Universität Stuttgart

Aspekte des Change Management in großen koordinierten Systemverbünden

Von der Graduate School of Excellence advanced Manufacturing Engineering der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von

Jan Königsberger

aus Stuttgart

Hauptberichter: Prof. Dr.-Ing. habil. Bernhard Mitschang

Mitberichter: Prof. Dr.-Ing. Norbert Ritter

Tag der mündlichen Prüfung: 20. August 2019

Institut für Parallele und Verteilte Systeme (IPVS)

Abteilung Anwendersoftware

der Universität Stuttgart

2019

Vorwort

Diese Arbeit entstand an der Graduate School of Excellence advanced Manufacturing Engineering (GSaME) in Zusammenarbeit mit der Abteilung Anwendersoftware am Institut für Parallele und Verteilte Systeme (IPVS) sowie in Kooperation mit der Daimler AG. Mein Dank gilt vielen Menschen, mit denen ich während meiner Promotionszeit zusammenarbeiten durfte.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr.-Ing. habil. Bernhard Mitschang, der mir viele Freiheiten in der Gestaltung meines Themas ließ, aber jederzeit mit konstruktiver Unterstützung und für Diskussionen bereitstand.

Prof. Dr.-Ing. Norbert Ritter möchte ich für seine Übernahme des Mitberichts meiner Arbeit danken.

In meiner Zeit am IPVS hatte ich die Gelegenheit mit vielen Kollegen zusammenzuarbeiten. Besonders bedanken möchte ich mich bei den Kollegen, mit denen ich im Rahmen von Publikationen oder auch in der Betreuung von Studenten eng zusammenarbeiten durfte: Stefan Silcher, Peter Reimann, Laura Kassner, Christian Weber, Eva Hoos, Mathias Mormul, Pascal Hirmer, Frank Steimle, Christoph Stach, Holger Schwarz und Matthias Wieland.

Auch auf Seiten des Kooperationspartners gab es Personen, die mich in meiner Arbeit unterstützt und mir einen tiefen Einblick in die Industrie ermöglicht haben. Mein Dank gilt hier Martin Schaaf, Henrik Berner und Frank Stahlhut.

Ein weiterer Dank gilt den Kollegen der GSaME, mit denen ich im Rahmen von interdisziplinären Projekten, aber auch im Verein der Freunde der GSaME zusammenarbeiten durfte: Tobias Helbig, Tobias Tauterat, Dominik Brenner, Johannes Nickel und Dominik Herr. Danke, dass ihr dazu beigetragen habt, die GSaME zu einem einzigartigen Promotionsumfeld zu machen!

Auch der Administration des IPVS und der GSaME gilt mein Dank für die Unterstützung in technischen und organisatorischen Fragen.

Ebenfalls bedanken möchte ich mich bei einigen Studenten, die ich im Rahmen von Abschlussarbeiten und Studienprojekten betreuen durfte und in denen Prototypen für die Konzepte und Methoden dieser Arbeit entstanden sind: Philipp Kraus, Dominik Bilgery, Alexander Blehm und David Krauss.

Zuletzt möchte ich meiner Familie danken, die mich in all den Jahren immer unterstützt hat und mir Verständnis und (meist) Geduld entgegengebracht hat: Meinen Eltern Manfred und Anne, meinem Bruder Lennart und ganz besonders meiner Frau Lena. Mein Dank gilt auch meinem Sohn Joscha, der die letzte Phase dieser Arbeit zwar nicht unbedingt erleichtert, aber ungemein bereichert hat.

Ehningen, im August 2019 Jan Königsberger

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

BO$^+$ Business Object *plus*

BPEL Business Process Execution Language

CPS Cyber-physisches System

CSW Corporate Semantic Web

EAI Enterprise Application Integration

EAM Enterprise Architecture Management

FOAF Friend-of-a-Friend

SOA-GovMM SOA Governance Meta Model

HATEOAS Hypermedia as the Engine of Application State

HTTP Hypertext Transfer Protocol

IT Informationstechnologie

JSON JavaScript Object Notation

MVVM Model View ViewModel

OWL Web Ontology Language

OWL-S Web Ontology Language for Web Services

R2SMA REST-to-SOAP Middleware Architecture

RAML RESTful API Modeling Language

RDF Ressource Description Framework

RDFS Resource Description Framework Schema

REST Representational State Transfer

SAML Security Assertion Markup Language

SemVer Semantic Versioning

SGR SOA Governance Repository

SITAM Stuttgart IT Architecture for Manufacturing

SOA serviceorientierte Architektur

SOAP SOAP, ursprünglich für Simple Object Access Protocol

SPARQL SPARQL Protocol and RDF Query Language

SPIN SPARQL Inferencing Notation

UML Unified Modeling Language

URI Uniform Resource Identifier

WADL Web Application Description Language

WS-* *Eine Menge von Spezifikationen im Kontext SOAP-basierter Webservices*

WSDL Web Service Description Language

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

Kurzfassung

Diese Arbeit untersucht verschiedene Aspekte von Änderungsvorhaben im Rahmen großer Systemverbünde in serviceorientierten Architekturen (SOA). Entsprechende Änderungsaktivitäten und -prozesse werden unter dem Begriff *Change Management* zusammengefasst und sind ein Teilbereich der SOA Governance. Die SOA Governance definiert Prozesse und Richtlinien zur Steuerung und Überwachung einer SOA. Änderungsprozesse müssen für jede Änderung an einem Bestandteil eines SOA-Systemverbundes, wie etwa eines Services oder seiner Schnittstellen, durchlaufen werden. Daher ist es essentiell, dass solche Prozesse klar dokumentiert und möglichst schlank gehalten werden. Aufgrund der Komplexität eines Systemverbundes ist die Unterstützung der Änderungsprozesse durch spezialisierte Softwareanwendungen unabdingbar zur effizienten Durchführung von Änderungsvorhaben. Das übergeordnete Ziel dieser Arbeit ist daher die Entwicklung von Methoden und Verfahren zur Unterstützung der Änderungsprozesse und der Governance serviceorientierter Architekturen.

Zur Erreichung dieses Ziel liefert die vorliegende Arbeit mehrere Beiträge. Es werden zunächst Möglichkeiten zur Vereinfachung der Integration von neuen Service-Consumern in eine SOA vorgestellt. Hierzu wurde das Konzept der Business Objects *plus* entwickelt. Dieses zielt auf eine Vereinheitlichung von häufig genutzten Datenobjekten über Domänengrenzen hinweg

ab, wodurch die Anbindung von Consumern vereinfacht wird. Einen weiteren Beitrag aus diesem Themenfeld stellt die *REST-to-SOAP-Middleware Architecture* dar. Sie ermöglicht die Anbindung von existierenden, klassischen SOAP-basierten Webservices in Anwendungsfällen, die das leichtgewichtigere *Representational State Transfer*-Architekturparadigma nutzen. Durch den Einsatz moderner Technologien können sich neue Möglichkeiten bei der Entwicklung von Softwaresystemen eröffnen. Konkret untersucht diese Arbeit dazu die Einsatzmöglichkeiten semantischer Technologien in der Entwicklung eines SOA-Governance-Informationssystems, das Stakeholdern einer SOA eine effiziente Erledigung ihrer Aufgaben ermöglichen soll. Ein weiterer wichtiger Themenkomplex ist die Durchführung von Software- und Schnittstellentests im Rahmen eines Änderungsprozesses. Insbesondere in einem Systemverbund sind dabei eine Vielzahl an Abhängigkeiten zwischen Systemen und Services zu beachten. Diese Arbeit liefert dazu eine Methode zur Risikobewertung von Änderungen, wodurch eine zielgerichtete und ressourcenschonende Testdurchführung ermöglicht wird. Zur Unterstützung der Testplanung und -durchführung wurde in einem weiteren Beitrag ein Konzept zur automatischen Generierung und Optimierung von Testzeitplänen entwickelt, welches existierende Abhängigkeiten und Randbedingungen mit einbezieht, durch die eine manuelle Erstellung eines solchen Zeitplans komplex wäre. Die in dieser Arbeit entwickelten Methoden und Konzepte wurden als Prototyp eines SOA-Governance-Informationssystems, dem *SOA Governance Repository* implementiert, das ebenfalls vorgestellt wird.

Abstract

This thesis examines different aspects of change projects within the framework of large system compounds in service-oriented architectures (SOA). Corresponding change activities and processes are summarized under the term *Change Management* and are a part of SOA governance. SOA governance defines processes and guidelines for controlling and monitoring a SOA. Change processes need to be performed for every change to a component within a SOA system compound, such as a service or its interfaces. It is therefore essential that such processes are clearly documented and kept as lean as possible. Due to the complexity of a system compound, the support of change processes by specialized software applications is indispensable for the efficient implementation of change projects. The overall goal of this work is therefore the development of methods and procedures to support change processes and the governance of service-oriented architectures.

To achieve this goal, this thesis provides several contributions. One research area studies possibilities for the simplification of the integration of new service consumers into a SOA. The concept of Business Objects *plus* addresses this topic by aiming at a standardization of frequently used data objects across domain boundaries. This allows for an easier integration of new consumers into the system compound. Another contribution in this area is the *REST-to-SOAP-Middleware Architecture*. It enables the integration

of existing, SOAP-based web services in use cases that build on the lighter *Representational State Transfer* architecture paradigm. The use of modern technologies can open up new possibilities in the development of software systems. Specifically, this thesis examines the possible applications of semantic technologies in the development of a SOA governance information system, which will enable stakeholders of a SOA to perform their tasks more efficiently. Another important topic are software and interface tests within the scope of a change process. Especially in a system compound, a large number of dependencies between systems and services must be taken into account when testing. This work provides a method to perform a risk assessment of changes, which enables a targeted and resource-saving test execution. To support the test planning and execution, a concept for the automatic generation and optimization of test schedules was developed in a further contribution. The concept considers existing dependencies and constraints that render the manual creation of such a schedule extremely complex. The methods and concepts developed in this thesis were implemented as a prototype of a SOA governance information system, the *SOA Governance Repository*, which is also presented.

Kapitel 1

Einleitung

Die vorliegende Arbeit befasst sich mit verschiedenen Aspekten des Change Management im Rahmen der Governance serviceorientierter Architekturen. Das Change Management (deutsch: Änderungsmanagement) umfasst in diesem Kontext die Tätigkeiten und Prozesse zur Verwaltung und Durchführung von Änderungen an den Bestandteilen einer serviceorientierten Architektur, insbesondere also an den Services, Schnittstellenbeschreibungen und den beteiligten Stakeholdern.

In diesem Kapitel wird dazu in Abschnitt 1.1 die Motivation und die in dieser Arbeit adressierte Forschungslücke genauer vorgestellt. Abschnitt 1.2 ordnet die Arbeit in den übergeordneten Forschungskontext des Advanced Manufacturing Engineering ein. In Abschnitt 1.3 werden die dieser Arbeit zugrundeliegenden Forschungsfragen zusammengefasst und die zugehörigen Forschungsbeiträge vorgestellt. Abschließend liefert Abschnitt 1.4 einen Überblick über den weiteren Aufbau dieser Arbeit.

1.1 Motivation und Forschungslücke

In den vergangenen Jahren kamen durch die Globalisierung und die dadurch zusammenwachsenden Märkte neue Herausforderungen auf Unternehmen zu. Durch steigende Anforderungen der Kunden und dem Wunsch nach ständig neuen Produkten haben sich die Innovationszyklen wie auch Produktlebenszyklen drastisch verkürzt. So ist es heute nicht mehr ausreichend alle fünf Jahre ein neues Produkt auf den Markt zu bringen, Kunden erwarten inzwischen in deutlich kürzeren Zeitabständen ein neues, innovatives und technisch verbessertes Produkt. Diese Entwicklung ist insbesondere bei Smartphones und anderer Unterhaltungselektronik zu sehen, aber auch in der Automobilindustrie sind ähnliche Entwicklungen zu beobachten. Automobilhersteller hatten in der Vergangenheit Innovationszyklen von circa sieben Jahren, inzwischen werden neue oder zumindest überarbeitete Modelle in der Regel alle zwei Jahre präsentiert. Zusätzlich erwarten Kunden inzwischen auch immer individuellere und auf ihre speziellen Bedürfnisse zugeschnittene Produkte. Die dadurch steigende Variantenvielfalt in der Produktion kann bisher von vielen Unternehmen nur schwer oder gar nicht umgesetzt werden [WSCL13]. Durch diese Erwartungen ergeben sich für alle Beteiligten neue Herausforderungen.

Unternehmen müssen nun noch enger mit Zulieferern zusammenarbeiten. Gut erkennbar ist dies beispielsweise bei neuen Logistik-Konzepten, die auf Just-in-time- oder Just-in-sequence-Anlieferung [Ver08] von Bauteilen setzen. Bei beiden ist es unabdingbar, dass die gesamte Prozess- und Lieferkette exakt durchgeplant ist. Ebenfalls ist es wichtig, dass die Prozesse schnell und kurzfristig auf Änderungen reagieren können. Dies ist nicht möglich ohne eine vollständige Integration der beteiligten IT-Systeme [WSCL13]. Auch neue Konzepte wie Industrie 4.0 [Bun17], das Industrielle Internet [JBSR17] und Cyber-physische Systeme [JBSR17] haben einerseits das Ziel Unternehmen diese Wandlungsfähigkeit zu ermöglichen, andererseits stellen sie diese vor neue Herausforderungen bezüglich der IT-Systemintegration. So ist es auch hierfür notwendig die Produktion zu digitalisieren und IT-Systeme domänenübergreifend zu integrieren. Besonders in vielen kleinen

und mittelständischen Unternehmen, aber auch in großen Konzernen, gibt es hierbei noch Nachholbedarf [GGH+13].

Produktionsunternehmen sind also darauf angewiesen, die ihnen zur Verfügung stehenden Ressourcen optimal zu nutzen. Dies umfasst einerseits die optimale Auslastung der Produktionsmaschinen mit möglichst niedrigen Rüst- und Wartungszeiten. Erreicht werden kann dies insbesondere durch eine flexible Produktionsplanung und -steuerung, wofür wiederum eine umfangreiche Integration aller beteiligten IT-Systeme notwendig ist [Sil14]. Andererseits betrifft es die Mitarbeiter, die Ihre Arbeitszeit möglichst effizient nutzen sollen. Dazu sind schlanke Prozesse und eine optimale Unterstützung durch Software- und IT-Systeme notwendig. Auch hier haben sich in den letzten Jahren die Anforderungen geändert. Mitarbeitern ist heute eine gewisse Flexibilität in ihrer Arbeitszeitgestaltung deutlich wichtiger als noch vor einigen Jahren. Bei Entwicklern und anderen klassischen Sachbearbeitertätigkeiten äußert sich das beispielsweise durch eine deutliche Zunahme von Telearbeit, sei es von zuhause aus oder vor Ort bei Kunden [Bit17]. Dies erfordert einen kontinuierlichen Zugang zum Unternehmensnetzwerk und bringt neue Herausforderungen hinsichtlich Datenschutz und IT-Sicherheit mit. Insbesondere Führungskräfte müssen heute oft örtlich flexibel sein und benötigen daher auch von mobilen Endgeräten aus Zugriff auf Unternehmensdaten [SGK+15; NB16].

Alle genannten Entwicklungen und Herausforderungen haben gemeinsam, dass sie nicht realisierbar sind, ohne eine umfangreiche Integration *aller* IT-Systeme eines Unternehmens, übergreifend über Arbeitsbereiche und Geschäftsdomänen hinweg.

In den letzten Jahren haben Unternehmen zur Lösung unterschiedlichster Integrationsprobleme vermehrt auf serviceorientierte Architekturen (SOA) gesetzt. Sie versprachen sich durch den Einsatz dieses Architekturparadigmas Verbesserungen in der Agilität ihrer IT-Anwendungen, Kosteneinsparungen und eine bessere Abstimmung zwischen IT- und Geschäftsprozessen [Bie06; Erl08]. Häufig jedoch wurden diese Ziele nicht erreicht, da das notwendige Umdenken nicht stattfand und die SOA-Umstellung als ausschließlich technisches Projekt angesehen wurde [Mee08; Swa12]. Bei der Umstellung

auf eine SOA ist es allerdings notwendig, auch personelle und organisatorische Änderungen durchzuführen. Diese beginnen bei der Einführung neuer Verwaltungsgremien (englisch: *Boards*) und enden beim Umdenken der Mitarbeiter – weg von traditionellem, technisch aus Applikationssicht getriebenem Denken, hin zu einem Denken in applikationsübergreifenden serviceorientierten Prozessen, die eingegliedert in ein Großes Ganzes sind [PSVS07]. Der Hauptgrund für das Scheitern dieser SOA-Initiativen war häufig das Fehlen strukturierter Governance-Mechanismen [Mee08; Gar07]. Das Ziel einer solchen SOA Governance ist die Definition von Prozessen und Richtlinien zur Steuerung und Überwachung einer SOA. Das Hauptaugenmerk dieser Arbeit liegt auf dem Teilbereich Change Management, der sich primär mit Änderungen an bestehenden Bestandteilen einer SOA beschäftigt. Dabei handelt es sich insbesondere um Services und ihre Schnittstellen, sowie Personen, die ein Interesse an der erfolgreichen Durchführung von Änderungen haben (englisch: Stakeholder). Trotz der Einführung von Governance-Strukturen haben Unternehmen in der tagtäglichen Arbeit an einer SOA weiterhin häufig mit Problemen zu kämpfen. Diese Probleme treten vermehrt auch im Kontext von Änderungen auf. Änderungsaktivitäten müssen entsprechend häufig durchgeführt werden, beispielsweise dann, wenn Serviceschnittstellen angepasst werden, um neue Funktionalität einzuführen. Solche Änderungen erfordern die Beteiligung vieler Stakeholder, sowohl auf Seiten des Service-Providers als auch auf der Seite des Service-Consumers. Ein häufiges Problem ist, dass im Unternehmen keine definierten Änderungsprozesse existieren oder es sind keine Informationen dazu verfügbar welche Systeme und Stakeholder von den Änderungen betroffen sind. Die Verfügbarkeit dieser Informationen ist ebenso notwendig um effiziente Tests von Services zu ermöglichen.

Teil des Change Management ist auch die Anbindung neuer Service-Consumer in einen SOA-Systemverbund. Dabei handelt es sich um einen Zusammenschluss von Softwaresystemen, die im Rahmen einer SOA Services anbieten und zwischen denen es Nutzungsabhängigkeiten gibt. Diese Anbindung ist auch wichtig für neue, häufig mobile, Anwendungsfälle wie beispielsweise die Nutzung von Tablets oder Smartphones in der Fertigung.

Solche Anwendungsfälle werden von derzeitigen Unternehmens-SOA aufgrund der eingesetzten Technologien häufig nicht unterstützt. Hier ist ein generischer Ansatz notwendig, der eine Anbindung an Bestandsservices ermöglicht.

Ebenfalls werden Probleme sichtbar, wenn Unternehmen mit Zulieferern und Partnerunternehmen zusammenarbeiten und Daten untereinander austauschen müssen. Dazu ist es notwendig eine Integration der beteiligten IT-Systeme zu schaffen. Meist nutzt allerdings jedes System eigene Schnittstellen und Datenmodelle, so dass für jedes System-Paar eine eigene Transformation zwischen den Schnittstellen geschaffen werden muss. Die eigentlich benötigte flexible Integration von IT-Systemen wird dadurch behindert.

Die genannten Probleme sind weder von wissenschaftlicher Seite noch in der Praxis vollständig gelöst. Insbesondere vor dem Hintergrund aktueller Entwicklungen, wie der durch Industrie 4.0 geförderten Digitalisierung der Produktion, ist es wichtig, diese Probleme zu adressieren [GGH+13]. Damit können IT-Umgebungen etabliert werden, welche die notwendige Flexibilität und Agilität besitzt, um Unternehmen über den gesamten Produktlebenszyklus hinweg eine moderne und zukunftsorientierte Gestaltung ihrer Prozesse zu ermöglichen.

1.2 Forschungskontext

Diese Arbeit bewegt sich einerseits im Forschungsumfeld der Graduate School of Excellence advanced Manufacturing Engineering (GSaME)[1], andererseits im Kontext der Engineering-Domäne, also der Produktentwicklung großer Automobilkonzerne.

Im Rahmen der GSaME werden interdisziplinäre Forschungsprojekte im Bereich einer wandlungsfähigen Produktion durchgeführt. Dabei wurden und werden Forschungsprojekte durchgeführt, die Berührungspunkte mit den Inhalten dieser Arbeit haben. Insbesondere betrifft dies Vorarbeiten von Minguez [Min12] und Silcher [Sil14], die sich ebenfalls mit Aspekten

[1] http://www.gsame.uni-stuttgart.de

serviceorientierter Architekturen im Produktionsumfeld auseinandersetzen. Ebenso existieren vorhergehende und aktuelle Arbeiten im Bereich Daten- und Textanalyse [Kas17], Datenqualität, Wissensmanagement und Engineering Apps. Einen zusammenhängenden Blick auf diese Einzelthemen bietet die *Stuttgart IT Architecture for Manufacturing* (SITAM, [KGK+17]), die in Abschnitt 2.3 kurz vorgestellt wird.

Die in dieser Arbeit vorgestellten Konzepte fokussieren sich speziell auf die Engineering-Domäne in der Automobilindustrie und sind praxisnah motiviert. Aspekte zur Praxisrelevanz wurden mit einem süddeutschen Automobilhersteller diskutiert und auch evaluiert. Dementsprechend finden sich in dieser Arbeit teilweise Fallbeispiele und Herausforderungen, die sich insbesondere auf den Produktentwicklungsbereich von Automobilherstellern beziehen.

1.3 Ziele und Forschungsfragen

Ausgehend von der in Abschnitt 1.1 umrissenen Forschungslücke, werden in diesem Abschnitt die Forschungsziele dieser Arbeit vorgestellt. Im Anschluss werden diese weiter aufgegliedert in einzelne Forschungsbeiträge.

1.3.1 Forschungsziele

Das übergeordnete Ziel dieser Arbeit ist die Entwicklung von Methoden und Verfahren zur Unterstützung der Änderungsprozesse und der Governance serviceorientierter Architekturen. Aus der in Abschnitt 1.1 diskutierten Forschungslücke wurden die folgenden drei Forschungsziele abgeleitet:

Forschungsziel Z_1: Vereinfachung der Integration von neuen Consumern in eine SOA Die flexible Anpassung einer Unternehmens-IT erfordert es, dass neue Service-Consumer schnell an existierende Services angebunden werden. Wie in Abschnitt 1.1 beschrieben, sind aktuelle IT-Umgebungen dieser Aufgabe nicht gewachsen, was zu langen Wartezeiten und Frust bei neuen Service-Consumern führt. Zur Adressierung dieses Forschungsziels wird daher untersucht, inwiefern sich diese Prozesse beschleunigen lassen.

Forschungsziel Z_2: Einsatz von semantischen Technologien zur Unterstützung der SOA Governance Um eine SOA mittels Governance-Prozessen effizient verwalten und unterstützen zu können, ist es erforderlich, dass eine Vielzahl von Informationen dokumentiert wird. Gleichzeitig ist es notwendig die Dokumentationsprozesse möglichst schlank zu halten, um Anwender nicht abzuschrecken. Eine Möglichkeit Daten unternehmensweit zu vernetzen ist der Einsatz von semantischen Technologien, wie beispielsweise Ontologien. In diesem Themenfeld soll daher untersucht werden, wie sich semantische Technologien im Unternehmen so einsetzen lassen, dass eine effizientere Durchführung der SOA-Governance-Prozesse möglich ist.

Forschungsziel Z_3: Unterstützung des Testens von Systemverbünden Durch die Abhängigkeiten zwischen einzelnen Services kann es durch nicht ausreichend getestete Änderungen zu größeren Ausfällen kommen, die im schlimmsten Fall Auswirkungen auf die Produktion haben können. Dementsprechend ist es wichtig, neue Serviceversionen gründlich, das heißt unter Einsatz von entsprechenden Zeit- und Personalressourcen zu testen, bevor sie produktiv eingesetzt werden. Andererseits ist es ebenfalls wichtig, diese Tests möglichst zielgerichtet durchzuführen, um nicht unnötig Ressourcen aufzuwenden. In diesem Themenbereich werden daher Möglichkeiten untersucht, Tests entsprechend dieser Rahmenbedingungen durchführen zu können.

1.3.2 Forschungsbeiträge

Im Rahmen dieser Arbeit wurden mehrere Forschungsbeiträge erbracht. Ein Großteil dieser Beiträge wurde bereits in verschiedenen Konferenzpublikationen veröffentlicht [KSM14; KM16; KM17; KM18]. Im folgenden soll eine Übersicht gegeben werden, wie die einzelnen Beiträge den vorgestellten Forschungszielen zuzuordnen sind. Abbildung 1.1 zeigt diese Zuordnung.

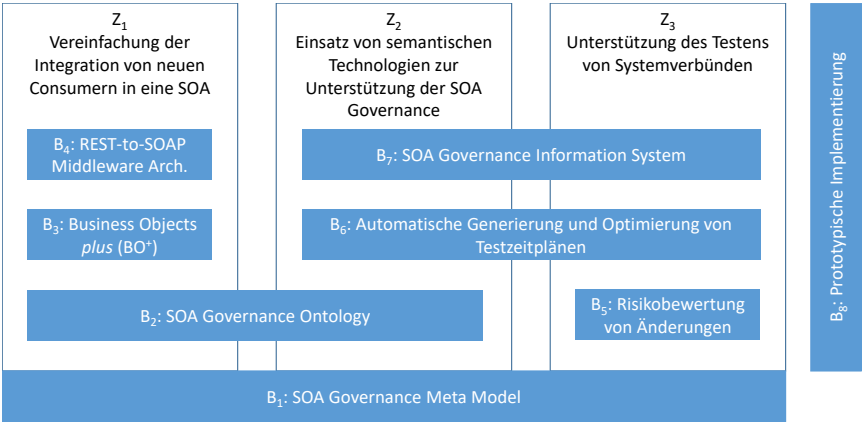


Abbildung 1.1: Forschungsziele und -beiträge

B_1: **SOA Governance Meta Model:** Das SOA Governance Meta Model (SOA-GovMM, [KSM14]) dient als Basis der weiteren Beiträge und bietet eine Strukturierung der Elemente, die zur effektiven Governance einer SOA benötigt werden.

B_2: **SOA Governance Ontology:** Die SOA-Governance-Ontologie [KM16] baut auf dem SOA-GovMM auf und bietet eine semantische Beschreibung der im Rahmen der SOA-Governance verwalteten Elemente, Personen und Beziehungen. Zusätzlich dient sie als Grundlage für die weitere Untersuchung der Einsatzmöglichkeiten semantischer Technologien im Rahmen des Forschungsziels Z_2.

B_3: **Business Objects *plus*:** Das Konzept der Business Objects *plus* (BO^+) [KM17] unterstützt Forschungsziel Z_1, indem es häufig genutzte Datenobjekte vereinheitlicht und an die Unternehmenshierarchie anpasst. Gleichzeitig wird durch das Konzept eine einheitliche Grundlage für Serviceschnittstellen geschaffen.

B_4: **REST-to-SOAP Middleware Architecture**: Einen Beitrag zu Z_1 liefert die REST-to-SOAP Middleware Architecture [KM16; KM18]. Ziel ist es, automatisiert REST-API-Proxys zu existierenden SOAP-Webservices

zu erzeugen, die eine schnelle und einfache Anbindung von REST-basierten Anwendungen in eine SOA erlauben.

B_5: **Risikobewertung von Änderungen**: Um zielgerichtete Testvorgänge zu ermöglichen, ist es notwendig einschätzen zu können, welche Risiken mit einer Änderung verbunden sind und wie sie sich auf einen SOA-Systemverbund auswirken können. Dieser Forschungsbeitrag zu Z_3 beschäftigt sich daher mit der Risiko- und Kritikalitätsbewertung von Änderungsvorhaben.

