifecycleState ;
13                         spin:violationValue ?lifecycleState ;
14                         spin:violationLevel spin:Warning ;
15                         rdfs:label "Missing endpoint for released ServiceVersion"
16                   }
17                   WHERE {
18                         ?this sgr:lifecycleState ?lifecycleState .
19                         ?lifecycleState sgr:state ("released" || "deprecated" ||
                               "retired").
20                         FILTER NOT EXISTS {?this sgr:endpoint ?endpoint} .
21                   }
22                   """
23       ] .
```

Listing 5.8: SPIN-Constraint zur Prüfung einer fehlenden Endpunktadresse

Die Regel nutzt in Zeile 20 einen Filter, um auf Nicht-Existenz eines Triples zu prüfen. Ist das Endpunkt-Triple nicht vorhanden, so wird in der CONSTRUCT-Klausel der SPIN-Constraint aktiv (und schreibt in diesem Fall die Regelverletzung in eine Log-Datei). Es sei angemerkt, dass der ODER-Operator in SPARQL valide ist, jedoch nicht korrekt vom SPIN-Reasoner in RDF4j interpretiert wird. Daher muss für jede ODER-Bedingung eine neue Regel geschrieben werden (was aus Übersichtsgründen in dieser Arbeit trotzdem mit dem ODER-Operator zusammengefasst ist). SPIN-Constraints sind in der Lage je nach *spin:violationLevel* eine Regelverletzung einfach nur in die Log-Datei zu schreiben oder zu erzwingen. Die Level auf der Höhe *spin:Warning* und darunter schreiben die Fehler lediglich in die Log-Datei. Alle Level darüber (*spin:Error* und höher) sind in der Lage, ganze Transaktionen mit der Datenbank abzubrechen, die ihre definierte Regel verletzen würden. Administratoren des SGR haben auf diese Weise die Möglichkeit, in Log-Dateien solche Inkonsistenzen im Datenmodell festzustellen und darauf zu reagieren.

SPIN-Regel für fehlende Endpunktadresse

Für die Regelverletzungen auf Level *spin:Warning* und darunter ist es von Vorteil, auch un-

abhängig vom Log zu wissen, an welchen Stellen eine Regel gebrochen wurde. Daher kann für einen solchen SPIN-Constraint eine zusätzliche SPIN-Regel definiert werden, die eine Art Markierung für die Regelverletzung mit Hilfe des Reasoners in der Datenbank inferiert. Das Listing 5.9 beschreibt eine passende Markierungsregel zum zuvor definierten SPIN-Constraint.

```
1  sgr:ServiceVersion
2        spin:rule
3        [     rdf:type sp:Construct;
4              sp:text """
5                    PREFIX sgr: <http://sgr#>
6                    PREFIX sgrspin: <http://sgrspin#>
7                    PREFIX spin: <http://spinrdf.org/spin#>
8                    PREFIX sp: <http://spinrdf.org/sp#>
9                    CONSTRUCT {
10                         ?this sgrspin:violationDetected "Missing endpoint for
                               released ServiceVersion"^^xsd:string .
11                   }
12                   WHERE {
13                         ?this sgr:lifecycleState ?lifecycleState .
14                         ?lifecycleState sgr:state ("released" || "deprecated" ||
                               "retired").
15                         FILTER NOT EXISTS {?this sgr:endpoint ?endpoint} .
16                   }
17                   """ ;
18       ] .
```

Listing 5.9: SPIN-Regel zur Markierung einer fehlenden Endpunktadresse

Die Regel nutzt in Zeile 15 einen *Filter*, um auf Nicht-Existenz des Endpunkt-Triples zu prüfen. Wenn das Endpunkt-Triple fehlt, so wird ein Markierungs-Triple von der SPIN-Regel an die jeweilige Serviceversion angefügt. Dies wird in Abbildung 5.14 grafisch dargestellt. Die *geschweiften Klammern* mit dem Prädikat *sgr:endpoint* und dem Symbol einer *leeren Menge* sollen die Nicht-Existenz eines Endpunkt-Triples an der Serviceversion (*SV*) darstellen.