B_6: **Automatische Generierung und Optimierung von Testzeitplänen**: Durch die Abhängigkeiten zwischen Services kann es bei mehreren Änderungen im SOA-Systemverbund notwendig sein, diese Abhängigkeiten auch bei den Testvorgängen zu beachten. Dieser Beitrag zu den Forschungszielen Z_2 und Z_3 erstellt basierend auf einem Abhängigkeitsgraphen der Services sowie weiteren Randbedingungen Zeitpläne für das Testen eines SOA-Systemverbundes. Diese Pläne lassen sich zudem mittels unterschiedlicher Verfahren optimieren.

B_7: **SOA Governance Information System**: Die an einer SOA beteiligten Stakeholder können nur effizient ihrer Arbeit nachgehen, wenn ihnen möglichst zielgerichtet auf ihre Rolle zugeschnitte Informationen bereitgestellt werden. Der Beitrag zu den Forschungszielen Z_2 und Z_3 stellt SOA-Stakeholdern im Unternehmensumfeld daher ein Informationssystem mit rollenbasiertem Dashboard zur Verfügung, das eine individuelle Anpassung an die Bedürfnisse und den Aufgabenbereich der jeweiligen Stakeholder ermöglicht [KM16].

B_8: **Prototypische Implementierung**: Ziel dieser Arbeit ist es auch, die entwickelten Forschungsbeiträge zu den Zielen Z_1–Z_3 für einen praktischen Einsatz prototypisch vorzubereiten und zu erproben. Zu diesem Zweck wurde ein Softwaresystem, das SOA Governance Repository, entwickelt. Mit diesem Prototypen wurde ein Tool geschaffen, das eine vollumfängliche und effiziente Governance einer Unternehmens-SOA ermöglicht [KM16].

1.4 Gliederung der Arbeit

Die weiteren Kapitel dieser Arbeit sind wie folgt strukturiert: Kapitel 2 stellt den Hintergrund der Arbeit und den Stand der Forschung vor. Kapitel 3 beschäftigt sich mit den Schnittstellen serviceorientierter Architekturen und stellt mit den Business Objects *plus* (B_3) einen Ansatz zu deren Harmonisierung vor. In Kapitel 4 wird mit dem SOA Governance Meta Model (B_1) und der SOA Governance Ontology (B_2) ein Verständnis für die Bestandteile serviceorientierter Architekturen geschaffen und die Grundlage für die Entwicklung semantischer SOA-Governance-Anwendungen gelegt. Kapitel 5 stellt die REST-to-SOAP Middleware Architecture zur Bereitstellung von klassischen Webservices über RESTful APIs vor (B_4). In Kapitel 6 wird das Testen in SOA-Systemverbünden betrachtet. Dazu wird ein Konzept zur automatischen Generierung von Testzeitplänen vorgestellt (B_6). Ebenfalls betrachtet wird die Risikobewertung von Änderungen (B_5). Kapitel 7 stellt das Konzept eines SOA Governance Information Systems (B_7) und den umgesetzten Prototypen (B_8) vor. Kapitel 8 schließt die Arbeit mit einer Zusammenfassung ab und gibt einen Ausblick auf weitere Forschungsrichtungen.

Kapitel 2

Hintergrund und Stand der Forschung

Dieses Kapitel stellt den Hintergrund der Arbeit und den Stand der Forschung vor, mit dem Ziel einen Überblick über die Themenbereiche der Arbeit und ihren Kontext zu vermitteln. Der Fokus liegt dabei speziell auf dieser Forschungsarbeit, detaillierte grundlegende Ausführungen zu den Gebieten Enterprise Architecture Management (EAM) [ARW08], Enterprise Application Integration (EAI) [HW03], serviceorientierte Architekturen (SOA) [Bur10; Pv07], Webservices [ACKM04; WCL+05] und Semantic Web [HKRS08] möge der Leser der soeben referenzierten Literatur entnehmen.

Abschnitt 2.1 stellt zunächst die informationstechnischen Grundlagen der Arbeit vor. Abschnitt 2.2 gibt daran anschließend einen Überblick über den Themenkomplex SOA Governance und Change Management. Abschließend gibt Abschnitt 2.3 einen Überblick über den Anwendungskontext dieser Arbeit.

2.1 Informationstechnische Grundlagen

Dieser Abschnitt stellt zunächst Service-Technologien vor und zieht daran anschließend einen Vergleich der beiden verbreitetsten Technologien in diesem Bereich. Im zweiten Teil des Abschnitts werden semantische Technologien diskutiert.

2.1.1 Service-Technologien

Ein Service ist eine abgeschlossene Komponente, die eine gewisse Funktionalität bietet und über eine definierte Schnittstelle angesprochen werden kann. Die konkrete Umsetzung, also verwendete Programmiersprache oder Plattform, sind durch die Schnittstelle verborgen. Services nutzen unter Umständen weitere Services im Hintergrund [The09b].

Das Paradigma der serviceorientierten Architektur (SOA) setzt auf lose gekoppelte Services zur Durchführung von Geschäftsprozessen [RLSB12]. Teilnehmer einer SOA sind auf der einen Seite Service-Provider, die Services anbieten und auf der anderen Seite Service-Consumer, die Services nutzen. Service-Provider und -Consumer sind dabei immer als Softwaresysteme zu verstehen. Für die Nutzung von Services werden zwischen Provider und Consumer, insbesondere im Unternehmenskontext, *Service Level Agreements* abgeschlossen. In diesen werden Rechte und Pflichten der Teilnehmer festgelegt, wie etwa die zur erbringende Service-Verfügbarkeit oder Reaktionszeiten bei Ausfällen.

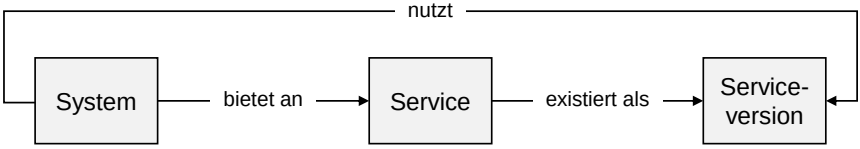


Abbildung 2.1: Beziehungen zwischen den Hauptkomponenten einer SOA

Basis-Services werden im Allgemeinen von einem Softwaresystem angeboten. Services, die eine Kombination mehrerer anderer Services darstellen, können dahingegen auch von Middleware-Komponenten wie einem Enter-

prise Service Bus angeboten werden. Im Weiteren wird unter dem Begriff „Service“ ein Basis-Service verstanden, der von einem Softwaresystem angeboten wird. Ein solches System kann mehrere Services anbieten, welche wiederum in unterschiedlichen Ausprägungen, sogenannten Serviceversionen, existieren. Ein Service kann nur in Form einer konkreten Serviceversion genutzt werden. Abbildung 2.1 verdeutlicht diese Beziehungen. Ein System kann dabei sowohl als Consumer wie auch als Provider agieren.

Zur Realisierung einer SOA können unterschiedliche Technologien zum Einsatz kommen. Dabei haben sich insbesondere klassische Webservices durchgesetzt, die Nachrichten über Internetprotokolle wie Hypertext Transfer Protocol (HTTP) übertragen. Da diese eine recht umfangreiche Beschreibung erfordern und weitere Technologien zur Umsetzung erfordern, sind in den letzten Jahren Services entstanden, die auf das leichtgewichtigere Architektur-Paradigma *Representational State Transfer* (REST) setzen.

2.1.1.1 Klassische Webservice

Als klassische Webservices werden im Kontext dieser Arbeit XML-basierte (*Extensible Markup Language*) Webservices bezeichnet, wie vom World Wide Web Consortium (W3C) definiert:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [BHM+04]

Die Beschreibung des Services, seiner Schnittstellen, Endpunkte und Nachrichtenformate erfolgt mittels der XML-basierten *Web Service Description Language* (WSDL, [CCMW01]). Diese liegt inzwischen in der Version 2.0 vor [W3C07b]. Der Nachrichtenaustausch erfolgt über das ebenfalls XML-basierte SOAP-Protokoll [W3C07a].

Um die Spezifikationen von WSDL und SOAP gruppieren sich eine Vielzahl weiterer Standards, die allgemein unter der Abkürzung *WS-** zusammengefasst werden. Diese WS-*-Standards erfüllen viele Anforderungen wie sie in Unternehmensarchitekturen entstehen, beispielsweise an die Sicherheit (WS-Policy, [W3C07d]), Orchestrierung von Webservices (WS-BPEL, [OAS07]) oder verlässliche Nachrichtenübermittlung (WS-Reliable Messaging, [OAS09]). Mit dem *WS-I Basic Profile* [Web10] existiert zudem eine Spezifikation, welche die Interoperabilität der WS-*-Spezifikationen sicherstellen soll. Dementsprechend häufig sind in Unternehmen SOA-Umgebungen zu finden, die sich auf die genannten Standards stützen.

2.1.1.2 RESTful Webservices

RESTful Webservices stützen sich auf das Architektur-Paradigma *Representational State Transfer*, das abstrakt auch die Funktionsweise des World Wide Web beschreibt. Der REST-Architekturstil wird von vier Prinzipien geleitet [RR07; PZL08]:

- **Ressourcenidentifikation durch URIs**: Ein RESTful Webservice ermöglicht den Zugriff auf Ressourcen, die eindeutig über einen Uniform Resource Identifier (URI) [Int05] identifizierbar sind.
- **Einheitliche Schnittstelle**: Der Zugriff auf Ressourcen erfolgt über eine definierte Menge von vier HTTP-Operationen [Int14]: *PUT*, *GET*, *POST*, *DELETE*. *PUT* erzeugt oder überschreibt eine Ressource, *GET* ermöglicht den Zugriff auf den aktuellen Stand einer Ressource, *POST* erzeugt einen neuen Zustand einer Ressource und *DELETE* löscht eine Ressource. Die Methoden *PUT*, *GET* und *DELETE* sind dabei idempotent, das heißt auch bei wiederholter Ausführung führen sie zum gleichen Resultat. Dies ist wichtig, da diese Operationen beispielsweise bei Zeitüberschreitungen automatisch erneut ausgeführt werden können und sichergestellt ist, dass immer das gleiche Resultat erzielt wird.
- **Selbstbeschreibende Nachrichten**: Ressourcen sind losgelöst von ihrer Repräsentation, das heißt sie können auf unterschiedliche Arten

in unterschiedlichen Formaten dargestellt werden. Unterstützt wird der Umgang mit Ressourcen durch Metadaten, beispielsweise zur Aushandlung des Formats.

- **Zustandsbehaftete Interaktionen über Hyperlinks**: Jede Anfrage an eine Ressource erfolgt isoliert, ist also zustandslos. Zustandsbehaftete Interaktionen werden durch die explizite Übertragung eines Zustandes ermöglicht, etwa durch zusätzliche Parameter oder die Rückgabe weiterer Interaktionspunkte.

REST beschränkt sich dabei auf die Rolle der Komponenten und die Bedingungen unter denen sie miteinander kommunizieren und lässt Vorgaben zur Implementierung der Komponenten und zur Protokoll-Syntax außen vor [FT02]. Im Zusammenhang mit RESTful Webservices wird immer häufiger auch der Begriff *REST-API* (Application Programming Interface) verwendet.

Auch für RESTful Webservices existieren verschiedene Beschreibungssprachen. Weit verbreitet ist inzwischen die *OpenAPI Specification* [Ope18].

2.1.1.3 Vergleich von klassischen und RESTful Webservices

Seit dem Aufkommen von RESTful-Services gab es immer wieder Diskussionen über Unterschiede und Gemeinsamkeiten zu SOAP-basierten, klassischen Webservices in teilweise emotional geführten Debatten. Als Ergebnis muss festgehalten werden, dass beide Ansätze Vor- und Nachteile haben und diese stark vom jeweiligen Anwendungsfall abhängen.

Ein ausführlicher Vergleich der beiden Varianten von Pautasso et al. [PZL08] kommt zum Schluss, dass diese sich sehr ähnlich sind, zumindest wenn für beide die gleichen Architekturentscheidungen getroffen werden. Als Vorteile klassischer Webservices nennen die Autoren die Protokoll-Unabhängigkeit und -Transparenz, die es erlaubt SOAP-Nachrichten mit einer Vielzahl an Middleware-Systemen zu transportieren. Durch die Beschreibung der Services mit einer WSDL-Datei wird zusätzlich eine Abstraktion von darunterliegenden Implementierungen und Kommunikationsprotokollen geschaffen, so dass auch bei sich ändernden Technologie-Anforderungen die Service-

schnittstelle nicht angepasst werden muss. Gleichzeitig wird der Umfang des WS-*-Stacks als Nachteil genannt, da es leicht zu Interoperabilitätsproblemen kommen kann. Hier schafft das WS-I Basic Profile Abhilfe.

RESTful Webservices werden häufig als „einfacher“ empfunden, da sie auf weit verbreitete und bekannte Standards aufsetzen, unter anderen HTTP und URI. Da die notwendige Infrastruktur bereits weit verbreitet ist, ist der Aufwand für die Entwicklung eines RESTful Webservice sehr gering. Dadurch, dass REST bereits Caching unterstützt, sind RESTful Webservices zudem sehr gut skalierbar [PZL08]. Die Flexibilität und Offenheit von REST ist gleichzeitig der Nachteil von RESTful Webservices. Nur bei korrekter Anwendung der REST-Prinzipien und der HTTP-Verben ist ein Service tatsächlich RESTful. Häufig erfolgt diese Anwendung falsch, so dass höchstens von HTTP-Schnittstellen, nicht aber von RESTful Webservice gesprochen werden kann [Fie08]. Das *Richardson Maturity Model* [Fow10] definiert drei Entwicklungsstufen zur Erreichung eines „echten“ RESTful Webservice: (1) Nutzung von Ressourcen statt eines einzigen umfangreichen Endpunktes, (2) Einführung und korrekte Nutzung der HTTP-Verben und (3) Selbstbeschreibung der Schnittstellen durch die Nutzung von HATEOAS (*Hypermedia as the Engine of Application State*).

Zusammengefasst eignen sich klassische Webservice demnach insbesondere für Unternehmensanwendungen, bei denen Wert auf Zuverlässigkeit und Sicherheit gelegt wird. RESTful Webservices bieten sich eher für leichtgewichtigere (hinsichtlich Performanz, Skalierbarkeit und Datenformaten) Integrationen über das Web und damit insbesondere auch für mobile Anwendungen an.

2.1.2 Semantische Technologien

Treiber für die Entwicklung und Nutzung semantischer Technologien ist die Idee des *Semantic Web* von Berners-Lee et al. [BHL01]:

> „The Semantic Web is [...] an extension of the current [web] in which information is given well-defined meaning, better enabling computers and people to work in cooperation.“ [BHL01]

Ziel ist es also, das „klassische“ Web um maschinenlesbare Semantik zu ergänzen und so auch Computern die Verarbeitung der dort vorliegenden Informationen zu ermöglichen.

Eine der Basistechnologien des Semantic Web ist das *Ressource Description Framework* (RDF, [W3C14a]). RDF ist ein Standard zur Beschreibung von Beziehungen zwischen Ressourcen in Form eines gerichteten Graphen. Ressourcen werden über URIs eindeutig identifiziert und Aussagen über sie in Form von Tripeln modelliert.

Abbildung 2.2: Grafische Darstellung eines Tripels: Die Ressource *Moby Dick* hat den Autor *Herman Melville*

Abbildung 2.2 zeigt ein einfaches Tripel in grafischer Darstellung. RDF ist unabhängig von einer bestimmten Syntax, es existieren allerdings verschiedene textbasierte Repräsentationen, wie RDF/XML [W3C14b] und N3 [W3C11a]. Listing 2.1 zeigt eine Repräsentation des Tripels aus Abbildung 2.2 in RDF/XML. Ein Tripel besteht immer aus einem *Subjekt*, einem *Prädikat* und einem *Objekt*. Subjekt und Prädikat werden immer durch einen URI beschrieben, ein Objekt kann sowohl durch einen URI als auch durch ein Literal beschrieben werden. Ein Literal ist eine Zeichenkette, unter Umständen ergänzt durch einen Datentyp. Im obigen Beispiel könnte also statt des Literals *Herman Melville* auch die Ressource *http://example.org/Persons/HermanMelville* referenziert werden. Durch solche Verknüpfungen entsteht aus einer Menge von Tripeln eine Graphstruktur.

```
1 <?xml version="1.0"?>
2 <rdf:RDF
3   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
      #"
4   xmlns:rel="http://example.org/rel#">
```

```
5     <rdf:Description
6     rdf:about="https://www.example.org/Books/MobyDick">
7         <rel:author>Herman Melville</rel:author>
8     </rdf:Description>
9 </rdf:RDF>
```

Listing 2.1: Einfaches Beispiel für ein RDF-Tripel in RDF/XML

Um Informationen in einem RDF-Graph auslesen und bearbeiten zu können, wurde mit der *SPARQL Protocol and RDF Query Language* (SPARQL, [W3C13a]) eine Anfragesprache definiert. SPARQL-Anfragen werden in Form von Graph-Mustern formuliert. Ein solches Muster entspricht einem RDF-Tripel, mit der Ausnahme, dass Subjekt, Prädikat und Objekt auch Variablen sein dürfen. Der RDF-Graph wird dann nach Teilgraphen durchsucht, die dem Graph-Muster entsprechen. Listing 2.2 zeigt eine einfache SPARQL-Anfrage an den in Listing 2.1 beschriebenen RDF-Graphen zum Auslesen des Autors.

```
1 SELECT ?author
2 WHERE
3 {
4   <https://www.example.org/Books/MobyDick> <http://
        example.org/rel#author> ?author .
5 }
```

Listing 2.2: Einfache SPARQL-Anfrage

Da RDF nur eine Syntax definiert, wurde mit Resource Description Framework Schema (RDFS, [W3C14c]) ein Vokabular zur Strukturierung von RDF-Ressourcen definiert, mit dem sich Taxonomien und einfache Ontologien beschreiben lassen.

2.1.2.1 Taxonomien und Ontologien

Mit *Taxonomien* können einfache Begriffshierarchien dargestellt werden. Sollen darüber hinaus noch zusätzliche Informationen und Relationen beschrieben werden, kommen *Ontologien* zur Anwendung, die zur Beschreibung und dem Austausch von Wissen über eine bestimmte Anwendungsdomäne dienen [Gru93].

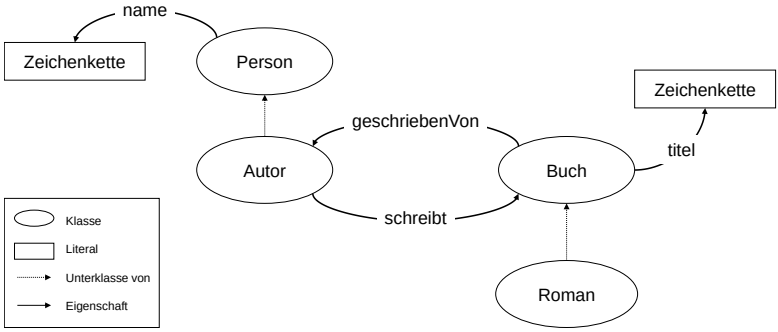


Abbildung 2.3: Ontologie zur Beschreibung von Büchern und Autoren

Abbildung 2.3 zeigt eine einfache Ontologie zur Beschreibung von Büchern und Autoren. Die Ontologie enthält Klassen (*Person*, *Buch*) mit Eigenschaften (*name*, *titel*), sowie Relationen (*schreibt*) zwischen den Klassen. Relationen und Eigenschaften können vererbt werden, im Beispiel erbt daher *Autor* als Unterklasse von *Person* die Eigenschaft *name* und *Roman* die Relation *geschriebenVon*.

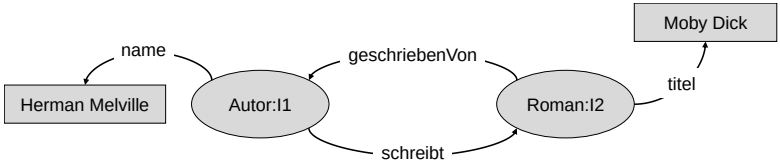


Abbildung 2.4: Instanz der Ontologie zur Beschreibung von Büchern und Autoren

Konkret nutzen lassen sich Ontologien durch die Instanziierung der Klassen, wie in Abbildung 2.4 dargestellt. Dabei steht $I1$, $I2$, ... für die eindeutigen Ressourcennamen der Instanzen, also jeweils einen URI.

Zur formalen Beschreibung von Ontologien wurde die *Web Ontology Language* (OWL, [W3C12]) spezifiziert. OWL setzt auf RDF und RDFS auf und erweitert diese um weitere Konstrukte. Grundlegend wird auch in OWL un-

terschieden zwischen Klassen, Eigenschaften und Instanzen. OWL definiert weitere Eigenschaften und Klassen, beispielsweise Disjunktion, Kardinalitäten und Gleichheit [W3C12]. OWL hat das erklärte Ziel, die Inhalte des World Wide Web besser für Maschinen zugänglich zu machen. So lassen sich auch mittels Reasonern logische Schlussfolgerungen ziehen. (Semantische) Reasoner sind Softwareprogramme, die aus semantischen Datensätzen automatisiert Wissen inferieren, also implizites Wissen explizit machen. Dabei gilt die Annahme der Weltoffenheit (engl.: Open World Assumption). Demnach werden Aussagen, die nicht explizit als wahr bekannt sind, nicht als falsch angesehen, sondern sind unbekannt.

2.1.2.2 Einsatz von Semantic-Web-Technologien im Unternehmensumfeld

Erst seit kurzem finden semantische Anwendungen auch ihren Weg in Unternehmen und damit in Unternehmensanwendungen. Solche Anwendungen werden zusammengefasst unter dem Begriff Corporate Semantic Web (CSW) [PCH+10; HHS+16]. Eine solche Anwendung "[...] wendet semantische Technologien im Unternehmenskontext an, sowohl zur Unterstützung interner (Webbasierter) Geschäftsinformationssysteme und –prozesse, als auch in Anwendungen und Diensten, die im öffentlichen Web (Public Semantic Web) von Unternehmen angeboten werden [...]"[EHR15]. Eine Hauptcharakteristik eines CSW ist die klar definierte Eingrenzung auf die Anwendungsdomäne eines Unternehmens. Dadurch lassen sich Probleme vermeiden, die durch die Offene-Welt-Annahme (*englisch: open world assumption*) bei „öffentlichen“ semantischen Netzwerken auftreten können. Die Offene-Welt-Annahme beschreibt das Konzept, dass Aussagen, die nicht explizit als wahr bekannt sind, nicht als falsch angesehen werden dürfen, sondern unbekannt sind. Dahingegen beschreibt das Konzept der Weltabgeschlossenheit (*englisch: closed world assumption*), dass alle Aussagen, die nicht als wahr bekannt sind, automatisch falsch sind. Zum Tragen kommen diese Probleme beispielsweise, wenn vorhandene relational beschriebene Konzepte und Abhängigkeiten in eine semantische Beschreibung überführt werden, da relationale Datenbanken von einer Annahme der Weltabgeschlossenheit ausgehen. Durch die

begrenzte Anwendungsdomäne ist es nun möglich in gewissem Rahmen auch im CSW von einer teilweisen Annahme der Weltabgeschlossenheit auszugehen, also in bestimmten Bereichen trotz fehlender Informationen bestimmte Aussagen zu treffen.

Zusammen mit dem Semantic Web selbst, haben sich auch die bereits vorgestellten Sprachen und Komponenten, die zur Umsetzung semantischer Anwendungen benötigt werden, weiterentwickelt. Auch RDF-Datenbanken bzw. Tripel-Speicher und Semantic Web Frameworks haben einen Reifegrad erreicht, der den Einsatz im Unternehmensumfeld mit seinen hohen Anforderungen erlaubt [HHS+16].

2.2 SOA Governance

Für den Begriff SOA Governance existieren viele Definitionen mit unterschiedlichem Fokus. Oracle [Ram13] definiert sie als

> „[...] the creation and administration of policies for the purpose of influencing and enforcing actions and behaviors that align with business objectives.“ [Ram13]

Allen & Wilkes [AW10] definieren SOA Governance als

> „The part of IT governance that refers to the organizational structures, policies and processes that ensure that an organization's SOA efforts sustain and extend the organization's business and IT strategies, and achieve the desired outcomes.“ [AW10]

Holley et al. [HPG06] sehen SOA Governance ebenfalls als Erweiterung der IT-Governance, mit Fokus auf den Lebenszyklus von Services.

Da eine SOA auch über Organisationsgrenzen hinweg wirkt und direkt von den Geschäftszielen eines Unternehmens beeinflusst wird, kann sie nicht als reine Erweiterung der IT-Governance gesehen werden, sondern muss alleinstehend betrachtet werden, wenn auch mit enger Kopplung zu den Prozessen der IT- und Unternehmens-Governance. Daher wurde aus

der Vielzahl der existierenden Definitionen im Rahmen dieser Arbeit die folgende Definition destilliert und wird im Weiteren verwendet:

> SOA Governance dient zur effektiven Steuerung der technischen, strategischen, organisationsbezogenen und personellen Anforderungen einer serviceorientierten Architektur. Sie ist zuständig für die Einführung von Richtlinien und Steuerungsmechanismen sowie Prozessen, um diese zu überwachen und durchzusetzen. Weiter ist sie dafür verantwortlich die Aktivitäten, die im Rahmen einer SOA durchgeführt werden, auf die Unternehmensziele auszurichten.

Detaillierte Anforderungen an SOA-Governance-Prozesse in Unternehmen werden in Kapitel 4 erhoben und diskutiert. Verschiedene Organisationen und Unternehmen haben SOA-Governance-Frameworks definiert, in denen allgemeine Herangehensweisen an das Thema, benötigte Prozesse oder beteiligte Stakeholder definiert werden. In Abschnitt 4.1.2 findet sich eine Übersicht über existierende SOA-Governance-Frameworks und ein detaillierter Vergleich.

Ein Teilbereich der SOA Governance ist das *Change Management* (deutsch: *Änderungsmanagement*), innerhalb dessen Änderungen an Softwaresystemen koordiniert begleitet und durchgeführt werden. Die Betriebswirtschaftslehre kennt den Begriff Change Management ebenfalls. Dort sind darunter die Prozesse zusammengefasst, die zu einer umfangreichen Veränderung einer Organisation durchgeführt werden müssen [BC12]. Im weiteren Verlauf dieser Arbeit ist unter Change Management immer das Änderungsmanagement von Softwaresystemen zu verstehen. Auch bei Änderungen an einer SOA müssen Änderungsprozesse als Teil der SOA Governance durchgeführt werden. Dabei handelt es sich insbesondere um Änderungen an Services und deren Schnittstellen. Diese müssen in einer Form durchgeführt werden, die möglichst effizient ist und möglichst geringe Auswirkungen auf den laufenden Betrieb hat.

Auch die *IT Infrastructure Library*[1], eine der bekanntesten Sammlungen

[1] https://www.axelos.com/best-practice-solutions/itil

von Best Practices in der IT, betrachtet im Teilbereich *Service Transition* das Change Management [Ran11]. ITIL definiert dazu Vorgehensweisen und Prozesse, beispielsweise zur Definition und zum Umgang mit Änderungsanforderungen (engl.: *Requests for Change*). Diese Prozesse können, da sehr allgemein gehalten, auch auf das Change Management der SOA Governance angewendet werden.

2.3 Anwendungskontext

Im Rahmen der vorliegenden Arbeit wurden im Speziellen Aspekte des Change Management großer SOA-Systemverbünde in Unternehmen untersucht. Als *SOA-Systemverbund* wird in dieser Arbeit ein Zusammenschluss von Softwaresystemen verstanden, die im Rahmen einer SOA Services anbieten und zwischen denen es Nutzungsabhängigkeiten gibt. Der Systemverbund wird dabei als Einheit betrachtet, die im Ganzen weiterentwickelt wird [KS11]. Dies schließt insbesondere auch die Veröffentlichung von Änderungen in gemeinsamen koordinierten *Verbundreleases* mit ein. Das dafür notwendige gemeinsame Testmanagement wird in Kapitel 6 näher betrachtet.

Wie auch innerhalb eines einzelnen Softwaresystems gibt es auch in einem SOA-Systemverbund eine Vielzahl an wechselseitigen Abhängigkeiten und beteiligten Personen, die eigene Interessen in den Betrieb und die Weiterentwicklung des Verbundes mit einbringen. Solche Personen werden im Weiteren mit dem englischen Begriff *Stakeholder* bezeichnet. Tabelle 2.1 gibt einen (keinesfalls vollständigen) Überblick über Stakeholder-Rollen, die an den Prozessen im Umfeld eines SOA-Systemverbundes beteiligt sind. Die Zusammenfassung der konkreten Stakeholder zu Rollen erfolgte im Rahmen dieser Arbeit bei einer Analyse der SOA-Governance-Frameworks. Stakeholder-Rollen fassen Aufgabenbereiche der Stakeholder zusammen und konkrete Stakeholder können dabei auch mehrere Rollen ausfüllen. Der Eigner (engl.: *owner*) eines Systems kann gleichzeitig auch als Projekt-Manager tätig sein und in anderem Kontext als Consumer eines Services auftreten.

Tabelle 2.1: Stakeholder-Rollen einer SOA

Rolle	Beschreibung	Beispiel
Manager	Ein Manager hat die Verantwortung und Kontrolle über einen Teilaspekt oder ein Teilprojekt im Systemverbund.	Projekt-Manager (vgl. [The09a; Bie06]), Service Manager (vgl. [Bis14])
Architekt	Ein Architekt ist verantwortlich für Architekturentscheidungen und die Festlegung von Standards für eine SOA im Gesamten oder Teilen davon, wie Systeme oder Services.	Enterprise-Architekt (vgl. [The09a]), Service-Architekt (vgl. [Bie06])
Entwickler	Entwickler sind zuständig für die Realisierung von Systemen und Services entsprechend der Vorgaben der Manager und Architekten.	System-Entwickler (vgl. [The09a]), Service-Entwickler (vgl. [Bie06; RB08])
Tester	Tester sind zuständig für die Planung und Durchführung von Softwaretests an den Bestandteilen der SOA.	Test Strategist (vgl. [The09a]), Tester (vgl. [The09a; Bie06])
Administrator	Administratoren sind zuständig für die Verwaltung der Infrastruktur sowie Installation und Betrieb der Systeme und Services.	Datenbank-Administrator (vgl. [The09a; Bie06]), Service-Administrator (vgl. [Afs07])
Consumer	Consumer sind Vertreter eines Systems, in dem ein bestimmter Service genutzt wird.	Consumer (vgl. [RB08]), Service Owner (vgl. [Afs07])
Eigner	Eigner sind verantwortlich für Bestandteile des SOA-Verbundes, wie beispielsweise Systeme oder Services.	Service Owner (vgl. [Afs07]), Business Domain Owner (vgl. [RB08; The09a])

Zur Verdeutlichung der Komplexität und Anzahl der Abhängigkeiten innerhalb eines SOA-Systemverbundes zeigt Abbildung 2.5 eine anonymisierte Darstellung eines realen SOA-Systemverbunds des Anwendungspartners. Abgebildet sind Systeme (blau), deren Services (orange) und zugehörige Serviceversionen (grün). Rot gestrichelt werden Nutzungsbeziehungen, also Consumer-Provider-Beziehungen, zwischen Systemen und Serviceversionen.

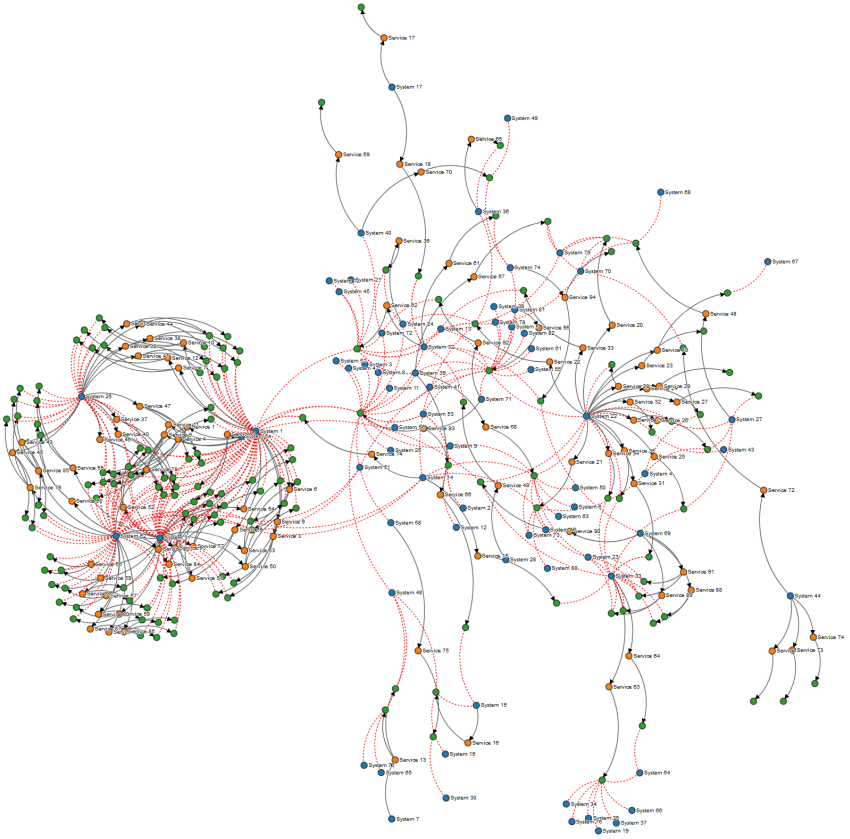


Abbildung 2.5: Abhängigkeiten in einem realen Systemverbund (anonymisierte Darstellung)

Diese Arbeit ist ebenfalls eingebettet in den Forschungskontext einer zukunftsfähigen und wandlungsfähigen Produktion. Daher sind die entwickelten Konzepte auch in die *Stuttgart IT Architecture for Manufacturing* (SITAM, [KGK+17] eingeflossen. Die SITAM beschreibt eine agile, lernende und menschenzentrische Unternehmensarchitektur, die den gesamten Produktlebenszyklus inklusive aller Prozesse, IT-Systeme, physischen Maschinen und anderer Datenquellen umfasst. Abbildung 2.6 zeigt eine vereinfachte Darstellung der Architektur.

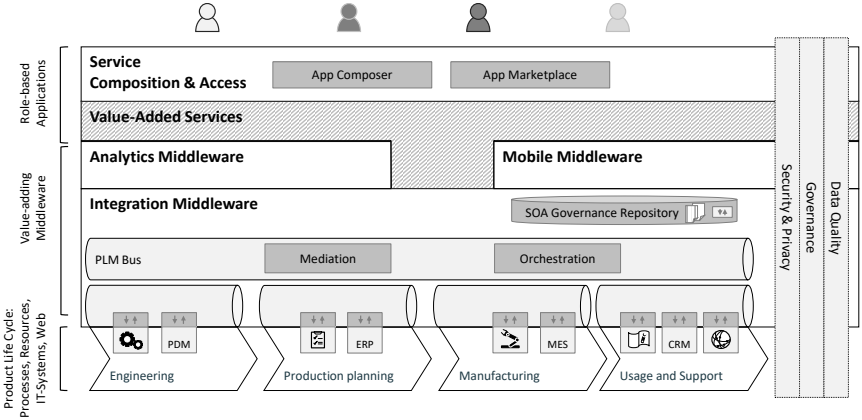


Abbildung 2.6: Stuttgart IT Architecture for Manufacturing. Vereinfachte Darstellung in Anlehnung an [KGK+17].
Adapted by permission from Springer: Springer Nature, Enterprise Information Systems. ICEIS 2016. Lecture Notes in Business Information Processing, vol 291. The Stuttgart IT Architecture for Manufacturing, Kassner et al., ©2017

Über eine Integrationsschicht wird serviceorientiert der Zugriff auf Ressourcen im gesamten Produktlebenszyklus ermöglicht. Auf dieser Schicht finden sich auch die Konzepte dieser Arbeit wieder. Speziell die Nutzung einheitlicher Datenobjekte (vgl. Kapitel 3) und ein Repository zur Verwaltung der bereitgestellten Services. Das in der SITAM nur abstrakt beschriebene Repository wurde in der vorliegenden Arbeit als Prototyp eines zentralen SOA Governance-Tools entwickelt. Im Kern bietet das *SOA Governance Repo-*

sitory (SGR) genannte Tool die Möglichkeit die Bestandteile einer SOA, die in Kapitel 4 beschrieben werden, umfassend zu dokumentieren. Auf dieser Basis stellt das SGR umfangreiche Funktionalitäten zur Verfügung, um den Stakeholdern einer SOA als zentrales Informationssystem eine effiziente Durchführung der SOA-Governance-Prozesse zu ermöglichen. Aufbauend auf der Integrationsschicht der SITAM stellen eine Analytics Middleware und eine Mobile Middleware Funktionalitäten zur Analyse strukturierter und unstrukturierter Daten und zum mobilen Datenmanagement bereit. Über die Mobile Middleware wird insbesondere die kontextbezogene Datenbereitstellung für mobile Endgeräte ermöglicht. Die in Kapitel 5 vorgestellte *REST-to-SOAP Middleware Architecture*, die sich in Teilen auch mit der Anbindung mobiler Endgeräte auseinandersetzt, würde die Integrationsschicht der SITAM ergänzen und als Grundlage für die Mobile Middleware dienen. Die Analytics und die Mobile Middleware bilden die Grundlage für wertschöpfende Services, die als rollenbasierte Anwendungen über einen App-Markplatz den Nutzern zur Verfügung gestellt werden.

Kapitel 3

Business Object Management

Das folgende Kapitel stellt das Konzept der Business Objects *plus* (BO^+) vor. Die Inhalte dieses Kapitels wurden bereits in [KM17] veröffentlicht.

Abschnitt 3.1 gibt einen Überblick über die Motivation und die Herausforderungen und skizziert die Forschungslücke. Daran anschließend werden in Abschnitt 3.2 verwandte Arbeiten vorgestellt und gegenüber dem Konzept der BO^+ abgegrenzt. Abschnitt 3.3 – 3.5 stellen das Konzept und zugehörige Management- und Governance-Prozesse im Detail vor. Abschnitt 3.6 diskutiert technische Aspekte und demonstriert den domänenübergreifenden Datenaustausch. Abschnitt 3.7 schließt das Kapitel mit einem Fazit und Ausblick.

3.1 Einführung

Immer kürzer werdende Innovationszyklen sowie zunehmende Globalisierung und dadurch steigende Konkurrenz haben in den vergangen Jahren

zu gestiegenen Anforderungen hinsichtlich Flexibilität und Anpassbarkeit von IT-Architekturen geführt. Daraus sind neue IT-Konzepte hervorgegangen, wie das Industrielle Internet der Dinge (englisch: *Industrial Internet of Things*, [JBSR17]) oder Cyber-physische Systeme (CPS, [Lee08]). Diesen Konzepten ist gemein, dass Informationen durch eine Vielzahl von Softwaresystemen propagiert und entlang des kompletten Produktlebenszyklus und über verschiedene Geschäftsdomänen hinweg integriert werden müssen. Dazu werden wohl-definierte und einheitliche Schnittstellen für den Hersteller-unabhängigen Zugriff auf Daten benötigt.

Um diese Aufgabe zu lösen, sind serviceorientierte Architekturen (SOA), die seit einigen Jahren in Unternehmen eingesetzt werden, optimal geeignet. Leider wurden die anvisierten Verbesserungen in der Agilität von Geschäfts- und IT-Prozessen häufig nicht erreicht. Einer der Gründe dafür ist, dass Serviceschnittstellen aus einer sehr technischen Sichtweise heraus definiert werden. Entwickler kennen meist nur die technischen Spezifikationen, nicht aber die Geschäftsprozesse, in denen die Services zum Einsatz kommen. Als Ergebnis entstehen Schnittstellen, die zu eng an die Anwendung oder einzelne Anwendungsfälle gekoppelt sind.

Des Weiteren entstehen durch diese Art der Entwicklung zwei grundlegende Probleme:

- **Semantische Mehrdeutigkeit**: Die Namen der Felder von Serviceschnittstellen entsprechen häufig nicht dem im Unternehmen etablierten Sprachgebrauch, sondern sind technisch motiviert. Dadurch können Kommunikationsprobleme zwischen fachlichen und technischen Bereichen entstehen. Durch die unabhängige Entwicklung mehrerer Schnittstellen werden semantisch äquivalente Felder in verschiedenen Services unterschiedlich benannt [Jos07] (*Synonyme*). Ebenso kann es vorkommen, dass semantisch unterschiedliche Felder gleich benannt werden (*Homonyme*).
- **Inkompatible Datenformate**: Da Serviceschnittstellen getrennt voneinander entwickelt werden, sind die Datenformate dieser Schnittstellen in den seltensten Fällen zueinander kompatibel. Dies führt dazu,

dass für den Datenaustausch zwischen Systemen mit unterschiedlichen Daten-Repräsentationen transformiert werden muss. Im schlimmsten Fall ist so für jede Kombination zweier Systeme eine eigene Transformation notwendig [BGL07; Erl09].

Das Problem der semantischen Mehrdeutigkeit ließe sich durch die Definition und Durchsetzung von Governance-Richtlinien lösen. Diese könnten von Entwicklern verlangen, dass vor der Definition einer Schnittstelle ein Abgleich von Feldnamen mit einer zentralen Instanz, wie einem Architekturgremium, oder mit einem kontrollierten Vokabular [Nat05] stattfindet. Allerdings zeigt die Praxis, dass solche Richtlinien nur schwer durchzusetzen sind [PSVS07; KAS09].

Das Problem der inkompatiblen Datenformate lässt sich im Gegensatz dazu allerdings nicht so einfach in den Griff bekommen. Dies liegt unter anderem daran, dass Firmen häufig gar keinen Einfluss auf die Datenmodelle von Serviceschnittstellen haben, beispielsweise wenn Standardsoftware eingesetzt wird oder externe Entwickler eigene Code-Richtlinien verwenden. Für letztere lassen sich zwar Architektur- oder Programmier-Richtlinien vorgeben, sofern die Entwickler vom Unternehmen direkt beauftragt wurden, der Einfluss auf die Schnittstellen von Standardsoftware ist allerdings begrenzt.

In der Vergangenheit gab es bereits Ansätze Datenmodelle von Anwendungen und Services zu vereinheitlichen, indem sogenannte kanonische Datenmodelle definiert wurden, die dann unternehmensweit gelten sollten. Versuche kanonische Datenmodelle großflächig einzuführen schlugen allerdings fehl, da die Modelle zu umfangreich und damit unwartbar wurden [Til15]. Ebenfalls war es kaum möglich alle Anwendungsfälle damit abzudecken. Andere Ansätze betrachten einzelne Geschäftsobjekte (englisch: *Business Objects*) im Kontext bestimmter Geschäftsprozesse. Entsprechende Ansätze lassen jedoch den Aspekt der Wiederverwendung im gesamten Unternehmen und damit entstehende Vorteile außer acht. Geschäftsobjekte sind zwar sehr implementierungsspezifisch, aber verfolgen zu einem gewissen Grad schon die Idee der einheitlichen Definition von geschäftsrelevanten

Entitäten. Das folgende Konzept der Business Objects *plus* stützt sich daher auf diesen Grundgedanken und verfolgt ihn weiter für eine Nutzung in großen, domänenübergreifenden SOA-Systemverbünden.

Eines der grundlegenden Konzepte serviceorientierter Architekturen ist die Wiederverwendbarkeit von Services [Erl08]. Allerdings gilt dies nur für die Services selbst, nicht aber für ihre Schnittstellen-Definitionen. Da Services individuell entwickelt werden, werden existierende Schnittstellen-Definitionen nicht wiederverwendet, auch wenn dies eigentlich möglich wäre. Häufig werden in Serviceschnittstellen semantisch identische Mengen von Feldern benötigt, zum Beispiel um einen Kunden abzubilden. Da Schnittstellen allerdings unabhängig voneinander definiert werden, sind solche Felder meist unterschiedlich benannt.

Ein weiterer Aspekt ist die Stabilität der Serviceschnittstellen. Wenn diese nicht korrekt verwaltet werden, sind sie häufig höchst instabil, d.h. sie werden sehr oft geändert und angepasst. Dies geschieht einerseits weil Entwickler Schnittstellen einfach ändern um sie „schöner“ oder sauberer zu designen, ohne an die Auswirkungen zu denken. Jede durchgeführte Änderung an einer Schnittstelle hat zur Folge, dass alle Anwendungen, die diese Schnittstelle nutzen, auch an die neue Schnittstellen-Version angepasst werden müssen. Wenn nun solche Änderungen durchgeführt werden und unter Umständen nicht einmal dokumentiert sind, führt dies zu Fehlern in der Anbindung des Services mit unklaren und unter Umständen weitreichenden Auswirkungen auf den SOA-Systemverbund. Andererseits kommt es natürlich ebenfalls zu Änderungen, wenn Schnittstellen um neue Funktionalitäten erweitert werden müssen. Solche Änderungen müssen aber in einem koordinierten Änderungsprozess ablaufen.

In den folgenden Abschnitten wird das Konzept für die Definition von Business Objects *plus* vorgestellt und die zugehörigen Management- und Governance-Prozesse werden definiert. Hauptziel der BO^+ ist es, die oben geschilderten Probleme durch die Definition kleiner, abgeschlossener Datenobjekte zu lösen. Diese Datenobjekte sind fachlich motiviert, beschreiben also Entitäten aus fachlicher, nicht technischer Sicht.

3.2 Verwandte Arbeiten

Der folgende Abschnitt stellt verwandte Arbeiten vor und grenzt sie vom Konzept der BO^+ ab. Die verwandten Arbeiten können in zwei Kategorien eingeteilt werden:

- Getestete und in der Praxis fehlgeschlagene Ansätze wie das *kanonische Datenmodell*.
- Aktuelle Ansätze, die auf einer deutlich differenzierten Sichtweise beruhen, unterstützt durch ein besseres Verständnis von Geschäftsprozessen. Hervorzuheben sind hier die bereits genannten *Business Objects* und das Konzept der *Business Artifacts*.

Kanonisches Datenmodell Das Konzept eines kanonischen (oder einheitlichen) Datenmodells war seit Anfang an Teil der SOA-Forschung und Umsetzung, bzw. ganz allgemein Teil des Bereichs Enterprise Application Integration [HW03; BKM+08]. Der Gedanke dahinter ist die Minimierung der Datenmodelltransformationen zwischen unterschiedlichen Anwendungen und ein dadurch erleichterter Datenaustausch sowie eine vereinfachte Integration von Anwendungen. Dazu wird ein zentrales, unternehmensweit gültiges Datenmodell definiert, das die „ganze Welt“, also die komplette Anwendungsdomäne, beschreibt. Inzwischen hat sich gezeigt, dass kanonische Datenmodelle im Unternehmensumfeld nicht umsetzbar sind. Sie werden zu schnell zu groß und zu verworren um noch wart- oder verwaltbar zu sein [Til15]. Zudem ist es mit einem solchen Datenmodell nie möglich, alle Anwendungsfälle aller Applikationen gleichzeitig abzudecken, da diese je nach Nutzungskontext und Geschäftsbereich völlig unterschiedliche Anforderungen haben können. Der im Rahmen dieser Arbeit entwickelte Ansatz setzt zur Vermeidung dieser Probleme daher auf kleinere und mehrschichtige Modelle.

Business Objects Das Konzept der Business Objects entstammt der objektorientierten Programmierung. Hier dienen sie zur Modellierung von Datenobjekten und Prozessen in Informationssystemen. Neben den Daten selbst enthalten sie zudem noch Prozesslogik [Lho98]. Der Begriff wird außerdem von Firmen wie IBM, SAP und Oracle zur Beschreibung bestimmter Datenstrukturen in ihren Softwareprodukten verwendet[1,2,3].

Business Objects sind in allen genannten Ausprägungen sehr implementierungsspezifisch und werden, anders als der Name vermuten lässt, nicht übergreifend eingesetzt, sondern sind eng gekoppelt mit den Prozessen und Anwendungen, in denen sie verwendet werden. Entsprechend sind Business Objects auch nicht für die Datenintegration mittels Services gedacht. Das hier vorgestellte Konzept setzt auf die Grundidee der Business Objects auf, um einen Plattform-unabhängigen Weg zur Standardisierung von Datenmodellen innerhalb eines Unternehmens zu schaffen. Dabei wird insbesondere der Fokus auf Wiederverwendbarkeit und die Integration in domänenübergreifenden SOA-Systemverbünden gelegt.

Business Artifacts *Cohn and Hull* [CH09] stellen einen Ansatz vor, der Daten und Prozesse von geschäftsrelevanten Objekten in sogenannte *Business Artifacts* kombiniert. Diese Artefakte haben eine bestimmte Lebensdauer, bestimmt durch ihren jeweiligen Lebenszyklus. Sie entwickeln sich weiter (im Sinne der Informationen die sie enthalten und der Operationen die auf ihnen ausgeführt werden können) während sie von System zu System, bzw. Prozess weitergereicht werden. Der Ansatz betrachtet einzelne Artefakte und Artefakt-Modelle im Kontext einzelner Geschäftsprozesse. Der im Rahmen dieser Arbeit vorgestellte Ansatz betrachtet im Gegensatz dazu auch die semantische Eindeutigkeit und Wiederverwendbarkeit der Datenmodelle über Prozess- und Servicegrenzen hinweg.

[1] https://www.sap.com/products/bi-platform.html
[2] https://www.ibm.com/support/knowledgecenter/en/SS4KMC_2.5.0/com.ibm.ico.doc_2.5/enablement/businessobjects.html
[3] https://docs.oracle.com/cd/E15586_01/doc.1111/e15176/model_bus_objs_bpmpd.htm

Informationsintegration Das Feld der Informationsintegration bietet eine Vielzahl an Konzepten, Technologien und Tools um die Integration von heterogenen Datenquellen zu lösen. Der Hauptfokus liegt dabei allerdings häufig auf der Integration von Informationen aus unterschiedlichen Datenquellen in eine einzelne, uniforme Datenstruktur, die dann dazu verwendet werden kann die Daten besser zu nutzen und Informationen mittels Data-Mining-Techniken zu gewinnen. Das in diesem Kapitel vorgestellte Konzept verfolgt in Teilen das gleiche Ziel, ein einheitliches Datenmodell zu erstellen. Allerdings geschieht das in kleinerem Umfang und verfolgt auch nicht das Ziel alle Daten in eine neue Datenquelle zu vereinen.

Zusammenfassend lässt sich festhalten, dass alle bisherigen Ansätze sich als zu komplex erwiesen haben und häufig mehr Probleme erzeugen als sie lösen. Andere Ansätze verfolgen ähnliche Grundideen, erfüllen aber nicht vollständig die Anforderungen. Daher wird im Folgenden das Konzept der Business Objects *plus* im Detail vorgestellt.

3.3 Konzept der Business Objects *plus*

Ein Ziel des Konzepts ist es, eine semantische Basis für Datenmodelle in einem Unternehmen zu schaffen, und diese in unterschiedlichen Anwendungsbereichen zu nutzen.

Die Notwendigkeit ergibt sich neben den oben bereits geschilderten Problemen und Herausforderungen auch aus der Praxis und der Zusammenarbeit mit dem Kooperationspartner im Rahmen dieser Forschungsarbeit. Zusätzlich zu den bereits genannten Herausforderungen haben Unternehmen häufig insbesondere in der Zusammenarbeit mit Partnern und Zulieferern eine Vielzahl an Problemen zu lösen. Für jede solche Kooperation ist es notwendig Daten auszutauschen und die Semantik dieser Daten aufeinander abzubilden. Meist werden von unterschiedlichen Partnern auch unterschiedliche Softwaresysteme zur Speicherung und Verarbeitung von Daten eingesetzt. Um Begrifflichkeiten des einen Partners denen des Anderen zuordnen zu

können, wird derzeit häufig auf lange, manuell erstellte, (Text-)Listen mit Begriffspaaren gesetzt. Hinzu kommt, dass häufig nicht einmal die Systeme innerhalb einer Firma die gleichen Begriffe für semantisch identische Attribute einsetzen, was eine solche Aufgabe zusätzlich erschwert. Es muss also mit Datenmodellen gearbeitet werden, die nicht miteinander übereinstimmen und eine unklare Semantik haben. Den Datenaustausch zwischen unterschiedlichen Partnern bzw. Systemen zu realisieren, erfordert also einen enormen Aufwand für die Transformation von Nachrichten.

Um dieses Problem zu lösen, oder zumindest abzumildern, wurde im Rahmen dieser Arbeit das Konzept der Business Objects *plus*, kurz BO^+, entwickelt. BO^+ sind kleine, in sich abgeschlossene Datenmodelle und beschreiben fachliche Entitäten, die im gesamten Unternehmen an unterschiedlichen Stellen zum Einsatz kommen und in ihren eigenen Lebenszyklen verwaltet werden. Durch diese getrennte Verwaltung lässt sich eine Entkopplung der Datenmodelle von den Serviceschnittstellen erreichen.

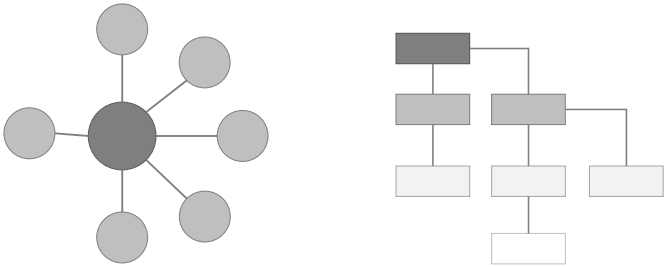
Ein BO^+ enthält eine abstrakte semantische Beschreibung eines Datenmodells und ist losgelöst von spezifischen Implementierungen. Da BO^+ auch in unterschiedlichen Anwendungsdomänen nutzbar sein sollen, muss es möglich sein, für ein BO^+ unterschiedliche Ausprägungen zu definieren. Dadurch ist es möglich die leicht unterschiedlichen Anforderungen der Anwendungsdomänen abzubilden. Dementsprechend wird eine logische Struktur benötigt, die es erlaubt die Attribute eines BO^+ für diese unterschiedlichen Domänen abzubilden.

In den folgenden Abschnitten werden mehrere Varianten zur Definition einer solchen Struktur vorgestellt und bewertet.

3.3.1 Logische Struktur der Business Objects *plus*

Ein BO^+ besteht aus einer Menge von Attributen. Unter Berücksichtigung der oben genannten Anforderungen sind unterschiedliche Varianten für die logische Struktur denkbar. In diesem Abschnitt werden vier Varianten vorgestellt: Zwei hierarchische Ansätze, sowie jeweils ein Sichten-basierter und ein typisierter Ansatz. Alle vier Varianten sind von klassischen Daten-

strukturen abgleitet, wie sie aus der objektorientierten Programmierung und Datenmodellierung bekannt sind.



(a) Erweiterbarer Kern (b) Hierarchisches Modell

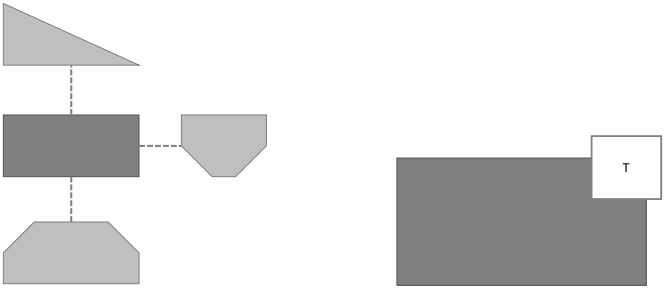
Abbildung 3.1: Business Object *plus*-Varianten, Teil 1 [KM17] ©2017 IEEE

3.3.1.1 Variante 1: Erweiterbarer Kern

Diese in Abbildung 3.1a gezeigte Variante besteht aus einem "Kern", der die grundlegenden Attribute enthält, die für alle Anwendungsdomänen gelten bzw. benötigt werden. Dieser Kern kann mit zusätzlichen, domänenspezifischen Attributen erweitert werden. Die Variante ähnelt dem Konzept der Spezialisierung aus der objektorientierten Programmierung.