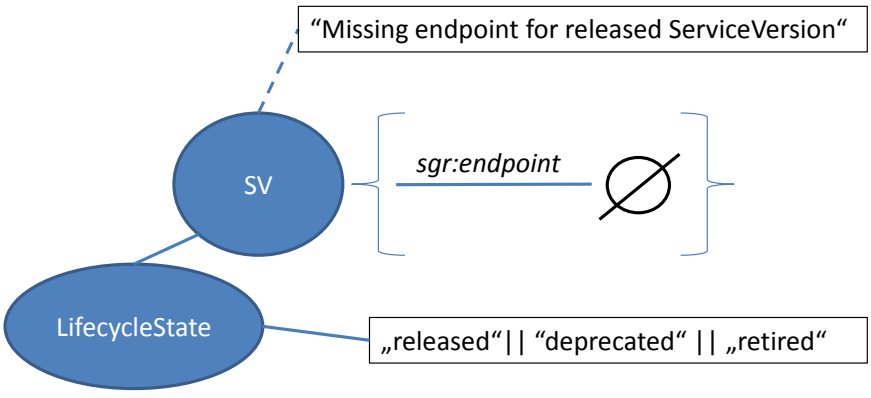

Abbildung 5.14.: Markierung für Serviceversion mit fehlender Endpunkt-Adresse

Mit Hilfe der zusätzlichen Regel können nun auch Entwickler solche Inkonsistenzen im Datenmodell identifizieren und mit entsprechender Programmlogik darauf reagieren. Beispielweise könnte man aufbauend auf der Regel eine Ansicht für Administratoren im SGR erstellen, die alle Inkonsistenzen auflistet. Damit müssten SGR-Administratoren nicht mehr den erschwerten Weg zum Lesen einer Log-Datei nutzen und könnten direkt solche Markierungen sehen.

Weitere Regeln

Die folgenden SPIN-Constraints und SPIN-Regeln wurden ebenfalls im Rahmen dieser Arbeit erstellt. Auch diese Regeln werden als initiale Regeln im SGR eingesetzt und sind in der folgenden Aufzählung mit den relevanten Inhalten der Regel-Funktionen abgekürzt (in Klammern wird immer der Regel-Typ beschrieben):

- Überprüfung des Service-Konsums für Serviceversionen, die den Status „deprecated“ oder „retired“ haben (als SPIN-Constraint mit Warning-Level und SPIN-Regel). Service-Konsumenten sollten keine veralteten Serviceversionen nutzen und können damit darauf hingewiesen werden (Anhang A.1).
- Überprüfung des Service-Konsums für Serviceversionen, die den Status „integrated“, „implemented“, „specified“, „announced“ oder „identified“ haben (als SPIN-Constraint mit Warning-Level und SPIN-Regel). Serviceversionen in sehr frühen Lebenszyklus-Phasen sind oft ungetestet und bringen auf diese Weise potenzielle, größere Fehler mit sich. Diese Regeln sollen die Service-Konsumenten auf diesen Sachverhalt hinweisen (Anhang A.2).
- Regel zum transitiven Herleiten von Hierarchiebeziehungen im hierarchischen Kategoriebaum einer Taxonomie für Serviceversionen (als SPIN-Regel). Diese Regel soll Entwicklern helfen, diese transitiven Beziehungen nicht im Programmcode herleiten zu müssen. Anfragen, wie „finde alle Kinder einer Kategorie“ in potenziell größeren Taxonomien sind so schon im Datenmodell durch die Regel beantwortet und müssen nicht erst im Programmcode umgesetzt werden (Anhang A.3).