3.3.1.2 Variante 2: Hierarchisches Modell

Diese Variante baut auf der Variante *erweiterbarer Kern* auf und führt zusätzliche Spezialisierungsstufen ein. So entsteht eine Hierarchie bzw. ein Baum von BO^+, wobei jedes BO^+ die Attribute der höheren Ebene erbt. Abbildung 3.1b zeigt grafische Darstellung der Struktur. Wie auch in der vorherigen Variante enthält die Wurzel des Baumes domänenübergreifende Attribute und mit jedem Hierarchielevel wird das BO^+ spezifischer für eine bestimmte Anwendungsdomäne oder einen bestimmten Anwendungsfall.



(c) Sichtenbasiertes Modell (d) Typisiertes Modell

Abbildung 3.1: Business Object *plus*-Varianten, Teil 2 [KM17] ©2017 IEEE

3.3.1.3 Variante 3: Sichtenbasiertes Modell

In dieser in Abbildung 3.1c gezeigten Variante wird ein zentrales Modell definiert, in dem alle Attribute, sowohl domänenübergreifend als auch domänenspezifisch, enthalten sind. Auf dieses Modell werden dann anwendungsfallspezifische Sichten definiert, die jeweils eine Teilmenge der Attribute beinhalten.

3.3.1.4 Variante 4: Typisiertes Modell

Wie beim Sichtenbasierten Modell, wird auch für diese Variante ein zentrales Modell definiert, in dem alle Attribute enthalten sind. Domänenspezifische BO^+ werden in dieser Variante gebildet, indem Untermengen der Attribute als domänenspezifisches BO^+ definiert werden. Ein spezielles Attribut gibt dabei den Typ des BO^+ an, alle nicht benötigten Attribute werden einfach leer bzw. weg gelassen. Abbildung 3.1d zeigt die Variante.

3.3.2 Vergleich und Bewertung der BO^+-Varianten

Im Folgenden soll eine Bewertung der vorgestellten BO^+-Varianten durchgeführt werden. Die Varianten werden dazu anhand von fünf Kategorien k_n, basierend auf den diskutierten Anforderungen, verglichen:

k_1: **Komplexität der Einführung:** Wie groß ist der notwendige Abstimmungsaufwand um ein neues BO^+ zu definieren und einzuführen? Abhängig von der logischen Struktur kann dieser Aufwand sich stark unterscheiden, beispielsweise in der Anzahl der miteinzubeziehenden Stakeholder.

k_2: **Abbildung auf die Unternehmenshierarchie:** Wie gut kann die logische Struktur der BO^+-Variante auf die Unternehmenshierarchie und damit unterschiedliche Anwendungsdomänen abgebildet werden?

k_3: **Abhängigkeiten:** Welche Abhängigkeiten existieren zwischen einem BO^+ und einem untergeordneten abgeleiteten BO^+? Abhängig von der Struktur kann die Bindung eines BO^+ und eines davon abgeleiteten BO^+ unterschiedlich stark ausgeprägt sein. Dies ist insbesondere von Bedeutung, wenn neue BO^+-Versionen entworfen werden und wirkt sich darauf aus, wie gut unterschiedliche Versionen des gleichen BO^+ gleichzeitig existieren können. Zudem sind BO^+ umso schwerer zu managen, je mehr Abhängigkeiten existieren. Bei stärkeren Abhängigkeiten sind zudem die Auswirkungen von Änderungen deutlich umfangreicher.

k_4: **Anomalien aufgrund Redundanz:** Ist die Definition von redundanten Attributen in abgeleiteten BO^+ möglich und wie werden diese dadurch beeinflusst? Identisch benannte, also redundante, Attribute können Mehrdeutigkeiten verursachen, wenn sie unabhängig voneinander, zum Beispiel für unterschiedliche Anwendungsbereiche, definiert werden, aber nicht die gleiche Semantik haben.

k_5: **Versionsübergänge:** Wie gut lässt sich der Übergang von einer BO^+-Version auf eine neue gestalten und welche Auswirkungen ergeben sich auf abgeleitete BO^+? Die Anzahl der möglichen abgeleiteten BO^+ hat hierauf einen direkten Einfluss. Je mehr Hierarchiestufen existieren, desto mehr BO^+ existieren und desto mehr Services müssen an eine neue BO^+-Version angepasst werden.

Tabelle 3.1 gibt einen Überblick über die Evaluations-Kategorien sowie über die Ergebnisse des Vergleichs der vier BO$^+$-Varianten. Jede Variante erhält eine Bewertung zwischen 0 (schlecht) und 3 (gut), abhängig davon wie gut das jeweilige Kriterium erfüllt wird. Die Summe dieser Werte wird dann mithilfe der folgenden Formel berechnet:

$$Summe_{v_n} = \frac{k_1}{2} + k_2 + k_3 + \frac{k_4}{2} + 2 \times k_5$$

Da ein BO$^+$ nur einmalig neu eingeführt wird, ist die Kategorie k_1 nur zur Hälfte gewichtet. Ebenso wird mit Kategorie k_4 verfahren, da Redundanz nicht zwangsläufig negativ zu bewerten ist. Korrekt eingesetzt ermöglicht sie eine bessere Strukturierung eines BO$^+$ für verschiedene Anwendungsfälle, Geschäftsprozesse oder Domänen. Kategorie k_5 fließt mit doppelter Gewichtung in die Gesamtbewertung ein, da Versionsänderungen sehr weitreichende Auswirkungen haben können. Sie treten sehr häufig auf, nämlich fast jedes Mal wenn eine Schnittstelle geändert werden muss. Zusätzlich wirken sie sich immer direkt auf alle Services und Consumer aus, die ein geändertes BO$^+$ einsetzen.

Tabelle 3.1: Vergleichsergebnisse der unterschiedlichen BO$^+$-Varianten

	1 Erw. Kern	**2 Hierarchisch**	**3 Sichtenbasiert**	**4 Typisiert**
k_1	2	3	0	0
k_2	2	3	1	1
k_3	2	1	2	3
k_4	1	2	3	3
k_5	2	2	2	1
Summe	**9,5**	**10,5**	**8,5**	**7,5**

3.3.2.1 Bewertung der Varianten

Die folgenden Abschnitte bewerten die vier BO$^+$-Varianten im Hinblick auf die Vergleichskriterien.

Komplexität der Einführung Der Einführungsprozess, der im folgenden Abschnitt 3.5 noch genauer vorgestellt wird, gestaltet sich für die Varianten unterschiedlich. BO^+ der Varianten 1 (Erweiterbarer Kern) und 2 (Hierarchisches Modell) ermöglichen es, dass eine kleine Anzahl an Personen, wie beispielsweise ein bereits existierendes Architekturgremium, ein Kern-BO^+ definieren. Untergeordnete BO^+ können dann in einem nächsten Schritt von den Architekturen der jeweiligen Geschäftsbereiche definiert werden. Für Variante 2 können weitere Hierarchiestufen durch die Abteilungs- oder Anwendungs-Architekten ausdetailiert werden. In den Varianten 3 (Sichtenbasiert) und 4 (Typisiert) bestehen BO^+ im Gegensatz dazu aus einer großen Menge an Attributen. Zur Definition dieser Attribute ist eine umfangreiche Koordination zwischen den Stakeholdern aller Hierarchieebenen notwendig. Dies verlängert den Prozess bis eine Einigung über die im BO^+ enthaltenen Attribute erzielt werden kann.

Abbildung auf die Unternehmenshierarchie Im Allgemeinen sind große Unternehmen in verschiedene Geschäftsbereiche untergliedert, die wiederum weiter unterteilt sind in Abteilungen und Teams. Daraus ergeben sich, je nach Unternehmensgröße, drei bis vier Hierarchieebenen. Meist gibt es auf allen Ebenen auch IT- bzw. Software-Architekten mit unterschiedlichen Verantwortlichkeiten. Auf der obersten Ebene legen Enterprise-Architekten die Architektur-Richtlinien für das Gesamtunternehmen fest. Auf Ebene der Geschäftsbereiche werden diese Richtlinien ergänzt und auf den Geschäftsbereich zugeschnitten. Darunter sind Anwendungs- oder Service-Architekten dafür zuständig die Richtlinien in konkrete Anwendungs- bzw. Service-Architekturen zu überführen.

Variante 1 ermöglicht die Abbildung der BO^+ auf zwei Hierarchiestufen. So kann beispielsweise der Kern die Unternehmensebene abdecken und die abgeleiteten BO^+ die darunterliegende Geschäftsbereichebene. Variante 2 ermöglicht eine Abbildung auf beliebige Hierarchien und kann so auch alle Hierachieebenen eines Unternehmens widerspiegeln. Varianten 3 und 4 sind nicht vollständig kompatibel mit einer solchen Hierachie-Struktur

sondern sind besser geeignet um einzelne Anwendungsfälle abzubilden. Mit Variante 3 ließen sich zwar Sichten für unterschiedliche Ebenen definieren, da diese aber nicht aufeinander aufbauen ist das Vorgehen nicht optimal geeignet.

Abhängigkeiten Varianten 1 und 2 verhalten sich für dieses Kriterium ähnlich. Ein abgeleitetes BO^+ erbt alle Attribute des Eltern-BO^+. Für Variante 2 sind die Abhängigkeiten gegebenenfalls stärker, da sie sich über mehrere Ebenen hinweg fortsetzen können. Abgeleitete BO^+ der Variante 3 haben ebenfalls Abhängigkeiten, allerdings sind diese auf die Menge der Attribute beschränkt, die in einer Sicht enthalten sind. Für Variante 4 existieren keine abgeleiteten BO^+, dementsprechend gibt es keine Abhängigkeiten, die negative Effekte haben können.

Anomalien aufgrund Redundanz Varianten 1 und 2 ermöglichen die Definition semantisch identischer, also redundanter, Attribute in abgeleiteten BO^+. Dadurch ist es allerdings auch möglich, dass gleich benannte Attribute definiert werden, die nicht semantisch identisch sind was wiederum zu den eingangs genannten Problemen führen kann. Variante 2 ermöglicht es, semantisch gleiche Attribute in einer zusätzlichen Hierarchiestufe zusammenzufassen. Dadurch wird die Redundanz verringert, gleichzeitig aber die Komplexität erhöht. In Variante 1 ließen sich solche Attribute nur dem Kern-BO^+ hinzufügen, was sich wiederum auf alle abgeleiteten BO^+ auswirkt und damit einen noch höheren Aufwand für die Restrukturierung auslöst. Im Gegensatz dazu sind Varianten 3 und 4 komplett frei von Redundanz, da in beiden Fällen alle Attribute im zentralen Modell enthalten sind.

Versionsübergänge Bei den Varianten 1 und 2 wirken sich Änderungen an einem BO^+ direkt auf alle abgeleiteten Objekte aus, die dann auf eine neue Version des Eltern-BO^+ aktualisiert werden müssen, sobald die alte Version außer Betrieb genommen wird. Für Variante 3 sind die Auswirkungen abhängig von der Art der durchzuführenden Änderung. Ein neu hinzugefügtes

Attribut hat zunächst keinerlei Auswirkungen auf existierende Sichten, wohingegen geänderte oder entfernte Attribute alle Sichten betreffen, in denen die entsprechenden Attribute enthalten sind. Änderungen eines BO^+ der Variante 4 betreffen immer das komplette BO^+, da dieses unabhängig vom Typ immer alle Attribute enthält. Für alle Varianten gilt, dass je größer die Anzahl der miteinzubeziehenden Stakeholder ist, desto größer wird auch der zusätzliche Kommunikations-Overhead. Ebenso hat jede Änderung immer direkte Auswirkungen auf alle Services die ein geändertes BO^+ nutzen.

3.3.2.2 Bewertungsergebnis

Aus der Evaluation in den vorherigen Abschnitten und den Vergleichsergebnissen in Tabelle 3.1 ergibt sich, dass das hierarchische BO^+-Modell die geschilderten Anforderungen am besten abdeckt und erfüllt. Es ist geeignet für den Einsatz in einem großen Unternehmen mit domänenübergreifendem Datenaustausch. Um zu verhindern, dass die BO^+-Hierarchie zu komplex und damit schwer wartbar wird, sollte die Anzahl der Hierarchiestufen auf eine angemessene Zahl begrenzt werden. Es bietet sich an, sich dabei an der Unternehmenshierarchie zu orientieren. Hauptnachteil dieser Variante sind Anomalien die durch redundante Attribute entstehen können und von der hierarchischen Struktur begünstigt werden. Dies wird bereits durch die Begrenzung der Hierarchiestufen eingeschränkt. Zusätzlich können Prozesse etabliert werden, die beispielsweise durch Reviews sicherstellen, dass redundante Attribute auch semantisch identisch sind und gegebenenfalls auf höheren Hierarchiestufen zusammengezogen werden.

Für Firmen mit flachen Hierarchien oder spezielle Anwendungsfälle mit einer eingeschränkten Anzahl an benötigten Hierarchiestufen, könnte alternativ auch Variante 1 zum Einsatz kommen. Variante 4 ist grundsätzlich nicht optimal geeignet, insbesondere durch den enormen Abstimmungsaufwand in der Entwurfsphase eines BO^+.

3.3.3 Detaillierungsgrade für Business Objects *plus*

Ein BO^+ wird zu unterschiedlichen Zeitpunkten in seinen Lebenszyklus auch mit unterschiedlichen Detaillierungsgraden dargestellt. In frühen Stadien wird das BO^+ zunächst nur konzeptuell fachlich beschrieben, beispielsweise als Zeichnung oder Attributliste auf einem Flipchart. Während es in seinem Lebenszyklus fortschreitet, wird diese Darstellung immer konkreter und nähert sich einer IT-Darstellung an. Abbildung 3.2 zeigt, wie hier klassische Stufen der Datenmodellierung zum Einsatz kommen können.

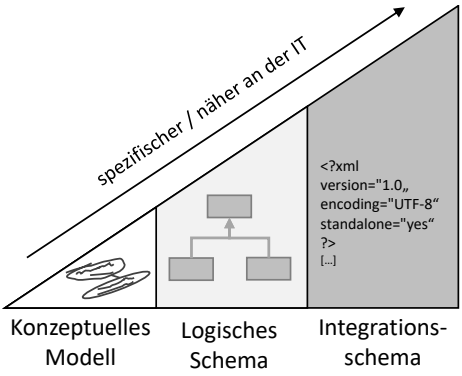


Abbildung 3.2: Detaillierungsgrade eines BO^+ in Anlehnung an [KM17] ©2017 IEEE

Konzeptuelles Modell Das konzeptuelle Modell ist losgelöst von formalen Beschreibungssprachen. In frühen Phasen des Entwurfsprozesses werden hier die Kernaspekte eines BO^+ festgelegt. Durch die Loslösung von formalen Beschreibungen eignet sich dieses Modell sehr gut als Basis für Diskussionen auf fachlicher Ebene.

Logisches Schema Das logische Schema enthält eine formale Beschreibung eines BO^+. Hier können grafische Modellierungssprachen wie beispielsweise die Unified Modeling Language (UML) [Obj17] zum Einsatz kommen.

Das Schema kann in späten Phasen des Entwurfsprozesses genauso eingesetzt werden, wie es in späteren Lebenszyklusphasen als Entwurfs-Tool und Vorstufe für ein Integrationsschema genutzt werden kann. Ebenfalls herangezogen werden kann es für die Definition neuer Versionen oder abgeleiteter BO^+.

Integrationsschema Das Integrationsschema stellt eine maschinenlesbare Beschreibung eines BO^+ dar, die direkt in Anwendungen verwendet werden kann. Zum Einsatz könnte hier beispielsweise ein XML-Schema-Dokument kommen. Abschnitt 3.6 stellt weitere Möglichkeiten zur Darstellung von BO^+ vor.

3.4 Governance-Prozesse und Dokumentationsanforderungen

Die Nutzung von BO^+ ist nicht möglich ohne die Etablierung organisatorischer Prozesse, die eine effektive Verwaltung der BO^+ ermöglichen. Um diese Prozesse zu unterstützen, ist es notwendig bestimmte Informationen zu dokumentieren. Dabei handelt es sich insbesondere um Metadaten wie beispielsweise Lebenszyklusstatus, Nutzungsbeziehungen oder Spezifikationsdokumente. Zudem sollten alle Prozesse in bereits existierende Enterprise-Architecture- und Governance-Prozesse eingebunden werden.

In den folgenden Abschnitten werden diese Prozesse und Dokumentationsanforderungen genauer vorgestellt.

3.4.1 Änderungsverwaltung

Im Normalfall werden BO^+ während ihrer Lebenszeit mehrere Änderungen durchlaufen. So kann es beispielsweise sein, dass aufgrund von Änderungen einer Serviceschnittstelle neue Attribute hinzugefügt werden müssen. Dazu muss ein existierendes BO^+ geändert werden oder ein neues, abgeleitetes, erstellt werden. Dementsprechend ist es wichtig solche Änderungen innerhalb eines definierten, systematischen und robusten Änderungsprozesses ablaufen zu lassen.

Im Falle der BO^+ müssen die folgenden Änderungsoperationen gehandhabt werden können:

I: Erstellung eines neuen BO^+

II: Erstellung eines neuen abgeleiteten BO^+

III: Änderung eines existierenden BO^+

IIIa: Hinzufügen neuer Attribute

IIIb: Entfernung nicht mehr benötigter Attribute

IIIc: Änderung eines existierenden Attributs (bspw. Umbenennung, Datentypänderung, Änderung von Einschränkungen)

Die aufgelisteten Änderungsoperationen haben unterschiedliche Ergebnisse: *I* und *II* resultieren in einem von Grund auf neu definierten BO^+. *IIIa-IIIc* machen die Änderung eines existierenden BO^+ erforderlich. Da eine Änderung eines BO^+ allerdings existierende Serviceschnittstellen betrifft ist es notwendig, dass im Rahmen einer solchen Änderung eine neue BO^+-Version entsteht. Diese unterscheidet sich dann von der Alten nur in den durchgeführten Änderungen. Neue Services nutzen dann direkt das neue BO^+, wohingegen existierende Services noch für eine definierte Übergangszeit das alte BO^+ einsetzen können.

3.4.1.1 Versionierungskonzept

Eine systematische und robuste Versionierung von BO^+ erfordert ein zuverlässiges Versionierungsschema. Software-Entwickler stützen sich heute oft auf ein numerisches Schema nach dem Muster *MAJOR.MINOR.PATCH*. Mit jeder neuen Version wird dann die entsprechende Major-, Minor- oder Patch-Ziffer erhöht. Die *Semantic Versioning*-Spezifikation (SemVer, [Pre13]) formalisiert dieses Schema. In der Spezifikation werden insbesondere Versionierungs-Richtlinien für öffentliche Application Programming Interfaces (APIs) beschrieben, womit SemVer optimal geeignet ist, um BO^+ zu versionieren.

Die Spezifikation enthält ebenfalls Regeln dazu, wann die Major-, Minor- und Patch-Ziffern erhöht werden müssen. Für die oben aufgelisteten Änderungsoptionen ergeben sich dabei die in Tabelle 3.2 gezeigten Änderungen der Versionsnummern. Großbuchstaben geben in der Tabelle die zu erhöhende Ziffer an. Da Operation *IIIa* nur neue Attribute hinzufügt und damit keinen *breaking change* darstellt, wird nur die Minor-Stelle angehoben. Bei einem *breaking change* handelt es sich um eine Änderung, die dazu führt, dass vorherige Softwareversionen eine Schnittstelle nicht mehr ohne Anpassung nutzen können. Daher führen auch alle anderen Operationen immer zwangsläufig zu neuen Major-Versionen.

Tabelle 3.2: Auswirkungen von Änderungen auf die Versionierung von BO^+

Operation	**Auswirkung**	
I	**1**.0.0	Initiale Erstellung
II	**1**.0.0	Initiale Erstellung
IIIa	x.**Y**.0	Erhöhung der Minor-Version
IIIb	**X**.y.0	Erhöhung der Major-Version
IIIc	**X**.y.0	Erhöhung der Major-Version

Für BO^+ werden Patch-Versionen nicht benötigt, da durch sie nur rückwärtskompatible interne Änderungen einer Anwendung dargestellt werden. Diese sind hier allerdings ohne Bedeutung, da BO^+ losgelöst von spezifischen Anwendungen verwaltet werden. Die konsistente Nutzung des Versionierungsschemas ermöglicht es Nutzern der BO^+ sowie Entwicklern mit einem Blick auf die Versionsnummer zu erkennen, welche Art der Änderung der Auslöser für den Versionssprung war.

3.4.1.2 Rückwärtskompatibilität

Wie bereits erwähnt erfordert jede Änderung eines BO^+ die Erstellung einer neuen Version. Für *breaking changes*, also wenn die Major-Version angehoben wird, müssen alle Services, in denen die vorherige Version eingesetzt wird ihre Schnittstellen an die Änderung anpassen. Ebenso müssen auch alle Nutzer eines dieser Services ihre Service-Proxys anpassen. Solche Änderun-

gen sind zum einen mit enormem Aufwand verbunden, wodurch es sehr wichtig ist, Änderungen langfristig zu planen und früh zu kommunizieren. Da selbst dann die Änderung aller Abhängigkeiten einige Zeit benötigen kann, ist es notwendig eine Rückwärtskompatibilität der Schnittstellen zu schaffen. Dies wird erreicht, indem alte BO^+ noch für einen definierten Übergangszeitraum parallel zu den neuen BO^+ in Schnittstellen verwendet werden dürfen. In diesem Zeitraum müssen Anwendungen also mehrere Schnittstellen parallel betreiben und Anfragen an beide korrekt beantworten. Alternativ kann eine Middleware-Komponente wie beispielsweise ein Enterprise Service Bus über eine zusätzliche Abstraktionsebene die Bereitstellung der Rückwärtskompatibilität übernehmen.

3.4.2 Lebenszyklusverwaltung

Zum effektiven Umgang mit BO^+-Versionen, ihren Abhängigkeiten und ihrem Gesamtzustand, müssen diese in ihrem eigenen Lebenszyklus verwaltet werden. Dieser Lebenszyklus ist losgelöst vom Lebenszykus der Schnittstellen, in denen ein BO^+ eingesetzt wird.

Abbildung 3.3 zeigt den 6-stufigen Zyklus, den ein BO^+ im Laufe seines Lebens durchläuft. Er sollte eingebunden sein in das Gesamtarchitekturmanagement, um sicherstellen zu können, dass bestimmte Statusübergänge erst nach Review und Freigabe durch ein Architektur-Gremium erfolgen. Notwendige Reviews sind in Abbildung 3.3 mit einem Häkchen dargestellt.

Zusätzlich zur Verwaltung des Lebenszyklusstatus in einem IT-System kann dieser auch als alphanumerischer Identifikator an die Versionsnummer eines BO^+ angefügt werden. Eine Version die sich derzeit in der Phase *Entwurf* befindet, wird dann beispielsweise als *2.5.0-entwurf* beschrieben.

Um die nahende Außerbetriebnahme einer alten BO^+-Version anzukündigen, wird diese in den Lebenszyklusstatus *deprecated* überführt und ein Datum für die Stilllegung wird festgelegt. Unter Zuhilfenahme definierter Governance-Prozesse wird dann sichergestellt, dass alle Services, in denen die veraltete Version noch genutzt wird, bis zur Deadline auf die neue Version umgestellt haben.

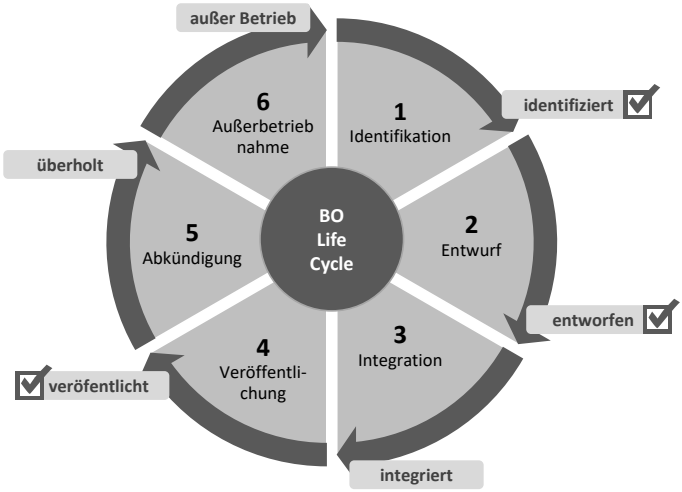


Abbildung 3.3: Lebenszyklus eines BO$^+$ in Anlehnung an [KM17] ©2017 IEEE

3.4.3 Stakeholder-Verwaltung

Neben den BO$^+$ und ihren Versionen ist es zudem wichtig, dass Metadaten über die beteiligten Stakeholder dokumentiert werden. Dabei handelt es sich beispielsweise um Personen, die für ein bestimmtes BO$^+$ hauptverantwortlich sind, sogenannte BO$^+$-Eigner. Ebenso wichtig ist es, Ansprechpartner für technische Fragen zu dokumentieren. Zu beachten ist, dass ein BO$^+$-Eigner nicht zwangsläufig eine Person sein muss, es könnte auch das Architektur-Board einer Abteilung als Gremium gemeinsam über ein BO$^+$ entscheiden. Entsprechende Personen und Gremien sind dann Ansprechpartner für geplante Änderungen und sind dafür zuständig die notwendigen Änderungsprozesse anzustoßen.

3.4.4 Dokumentation

Eine aktuelle Dokumentation der BO$^+$ und ihrer Metadaten wie beispielsweise Lebenszyklus oder Stakeholder ist wichtig und notwendig, um die

Aktivitäten innerhalb der vorgestellten Governance-Prozesse effizient durchführen zu können. Ohne diese Dokumentation lassen sich die gesteckten Ziele hinsichtlich Wiederverwendbarkeit und semantischer Eindeutigkeit nicht erreichen. Eine vollständige Dokumentation muss auch die Abhängigkeiten zwischen BO^+ und den Services in deren Schnittstellen sie zum Einsatz kommen berücksichtigen. Dadurch können Architekten und Entwickler einen Überblick über alle BO^+ und ihre Nutzung behalten. Spezifikationsdokumente und grafische Darstellungen der BO^+ aus frühen Entwurfsphasen sollten ebenso dokumentiert werden und können auch später noch hilfreich für die Abstimmungen zwischen IT und Fachbereich sein.

Die folgenden Abschnitte stellen zusätzliche Metadaten vor. Diese können ebenfalls dokumentiert werden und schaffen zusätzlichen Nutzen.

Sicherheits-Klassifizierung In Firmen werden Daten und die Systeme, über die auf diese Daten zugegriffen werden kann, häufig mittels Sicherheitsstufen klassifiziert. Eine mögliche Klassifizierung ist die Unterscheidung in *öffentlich*, *intern*, *vertraulich* und *geheim*, angelehnt an DIN 66399. Für jedes Level können dann entsprechende Sicherheitsrichtlinien festgelegt werden. Je nach Position und Aufgabenbereich erhalten Mitarbeiter dann Zugriff auf die Daten, beziehungsweise Systeme. Wenn BO^+ definiert werden, die Daten einer bestimmten Sicherheitsstufe repräsentieren, sollte dies dokumentiert werden. Dadurch lässt sich zum Beispiel die Verwendung solcher BO^+ auf Services und Anwendungen beschränken, die der gleichen Sicherheitsstufe unterliegen.

Persönliche Daten Persönliche Daten wie Namen und Adressen von Kunden oder Kreditkartendaten unterliegen strengen Datenschutzvorschriften. Entsprechend ist es notwendig solche Daten sicher zu speichern und den Zugriff darauf einzuschränken. Dementsprechend muss auch dokumentiert werden, wenn BO^+ solche Daten repräsentieren können. Dadurch ergibt sich zusätzlich noch der Vorteil, dass Services und Prozesse, die persönliche Daten verarbeiten, leicht identifiziert werden können.

Archivierung In vielen Ländern sind Unternehmen gesetzlich verpflichtet gewissen Dokumentationspflichten nachzukommen. Dokumentiert werden müssen beispielsweise Produktionsprozesse, Geschäftsbeziehungen oder die Lieferkette. Um diesen Verpflichtungen leichter nachkommen zu können, ist es hilfreich für ein BO^+ zu definieren ob es Daten repräsentiert, die der Archivierungspflicht unterliegen. Service- und Applikations-Entwickler können diese Information dann nutzen um notwendige Archivierungsfunktionalität in ihren Anwendungen vorzusehen.

Zur Dokumentation der BO^+ und ihrer Metadaten bietet sich die Verwendung eines zentralen Informationssystems, wie das im Rahmen dieser Arbeit entworfene SOA Governance Repository (SGR), an.

3.5 Einführung von BO^+

Die Einführung von BO^+ in eine Firma oder Organisation ist keine leichte Aufgabe. Wie sich bereits bei der Einführung von SOA gezeigt hat, erfordern große Architekturänderungen gute Vorbereitung. Ebenfalls ist es erforderlich, dass Entscheider und Mitarbeiter vorab von der Notwendigkeit und den Vorteilen der Veränderung überzeugt werden. Andernfalls lassen sich die erhofften Vorteile nicht erzielen [KAS09]. Dementsprechend kann ein klar strukturierter Prozess, der hilft die positiven Aspekte der Änderung hervorzuheben, zur Überzeugung von Skeptikern beitragen.

Abbildung 3.4 zeigt den Prozess zur Einführung neuer BO^+ in ein Unternehmen. In jedem Prozessschritt werden dabei zugehörige Informationen und Metadaten dokumentiert.

Der Prozess besteht aus folgenden Schritten:

1. Zunächst müssen geeignete Datenobjekte als BO^+-Kandidaten ausgewählt werden.
2. Im Anschluss werden die Serviceeigner oder ihre technischen Vertreter, die von der Einführung des BO^+ betroffen sind, in den Entwurfsprozess mit einbezogen.

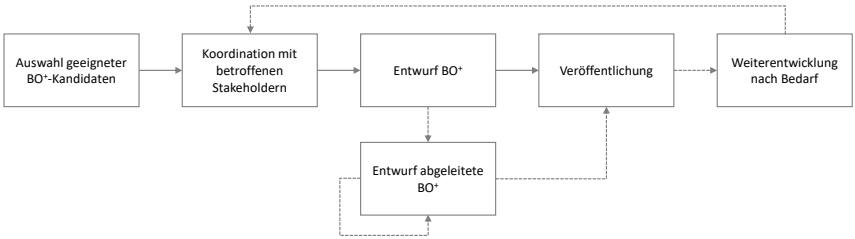


Abbildung 3.4: Prozess zur Einführung neuer BO^+ in ein Unternehmen in Anlehnung an [KM17] ©2017 IEEE

3. Nachdem ein BO^+ definiert wurde, können (sofern benötigt) abgeleitete BO^+ von Domänen-, Abteilungs- oder Anwendungs-Architekten definiert werden.
4. Im Anschluss werden die neu definierten BO^+ veröffentlicht und können in Schnittstellen und Anwendungen eingesetzt werden.
5. Während der Nutzungsphase werden BO^+ dann weiterentwickelt wodurch neue BO^+-Versionen entstehen.

Die umfangreichsten Schritte des Prozesses sind die Bewertung von BO^+-Kandidaten (Schritt 1) und die Koordination zwischen Stakeholdern (Schritt 2). Diese werden in den folgenden Abschnitten daher im Detail beleuchtet.

3.5.1 Bewertung von BO^+-Kandidaten

Wie die Nachteile und Fehlschläge von unternehmensweiten Datenmodellen bereits eindrücklich demonstriert haben (vgl. Abschnitt 3.2), ist es nicht zielführend alle Datenmodelle eines Unternehmens zu regulieren und zusammenzufassen. Daher ist es für die effiziente Nutzung von BO^+ notwendig, vorab zu beurteilen, ob ein Datenmodell tatsächlich dazu geeignet ist unternehmensweit als BO^+ standardisiert zu werden. Solche Modelle werden zunächst als BO^+-Kandidaten bezeichnet. Im Folgenden wird eine Methode zur Bewertung dieser Kandidaten vorgestellt, die Architekten dabei helfen soll zu entscheiden, ob ein Datenmodell sich tatsächlich als BO^+ eignet.

Dazu wird eine Bewertung anhand eines Kriterienkatalogs durchgeführt. Aus diesem ergeben sich zwei Werte $x_{suitability}$ und $y_{reuse\_potential}$. Diese dienen als Koordinaten und positionieren den BO$^+$-Kandidaten in einem der Quadranten des in Abbildung 3.5 gezeigten Gitters.

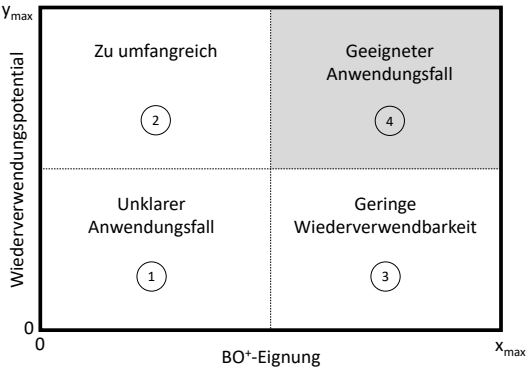


Abbildung 3.5: Bewertung von BO$^+$-Kandidaten in Anlehnung an [KM17] ©2017 IEEE

Je weiter ein Kandidat in der rechten oberen Ecke positioniert ist, desto besser ist er als BO$^+$ geeignet. Die Tabellen 3.3 und 3.4 zeigen den Kriterienkatalog. Jedem Kriterium $c_n \in \mathbb{N}$ wird ein numerischer Wert zwischen 1 (niedrig) und 3 (hoch) bzw. 1 und 2 zugewiesen. Zusätzlich wird ein Gewichtungskoeffizient $w_n \in \mathbb{Q}$ zwischen 1 und 2 vergeben. Dieser erlaubt es Architekten bestimmte Kriterien stärker als andere zu bewerten und damit den Kriterienkatalog auf die jeweiligen Anforderungen des Unternehmens maßzuschneidern. Die Eignung eines Kandidaten lässt sich dann folgendermaßen berechnen:

$$BO^+Capability = (x_{suitability}, y_{reuse\_potential})$$

$$mit$$

$$x_{suitability} = \sum_{n=1}^{4}(c_n \times w_n) \quad und \quad y_{reuse\_potential} = \sum_{n=5}^{7}(c_n \times w_n)$$

Tabelle 3.3: BO$^+$-Eignung

Kriterium	Wertebereich
c_1: Klar abgegrenzter Umfang	1..3
c_2: Angemessene Anzahl an Attributen (≤ 15)	1..3
c_3: Eindeutige Semantik	1..3
c_4: Beschreibt ein reales Objekt der Anwendungsdomäne	0..1

Tabelle 3.4: BO$^+$ Wiederverwendungspotential

Kriterium	Wertebereich
c_5: (Geplanter) Einsatz in mehreren Geschäftsprozessen	1..3
c_6: (Geplanter) Einsatz in mehreren Services	1..3
c_7: Nutzung über Domänengrenzen hinweg möglich	0..1

Ein Kandidat ist als BO$^+$ geeignet, wenn seine $BO^+Capability = (x, y)$ $mit\ x \geq \frac{x_{max}}{2}\ und\ y \geq \frac{y_{max}}{2}$, also in Quadrant 4 verortet ist. x_{max} und y_{max} sind abhängig von den zugewiesenen Gewichtungskoeffizienten, da dadurch die maximale Punkteanzahl bestimmt wird. BO$^+$-Kandidaten, die in Quadrant 1 verortet werden, können nicht klar bestimmten Anwendungsfällen zugeordnet werden, was es schwierig macht die Semantik der Attribute festzulegen. Für Kandidaten die in Quadrant 2 verortet sind gilt das gleiche, allerdings gibt es für sie einen höheren Wiederverwendungswert, was nach eingehender Prüfung die Nutzung in wenigen eingeschränkten Anwendungsfällen ermöglichen könnte. Kandidaten in Quadrant 3 wiederum bieten einen abgegrenzten Umfang und eine klare Semantik, allerdings ergeben sich für sie keine ausreichend hohe Wiederverwendungsmöglichkeit im Unternehmen. Dadurch ist der Aufwand zur Definition eines BO$^+$ häufig nicht gerechtfertigt. Abbildung 3.6 in Abschnitt 3.6 zeigt beispielhaft die Definition eines geeigneten BO$^+$ auf Basis eines existierenden Datenmodells.

3.5.2 Koordination zwischen Stakeholdern

Um den bereits erwähnten Bedenken der Mitarbeiter bezüglich großer Architekturänderungen begegnen zu können, ist insbesondere zu Anfang ein umfangreicher Koordinations-, Moderations- und Mediations-Prozess notwendig, um Kritiker zu überzeugen. In diesen Prozess müssen insbesondere die Domänen- und Serviceeigner, die später für die BO^+ verantwortlich sind, mit einbezogen werden. Zur Einführung ist es ratsam das Konzept in wenigen Leuchtturm-Projekten zu erproben und so die Akzeptanz der Mitarbeiter zu fördern.

Sobald BO^+ im Unternehmen etabliert sind, werden Reibungspunkte hauptsächlich entstehen, wenn Stakeholder unterschiedlicher Abteilungen oder Geschäftsbereiche in Abstimmungsprozesse involviert sind. Im Allgemeinen hat jeder dieser Stakeholder eine eigene Agenda und möchte diese durchsetzen. Durch die hierarchische Struktur der BO^+ lassen sich solche Auseinandersetzungen allerdings deutlich reduzieren, da jeder Geschäftsbereich und jede Abteilung selbst abgeleitete BO^+ definieren kann. Dadurch haben sie die Freiheit, BO^+ besser an die Anforderungen ihres jeweiligen Bereiches anzupassen.

Unabhängig davon ist es wichtig die Personen mit einzubeziehen, die tatsächlich mit den BO^+ arbeiten und nicht fertig definierte BO^+ von oben in die Unternehmenshierarchie zu pressen.

3.6 Technische Umsetzung

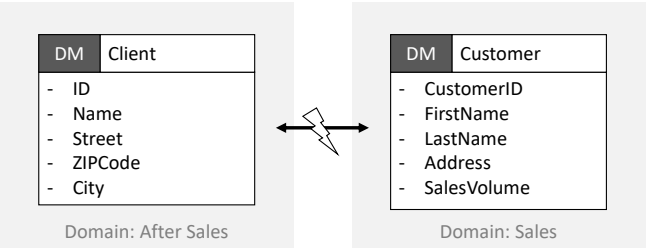
Wie bereits erwähnt, ist es notwendig, BO^+ und ihre Metadaten zu dokumentieren, um darauf aufbauend eine Wiederverwendung der BO^+ in verschiedensten Anwendungsfällen zu ermöglichen. In einem geeigneten System wie dem SGR werden dann die logischen Schemata der BO^+ gespeichert. Das SGR setzt dazu auf eine Speicherung mittels semantischer Technologien, wie dem Ressource Description Framework (RDF) [W3C14a] in einem Tripel-Speicher. Dadurch lässt sich die semantische Beschreibung eines BO^+ implementierungsunabhängig beschreiben. Eine Speicherung mittels semantischer

Technologien hat zudem den Vorteil, dass diese auf existierende semantische Vokabulare und Ontologien wie Friend-of-a-Friend (FOAF, [BM14]) oder das Dublin Core Metadata Element Set [Int07] aufbauen können. Dadurch erhöht sich der Grad der semantischen Beschreibung und auch eine Integration mit anderen Produktlebenszyklustools, die diese Vokabulare bereits einsetzen, wird vereinfacht. Tools, welche die RDF-basierten *Open Services for Lifecycle Collaboration* Spezifikationen [OAS18] umsetzen, kommen beispielsweise hierbei in Frage. Aus dieser implementierungsunabhängigen Beschreibung eines BO^+ lassen sich dann Integrationsschemata für verschiedene Programmiersprachen erzeugen, beispielsweise:

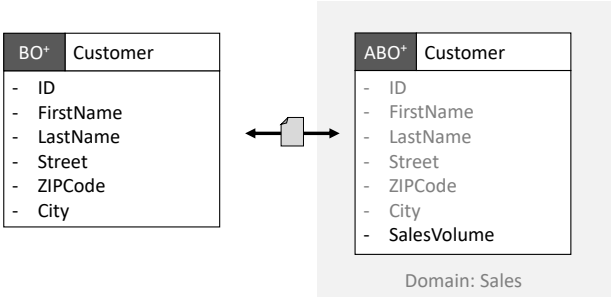
- XML-Schemabeschreibungen für die Nutzung in WSDL-Servicebeschreibungen,
- JSON-Repräsentationen für die Nutzung mit REST-basierten Services,
- „Plain Old Java Objects“ oder „Plain Old CLR Objects“ (POJO, POCO).

Solche generierten Integrationsschemata unterstützen auch Entwickler bei der Implementierung neuer Serviceschnittstellen oder der Entwicklung von Service-Proxys für den Zugriff auf existierende Services. Der schnelle Zugriff auf vorgefertigte Implementierungsartefakte kann zusätzlich helfen die Akzeptanz von BO^+ im Unternehmen zu steigern.

Abbildung 3.6a zeigt beispielhaft semantisch ähnliche Datenmodelle aus unterschiedlichen Domänen und im Vergleich dazu in Abbildung 3.6b eine Vereinheitlichung dieser Datenmodelle mittels BO^+. An diesem Beispiel ist leicht zu erkennen, dass der Datenaustausch zwischen zwei Systemen, die diese unterschiedlichen Datenmodelle in ihren Services aufgrund der unterschiedlichen Benennung semantisch gleicher Attribute (*Street, ZipCode, City* gegenüber *Address*) ohne eine Transformation zwischen den beiden Datenmodellen nicht möglich ist. Kommen dahingegen BO^+ zum Einsatz, ist dieser Datenaustausch möglich. Das BO^+ *Customer* vereinheitlicht dazu die gemeinsamen Attribute der beiden Datenmodelle, das abgeleitete BO^+ (ABO^+) wurde für die Domäne Sales um ein zusätzliches Attribut erweitert, das in anderen Domänen nicht benötigt wird.



(a) Datenmodelle ohne BO^+



(b) Vereinheitlichung durch BO^+ und abgeleitetes BO^+

Abbildung 3.6: Beispiel eines Business Object *plus* und Vergleich mit klassischem Datenmodell

Im einfachsten und häufigsten Fall findet die Kommunikation zwischen Systemen und Services innerhalb einer Abteilung oder zumindest einer Geschäftsdomäne statt. Hierbei ist der Einsatz von BO^+ unkompliziert: Die BO^+ werden als Teil der Service-Beschreibungen, also beispielsweise in der WSDL-Beschreibung, eingesetzt und Nachrichten zwischen Systemen und Services dementsprechend formatiert.

In den Fällen des Nachrichtenaustausches zwischen unterschiedlichen Hierarchieebenen einer Firma oder zwischen Firmen muss der Austausch leicht abgeändert gehandhabt werden. Abbildung 3.7 zeigt den Nachrichtenaustausch über Domänengrenzen hinweg. Die unterschiedlichen Formen repräsentieren dabei Mengen an Attributen; Pfeile zeigen die möglichen Kommunikationswege über Hierarchiegrenzen hinweg.

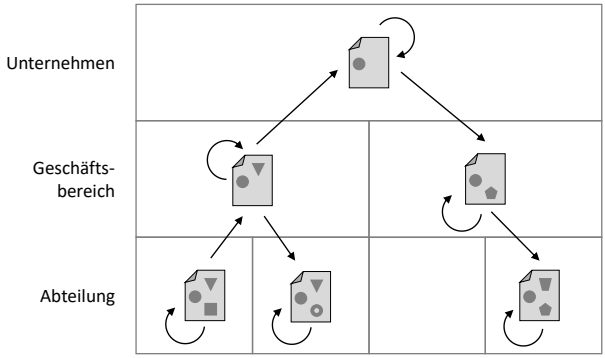


Abbildung 3.7: Domänenübergreifender Datenaustausch in Anlehnung an [KM17] ©2017 IEEE

Jedes BO$^+$ erbt dabei die Attribute höherer Hierarchiestufen und kann noch weitere Attribute spezifisch für den eigenen Anwendungsfall ergänzen.

Jedes BO$^+$ enthält eine echte Teilmenge der Attribute der von ihm abgeleiteten BO$^+$ ($BO^+_{parent} \subset BO^+_{child}$). Dies führt dazu, dass beim Nachrichtenaustausch zwischen Services auf unterschiedlichen Ebenen die Attribute, die nicht in der Teilmenge enthalten sind, weggelassen werden müssen, damit das Datenmodell noch „verstanden“ wird. Dies erfordert allerdings nicht wie bisher die manuelle Definition von Datenmodelltransformationen, sondern kann automatisch und generisch von einer Middleware-Komponente wie einem Enterprise Service Bus gehandhabt werden. Da die weggelassenen Attribute für die höhere Hierarchiestufe keine Bedeutung haben und dort nicht zum Einsatz kommen, hat dieses Weglassen keine negativen Nebeneffekte. Sollte sich später zeigen, dass bestimmte Attribute in mehreren Domänen benötigt werden, können diese auch nachträglich noch mittels der geschilderten Änderungsprozesse in neue Versionen der übergeordneten BO$^+$ aufgenommen werden.

3.7 Zusammenfassung und Ausblick

Die in diesem Kapitel vorgestellten Business Objects *plus* und die zugehörigen Management- und Governance-Prozesse können Firmen helfen eine semantische Basis für Serviceschnittstellen zu schaffen. Dadurch kann die Wiederverwendung auf Schnittstellenebene unterstützt und gefördert werden. Die semantische Basis hilft dabei inkompatible Datenformate zu eliminieren und vereinfacht oder erübrigt Datenmodelltransformationen zwischen Services. Durch die unterschiedlichen Detaillierungsgrade der BO^+ und ihrer Fachlichkeit können außerdem Kommunikations- und Verständnisprobleme zwischen IT- und Fachbereichen ausgeräumt werden. Aufgrund ihrer hierarchischen Struktur lassen sich BO^+ optimal in Umgebungen einsetzen, in denen Systeme und Informationen über Domänengrenzen hinweg integriert und ausgetauscht werden müssen.

Die Etablierung einer einheitlichen semantischen Basis innerhalb einer Firma erleichtert auch den Datenaustausch mit Partnerfirmen und Zulieferern. Transformationen zwischen den Firmen müssen nur einmalig für jedes BO^+ definiert werden und nicht für jeden Service separat. Zudem ist die Semantik der BO^+ auch eindeutig definiert, so dass es zu keinen Missverständnissen in der Bedeutung kommen kann. Insgesamt lassen sich dadurch die Aufwände für die Etablierung von Kooperationsprojekten mit anderen Firmen oder Zulieferern drastisch reduzieren.

Die Nutzung von Standardsoftware in einem SOA-Systemverbund erfordert weiterhin Transformationsaufwand. Allerdings ist dieser deutlich verringert, da die Schnittstelle nur mittels eines einzelnen Adapters in die firmeninterne Semantik übersetzt werden muss. Damit sind für alle weiteren Nachrichtenaustausche mit der Standardsoftware keine zusätzlichen Transformationen notwendig.

Der gemanagte Lebenszyklus der BO^+ hilft unautorisierte und ungeplante Änderungen an Serviceschnittstellen zu verhindern und verringert damit das Risiko von unerwarteten Service-Ausfällen. Zusätzlich erlaubt die Dokumentation der BO^+ und ihrer Abhängigkeiten die frühzeitige Kommunikation von Änderungen an alle Stakeholder. Dadurch können diese sich früher auf

bevorstehende Änderungen einstellen und entsprechend reagieren, um die eigenen Schnittstellen anzupassen.

BO^+ können zusätzlich dabei helfen Services zu identifizieren, die ähnliche Funktionen erfüllen. Wenn Services gleiche oder ähnliche Kombinationen von BO^+ nutzen, kann dies ein Indikator dafür sein, dass diese Services ähnliche Funktionen anbieten. Die Services können dann unter Umständen zusammengefasst werden. Dadurch wird einerseits die Gesamtkomplexität des SOA-Systemverbunds reduziert und andererseits der Wiederverwendungsgrad der Services erhöht, da eine gewisse Funktionalität nur noch von einem Service angeboten wird.

Das BO^+-Konzept kann auch als Basis für eine feingranulare Zugriffskontrolle dienen. Kombiniert mit einem Ansatz wie *RestACL* [HS16], könnten BO^+ mit Richtlinien annotiert werden, die es erlauben würden, die Zugriffskontrolle aus den Anwendungen heraus auf eine Middleware-Schicht zu verlagern. Dadurch ließen sich effektiv Zugriffsrechte systemübergreifend an einzelne Nutzer oder Nutzergruppen vergeben.

Kapitel 4

SOA Governance Meta Model

Ein SOA-Verbund im Unternehmen umfasst eine Vielzahl an Bestandteilen. Dies beginnt bei den Services selber und geht über zugehörige Meta-Informationen bis hin zu den unterschiedlichen Stakeholdern. Das in diesem Kapitel vorgestellte *SOA Governance Meta Model* soll dazu dienen all diese Bestandteile miteinander in Beziehung zu setzen und somit eine Grundlage für eine umfangreiche und vollumfängliche Dokumentation aller SOA-Artefakte zu schaffen. Gleichzeitig bildet das Meta-Modell die logische Basis der weiteren Forschungsarbeiten, indem es eine einheitliche Grundlage und ein einheitliches Verständnis für alle an einer serviceorientierte Architektur (SOA) beteiligten Artefakte schafft. Inhalte dieses Kapitels wurden bereits in [KSM14] und [KM16] publiziert.

Im Folgenden werden in Abschnitt 4.1 zunächst die allgemeinen Anforderungen an eine SOA Governance definiert und existierende SOA Governance Frameworks vorgestellt, bewertet und verglichen. Daran anschließend wird in Abschnitt 4.2 das auf diesen Anforderungen aufgebaute *SOA Governance*

Meta Model vorgestellt. Abschnitt 4.3 untersucht existierende Ontologien und Vokabulare und stellt die *SOA Governance Ontology* vor, die basierend auf dem SOA Governance Meta Model die Grundlage für die Entwicklung semantischer SOA Governance Anwendungen legen soll. Eine Bewertung und Zusammenfassung der vorgestellten Konzepte schließt das Kapitel ab.

4.1 Anforderungen an Governance-Prozesse in Unternehmen

Eine umfangreiche Literaturanalyse, Gespräche mit Fachexperten und eigene Arbeit im Bereich der Unternehmens-SOA des Kooperationspartners bilden die Grundlage für die in diesem Abschnitt vorgestellten Anforderungen an Governance-Prozesse in Unternehmen.

4.1.1 Anforderungen

Weitere Aspekte, wie beispielsweise Risikobewertung, Domänenverwaltung oder Mitarbeiterfortbildung sind ebenfalls wichtige Bausteine einer SOA. Da diese aber häufig in Unternehmensweite Programme eingebunden und nicht speziell der SOA zuzuordnen sind, werden sie im Weiteren nicht als eigene Anforderungen genannt.

4.1.1.1 Lebenszyklusverwaltung

Die Lebenszyklusverwaltung der Services ist einer der wichtigsten Prozesse in der Governance einer SOA. Die Nachverfolgung eines Services mag zwar zunächst einfach klingen, wird aber komplex, sobald immer mehr Services ins Portfolio aufgenommen werden oder sobald mehrere Versionen desselben Services parallel in Betrieb sind. Dies ist beispielsweise notwendig, um eine Rückwärtskompatibilität für Consumer zu ermöglichen, die noch nicht auf die neue Serviceversion umgestellt haben. Die Verwaltung eines Services wird noch herausfordernder, wenn man auch die unterschiedlichen Umgebungen miteinbezieht, die Firmen häufig betreiben. Neben der Produktivumgebung, auf der Services im Produktiveinsatz laufen, gibt es häufig noch Test- und

Integrationsumgebungen, auf denen neue Versionen der Services entwickelt und erprobt werden. In diesen Umgebungen können dann auch beliebige Kombinationen von Services und Serviceversionen in unterschiedlichem Lebenszyklusstatus aktiv sein.

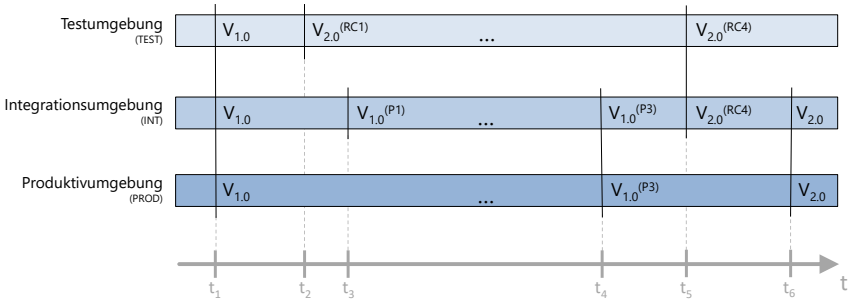


Abbildung 4.1: Lebenszyklusverwaltung über unterschiedliche Umgebungen hinweg

Abbildung 4.1 zeigt beispielhaft wie sich die Versionen eines Services über einen gewissen Zeitraum auf den unterschiedlichen Umgebungen entwickeln. So ist auf allen drei Umgebungen zum Zeitpunkt t_1 zunächst noch Version 1.0 des Services in Betrieb. Mit fortschreitender Entwicklung neuer Versionen werden diese auf den unterschiedlichen Umgebungen in Betrieb genommen. Dabei können auch unterschiedliche Versionsstränge unabhängig voneinander weiterentwickelt und in Betrieb genommen werden. Während also im Beispiel zum Zeitpunkt t_3 Version 2.0, Release Candidate 1 (RC1) auf der Testumgebung erprobt wird, wird gleichzeitig ein Patch (1.0P1) auf der Wartungsumgebung integriert, bevor er produktiv genommen wird. Vertikale Striche zeigen jeweils die Inbetriebnahme von Versionen auf einer neue Umgebung an. Der Parallelbetrieb von mehreren Serviceversionen auf der gleichen Umgebung ist hierbei nicht dargestellt.

Zwar ist es möglich die Services für jede der Umgebungen separat zu verwalten, allerdings erlaubt eine gemeinsame Verwaltung eine bessere Nutzung der zur Verfügung stehenden Informationen. Diese kann dann auch zur effektiveren Steuerung der Governance-Prozesse genutzt werden, indem

man mittels eines entsprechenden Governance-Tools Restriktionen einführt, die beispielsweise verhindern, dass ein Service auf der Produktivumgebung veröffentlicht wird, solange er nicht auf der Integrationsumgebung erprobt wurde.

Der Lebenszyklus eines Service selbst besteht im Allgemeinen aus mehreren Phasen, von der initialen Identifizierung bis zur finalen Außerbetriebnahme. Der in [KSM14] vorgestellte Lebenszyklus besteht aus den in Abbildung 4.2 gezeigten sieben Phasen.