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde eine SOA Governance Ontologie erstellt, erweitert und als prototypische Implementierung umgesetzt. Dazu wurde nach einer Betrachtung existierender Technologien im Bereich des Semantic Web zuerst das Datenmodell des SOA Governance Repositories als OWL-Modell formalisiert. Auf Basis des SOA-GovMM und des formalisierten OWL-Modells wurden dann Erweiterungen erstellt, die Services mit zusätzlicher Semantik beschreiben können: Schlüsselwörter für eine Kurzbeschreibung eines Service-Aspekts und Taxonomien zur Gruppierung von Services in Kategorien. Auf Basis der SOA Governance Ontologie wurden dann semantische Regeln formuliert, die die Einhaltung bestimmter Eigenschaften im Datenmodell erzwingen sollen und neue Informationen herleiten können, die für die Prozesse einer SOA Governance sinnvoll sind. Software-Tools die die SOA Governance Ontologie implementieren, können vollständig semantisch unterstützt werden und die Vorteile einer Ontologie nutzen. Diese Vorteile sind zum einen eine bessere Kontrolle für die Regeleinhaltung im Datenmodell. Sowohl syntaktische als auch semantische Regeln können auf der SOA Governance Ontologie definiert werden und deren Einhaltung mit Hilfe von Reasonern erzwungen werden. Zum anderen kann mit Hilfe von semantischen Regeln auch implizites Wissen aus dem Datensatz gewonnen werden, das für die Prozesse einer SOA Governance oder das Software-Tool im allgemeinen sinnvoll sein könnten. Neue Beziehungen und Objekte können durch die Evaluation semantischer Regeln von Reasonern inferiert werden. Schließlich können auch Performanceverbesserungen und Komplexitätsersparnisse aus der SOA Governance Ontologie resultieren, weil weniger Programmcode in einer Anwendung geschrieben werden muss, um bestimmte Regeln im Datenmodell einzuhalten. Des Weiteren wurde die SOA Governance Ontologie in Kapitel 4 erweitert durch: Schlüsselwörter zur semantischen Beschreibung von bestimmten Service-Merkmalen und Taxonomien zur Gruppierung von Serviceversionen in Kategorien. Eine semantische Suche und die Erkennung von ähnlichen Services sind die Vorteile dieser Erweiterung. Abschließend wurden im Kapitel 4 schon einige semantische Regeln eingeführt, die im Rahmen einer SOA Governance sinnvoll sind und problemlos auf der SOA Governance Ontologie anwendbar sind.

Dieses Konzept wurde als Erweiterung des SOA Governance Repositories implementiert. Das Basis-Datenmodell ist bereits eine mögliche Implementierung der SOA Governance Ontologie und musste nicht mehr angepasst werden; jedoch mussten die Erweiterungen (Schlüsselwörter und Taxonomie) implementiert werden. Neben der Datenmodellerweiterung des SOA Governance Repositories wurden auch auf der Benutzeroberfläche sichtbare Änderungen vorgenommen. Es gibt nun die Möglichkeit, beim Anlegen und Bearbeiten von Serviceversionen Schlüsselwörter anzugeben und die Serviceversionen in eine oder mehrere Kategorien

einzuordnen. In der detaillierten Ansicht für Serviceversionen können nun auch ähnliche Versionen angezeigt werden. Wenn zwei Serviceversionen ähnlich sind, haben sie mindestens eines oder mehrere gleiche Schlüsselwörter und haben eine Art semantische Ähnlichkeit, die im Rahmen dieser Arbeit mit der Anzahl an geteilten Schlüsselwörtern gewichtet werden kann. Eine semantische Suche wurde implementiert, die auf Basis von Servicenamen und Schlüsselwörtern Serviceversionen finden kann und mit Hilfe von Kategorien filtern kann. Im letzten Teil der Implementierung wurde das SOA Governance Repository nun auch mit SPIN-Regelunterstützung erweitert. SPIN-Regeln können nun in der Oberfläche erstellt und auf der RDF4j-Datenbank zur Laufzeit ausgeführt werden. Es gibt ebenfalls die Möglichkeit SPIN-Regeln beim Login zu importieren. Alle zusätzlichen Regeln des Konzepts wurden als SPIN-Regeln implementiert und können als Standard in der Datenbank für das SOA Governance Repository eingesetzt werden.

Ausblick

Die Technologien für eine semantische Unterstützung sind vorhanden, jedoch ist der Einstieg diese Technologien zu verwenden, erschwert, weil wenig einheitlich gelöst ist. Dies fängt schon bei der Wahl eines Vokabulars für eine eigene Ontologie an. Verzeichnisse wie „Linked Open Vocabularies“[1] erlauben das Auffinden von existierenden Vokabularen, jedoch ist es immer noch relativ aufwendig herauszufinden, ob ein Vokabular einer fremden Domäne wirklich zu der eigenen passt. Beispielsweise war es im Rahmen dieser Arbeit einfach, Vokabulare zu finden, die eine Art Kategorisierung beschreiben. Inhaltlich waren diese Kategorisierungen aber für Pflanzen- und Tierarten zugeschnitten, sodass diese nicht im Rahmen von Services und Serviceversionen verwendet werden konnten. Im Bereich der Vokabulare könnte sich die Auffindbarkeit noch verbessern und es müssten sich weitere Vokabular-Standards etablieren, die häufige Nutzdomänen (z.B. SOA) unterstützen.

Auch die Reasoner-Technologien sind von Anwendung zu Anwendung oft unterschiedlich. Es gibt Reasoner, die auf eine bestimmte Datenbanktechnologie zugeschnitten sind und Reasoner die auf eine bestimmte Sprache zugeschnitten sind. Die Reasoner auf Datenbankebene können nur zusammen mit der Datenbankimplementierung verwendet werden und sind nicht ohne weiteres auf andere Datenbanken übertragbar. Der Vorteil dieser Reasoner ist jedoch, dass sie performanter sind, weil es nicht nötig ist, zur Evaluierung von semantischen Regeln ganze Datensätze in den Arbeitsspeicher einer Anwendung zu laden. Reasoner, die eine Schnittstelle anbieten, um auf einer bestimmten Sprache zu inferieren, sind flexibler, weil sie in Datenbanken eingesetzt werden können, die Ontologiesprachen wie OWL unterstützen. Der Nachteil dieser Reasoner ist jedoch, dass sie weniger performant sind. Das Reasoning auf diese Weise muss mehrere Datensätze temporär in den Arbeitsspeicher laden, um komplexere semantische Regeln

[1] Linked Open Vocabularies: http://lov.okfn.org/dataset/lov

zu evaluieren. Es ist denkbar, dass sich die Reasoner-Technologien noch weiterentwickeln und Reasoner in Zukunft als flexible „Plugins“ für Datenbanken existieren könnten.

Diese Arbeit hat es dem SOA Governance Repository ermöglicht, semantisch unterstützt zu werden und hat den ersten Schritt durch die Definition erster, semantischer Regeln demonstriert. Der Datensatz kann an beliebigen Stellen semantisch annotiert werden und es können zusätzliche Regeln definiert werden, um bestimmte Regeln im Datensatz einzuhalten oder neues Wissen herzuleiten. Dafür kann sowohl der Datenbank-eigene Reasoner verwendet werden oder flexiblere Reasoner (wie OWL-Reasoner) angebunden werden. Folglich könnten in zukünftigen Arbeiten weitere semantische Regeln definiert werden, die bestimmte Semantiken im Datenmodell unterstützen sollen.

Es wurden einige SPIN-Constraints und SPIN-Regeln erstellt, um Inkonsistenzen und fehlende Informationen in der Datenbank des SOA Governance Repository zu identifizieren. Beispielsweise werden fehlende E-Mail-Adressen von Benutzern und fehlende Endpunkt-Adressen von freigegebenen (released) Serviceversionen erkannt. Auch der Service-Konsum von ungetesteten oder veralteten Serviceversionen wird in der Datenbank entsprechend mit SPIN-Constraint und SPIN-Regel behandelt. SPIN-Constraints schreiben diese Befunde in eine Log-Datei während SPIN-Regeln ein Markierungs-Triple an das Objekt anfügen, das diese Inkonsistenz aufweist. Für alle markierten Objekte in der Datenbank könnte eine Ansicht in der Benutzeroberfläche für Administratoren erstellt werden, um diese Inkonsistenzen aufzulisten und gegebenenfalls sogar von dort aus zu beheben (oder betroffene Benutzer darauf hinzuweisen).