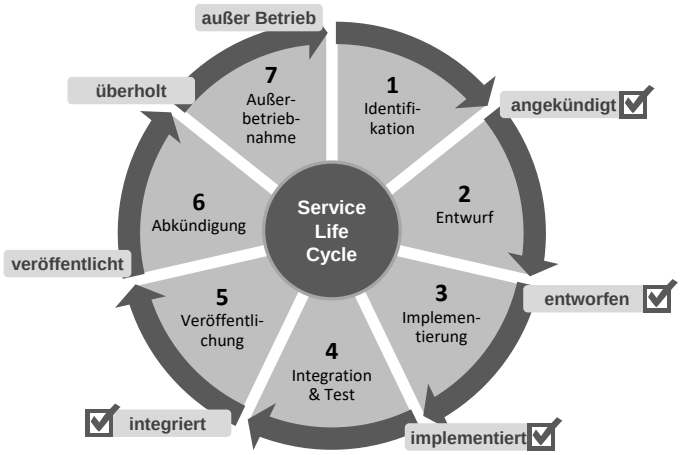


Abbildung 4.2: Service-Lebenszyklus in Anlehnung an [KSM14] ©2014 IEEE

In Phase 1 (*Identifikation*) werden die Anforderungen an einen neuen Service oder eine neue Serviceversion identifiziert. Es wird ebenfalls abgeglichen, ob diese Neuentwicklung an den Geschäftszielen ausgerichtet ist, bzw. ob schon Services mit einem ähnlichen Funktionsumfang existieren. In Phase 2 (*Entwurf*) wird der Service und seine Schnittstelle entsprechend den unternehmensinternen Architektur-Richtlinien spezifiziert. Nach einem

Architektur-Review wird der Service implementiert (Phase 3, *Implementierung*). Im Anschluss werden in Phase 4 (*Integration & Test*) die Integration und Tests des Services durchgeführt. Nach einem erfolgreichen Abschluss der Tests kann der Service veröffentlicht und produktiv eingesetzt werden (Phase 5, *Veröffentlichung*). Sobald der Service nicht mehr benötigt wird, weil eine neue Version entwickelt wurde oder seine Funktionalität obsolet ist, geht der Service in Phase 6 (*Abkündigung*) über. In dieser Übergangsphase werden alle noch aktiven Consumer über die baldige Abschaltung des Services informiert. Im Anschluss an diese Phase wird der Service endgültig außer Betrieb genommen (Phase 7, *Außerbetriebnahme*) und steht nicht mehr zur Verfügung.

4.1.1.2 Consumer-Verwaltung

Im Unternehmensumfeld wird meist nicht auf die bekannte „Selbstbedienung“ bei der Nutzung von Services gesetzt. Neue Consumer können zwar die angebotenen Services in einem Verzeichnis finden, allerdings ist zur Nutzung normalerweise eine Authentifizierung notwendig. Dazu müssen Service-Provider und Service-Consumer zunächst einen formalen Vertrag schließen, ein sogenanntes *Service Level Agreement*. Darin werden die Rechte und Pflichten der beiden Vertragsparteien festgehalten. Insbesondere geht es dabei um die Bedingungen unter denen der Consumer einen Service nutzen darf und unter denen der Provider ihn zur Verfügung stellt. Darin eingeschlossen sind Vergütungsmodelle (bspw. als Flatrate oder nutzungsabhängig) oder Nutzungsbeschränkungen, wie eine maximale Anzahl Anfragen pro Stunde, aber auch die Service-Verfügbarkeit und maximale Antwortzeiten. Die Consumer-Verwaltung schließt zudem auch die Nachverfolgung und Dokumentation aller Consumer und der von ihnen genutzten Services mit ein. Dies erlaubt beispielsweise die Stakeholder der Consumer gezielt zu informieren, wenn eine Serviceversion außer Dienst gestellt wird.

4.1.1.3 Organisationsstruktur und Stakeholder-Verwaltung

Wie bereits erwähnt besteht ein SOA-Verbund nicht nur aus den technischen Services, sondern ebenso aus einer Vielzahl an Personen, Prozessen und weiteren Technologien. Eine erfolgreiche SOA-Strategie muss daher auch Änderungen an den etablierten Team- und Rollenstrukturen mit sich bringen. Diese neuen Prozesse sind häufiger als in der Vergangenheit abteilungs- oder domänenübergreifend. Dadurch wird der Fokus von Applikationsdenken hin zu einer breiter gefassten Sicht gelenkt und kann dadurch schnell zu Spannungen innerhalb des Unternehmens führen. Daher ist es wichtig, Rollen und Zuständigkeiten klar zu definieren und innerhalb des Unternehmens entsprechend zu kommunizieren. Insbesondere eine klare Definition der Zuständigkeiten ist wichtig, da Stakeholder wissen müssen wer der Eigner eines Services ist und an wen sie sich im Falle von Problemen wenden müssen. Genauso wichtig ist es den Zweck von geplanten Strukturänderungen offensiv und transparent an die Mitarbeiter zu kommunizieren, da diese sich sonst häufig übergangen fühlen und sich gegen die Änderungen sperren.

Die Dokumentation dieser Personen und Zuständigkeiten ist im Rahmen der Governance-Prozesse ebenfalls wichtig und wird als Stakeholder-Verwaltung bezeichnet.

4.1.1.4 Portfolioverwaltung

Ziel der Portfolioverwaltung ist es, die Neu- und Weiterentwicklung von Services zu steuern. Insbesondere wird also darauf geachtet, dass nicht die „falschen“ Services entwickelt werden, also solche die entweder bereits existierende Funktionalität umsetzen oder nicht an den Geschäftszielen ausgerichtet sind. Um diesen Prozess umzusetzen, werden an den in Abbildung 4.2 hervorgehobenen Kontrollpunkten (Architektur-)Reviews durchgeführt. Dabei wird im Falle der Portfolioverwaltung sehr früh die Spezifikation dahingehend überprüft, ob es Überschneidungen mit bestehenden Services gibt und dann gegebenenfalls für eine Weiterentwicklung freigegeben.

4.1.1.5 Architekturstandards

Die Etablierung von Architekturstandards unterstützt Service-Architekten und -Programmierer bei der Entwicklung von Services. Dieser Prozess wird im Allgemeinen als Teil des Enterprise Architecture Management durchgeführt und umfasst beispielsweise die Definition von SOA-Patterns [Erl09], die Nutzung bestimmter Middleware-Komponenten, wie eines Enterprise Service Bus, und Richtlinien für Serviceschnittstellen. Der Einsatz einheitlicher Standards führt zu homogeneren Schnittstellen der Services und damit auch zu geringerem Aufwand bei der Anbindung von Services, da Entwickler sich nicht von Grund auf in für sie neue Schnittstellen eindenken müssen.

4.1.1.6 Governance-Hierarchie

SOA Governance ist kein alleinstehendes Konzept, sondern muss in alle anderen Governance-Konzepte mit eingebunden sein, insbesondere die IT- und Unternehmens-Governance. Die Literatur betrachtet die SOA Governance häufig als Teilmenge der IT Governance [RB08; Ben12]. Die Einschätzung wird in dieser Arbeit nicht geteilt. Da eine SOA auch über Organisationsgrenzen hinweg wirkt und direkt von der Geschäftstätigkeit eines Unternehmens beeinflusst wird, wird sie als alleinstehend betrachtet. Allerdings existieren enge Beziehungen sowohl zur IT- als auch zur Unternehmens-Governance.

4.1.1.7 Vergütungsmodelle

Mit der Bereitstellung von Services an andere Geschäftsbereiche eines Unternehmens oder sogar an Consumer außerhalb des Unternehmens, gewinnen auch Vergütungsmodelle für die Nutzung dieser Services an Bedeutung. Dies können beispielsweise nutzungsbasierte oder Flatrate-Modelle sein, die eine uneingeschränkte Nutzung eines Services zulassen. Das Vorgehen, auch unternehmensinterne Nutzer zur Kasse zu bitten hat den Vorteil, dass Entwicklungs- und Betriebskosten der Services auf alle Bereiche oder Costcenter eines Unternehmen verteilt werden können.

4.1.1.8 Service-Monitoring

Eine weitere wichtige Tätigkeit beim Betrieb einer SOA ist das Monitoring der Services. Mittels Monitoring-Mechanismen kann der Zustand von Services automatisiert überwacht werden. Sobald bestimmte Grenzwerte über- oder unterschritten werden, kann automatisch Alarm ausgelöst werden und der zuständige Betriebsverantwortliche informiert werden. Dies erlaubt eine schnelle Reaktion bei Ausfällen und Beeinträchtigen und damit möglichst kurze Ausfallzeiten. Da Services häufig sehr tief in Geschäftsprozesse integriert sind, kann der Ausfall eines scheinbar unwichtigen Services unter Umständen umfangreiche Auswirkungen haben.

4.1.1.9 Reifegradbewertung

Ein SOA-Verbund wird nicht einmalig aufgebaut und bleibt dann unverändert im Einsatz. Wie bereits erwähnt ist die Einführung einer SOA ein mehrstufiger Prozess, der regelmäßig überprüft und angepasst werden muss.

Reifegradmodelle können Unternehmen dabei helfen den aktuellen Stand einer Technologie- oder Architekturumsetzung zu bewerten und zeigen notwendige weitere Schritte auf. Dies gilt für eine SOA allgemein, wie auch für die SOA Governance.

4.1.1.10 Schnittstellenverwaltung

Ein Grundgedanke des SOA-Architekturparadigmas ist die Abstraktion von Funktionalität in Services sowie die Nutzung einer einheitlichen Standardisierungssprache zur Beschreibung ihrer Schnittstellen. Ein Ziel der SOA Governance sollte es zudem sein, diese Schnittstellen und insbesondere die Änderungen daran zu verwalten und zu steuern. Diese Verwaltung muss getrennt vom Lebenszyklus des eigentlichen Service erfolgen. Das in Kapitel 3 vorgestellte Konzept der Business Objects *plus* unterstützt dieses Vorgehen und erlaubt Unternehmen die Entwicklung domänenübergreifender Datenobjekte die in unterschiedlichen Services wiederverwendet werden können.

4.1.2 Bewertung existierender Ansätze

In den letzten Jahren wurde verschiedene Frameworks für SOA Governance definiert und veröffentlicht. Diese unterscheiden sich teilweise stark im Fokus und werden daher in diesem Abschnitt kurz vorgestellt und einem Vergleich hinsichtlich der im vorherigen Abschnitt vorgestellten Anforderungen unterzogen.

Kohnke et al. [KSH08] beschreiben acht Handlungsfelder der SOA Governance, aufgeteilt in die drei Bereiche *Strukturen*, *Prozesse* und *Mitarbeiter*. Die Betrachtung erfolgt dabei deutlich aus einer Beratersicht, die Autoren sind Mitarbeiter von SAP, mit Fokus auf Prozesse aber auch auf die Einbindung der Mitarbeiter. Konzepte werden dabei nur kurz genannt und kaum im Detail vorgestellt.

Niemann et al. [NERS08] stellen ein zweiteiliges SOA Governance-Modell vor. Dieses besteht aus einem Steuerungszyklus, der einen Überblick über SOA Governance-Prozesse auf einer hohen Ebene liefern soll und die Einführung und Verbesserung einer SOA steuern soll. Der zweite Teil, das SOA Governance Operational Model, soll die Erfüllung von vorab definierten SOA-Zielen unterstützen. Dazu werden Beziehungen zwischen verschiedenen Governance-Mechanismen (unter anderen Metriken, Richtlinien, Best Practices) beschrieben die der Zielerreichung dienlich sein können.

Rieger und Bruns [RB08] fokussieren sich in ihrem Beitrag stark auf die Organisationsstruktur und beschreiben ein umfangreiches Rollenmodell. Weitere Aspekte der SOA Governance werden als wichtig erwähnt, allerdings nicht im Detail beschrieben. Als einziges Framework im Vergleich wird hier, wenn auch nur kurz, auf die Schnittstellenverwaltung und Geschäftsobjekte eingegangen.

Oracle [Ram13; Ben12; HD13] beschreibt in mehreren Publikationen ein umfangreiches SOA Governance Model, bestehend aus fünf von einander abhängigen Bereichen. Für die meisten der Bereiche stellt das Framework detaillierte Konzepte vor. Zusätzlich definiert Oracle das *SOA Maturity Model* zur Bewertung des Reifegrades einer SOA. Eine separate Schnittstellenverwaltung ist aber auch in diesem Framework nicht vorgesehen.

HP und Systinet [Sys06; Hew08] beschreiben sehr oberflächlich verschiedene Aspekte der SOA Governance mit einem Framework, das sich auf vier Schlüsselaspekte der SOA Governance stützt sowie in Form von Best Practices, mit dem Ziel der besseren Vermarktung des Produkts *HP SOA Systinet*.

IBM [HPG06; BMT06] definiert ein SOA Governance Framework als Erweiterung der IT Governance mit Prozessen und Richtlinien zugeschnitten auf eine SOA. Konzepte werden dabei nur erwähnt, Details finden sich kaum. Zur Unterstützung in der Praxis wird ein Katalog mit Best Practices angeboten.

Everware-CBDI [AW10] beschreibt das *CBDI-SAE SOA Governance Framework*, das die zentralen Fragen *was*, *wie*, *wer* und *wann* beantworten soll. Dazu werden verschiedene Sichten auf eine zentrale Richtlinienhierarchie definiert. Teil des Frameworks ist außerdem ein umfangreiches Reifegradmodell.

PricewaterhouseCoopers (PwC) [Pri09] beschreibt ein SOA Governance Framework, das die drei Dimensionen *Personen*, *Prozesse* und *Technologie* betrachtet. Diese sind in weitere Kategorien aufgeteilt. Ebenfalls definiert wird ein mehrstufiger Einführungsprozess für SOA Governance parallel zur Einführung einer SOA.

Die Open Group [The09a] stellt ein umfangreiches SOA Governance Framework zur Verfügung. Dieses besteht aus dem *SOA Governance Reference Model* (SGRM) und der *SOA Governance Vitality Method* (SGVM). Das SGRM beschreibt eine Menge von Leitmotiven (*Guiding Principles*), die den Anwender dabei unterstützen sollen eine SOA zu überwachen. Für diese Leitmotive werden empfohlene Aktivitäten zur Umsetzung aufgelistet. Die SGVM beschreibt vier Phasen zur Einführung, Überwachung und Verbesserung einer SOA Governance auf Basis des SGRM.

Tabelle 4.1 zeigt im Detail das Ergebnis der vollständigen Analyse der Frameworks. Zur besseren Vergleichbarkeit wurde jeder der Aspekte für jedes der Frameworks bewertet. Der Füllgrad der Kreise gibt an wie umfangreich das jeweilige Kriterium erfüllt wird.

Der Großteil der Frameworks beschränkt sich auf wenige ausgewählte Aspekte, für die ein detailliertes Konzept vorgestellt wird (bspw. [RB08; Hew08; Sys06]). Darüber hinaus wird die SOA Governance nicht im Ganzen

betrachtet. Zutreffend ist dies insbesondere für Frameworks, die von Softwareherstellern entwickelt wurden. Deren Definition der SOA Governance basiert, verständlicherweise, immer auf dem Funktionsumfang der jeweils angebotenen Software-Produkte. Einige Aspekte werden in fast allen Frameworks aufgeführt. Dabei handelt es sich zum einen natürlich um die grundsätzliche Notwendigkeit der SOA Governance. Ebenfalls häufig beschrieben werden die Lebenszyklusverwaltung von Services und Organisationsstrukturen, oft in Verbindung mit einem Rollenkonzept (bspw. [RB08]). Nur von knapp der Hälfte der Frameworks wird der Aspekt des Service-Monitorings betrachtet. Bis auf eines ([RB08]) wird in keinem der untersuchten Frameworks das Konzept der Schnittstellenverwaltung separat betrachtet. Dieses wird meist als Teil der Lebenszyklusverwaltung von Services gesehen und nicht, wie in dieser Arbeit, als serviceübergreifendes Konzept zur Erreichung von Schnittstellenstabilität und Agilität in den Geschäftsprozessen. Sehr umfangreiche Frameworks beschreiben Oracle und die Open Group. Insbesondere im Framework von Oracle sind Konzepte teilweise detailliert ausgearbeitet, beispielsweise für die Lebenszyklusverwaltung und die Reifegradbewertung einer SOA. Andere Aspekte wie Vergütungsmodelle und die Schnittstellenverwaltung finden hingegen kaum Beachtung. Auch das Framework der Open Group umfasst viele Aspekte, allerdings sind diese nicht so detailliert herausgearbeitet. In Teilen ist das Framework auch in sich nicht stimmig und erscheint sehr wahllos beschrieben.

Tabelle 4.1: Vergleich von SOA Governance Frameworks

	Kohnke et al. [KSH08]	Niemann et al. [NERS08]	Rieger & Bruns [RB08]	Oracle [Ram13], [Ben12], [HD13]	HP/ Systinet [Sys06], [Hew08]	IBM [HPG06] [BMT06]	Everware -CBDI [AW10]	PwC [Pri09]	Open Group [The09a]
Lebenszyklus-Verw.	◔	◔	◕	●	◕	◕	◕	◔	◕
Consumer-Verw.	○	◕	◔	◕	◕	◔	◔	○	◕
Organisationsstruktur	◕	●	●	●	○	◕	◔	◕	◕
Portfolio-Verw.	◔	○	○	●	◔	◔	◕	◔	◕
Architekturstandards	◕	○	○	◕	○	◔	○	◕	◔
Governance-Hierarchie	◕	○	◕	◕	◔	◔	○	○	◕
Vergütungsmodelle	◔	◔	○	◔	○	◔	◕	◕	◔
Service-Monitoring	◕	◕	○	◕	○	◔	○	○	●
Reifegradbewertung	○	◕	○	●	○	◔	●	○	◕
Schnittstellen-Verw.	○	○	◔	○	○	○	○	○	○

● Detailliertes Konzept ◕ Definiert ◔ Erwähnt ○ Nicht vorgesehen

4.2 SOA Governance Meta Model

Das im Folgenden vorgestellte SOA Governance Meta Model (SOA-GovMM) bildet die in Abschnitt 4.1.2 genannten Aspekte der SOA Governance und ihre Beziehungen untereinander ab und dient als Grundlage für die Entwicklung von Konzepten und Anwendungen zur Unterstützung der SOA Governance. Es soll auch dabei helfen die Beziehungen und Artefakte einer SOA leichter erfassen zu können, was bei vielen existierenden Frameworks durch die textlastigen Beschreibungen nur schwer möglich ist.

Abbildung 4.3 zeigt das Metamodell. Es ist in vier Bereiche aufgeteilt, die in den folgenden Abschnitten detaillierter vorgestellt werden:

- Service-Provider,
- Service-Consumer,
- Organisationsstrukturen,
- Business Objects.

4.2.1 Service-Provider

Teil A des SOA-GovMM beschreibt Artefakte der Service-Provider. Dies umfasst die in Abbildung 4.3 gezeigten Artefakte *Service* und *Serviceversion*, den zugehörigen Lebenszyklus (*Life Cycle State*) und Informationen zum Deployment (*Environment*).

Wie im Rahmen der Anforderungen bereits diskutiert, ist die Verwaltung des Service-Lebenszyklus eine wichtige Aufgabe der SOA Governance. Das Modell bezieht daher mit ein, dass es mehrere Versionen eines Service zu gleichen Zeit geben kann. Dies ist durch die 1:n-Relation *Service-Service Version* beschrieben. Jede dieser Serviceversionen kann dann wiederum mehreren Deployment-Umgebungen (*Environment*) zugeordnet werden, was die Verwaltung von Installationen auf unterschiedlichen IT-Umgebungen erlaubt. Da sich die Serviceversionen auf den unterschiedlichen Umgebungen

in unterschiedlichen Lebenszyklus-Phasen befinden dürfen, ist der Lebenszyklusstatus (*Life Cycle State*) an diese Relation geknüpft. Mittels *Operating Resources* lassen sich zudem Hardware-Ressourcen beschreiben.

Das Meta-Modell erlaubt das Anfügen von weiteren Informationen in Form von Serviceartefakten (*Service Artifact*). Zwei spezialisierte und immer benötigte Artefakte werden bereits durch das Modell definiert: Eine Servicebeschreibung (*Service Description*) enthält die Beschreibung eines Services, also beispielsweise die in einer WSDL-Datei (Web Service Description Language) enthaltenen Informationen. Ebenso sind die Schnittstellenbeschreibungen in Form von Business Objects als Serviceartefakte modelliert. Über Endpunkte (*Endpoint*) wird die Zuordnung von Service-Consumern (*Service-Consumer*) zu Serviceversionen (*Service Version*) abgebildet. Stakeholder werden als Kombination einer Rolle (*Role*) mit einer Person (*Person*) dargestellt und sind über die Rolle anderen Artefakten wie Services und Serviceversionen zugeordnet.

4.2.2 Service-Consumer

System und *System Version* beschreiben Softwaresysteme, die bestimmte Services nutzen. Diese Nutzung wird mittels Relation zu einem Service-Endpunkt dargestellt. Die Relation ist zudem assoziiert mit einem *Consumer Contract*, in dem mittels *Contract Property* die Einzelheiten des Service Level Agreements zwischen Service-Consumer und Service-Provider beschrieben werden. Diese Daten können ebenfalls für das automatische Monitoring von Services weiterverwendet werden. Auch in diesem Bereich sind Stakeholder wieder über die Relation zu einer Rolle modelliert.

4.2.3 Organisationsstrukturen

Wie bereits beschrieben, stützt sich eine SOA sehr stark auf eine klar definierte Organisationsstruktur und definierte Zuständigkeiten.

Kernelement dieses Teilbereichs ist die Entität *Role*. Sie ist Anknüpfungspunkt in die anderen Bereiche des Meta-Modells und modelliert damit

gewisse Zuständigkeiten im Rahmen der SOA Governance. Jede Rolle ist verknüpft mit bestimmten Zuständigkeiten (*Responsibilities*) die eine Person (*Person*), welche diese Rolle inne hat, erhält. Personen sind einerseits Vorgesetzten (*line manager*) zugeordnet, andererseits sind sie über die Relation zu einer Abteilung (*Department*) in die Unternehmenshierarchie eingegliedert. Zur Modellierung unterschiedlicher Unternehmensebenen sind Abteilungen bestimmten Geschäftseinheiten (*Business Unit*) zugeordnet.

Sowohl Personen als auch Rollen sind mit Fertigkeiten (*Skill*) verknüpft. Für eine Rolle werden dadurch bestimmte Anforderungen definiert, die zum Ausfüllen einer Rolle benötigt werden. Im Falle einer Person werden damit die Fertigkeiten repräsentiert, die diese Person hat. Dieses Konstrukt erlaubt es beispielsweise qualifizierte Kandidaten für eine Rolle zu identifizieren.

4.2.4 Business Objects

Die in Kapitel 3 vorgestellten Business Objects *plus* bilden einen weiteren zentralen Baustein des SOA Governance Meta Models. Abbildung 4.3 zeigt das zugehörige, leicht abstrahierte, Meta-Modell.

Jedes *Business Object* kann eine oder mehrere Versionen (*Business Object Version*) haben. Für jede Version wird ein eigener Lebenszyklusstatus (*Life Cycle State*) definiert. Ebenso ist jeder Version ein Datenmodell (*Data Model*) zugeordnet. Die Hierarchie der Business Objects wird über die Relation *superior* modelliert. Zusätzlich ist jedes Business Object auch einer Geschäftseinheit (*Business Entity*) im Quadranten Organisationsstruktur zugeordnet.

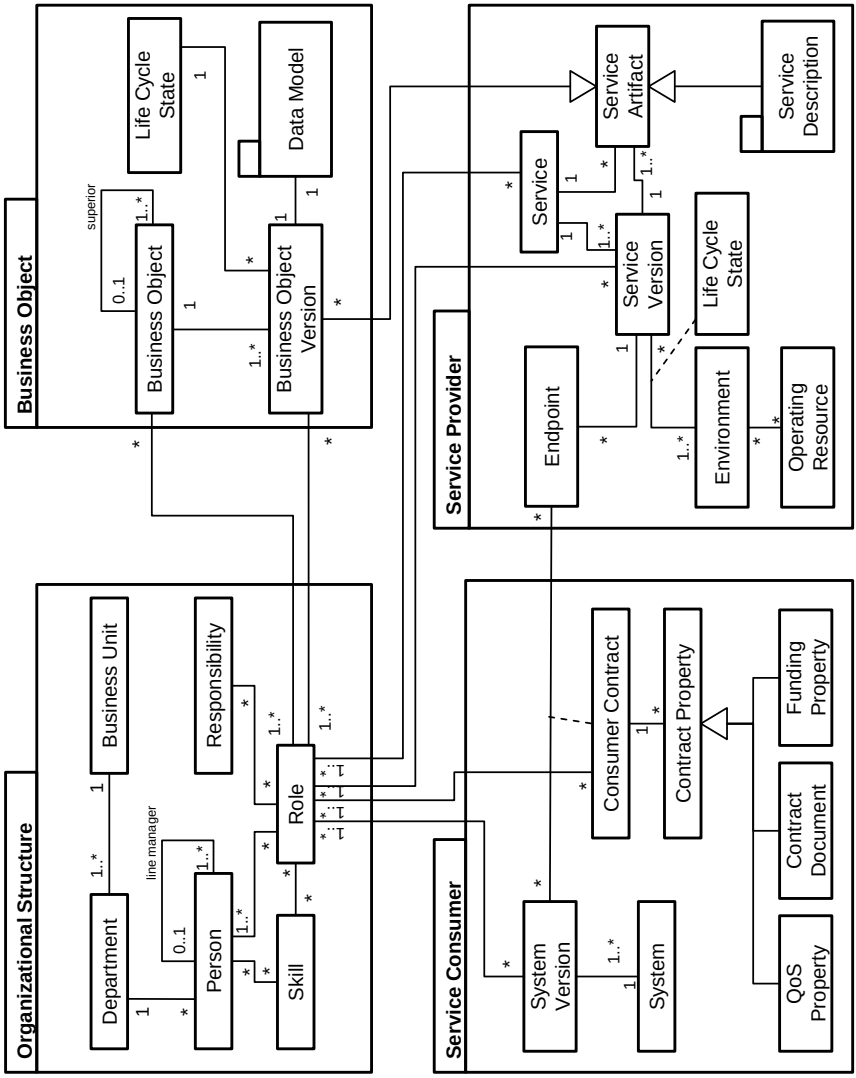


Abbildung 4.3: SOA-GovMM in Anlehnung an [KSM14] ©2014 IEEE

4.3 Die SOA Governance Ontology

Zur Nutzung in semantisch unterstützen Anwendungen wird in diesem Abschnitt eine auf dem SOA-GovMM basierende Ontologie, die *SOA Governance Ontology*, vorgestellt. Diese setzt auf dem SOA-GovMM auf und erweitert es um zusätzliche Beziehungen und Konzepte zwischen den Bestandteilen einer SOA. Im Folgenden werden zunächst verwandte Arbeiten und existierende Ontologien vorgestellt. Im Anschluss wird die SOA Governance Ontology im Detail diskutiert.

4.3.1 Verwandte Arbeiten

Im Kontext serviceorientierter Architekturen gibt es verschiedene Ontologien die in den letzten Jahren entstanden sind. Diese fokussieren sich meist sehr stark auf technische beziehungsweise Implementierungsaspekte, wie beispielsweise die *Web Services Modeling Ontology* [FD05].

Die Open Group stellt mit der *Service-Oriented Architecture Ontology* [The14] eine Ontologie vor, die „Konzepte, Terminologien und Semantik serviceorientierter Architekturen sowohl fachlicher als auch technischer Natur“ [The14] definiert, unter anderen mit den Zielen die Grundlage für weitere Arbeiten zu schaffen sowie die Kommunikation zwischen Fach- und IT-Personal zu unterstützen. Die beschriebene Ontologie ist allerdings sehr oberflächlich und beschreibt nur einige wenige grundlegende Konzepte wie beispielsweise Services, Service-Kompositionen oder Service Contracts. Personen werden durch einen generischen *HumanActor* abgebildet, dieser wird aber nicht direkt mit Services in Verbindung gebracht, sondern kann *Tasks* ausführen. Governance-Aspekte sind in der Ontologie nicht vorhanden, ebenso wenig nutzt die Ontologie Klassen oder Konzepte anderer Ontologien, obwohl sich dies insbesondere bei den Beziehungen anbieten würde und die Interoperabilität gefördert hätte. Die in dieser Forschungsarbeit definierte Ontologie geht im Umfang und der praktischen Anwendbarkeit daher deutlich über die von der Open Group definierte heraus.