Gerade im Bereich der Service-Ähnlichkeitsbewertung könnte noch ein Modell geschaffen werden, das eine Service-Ähnlichkeit besser bewertet und gewichtet, als mit der Anzahl gleicher Schlüsselwörter. Serviceversionen, die in der gleichen Kategorie sind, weisen schließlich auch eine Ähnlichkeit auf und ähneln sich umso mehr, wenn auch die Schnittstellenbeschreibung ähnlich sein sollte (ähnliches Datenmodell, ähnliche angebotene Operationen...). Wenn ähnliche Serviceversionen besser identifiziert werden können, so ist der Vorschlag ähnlicher Serviceversionen auch nützlicher für Service-Konsumenten, die Alternativen suchen.

Auch für die Einschätzung, wie abhängig ein Service ist, könnte ein Modell geschaffen werden. Im Moment gibt es durch die SOA Governance Ontologie die Möglichkeit, kritische Beziehungen (von Serviceversionen zu einem Vertrag/zu Service-Konsumenten) zu erkennen, jedoch ist das einfache Zählen dieser Beziehungen ein sehr triviales Modell zur Einschätzung, wie kritisch ein Service-Ausfall oder -Update wäre. Wenn man die Abhängigkeiten eines Services präzise bestimmen und gewichten könnte, so hätten Service-Anbieter eine genaue Vorstellung davon, wann sie ihren Service aktualisieren könnten und wann sie dies besser nicht tun sollten.

A. Anhänge

A.1. Regeln für Service-Konsum veralteter Serviceversionen

```
sgr:Contract
      spin:constraint
      [     a sp:Construct;
            sp:text """
                  PREFIX sgr: <http://sgr#>
                  PREFIX sgrspin: <http://sgrspin#>
                  PREFIX spin: <http://spinrdf.org/spin#>
                  PREFIX sp: <http://spinrdf.org/sp#>
                  CONSTRUCT {
                        _:violation a spin:ConstraintViolation ;
                        spin:violationRoot ?this ;
                        spin:violationPath sgr:serviceVersion ;
                        spin:violationValue ?sv ;
                        spin:violationLevel spin:Warning ;
                        rdfs:label "Usage of deprecated ServiceVersion! Choose a
                            more up-to-date one."
                  }
                  WHERE {
                        ?this sgr:serviceVersion ?sv .
                        ?sv sgr:lifecycleState ?lifecycleState .
                        FILTER EXISTS { ?lifecycleState sgr:state ("deprecated" ||
                            "retired")} .
                  }
                  """
      ] .
```

Listing A.1: SPIN-Constraint zur Ermittlung veralteter Serviceversionen

```
sgr:Contract
      spin:rule
      [     rdf:type sp:Construct;
            sp:text """
                  PREFIX sgr: <http://sgr#>
                  PREFIX sgrspin: <http://sgrspin#>
```

```
7                   PREFIX spin: <http://spinrdf.org/spin#>
8                   PREFIX sp: <http://spinrdf.org/sp#>
9                   CONSTRUCT {
10                        ?this sgrspin:violationDetected "Usage of deprecated
                              ServiceVersion! Choose a more up-to-date
                              one."^^xsd:string .
11                  }
12                  WHERE {
13                        ?this sgr:serviceVersion ?sv .
14                        ?sv sgr:lifecycleState ?lifecycleState .
15                        FILTER EXISTS { ?lifecycleState sgr:state ("deprecated" ||
                              "retired")} .
16                  }
17                  """ ;
18      ] .
```

Listing A.2: SPIN-Regel zur Markierung veralteter Serviceversionen

A.2. Regeln für Service-Konsum ungetesteter Serviceversionen

```
1  sgr:Contract
2       spin:constraint
3       [     a sp:Construct;
4             sp:text """
5                   PREFIX sgr: <http://sgr#>
6                   PREFIX sgrspin: <http://sgrspin#>
7                   PREFIX spin: <http://spinrdf.org/spin#>
8                   PREFIX sp: <http://spinrdf.org/sp#>
9                   CONSTRUCT {
10                        _:violation a spin:ConstraintViolation ;
11                        spin:violationRoot ?this ;
12                        spin:violationPath sgr:serviceVersion ;
13                        spin:violationValue ?sv ;
14                        spin:violationLevel spin:Warning ;
15                        rdfs:label "Usage of yet untested ServiceVersion! (below
                              release-state)"
16                  }
17                  WHERE {
18                        ?this sgr:serviceVersion ?sv .
19                        ?sv sgr:lifecycleState ?lifecycleState .
20                        FILTER EXISTS { ?lifecycleState sgr:state ("identified" ||
                              "announced" || "specified" || "implemented" ||
                              "integrated")} .
21                  }
22                  """
23      ] .
```

Listing A.3: SPIN-Constraint zur Ermittlung ungetesteter Serviceversionen

```
sgr:Contract
      spin:rule
      [     rdf:type sp:Construct;
            sp:text """
                  PREFIX sgr: <http://sgr#>
                  PREFIX sgrspin: <http://sgrspin#>
                  CONSTRUCT {
                        ?this sgrspin:violationDetected "Usage of yet untested
                            ServiceVersion! (below release-state)"^^xsd:string .
                  }
                  WHERE {
                        ?this sgr:serviceVersion ?sv .
                        ?sv sgr:lifecycleState ?lifecycleState .
                        FILTER EXISTS { ?lifecycleState sgr:state ("identified" ||
                            "announced" || "specified" || "implemented" ||
                            "integrated")} .
                  }
                  """ ;
      ] .
```

Listing A.4: SPIN-Constraint zur Markierung ungetesteter Serviceversionen

A.3. Regeln für transitive Beziehungen in Taxonomie

```
sgr:Category rdf:type rdfs:Class;
      spin:rule [
            rdf:type sp:Construct;
            rdfs:comment "Infer transitive Relation for downward
                sub-Categories"^^xsd:string ;
            sp:text """
                  PREFIX sgr: <http://sgr#>
                  PREFIX sgrspin: <http://sgrspin#>
                  CONSTRUCT {
                        ?this sgr:hasSubCategory ?categoryb .
                  }
                  WHERE {
                        ?this sgr:hasSubCategory ?categorya .
                        ?categorya sgr:hasSubCategory ?categoryb .
                  }
                  """ ;
      ] .
```

Listing A.5: SPIN-Regel für transitive Beziehungen in Taxonomie (abwärts)

```
1 # Infer transitive Relation for upward parent-Categories
2 sgr:Category rdf:type rdfs:Class;
3       spin:rule [
4             rdf:type sp:Construct;
5             rdfs:comment "Infer transitive Relation for upward
                  parent-Categories"^^xsd:string ;
6             sp:text """
7                   PREFIX sgr: <http://sgr#>
8                   PREFIX sgrspin: <http://sgrspin#>
9                   CONSTRUCT {
10                        ?this sgr:isSubCategoryOf ?categoryb .
11                  }
12                  WHERE {
13                        ?this sgr:isSubCategoryOf ?categorya .
14                        ?categorya sgr:isSubCategoryOf ?categoryb .
15                  }
16                  """ ;
17      ] .
```

Listing A.6: SPIN-Regel für transitive Beziehungen in Taxonomie (aufwärts)

Literaturverzeichnis

[Ahm+15] N. Ahmed, J. Bryant, G. Hasseler, M. Paulini. „Enabling semantic technologies in publish and subscribe middleware“. In: *2015 IEEE International Conference on Semantic Computing (ICSC)*. 2015, S. 338–343. DOI: 10.1109/ICOSC.2015.7050831 (zitiert auf S. 21).

[BBC12] BBC. *Creative Work Ontology*. 1.12.2012. URL: http://www.bbc.co.uk/ontologies/creativework (zitiert auf S. 47).

[Bil16] D. Bilgery. „Rollenbasiertes SOA-Governance Dashboard“. Bachelor Thesis. Universität Stuttgart, 2016 (zitiert auf S. 13, 46).

[Buz02] Buzzword.org.uk. *Biological Taxonomy Vocabulary*. 6.10.2008. URL: http://ontologi.es/biol/ns (zitiert auf S. 49).

[Buz03a] Buzzword.org.uk. *Biological Taxonomy Vocabulary (Botany)*. 18.03.2009. URL: http://ontologi.es/biol/botany (zitiert auf S. 49).

[Buz03b] Buzzword.org.uk. *Biological Taxonomy Vocabulary (Zoology)*. 18.03.2009. URL: http://ontologi.es/biol/zoology (zitiert auf S. 49).