Die *Web Ontology Language for Web Services* (OWL-S, [W3C04]) beschreibt

Services auf technischer Ebene, um das Auffinden, Ausführen und Überwachen von Webservices zu ermöglichen. Der Fokus liegt dabei auf Webservices und nicht auf einer Gesamtbetrachtung einer SOA. Dementsprechend sind auch keine Governance-Aspekte Teil der Ontologie. Die Ontologie kann als Erweiterung der hier vorgestellten SOA Governance Ontology zum Einsatz kommen, als Alternative ist sie nicht geeignet.

4.3.2 Analyse existierender Vokabulare und Ontologien

Ontologien sind per Definition offen und erweiterbar. Insbesondere machen sie sich häufig bereits existierende Ontologien zunutze, um gewisse Aspekte zu beschreiben. Dies erlaubt eine hochgradige Interoperabilität zwischen unterschiedlichen Anwendungen, welche die gleichen Ontologien einsetzen. Aus diesem Grund ist es ratsam, wo möglich, auf existierende Ontologien zu setzen und diese wieder- bzw. weiterzuverwenden. In diesem Abschnitt werden Ontologien verglichen und bewertet, die sich zum Einsatz in der SOA Governance Ontology anbieten. Tabelle 4.2 zeigt eine Kurzbewertung der untersuchten Ontologien und ob sie in der SOA Governance Ontology genutzt werden.

Tabelle 4.2: Kurzbewertung Ontologien

Ontologie	Verbreitung	Wiederverw.	Umfang	Nutzung
DCMI	◕	●	◑	ja
FOAF	◕	◕	◑	ja
vCard	◑	◕	◕	ja
Org	◑	◕	◑	ja
WSDL RDF	◔	◑	●	ja
FIBO	◔	◔	●	nein
SKOS	◕	●	◕	nein

Die **DCMI Metadata Terms** (Dublin Core Metadata Initiative, [DCM12]) sind ein weit verbreitetes und bekanntes Vokabular zur Beschreibung von Web-Ressourcen. Das *Dublin Core Metadata Element Set* besteht aus 15 Termen und ist als ISO 15836:2009 standardisiert [Int09]. Die *DCMI Metadata Terms* erweitern diese um zusätzliche Terme. In der SOA Governance Onto-

logy kommen unter anderem Terme wie *description*, *title* oder *isVersionOf* zum Einsatz.

FOAF (Friend-of-a-Friend, [BM14]) ist eine Ontologie, die Personen und deren Beziehung zu anderen Personen und Objekten beschreibt. FOAF ist eine der ältesten und bekanntesten Ontologien, das Projekt wurde im Jahr 2000 begonnen. Die Ontologie wird genutzt, um Teile der Organisationsstruktur der SOA Governance Ontology zu beschreiben.

Die **vCard Ontology** [W3C14e] ist eine Umsetzung der vCard-Spezifikation [Int11] als Ontologie. Sie beschreibt Personen und Organisationen und überlappt sich teilweise mit FOAF. Durch die Nutzung beider Ontologien ergibt sich ein größeres Integrationsspektrum mit anderen semantischen Anwendungen. Die SOA Governance Ontology nutzt die vCard-Ontologie zur Beschreibung zusätzlicher Eigenschaften für mittels FOAF beschriebener Personen.

Die **Organization Ontology** [W3C14d] beschreibt Organisationen und Organisationsstrukturen. Ebenfalls definiert sind Berichtsstrukturen innerhalb dieser Organisationen und Standortinformationen für Gebäude und Fabriken. Zur Beschreibung dieser Informationen nutzt die Ontologie auch Terme anderer Ontologien, insbesondere FOAF und vCard. In der SOA Governance Ontology werden mithilfe der Organization Ontology Teile der Organisationsstruktur beschrieben.

Mittels der Ontologie **Web Services Description Language Version 2.0: RDF Mapping** [W3C07c] lassen sich WSDL-Dokumente in Ressource Description Framework (RDF)/Web Ontology Language (OWL) beschreiben. So ist es möglich die Beschreibungen von Webservices umzuwandeln und in ein semantisches Datenmodell zu integrieren. Entsprechend wird die Ontology in der SOA Governance Ontology dazu genutzt die Service-Beschreibungen mit einzubeziehen.

Die **Financial Industry Business Ontology** (FIBO, [Obj15]) definiert Konzepte und Begriffe aus der Finanzindustrie. Die Ontologie ist sehr umfangreich, dadurch aber auch sehr spezifisch für die Finanzindustrie. Teile der Ontologie würden sich nutzen lassen um beispielsweise Verträge und Vergütungsmodelle zwischen Consumer und Provider zu beschreiben. Diese würden allerdings durch die in der Ontologie definierten Relationen in einen anderen Kontext gesetzt werden als beabsichtigt. Die FIBO-Ontologie kommt daher nicht zum Einsatz.

Das **Simple Knowledge Organization System** (SKOS, [W3C09]) ist ein RDF-Datenmodell zur Beschreibung von Klassifikationsschemata und Taxonomien. SKOS beschreibt dazu Konzepte und deren Hierachie und Abhängigkeiten innerhalb eines *Konzeptschemas*.

4.3.3 Bestandteile der SOA Governance Ontology

In diesem Abschnitt werden die Bestandteile der SOA Governance Ontology vorgestellt. Zum Einsatz kommen dabei auch die im vorherigen Abschnitt vorgestellten Ontologien und Vokabulare. Tabelle 4.3 listet die zugehörigen Präfixe auf, die in den Grafiken dieses Abschnitts verwendet werden.

Tabelle 4.3: Namespace Prefixes

Vocabulary	**Prefix**
SOA Governance Ontology	(sgo)
RDF	rdf
RDF Schema	rdfs
XML Schema Datatypes	xsd
Friend of a Friend	foaf
vCard	vcard
Organization Ontology	org
DCMI Metadata Terms	dct
WSDL RDF Mapping	wsdl

Zur besseren Übersicht wurde die Ontologie in fünf Bereiche gegliedert:

- Kernkomponenten,
- Organisationsstruktur,
- Service-Provider,
- Service-Consumer,
- Business Objects.

Einige Relationen werden ebenfalls aus Gründen der Übersicht nicht in den Abbildungen dargestellt.

4.3.3.1 Kernkomponenten

Dieser Teilbereich der Ontologie beschreibt mehrere abstrakte Kernkomponenten und Basisklassen, die in den anderen Teilbereichen weiterverwendet werden. Abbildung 4.4 zeigt diese Klassen, sowie eine Legende für die Abbildungen in diesem Abschnitt.

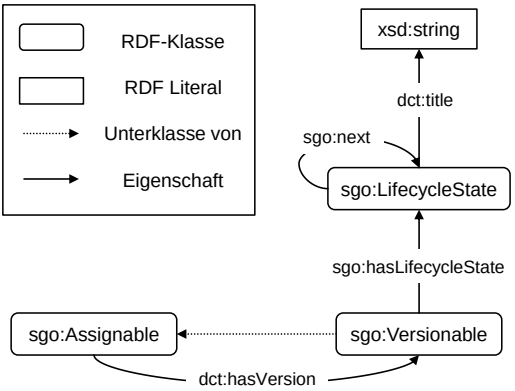


Abbildung 4.4: SOA Governance Ontology: Kernkomponenten in Anlehnung an [KM16] ©2016 IEEE

Kernkomponenten sind die Klassen *sgo:Assignable* und *sgo:Versionable*. Diese dienen als Superklassen für weitere Bestandteile der Ontologie. Ein

sgo:Assignable ist jede Klasse, der ein Posten (*org:Post*) zugewiesen werden kann. Die Person, die diesen Posten ausfüllt, ist dann dementsprechend ein Stakeholder der *sgo:Assignable*-Instanz. Subklassen von *sgo:Assignable* sind in allen Teilbereichen zu finden, beispielsweise *sgo:Service* oder *sgo:Contract*.

Die Klasse *sgo:Versionable* ist die Superklasse bestimmter Versionen eines *sgo:Assignable* (z.B. *sgo:ServiceVersion*). Durch die Modellierung als Subklasse von *sgo:Assignable* können auch einem *sgo:Versionable* Stakeholder zugeordnet werden. Beide Klassen werden durch die Relation *dct:hasVersion* veknüpft. Jedes *sgo:Versionable* hat einen Lebenszyklus, der durch die Relation *sgo:hasLifecycle* und die Klasse *sgo:LifecycleState* beschrieben wird.

4.3.3.2 Organisationsstruktur

Die Organisationsstruktur (Abbildung 4.5) nutzt die *Organization Ontology* und *FOAF*.

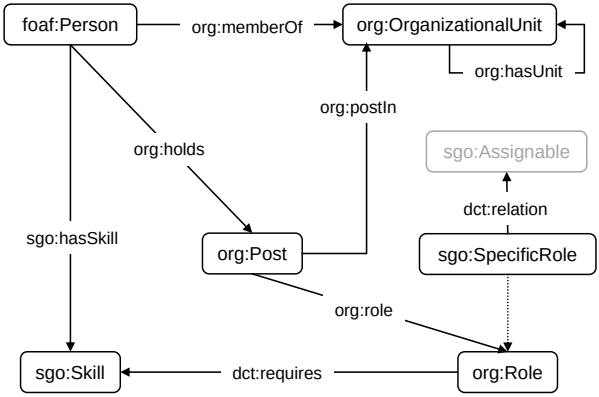


Abbildung 4.5: SOA Governance Ontology: Organizationsstruktur (ausgegraute Klassen zeigen die Verbindung zu anderen Teilbereichen der Ontologie) in Anlehnung an [KM16] ©2016 IEEE

Die Klasse *org:OrganizationalUnit* ist Teil der *Organization Ontology* und beschreibt die Unternehmenshierarchie. Personen (*foaf:Person*) sind dann jeweils diesen Organisationseinheiten zugeordnet. Mit den Klassen *org:Role*

und *org:Post* wird die Zuordnung von Personen zu Posten beschrieben und Instanzen von *sgo:Assignable* zugeordnet. Die Klasse *sgo:Skill* beschreibt Fertigkeiten einer Person bzw. Anforderungen einer Rolle.

4.3.3.3 Service-Provider

Der Bereich Service-Provider (Abbildung 4.6) beschreibt Services (*sgo:Service*, *sgo:ServiceVersion*) und zugehörige Artefakte, wie Endpunkte (*wsdl:Endpoint*) und Deployment-Informationen (*sgo:Deployment*).

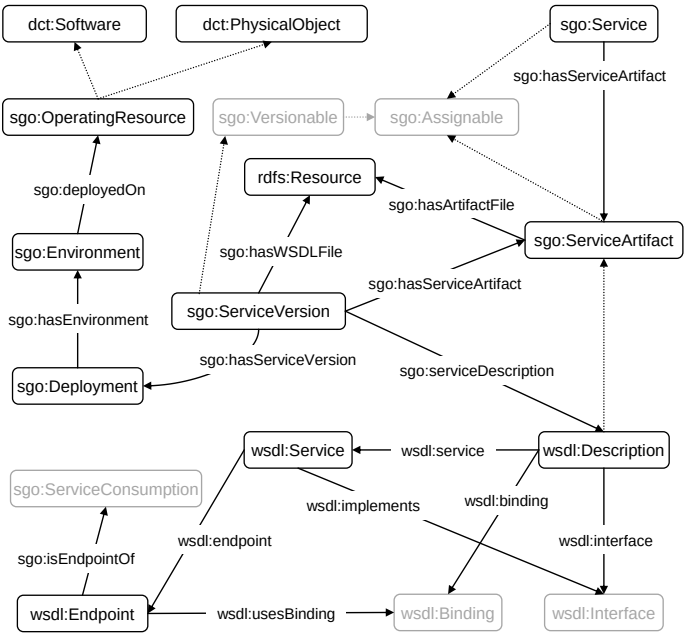


Abbildung 4.6: SOA Governance Ontology: Service-Provider in Anlehnung an [KM16] ©2016 IEEE

Die WSDL-Beschreibungen (*wsdl:description*) der Services werden über die Relation *sgo:serviceDescription* in die Ontologie eingebunden. Über die Relation *isEndpointOf* wird die Verbindung zum Teilbereich Service-Consumer beschrieben.

4.3.3.4 Service-Consumer

Der in Abbildung 4.7 gezeigte Teilbereich beschreibt die Nutzer der Services, also Systeme (*sgo:System*) und die Bedingungen, unter denen diese Services genutzt werden (*sgo:Contract* und *sgo:ServiceConsumption*). Über die Relation *regulatedBy* wird ein Vertrag einer Consumer-Provider-Beziehung zugeordnet.

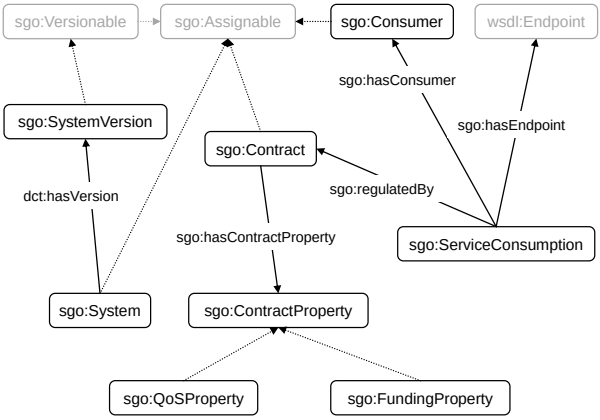


Abbildung 4.7: SOA Governance Ontology: Service-Consumer in Anlehnung an [KM16] ©2016 IEEE

4.3.3.5 Business Object

Dieser Teilbereich (Abbildung 4.8) beschreibt die Business Objects sowie die Datenmodelle, die in den Schnittstellen der Services zum Einsatz kommen. Diese Schnittstellenbeschreibung (*sgo:InterfaceModel*) kann aus einer Kombination von Datenobjekten (*sgo:DataObject*) und konkret definierten Business Objects (*sgo:BusinessObjectVersion*, *sgo:BusinessObject*) bestehen. Die hier beschriebenen Schnittstellen werden dann über ihr konkretes Datenmodell (*sgo:DataModel*) aus der WSDL-Beschreibung referenziert (*wsdl:InputMessage* und *wsdl:OutputMessage*).

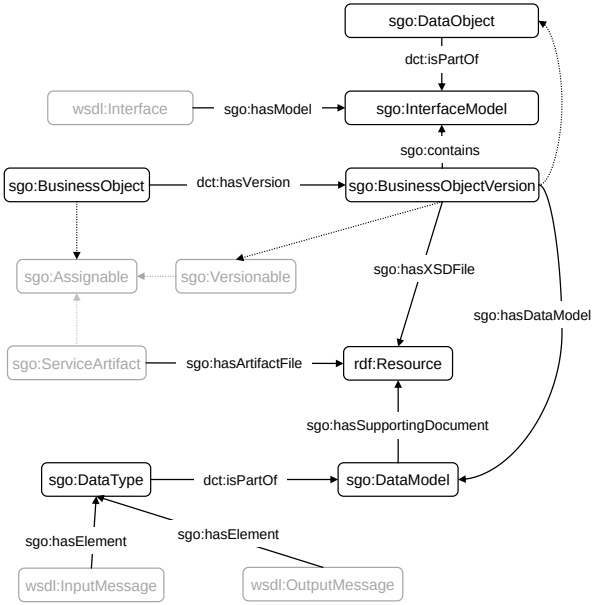


Abbildung 4.8: SOA Governance Ontology: Business Object in Anlehnung an [KM16] ©2016 IEEE

4.4 Bewertung und Zusammenfassung

Das vorgestellte SOA Governance Meta Model modelliert die Zusammenhänge zwischen den Governance-Artefakten einer SOA. Insbesondere werden dabei auch in anderen Frameworks sonst kaum beachtete Artefakte mit einbezogen. Dabei handelt es sich beispielsweise um das Deployment von Serviceversion in mehreren Umgebungen mit getrennter Lebenszyklusverwaltung und die Einbeziehung der Schnittstellenverwaltung in Form von Business Objects. Einige der eingangs genannten Anforderungen (vgl. Abschnitt 4.1) sind nicht als einzelne Entitäten modelliert, sondern ergeben sich aus dem gesamten Modell. Dies gilt für die Portfolioverwaltung genauso wie für das Service-Monitoring. Beides lässt sich durch Nutzung der mit dem Modell erfassbaren Informationen optimal unterstützen. Ebenfalls kann

sich mittels Statistiken über Service-Nutzung oder -Wiederverwendung eine fundierte Aussage über den Reifegrad eines SOA-Verbunds treffen lassen.

Aufbauend auf dem SOA-GovMM detailliert die ebenfalls vorgestellte SOA Governance Ontology das Meta-Modell für die Nutzung in semantisch unterstützten Anwendungen. Durch den Einsatz existierender Ontologien in der SOA Governance Ontology wird die Möglichkeit zur einfachen Interoperabilität mit einer Vielzahl an weiteren semantischen Anwendungen geschaffen. Die definierte Ontologie erlaubt ebenfalls den Einsatz von Semantischem Reasoning zur weitergehenden Informationsgewinnung und Unterstützung von SOA-Governance-Prozessen. Abschnitt 7.3 beschreibt die praktische Nutzung von Reasoning-Verfahren auf Basis der SOA Governance Ontology.

Kapitel 5

REST-to-SOAP Middleware Architecture

In diesem Kapitel wird eine Architektur vorgestellt, die eine semiautomatische Generierung von REST-Proxys (*Representational State Transfer*) für existierende „klassische“, meist SOAP-basierte Webservices ermöglicht. Die Inhalte dieses Kapitels wurden bereits in [KM18] veröffentlicht. Die Architektur hat zum Ziel, Unternehmen dabei zu unterstützen, diese klassischen Webservices schnell und ohne große Kosten auch über REST-APIs (Application Programming Interface) verfügbar und so in modernen Anwendungsfällen nutzbar zu machen.

In Abschnitt 5.1 werden zunächst die Anforderungen vorgestellt und in Abschnitt 5.2 gegen bereits existierende Ansätze abgegrenzt. Im Anschluss wird in Abschnitt 5.3 die REST-to-SOAP Middleware Architektur detailliert vorgestellt. Abschnitt 5.4 stellt die prototypische Umsetzung der Architektur vor und Abschnitt 5.5 verdeutlicht ihre Funktionsweise an einem kurzen Anwendungsbeispiel. Abschnitt 5.6 schließt das Kapitel mit einer Bewertung und Zusammenfassung ab.

5.1 Anforderungen

Durch den Einzug von mobilen Endgeräten in Unternehmen entstehen auch neue Anforderungen hinsichtlich der Agilität von IT-Systemen. Zur Integration von Systemen setzen Unternehmen häufig auf klassische, im Allgemeinen auf SOAP basierende Webservices und Messaging Middleware. Solche Webservices bieten gerade im Unternehmensumfeld viele Vorteile, unter anderem Wiederverwendbarkeit, Einfachheit der Integration und schnelle Anpassbarkeit. Allerdings ist damit die Integration mobiler Technologien und Endgeräte nicht ohne Weiteres möglich. Mobile Anwendungen stützen sich auf andere, leichtgewichtigere Technologien wie beispielsweise REST (REST, [Fie00]), das einen schnellen und flexiblen Zugriff auf Ressourcen und Informationen ermöglicht. Unternehmen stehen nun also vor der Herausforderung diese Technologien in ihren Technologie-Stack mit aufzunehmen und damit insbesondere auch den Zugriff auf existierende Webservices für mobile Endgeräte zu ermöglichen.

Neben der Integration von mobilen Anwendungen in ihre IT-Umgebungen, nehmen Unternehmen die zunehmende Digitalisierung auch zum Anlass neue Geschäftsfelder zu erschließen. So werden häufig Systeme für externe Nutzer geöffnet, um beispielsweise Pay-per-Use APIs anzubieten, die von Entwicklern genutzt werden können, um eigene Anwendungen zu entwickeln und zu vermarkten.

Im Unternehmensumfeld müssen bestimmte Anforderungen eingehalten werden, wenn Daten aus Systemen angefragt werden sollen. Dabei kann es sich um Sicherheitsaspekte im Umgang mit sensitiven Konstruktionsdaten oder um Einschränkungen bei der Verfügbarkeit von Entwicklungsressourcen handeln. Im Folgenden werden detaillierte Anforderungen an die Architektur vorgestellt. Diese entstammen intensiven Diskussionen mit Vertretern des Anwendungspartners.

5.1.1 Weiterverwendung existierender Services

Durch die teilweise bereits jahrelange Nutzung von serviceorientierten Architekturen beinhaltet das Service-Portfolio von Unternehmen eine Vielzahl an klassischen Webservices, die in existierende Geschäftsprozesse eingebunden und vollständig etabliert sind. All diese Services zusätzlich mit REST-APIs zu erweitern, würde einen enormen Entwicklungsaufwand und hohe Kosten mit sich bringen. Daraus ergeben sich zwei Möglichkeiten:

1. Eine Portierung genau der Services, für die wirklich ein Zugriff von mobilen Endgeräten benötigt wird oder
2. die Schaffung eines generischen Ansatzes, der es ermöglicht REST-Zugriffe auf alle Services zu ermöglichen, ohne dass diese manuell erweitert werden müssen.

Mit Möglichkeit 1 lassen sich zwar passgenaue REST-Services entwickeln, allerdings ist es durch den bereits genannten hohen Entwicklungsaufwand nur möglich eine kleine Anzahl an Services umzustellen. Variante 2 erfordert zunächst zwar auch einen hohen initialen Aufwand zur Etablierung einer entsprechenden Infrastruktur, allerdings können im Anschluss beliebige existierende Webservices über REST-APIs angesprochen werden.

5.1.2 Flexibler und schneller Zugriff

Wenn Unternehmen den Einstieg in den Markt mobiler Anwendungen gehen, ist häufig ein Umdenken in den Entwicklungsprozessen notwendig: Entwicklungszyklen in der „mobilen Welt“ sind um ein Vielfaches kürzer als in der eher gemächlichen Unternehmenswelt. Dementsprechend erwarten Nutzer regelmäßige Updates und die schnelle Umsetzung von Verbesserungen. Die Anbindung an existierende Unternehmensanwendungen über REST-APIs muss dann diesen schnellen Entwicklungszyklen folgen können. Besonders große Unternehmen kämpfen häufig, diesen Erwartungen gerecht zu werden. Traditionell arbeiten Unternehmen mit längeren Release-Zyklen, häufig sind beispielsweise zwei Verbund-Releases pro Jahr vorgesehen. Das erlaubt

es, Ausfallzeiten minimal zu halten und gibt den Entwicklern ausreichend Zeit neue Versionen vorab ausgiebig zu testen. Es wird also eine Architektur benötigt, die es erlaubt sowohl die sich schnell ändernden REST-Schnittstellen wie auch die sich seltener ändernden Backend-Services zu unterstützen. McKinsey hat dazu den Begriff „Architektur der zwei Geschwindigkeiten“ (Englisch: two-speed architecture, [BIL14]) geprägt.

Anfragen an REST-APIs werden in mobilen Anwendungen häufig im Hintergrund ausgeführt, um Daten nachzuladen während ein Nutzer die Anwendung verwendet. Um ein gutes Nutzungsverhalten sicherzustellen, ist es dafür notwendig, dass Anfragen an REST-APIs schnell beantwortet werden. Dies kann unter Umständen nicht sichergestellt werden, wenn Anfragen an monolithische Unternehmensanwendungen weitergeleitet werden müssen, die umfangreiche Datensätze zurückliefern. Daher ist es notwendig eine Möglichkeit zu schaffen, um Antworten auf solche Anfragen zwischenspeichern zu können und damit schnelle Antwortzeiten realisieren zu können.

5.1.3 Zuverlässigkeit

Auf die Zuverlässigkeit ergeben sich drei unterschiedliche Sichtweisen:

(1) REST bietet kein zuverlässiges Messaging, wie es von klassischen Webservices eingesetzt wird, die eine zuverlässige Nachrichtenzustellung auf Transportebene mittels WS-ReliableMessaging unterstützen [OAS09]. Entsprechende Funktionalität kann allerdings auch auf Anwendungsebene umgesetzt werden. Im Hinblick auf REST-Services ist Zuverlässigkeit aber insofern wichtig, dass wiederholte Anfragen an eine Ressource dieselben Ergebnisse zurückliefern müssen. In anderen Worten: Die vom REST-Paradigma geforderte Idempotenz der HTTP-Methoden (Hypertext Transfer Protocol (HTTP)) muss erfüllt werden (siehe Abschnitt 2.1.1.2).

(2) Der Zugriff auf Services muss zuverlässig sein, d.h. Services müssen stets erreichbar sein. Wartungsfenster müssen also vorab kommuniziert werden und das Risiko von Ausfällen muss nach Möglichkeit mit Failover-Systemen oder anderen Strategien verringert werden.

(3) Service-Nutzer erwarten, dass die Serviceschnittstelle dem REST-

Paradigma entsprechen. Dies betrifft insbesondere die korrekte Nutzung der HTTP-Verben zum Zugriff und zur Manipulation von Ressourcen.

5.1.4 Sicherheit

Der Zugriff auf existierende Webservices muss den gleichen Sicherheits-Anforderungen genügen, die an diesen bereits gestellt werden. Es darf also nicht zur Vereinfachung des Zugriffes auf Sicherheitsmechanismen verzichtet werden, in dem beispielsweise ein vom Backend-Service unabhängiges Identitäts- und Zugriffsmanagement in der Middleware-Komponente eingesetzt wird. Dementsprechend ist es erforderlich Authentifizierungsanfragen an den REST-Endpunkt ans Backend-System weiter zu propagieren.

5.1.5 Versionierbarkeit

Wie bereits in den vorherigen Kapiteln ausgeführt, ist beim Betrieb von Services ein kontrolliertes Änderungsmanagement notwendig. Dies gilt auch für REST-Proxys. Wenn sich an den Backend-Services Änderungen ergeben, müssen diese auch im zugehörigen REST-Proxy reflektiert oder durch ihn transparent gehalten werden. Dementsprechend ist es notwendig unterschiedliche Versionen eines Services sowohl sequentiell als auch parallel verwalten zu können. Ebenso müssen Service-Konsumenten in der Lage sein eindeutig zu identifizieren, auf welche Serviceversion sie zugreifen. Da sich Proxy-Services auch unabhängig vom Backend-Service weiterentwickeln können, ist es notwendig die Version des Proxys von der Version des Backend-Services zu entkoppeln

5.2 Existierende Ansätze und verwandte Arbeiten

Im Vorfeld der Entwicklung der REST-to-SOAP Middleware Architektur wurden existierende Ansätze und verwandte Arbeiten untersucht. Diese lassen sich in zwei Bereiche gliedern:

- **Wissenschaftliche Veröffentlichungen**: Diese beschäftigen sich insbesondere mit den grundlegenden Unterschieden zwischen SOAP und REST (u.a. [PZL08; Pre02]). Sehr wenige Arbeiten untersuchen auch den Zugriff auf SOAP-Services über REST-Schnittstellen.
- **API Middleware und zugehörige Industrie-Whitepaper**: Es gibt einige wenige API-Middleware-Lösungen, die bereits Ansätze für einen REST-Proxy-Zugriff auf Webservices unterstützen. Teilweise wird deren Funktionalität in zugehörigen Whitepapern beschrieben.