[Col05] G. Collaboration. *SCOT Core Ontology*. 22.05.2013. URL: http://rdfs.org/scot/ns (zitiert auf S. 47).

[DUD17] DUDEN. *Taxonomie*. 2017. URL: http://www.duden.de/rechtschreibung/Taxonomie (zitiert auf S. 15).

[Goo10] GoodRelations. *GoodRelations Language Reference*. 1.10.2011. URL: http://www.heppnetz.de/ontologies/goodrelations/v1.html (zitiert auf S. 49, 64).

[HB07] M. Hepp, J. d. Bruijn. „GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies“. In: *The Semantic Web: Research and Applications*. Hrsg. von D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, E. Franconi, M. Kifer, W. May. Bd. 4519. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 129–144. ISBN: 978-3-540-72666-1. DOI: 10.1007/978-3-540-72667-8\textunderscore11 (zitiert auf S. 20).

[Hit08] P. Hitzler. *Semantic Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-33993-9. DOI: 10.1007/978-3-540-33994-6 (zitiert auf S. 7, 8).

[JBW11] N. Joachim, D. Beimborn, T. Weitzel. „What Are Important Governance and Management Mechanisms to Achieve IT Flexibility in Service-Oriented Architectures (SOA)?: An Empirical Exploration“. In: *2011 44th Hawaii International Conference on System Sciences (HICSS 2011)*. 2011, S. 1–10. DOI: 10.1109/HICSS.2011.489 (zitiert auf S. 8).

[Jep09] T. C. Jepsen. „Just What Is an Ontology, Anyway?“ In: *IT Professional* 11.5 (2009), S. 22–27. ISSN: 1520-9202. DOI: 10.1109/MITP.2009.105 (zitiert auf S. 13).

[KM16] J. Königsberger, B. Mitschang. „A Semantically-Enabled SOA Governance Repository (Application Paper)“. In: *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*. 2016, S. 423–432. DOI: 10.1109/IRI.2016.63 (zitiert auf S. 13, 26, 45).

[KSM14] J. Königsberger, S. Silcher, B. Mitschang. „SOA-GovMM: A meta model for a comprehensive SOA governance repository“. In: *2014 IEEE International Conference on Information Reuse and Integration (IRI)*. 2014, S. 187–194. DOI: 10.1109/IRI.2014.7051889 (zitiert auf S. 12, 26, 45).

[KYA16] A. Kamilaris, S. Yumusak, M. I. Ali. „WOTS2E: A search engine for a Semantic Web of Things“. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. 2016, S. 436–441. DOI: 10.1109/WF-IoT.2016.7845448 (zitiert auf S. 7).

[Kra16a] P. Kraus. „Dokumentation und automatische Generierung von Service-Beschreibungen“. Bachelor Thesis. Universität Stuttgart, 2016 (zitiert auf S. 13, 46).

[Kra16b] D. Krauss. „Generierung und Optimierung von Testzeitplänen im Rahmen des SOA Change Managements“. Master Thesis. Universität Stuttgart, 2016 (zitiert auf S. 13, 46).

[LDB09] P. d. Leusse, T. Dimitrakos, D. Brossard. „A Governance Model for SOA“. In: *2009 IEEE International Conference on Web Services (ICWS)*. 2009, S. 1020–1027. DOI: 10.1109/ICWS.2009.132 (zitiert auf S. 12).

[LX10] Y. Li, Q. Xiong. „Enterprise Application Rebuilding Framework Based on Semantic SOA and Workflow“. In: *2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*. 2010, S. 424–426. DOI: 10.1109/DCABES.2010.93 (zitiert auf S. 18).

[Loh11] S. Lohmann. *Modular Unified Tagging Ontology (MUTO)*. 16.11.2011. URL: http://purl.org/muto/core (zitiert auf S. 48).

[ML16] V. Mala, D. K. Lobiyal. „Semantic and keyword based web techniques in information retrieval“. In: *2016 International Conference on Computing, Communication and Automation (ICCCA)*. 2016, S. 23–26. DOI: 10.1109/CCAA.2016.7813724 (zitiert auf S. 7).

[Mar08] E. A. Marks. *Service-oriented architecture governance for the services driven enterprise*. Hoboken: Wiley, 2008. ISBN: 978-0-470-17125-7 (zitiert auf S. 23).

[Miy15] V. Miyai. „RDF-Based Data Model for a SOA Governance Repository“. Master Thesis. University of Stuttgart, University of Crete, Tilbug University, 2015 (zitiert auf S. 8, 13, 45).

[NM01] N. Noy, D. McGuinnes. *Ontology Development 101: A Guide to Creating Your First Ontology*. 2001. URL: http://protege.stanford.edu/publications/ontology\textunderscoredevelopment/ontology101.pdf (zitiert auf S. 25).

[SBB13] S. Slimani, S. Baina, K. Baina. „Toward a semantic SOA implementation strategy“. In: *2013 3rd International Symposium ISKO-Maghreb*. 2013, S. 1–4. DOI: 10.1109/ISKO-Maghreb.2013.6728157 (zitiert auf S. 17, 18).

[SM01] U. Sattler, N. Matentzuglu. *List of Reasoners*. 26.01.2016. URL: http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/ (zitiert auf S. 16).

[SNK09] S. Seedorf, K. Nordheimer, S. Krug. „STraS: A framework for semantic traceability in enterprise-wide SOA life-cycle management“. In: *2009 13th Enterprise Distributed Object Computing Conference Workshops, EDOCW*. 2009, S. 212–219. DOI: 10.1109/EDOCW.2009.5331994 (zitiert auf S. 19).

[VDR16] J. M. Vallejo, J. G. R. Diaz, J. C. O. Rojas. „Construction of Semantic Web search engine from a specific context“. In: *2016 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*. 2016, S. 1–6. DOI: 10.1109/ROPEC.2016.7830640 (zitiert auf S. 7).

[W3C02] W3C. *RDF 1.1 Concepts and Abstract Syntax*. 25.02.2014. URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/Overview.html (zitiert auf S. 13).

[W3C03] W3C. *SPARQL 1.1 Overview*. 21.03.2013. URL: https://www.w3.org/TR/sparql11-overview/ (zitiert auf S. 14).

[W3C05] W3C. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. 21.05.2004. URL: https://www.w3.org/Submission/SWRL/ (zitiert auf S. 15).

[W3C09] W3C. *SPIN - SPARQL Syntax*. 12.09.2013. URL: http://spinrdf.org/sp.html (zitiert auf S. 15).

[W3C11] W3C. *OWL-S: Semantic Markup for Web Services*. 22.11.2004. URL: https://www.w3.org/Submission/OWL-S/ (zitiert auf S. 14).

[W3C12] W3C. *OWL 2 Web Ontology Language: Document Overview (Second Edition)*. 11.12.2012. URL: https://www.w3.org/TR/2012/REC-owl2-overview-20121211/ (zitiert auf S. 14).

[XLPf08] S. Xiao, Z. Lin, Y. Peng-fei. „Semantic SOA-Based Enterprise Information System Integration Technology“. In: *2008 International Conference on Smart Manufacturing application (ICSMA)*. 2008, S. 534–537. DOI: 10.1109/ICSMA.2008.4505582 (zitiert auf S. 18).

[ZS09] S. Zhou, Z. Sun. „Using WSMO to Enable Mediation of Heterogeneous Services and Semi-automation of Service Discovery and Execution in Semantic SOA“. In: *2009 International Conference on Web Information Systems and Mining (WISM)*. 2009, S. 638–642. DOI: 10.1109/WISM.2009.134 (zitiert auf S. 18).

[Zaf+16] R. Zafar, E. Yafi, M. F. Zuhairi, H. Dao. „Big Data: The NoSQL and RDBMS review“. In: *2016 International Conference on Information and Communication Technology (ICICTM)*. 2016, S. 120–126. DOI: 10.1109/ICICTM.2016.7890788 (zitiert auf S. 8).

Alle URLs wurden zuletzt am 30. 05. 2017 geprüft.

Abbildungsverzeichnis

Verzeichnis der Listings

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