Die relevanten wissenschaftlichen Veröffentlichungen betrachten das Thema hauptsächlich aus Sicht von Workflows:

De Giorgio et al. [dGRZ10] untersuchen, inwiefern SOAP- und REST-basierte Webservices in BPEL-Workflows eingesetzt werden können. Die Autoren diskutieren Möglichkeiten semantische Beschreibungen beider Service-Arten abzugleichen. Dadurch ermöglichen Sie zur Laufzeit, je nach Verfügbarkeit, einen dynamischen Wechsel zwischen SOAP- und REST-basiertem Service. Eine Möglichkeit automatisiert REST-APIs zu SOAP-Services zur Verfügung zu stellen wird dabei aber nicht geboten, alle angesprochenen Schnittstellen müssen bereits existieren. Peng et al. [PML09] stellen den entgegengesetzten Weg dar. Das SOAP2REST-Framework der Autoren erlaubt es, REST-Services mittels SOAP-basierten Services zu verpacken, um diese so in BPEL-Prozessen einsetzen zu können. Das Framework generiert dazu aus *Web Application Description Language*-Beschreibungen (WADL[1]) der REST-Services die SOAP-Proxys automatisch.

Apigee bietet den Nutzern seiner API-Plattform einen Assistenten, mit dessen Hilfe SOAP-basierte Webservices in REST-basierte Webservices umgewandelt werden können [Api18]. Der Assistent nutzt dazu die vorhandenen WSDL-Beschreibungen (Web Service Description Language) der Services, um ein Skelett für den REST-Service zu erzeugen. Der Nutzer kann dann die notwendigen Parameter wie Uniform Resource Identifiers (URIs) der Ressourcen und die HTTP-Verben händisch ergänzen. Erweiterte Funktiona-

[1] https://javaee.github.io/wadl/

litäten, die für den Unternehmenseinsatz notwendig sind, wie das Caching von Anfragen oder die Propagation von Authentifizierungsinformationen werden nicht unterstützt.

Mulesoft beschreibt einen zum Großteil manuellen Prozess zur Erstellung von REST-Proxys [Agr15]. Nachdem der gewünschte REST-Service mittels *RESTful API Modeling Language* (RAML) beschrieben wurde, wird die Anwendung *Anypoint Studio* des Herstellers genutzt, um die Nachrichtentransformation von der HTTP-Anfrage zur SOAP-Anfrage zu erzeugen. Eine automatische Generierung einer REST-API aus einer existierenden WSDL-Beschreibung ist nicht vorgesehen. Ebenso wird hier kein Caching und keine Propagation von Authentifizierungsinformationen unterstützt.

Zusammengefasst adressiert keiner der existierenden Arbeiten und Produkte die in Abschnitt 5.1 diskutierten Anforderungen, die notwendig sind um einen REST-zu-SOAP-Proxy für den Unternehmenseinsatz zu realisieren.

5.3 REST-to-SOAP Middleware Architecture

Klassische Webservices haben viele Vorteile, die sie zur idealen Technologie für die Integration von Unternehmensanwendungen machen. Durch den steigenden Bedarf an Flexibilität von Unternehmensarchitekturen in den letzten Jahren wurden allerdings auch leichtgewichtigere Technologien wie REST in die Architekturen integriert. Die Hauptunterschiede zwischen den beiden Architekturparadigmen wurden bereits in Abschnitt 2.1.1.3 diskutiert.

Die REST-to-SOAP Middleware Architecture (R2SMA), die in diesem Kapitel vorgestellt wird, ermöglicht es Unternehmen existierende klassische Webservices mittels eines Middleware-Proxys zu abstrahieren und REST-APIs für diese Services anzubieten. Transparent für die Servicenutzer bietet die Architektur also eine REST-zu-SOAP-Konvertierung. Im Folgenden zeigt Abschnitt 5.3.1 einen Gesamtüberblick über die Architektur, bevor die weiteren Abschnitte sich den einzelnen Komponenten widmen.

5.3.1 Überblick

Abbildung 5.1 zeigt eine Gesamtübersicht der Architektur. Die Kernfunktionalität der Architektur deckt die Komponente *REST-to-SOAP-Proxy* ab. Sie stellt die REST-Proxy-Endpunkte den Nutzern zur Verfügung. Die Proxy-Komponente bietet zudem Caching und Nachrichtentransformation für REST-Anfragen. Existierende Webservices werden entweder direkt oder über eine Middleware-Schicht, wie beispielsweise einen Enterprise Serivce Bus an die Transformations-Komponente angebunden. Entsprechende Middleware bietet meist bereits Sicherheits- und Orchestrationsfunktionalitäten. Ebenfalls angebunden werden können existierende REST-Services, für die der Cache des REST-to-SOAP-Proxys verwendet werden kann, um Anfragen zu beschleunigen. Vervollständigt wird die Architektur durch einen Enterprise Security Gateway, zuständig für Authentifizierung und Autorisierung, sowie eine Repository-Komponente für Schnittstellenverwaltung und Proxy-Generierung.

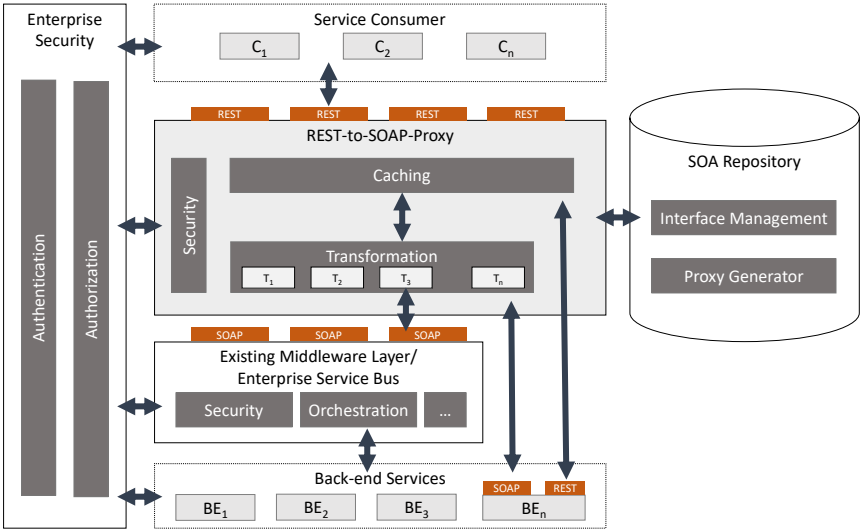


Abbildung 5.1: REST-to-SOAP Middleware Architecture in Anlehnung an [KM18]

5.3.2 Transformation

Jede Anfrage an einen Proxy-Endpunkt löst im Hintergrund eine Anfrage an den zugehörigen Backend-Service aus, sofern die entsprechende Anfrage nicht bereits gestellt wurde und daher im Cache zwischengespeichert ist. Dieses Verhalten ist für den Nutzer bzw. Anfrager allerdings komplett transparent. Die Transformationskomponente übersetzt eine Nachricht dann mittels eines Transformators (T_n in Abbildung 5.1) aus einem REST-kompatiblen JSON-Format (JavaScript Object Notation) in ein Datenformat, das der Backend-Service interpretieren kann. Meist handelt es sich dabei um SOAP, aber auch die Anbindung an eine Message-oriented Middleware oder ein proprietäres Datenformat wären möglich.

Das genaue Datenformat und die Transformation wird dazu bereits beim Erstellen eines Proxy-Endpunkts festgelegt und ist Teil der Schnittstellenbeschreibung. Die Transformationskomponente übernimmt auch die Validierung der eingehenden Nachricht und stellt sicher, dass sie in das SOAP-Format übersetzt werden kann. Eine sorgfältige Validierung ist notwendig, da JSON nur eingeschränkt Datentypen unterstützt. Beispielsweise wird bei Zahlenwerten nicht zwischen Ganz- und Fließkommazahlen unterschieden, was zu unerwartetem Verhalten führen kann.

5.3.3 Caching

Die Caching-Komponente ist ein wichtiger Teil der Architektur und erfüllt zwei Hauptaufgaben. Sie dient zum einen zur Reduzierung der Last auf den Backend-Systemen und zum anderen der Ermöglichung schneller Antwortzeiten.

Aufgrund der bereits in Abschnitt 2.1.1.3 diskutierten Unterschiede zwischen REST-APIs und Webservices, werden beim Einsatz des REST-Paradigmas deutlich mehr Anfragen an einen Service gestellt als beim Einsatz von klassischen Webservices. Der Grund dafür ist, dass REST Ressourcenbasiert arbeitet, wohingegen klassische Webservices über Operationen angefragt werden. Bei REST ist es daher häufig notwendig Anfragen an mehrere Res-

sourcen zu stellen, um die gleichen Informationen zu erhalten, die eine einzelne SOAP-Anfrage ergibt. Zudem ermöglicht REST auch das teilweise Abrufen von Ressourcen, sogenannter *content ranges*. Dazu wird bei Listen immer nur ein kleiner, in der Anfrage definierter Ausschnitt abgerufen. Dies ermöglicht kürzere Antwortzeiten und dadurch eine schnellere Darstellung auf dem Endgerät des Nutzers. Wenn nun dieses Mehr an Anfragen jedes Mal an den Backend-Service weitergereicht werden muss, erhöht sich die Last auf dem Backend-System um ein vielfaches, was im schlimmsten Fall zu Ausfällen oder gar Abstürzen führen kann. Daher ist der Einsatz einer Caching-Komponente unabdingbar.

5.3.3.1 Einbindung des Caches in die Sicherheitsinfrastruktur

Im Unternehmensumfeld, insbesondere in der Automobilindustrie arbeiten Nutzer häufig mit hochsensiblen Daten. Dies macht es erforderlich, dass der Cache in die übergeordnete Sicherheitsinfrastruktur eingebunden wird. Nur so lässt sich sicherstellen, dass Nutzer ausschließlich auf Daten zugreifen können, für die sie eine Berechtigung haben.

Ebenfalls kann für hoch-sensitive Daten das Caching komplett deaktiviert werden. Dies führt zwar zu einer höheren Last auf dem Backend-System, muss aber für den zusätzlichen Sicherheitsgewinn akzeptiert werden.

5.3.3.2 Arten von Caches

Die Realisierung des Caches kann auf unterschiedliche Arten erfolgen:

1. als zentraler, nutzerunabhängiger Cache
2. als nutzerabhängiger Cache
3. als nutzerabhängiger, sessionbasierter Cache

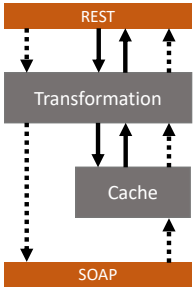
Der zentrale Cache lässt sich am besten mit einem Standard HTTP-Cache vergleichen, der jede angefragte Ressource einmalig speichert und diese bei nachfolgenden Anfragen unabhängig von der Identität des Anfragers

ausliefert. Für Services, die keinerlei Authentifizierung erwarten, wäre diese Art der Implementierung die offensichtlichste und effizienteste Lösung. Aufgrund der bereits geschilderten Sicherheitsanforderungen, ist diese Variante im Unternehmensumfeld allerdings nicht nutzbar. Es wird daher ein nutzerabhängiger Cache benötigt, der Ressourcen getrennt für jeden Nutzer vorhält. Dies führt zu höheren Anforderungen an den Speicherplatz, da die gleiche Ressource mehrmals zwischengespeichert werden muss. Um den notwendigen Speicherplatz zu verkleinern, kann ein sessionabhängiger Cache zum Einsatz kommen. Diese Spezialisierung des nutzerabhängigen Caches entfernt Ressourcen aus dem Cache, sobald die Session eines Nutzers abgelaufen ist, was deutlich kürzer als die Lebenszeit (englisch: *time-to-live, TTL*) der Ressource sein kann.

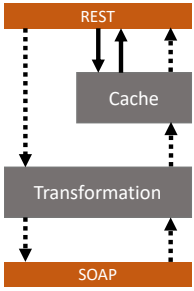
5.3.3.3 Positionierung des Caches

Jede Antwort eines Backend-Services wird mit Hilfe der Transformations-Komponente vom SOAP-Format in ein REST-geeignetes JSON-Format überführt. Für die Caching-Komponente erlaubt dies zwei Möglichkeiten der Positionierung, entweder vor oder nach der Nachrichtentransformation. Diese Positionierung hat auch Auswirkungen auf die Funktionalität des Caches. Abbildung 5.2 zeigt die beiden Varianten. Gestrichelte Pfeile zeigen den Nachrichtenfluss einer initialen Anfrage, durchgehende Pfeile den für nachfolgende Anfragen unter Nutzung des Caches.

Eine Positionierung vor der Transformation (Abbildung 5.2a) erlaubt es die zwischengespeicherte Antwort des Backend-Services für weitere, auch andere, Anfragen wiederzuverwenden. Allerdings erhöht dies auch die Last auf den Proxy, da die Antworten für jede neue Anfrage erneut transformiert werden müssen. Eine Positionierung nach der Transformation (Abbildung 5.2b) erlaubt kürzere Antwortzeiten bei Folgeanfragen, allerdings stehen die Daten anderen, nicht-identischen Anfragen nicht zur Verfügung. Aufgrund der bereits diskutierten Sicherheitsanforderungen im Unternehmensumfeld, müssen unterschiedliche Anfragen aber grundsätzlich einmalig vom Backend-System beantwortet werden, um sicherzustellen, dass der Anfra-



(a) Vor der Transformation



(b) Nach der Transformation

Abbildung 5.2: Positionierung des Caches

ger auch wirklich eine entsprechende Berechtigung hat. Eine Nutzung von zwischengespeicherten Antworten für andere Anfragen ist daher ohnehin nicht möglich. Dementsprechend ist in der Architektur der Cache nach der Transformation positioniert, so wie in Abbildung 5.2b dargestellt.

5.3.4 HATEOAS und Hypermedia Controls

Ein Grundkonzept von REST-Schnittstellen ist, dass diese sich selber beschreiben, also dem Nutzer selbstständig mitteilen, wie die Schnittstelle zu benutzen ist. Dieses Konzept wird als *Hypermedia as the Engine of Application State* (HATEOAS, [Fie08]) bezeichnet. In der Umsetzung bedeutet dies, dass Ressourcen dem Nutzer mitteilen, welche Operationen unterstützt werden und welche URIs dazu angefragt werden müssen. Dadurch ist es möglich, eine REST-API zu nutzen, ohne weitere Informationen als ein Einstiegs-URI zu benötigen. Im Unternehmensumfeld ist ein solches Verhalten aber häufig nicht gewünscht, da hier die Stabilität, und damit eindeutig definierte Schnittstellen, wichtiger gewertet wird als die Flexibilität. Dennoch müssen diese Aspekte natürlich auch hier betrachtet werden.

Die R2SMA unterstützt, auf konzeptuellem Level, HATEOAS und damit *Hypermedia Controls*. Bei der Generierung des Proxy-Endpunkts werden dazu die Operationen des Webservice einer Kombination von Ressource und HTTP-

Verb zugeordnet. Beispielsweise wird die Operation *retrieveEmployee* der Ressource /employee mit dem HTTP-Verb GET zugeordnet, die Operation *deleteEmployee* derselben Ressource aber mit dem HTTP-Verb DELETE (siehe dazu auch Abschnitt 5.5). Dadurch ist es möglich, bei Anfragen an eine Ressource auch die URI für weitere Anfragemöglichkeiten auf der Ressource mit auszuliefern.

5.3.5 Identitäts- und Zugriffsmanagement

Der Proxy enthält keine eigene Komponente zur Authentifizierung der Nutzer, sondern greift auf bereits existierende Sicherheitsmechanismen zurück. Dies ist zum Einen ein Sicherheits-Gateway, wie es im Allgemeinen in Unternehmen bereits existiert, zum Anderen wird zusätzlich auf das Rechte- und Zugriffsmanagement der Backend-Systeme vertraut. Entsprechende Anmeldeinformationen werden dazu vom Proxy direkt an die Backend-Systeme weitergeleitet.

Klassische WebsServices unterstützen eine Vielzahl erweiterter Sicherheitsmechanismen, wie beispielsweise XML-Verschlüsselung, XML-Signaturen oder das darauf aufbauende WS-Security. Für die Realisierung der Proxy-Endpunkte ist es nun notwendig, auf Sicherheitsverfahren zu setzen, die auch mit REST abzubilden sind. Die R2SMA setzt dazu auf *Security Assertion Markup Language*-Token (SAML, [Int15]), da diese sowohl mit SOAP, als auch mit REST-Schnittstellen genutzt werden können. Ebenso möglich ist eine Lösung mittels HTTP-Basisauthentifizierung und Weiterpropagation der Anmeldeinformationen an den Backend-Service.

5.3.6 Schnittstellenverwaltung

Neben der Proxy-Endpunkt-Erzeugung ist das SOA Repository ebenfalls zuständig für die Verwaltung der Schnittstellenbeschreibungen, sowohl für REST-APIs als auch für Webservices. Für die Beschreibung von REST-Services setzt die R2SMA auf OpenAPI [Ope18]. Dieser Standard hat sich in jüngster Vergangenheit durchgesetzt, nicht zuletzt aufgrund der Unter-

stützung durch große Firmen wie Microsoft, Google und Atlassian. Die Service-Beschreibungen werden über das Repository den Service-Nutzern zur Verfügung gestellt. Ebenso beinhaltet es die Versionsverwaltung der Schnittstellen.

5.3.7 Proxy-Generator

Die Proxy-Generator-Komponente dient zur Erstellung der REST-Proxy-Endpunkte. Mittels eines interaktiven Assistenten wird dazu aus der WSDL-Beschreibung eines Backend-Services ein REST-Proxy-Endpunkt erstellt. Dazu werden die Operationen des Webservices auf Ressourcen gemappt und die passenden HTTP-Verben festgelegt (also GET für eine Anfrage, DELETE zum Löschen usw.).

Durch das Mapping der Operationen auf Ressourcen und das bereits beschriebene Mapping der Datenmodelle (siehe Abschnitt 5.3.2) von SOAP auf JSON, lässt sich dann automatisch die Endpunkt-Komponente erzeugen und in der Middleware ausbringen.

5.4 Umsetzung

Die vorgestellte Architektur soll zur Umsetzung eines REST-zu-SOAP-Proxys dienen. Die konkrete Auswahl an Technologien zur Umsetzung ist dabei stark abhängig vom IT-Ökosystem des jeweiligen Unternehmens. Der folgende Abschnitt stellt einige Technologien vor, die bei einer Umsetzung zum Einsatz kommen können. Daran anschließend stellt Abschnitt 5.4.2 einen Teilbereich der Architektur vor, der prototypisch implementiert wurde.

5.4.1 Technologien zur Umsetzung

Zur Umsetzung der Proxy-Komponente inklusive Caching, Transformation und Security kann Standardsoftware zum Einsatz kommen. Application Server und verwandte Produkte verschiedener Hersteller bieten bereits von

Haus aus entsprechende Caching- und Security-Funktionalitäten an. Entsprechende Hersteller sind beispielsweise WSO2[1], Mulesoft[2] oder auch IBM[3].

Die jeweiligen Transformations-Komponenten können dann vom Proxy-Generator erzeugt und auf dem gewählten Application Server verfügbar gemacht werden. Als Technologien können hier beispielsweise Java-Bibliotheken oder XSLT-Stylesheets (*Extensible Stylesheet Language Transformations*) zum Einsatz kommen. Für die Transformation von XML zu JSON existieren bereits Konverter wie XSLTJSON[4].

Als Repository ist in der Architektur das im Rahmen dieser Arbeit entwickelte SOA Governance Repository vorgesehen (siehe Kapitel 7). Dieses dient als zentraler Anlaufpunkt für Informationen zum Zustand einer Unternehmens-SOA und stellt Dokumentation über Services und Stakeholder bereit. Die entsprechenden Funktionalitäten könnten auch als alleinstehendes Tool entwickelt werden oder in eines der in Abschnitt 5.2 vorgestellten am Markt erhältlichen Produkte integriert werden.

5.4.2 Prototypische Implementierung

Die Kernkomponente der Architektur ist der im SOA Governance Repository verortete Proxy-Generator. Dieser wurde in einer im Rahmen dieser Forschungsarbeit konzipierten und betreuten Bachelorarbeit teilweise implementiert [Kra16a].

5.4.2.1 Proxy-Generator

Das SOA Governance Repository speichert zu jedem dort verwalteten Webservice die Schnittstellenbeschreibungen als RDF-Tripel (Ressource Description Framework (RDF)) ab. Die Beschreibungen werden dazu beim Anlegen mithilfe des *WSDL Parsers* eingelesen und umgewandelt. Der Proxy-Generator

[1] https://www.wso2.com
[2] https://www.mulesoft.com
[3] https://www.ibm.com
[4] https://github.com/bramstein/xsltjson/

greift auf diese Daten zurück, um den REST-Proxy-Endpunkt für den jeweiligen Web Service zu erzeugen.

Aufgrund der bereits beschriebenen Paradigmen-Unterschiede – operationenbasiert bei SOAP, ressourcenbasiert bei REST – ist ein manueller Schritt nötig, um einen Proxy zu erzeugen. In diesem Schritt muss ein technischer Service-Verantwortlicher für die Operationen des Webservices Ressourcen-URIs und zugehörige HTTP-Verben definieren. Abbildung 5.3 zeigt die Benutzeroberfläche des SOA Governance Repository.

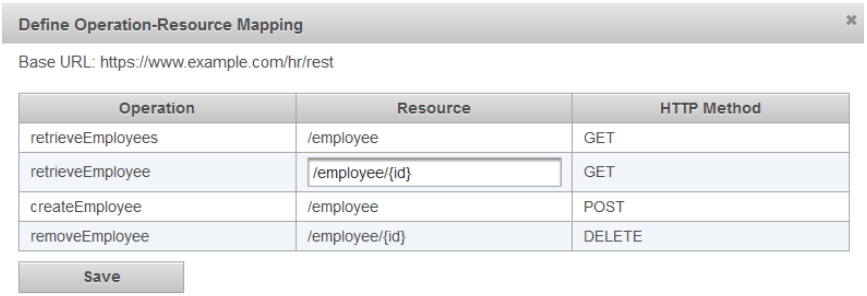


Abbildung 5.3: Benutzeroberfläche des SOA Governance Repository: Erstellung eines REST-Proxys

Dieser Vorgang wird sofern möglich unterstützt, indem aus den Operations-Namen HTTP-Verben abgeleitet werden. Beispielsweise wird für eine Operation *getXY* das HTTP-Verb *GET* vorgeschlagen. Die so ergänzten Informationen werden ebenfalls als Tripel in der Datenbank des SOA Governance Repository abgelegt und für die Generierung des REST-Proxy-Endpunkts genutzt. Ein solcher Proxy-Endpunkt besteht aus einer generierten Java-Bibliothek, die dann auf einem Application Server ausgebracht werden kann und den Nutzern eine REST-API zur Verfügung stellt. Eine zugehörige Transformation übersetzt ankommende Anfragen in SOAP und umgekehrt.

5.4.2.2 REST-Schnittstellenbeschreibung

Neben dem Proxy-Endpunkt wird auch eine Beschreibung der REST-Schnittstelle generiert und im Repository für Nutzer veröffentlicht. Für diese Be-

schreibung wird der OpenAPI-Standard [Ope18] der OpenAPI-Initiative genutzt. Die zur Generierung des Beschreibungsdokuments notwendigen Daten sind alle im SOA Governance Repository gespeichert und stammen in Teilen aus dem eingelesenen WSDL-Dokument und werden vor der Generierung des Proxy-Endpunkts wie oben beschrieben manuell ergänzt. Im SOA Governance Repository sind zudem die im zugrundeliegenden Backend-Service beschriebenen Datenobjekte bereits mit Business Objects *plus* (siehe Kapitel 3) verknüpft. Aus diesen wird dann ein kompatibles JSON-Schema [Int18] für die Verwendung in der REST-Schnittstellenbeschreibung erzeugt. Auch eine direkte Umwandlung aus einer XML-Schema-Definition ist möglich. Es existieren dazu bereits Ansätze und Tools, unter anderem [NF14], sowie owl2jsonschema[1] und JSONix Schema Compiler [2].

5.5 Anwendungsbeispiel

Im Folgenden wird ein kurzes Anwendungsbeispiel vorgestellt, um zu zeigen, wie ein REST-Proxy-Endpunkt für einen existierenden Webservice erzeugt wird und wie die Zuordnung von Operationen zu Ressourcen umgesetzt ist.

Das folgende Listing 5.1 beschreibt einen Webservice mittels WSDL 2.0. Der Service *MyHRPortal* stellt vier Operationen bereit, um Mitarbeiter zu verwalten. Mit diesen Operationen lassen sich ein oder mehrere Mitarbeiter abrufen, löschen oder neu anlegen. Die WSDL-Bescheribung wurde um nicht relevante Bestandteile gekürzt.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <description [...] >
3   <types>
4     <xs:schema [...] >
5       <xs:element name="employee"> ...
6       [...]
7     </xs:schema>
8   </types>
9
```

[1] https://github.com/redaktor/owl2jsonschema.js
[2] https://github.com/highsource/jsonix-schema-compiler

```
10   <interface name="HRPortal"> [...] <interface>
11
12   <binding name="SoapBinding"
13            interface="tns:HRPortal"
14            type="...w3.org/ns/wsdl/soap"
15            [...] >
16     <operation ref="tns:retrieveEmployees" />
17     <operation ref="tns:retrieveEmployee" />
18     <operation ref="tns:createEmployee" />
19     <operation ref="tns:removeEmployee" />
20   </binding>
21
22   <service name="MyHRPortal" interface="tns:HRPortal">
23     <endpoint name="SoapBinding"
24               binding="tns:SoapBinding"
25               address=".../hr/soap/"/>
26   </service>
27 </description>
```

Listing 5.1: Auszug aus einem WSDL-Beschreibungsdokument

Diese WSDL-Datei wird geparst und in der Repository-Komponente der R2SMA abgelegt. Aus diesen Informationen wird dann ein Skelett für den Proxy-Endpunkt generiert, der bereits den Großteil der abstrakten Inhalte der geparsten WSDL enthält. Dazu zählen Service-Name, -Beschreibung und die Datentypen. Im Anschluss kann ein Service-Entwickler oder ein anderer technischer Verantwortlicher die Operationen den Ressourcen zuordnen. Tabelle 5.1 zeigt die Zuordnung für diesen Anwendungsfall. Wie schon in Abschnitt 5.4.2.1 beschrieben und in Abbildung 5.3 gezeigt, wird hierfür eine Benutzeroberfläche angeboten. Basierend auf dieser Zuordnung kann dann der Proxy-Endpunkt über die R2SMA zur Nutzung angeboten werden.

Das vorgestellte Anwendungsbeispiel ist recht unkompliziert und die Zuordnung der Operationen zu Ressourcen wäre sogar automatisiert möglich. Allerdings ist die Zuordnung nicht für alle Services so einfach zu realisieren. Dies ist insbesondere dann der Fall, wenn Services unstrukturierte Operationen haben, wie es im Unternehmensumfeld manchmal vorkommt. Eine korrekte Zuordnung entsprechend der REST-Richtlinien ist dann nicht mög-

Tabelle 5.1: Operationen-Ressourcen-Zuordnung

WS-Operation	REST-Ressource	HTTP-Methode
retrieveEmployees	.../employee	GET
retrieveEmployee	.../employee/id	GET
createEmployee	.../employee	POST
removeEmployee	.../employee/id	DELETE

lich. In solchen Fällen müssen zunächst APIs mit niedrigerem Reifegrad akzeptiert oder die Backend-Services entsprechend angepasst werden.

5.6 Zusammenfassung und Ausblick

Die vorgestellte Architektur erreicht durch die Abstraktion der Backend-Services durch die REST-to-SOAP Proxy Middleware und die manuelle Definition der Operationen-Ressourcen-Zuordnung Level 2 des Richardson Maturiy Model (vgl. [Fow10] oder Abschnitt 2.1.1.3). Dennoch ist ein flexibler und schneller Zugriff auf Services möglich und eine korrekte Nutzung des REST-Paradigmas möglich, so wie in Abschnitt 5.1 gefordert. Zur Erreichung des höchsten Levels 3 wäre eine vollständige Unterstützung von HATEOAS notwendig. Die Grundlagen dazu wurden in diesem Forschungsbeitrag geschaffen. Zur vollständigen Erfüllung wäre es notwendig die Operationen bzw. Ressourcen weiter zu verknüpfen, um Abhängigkeitsbeziehungen zwischen ihnen zu beschreiben, die dann von der Proxy-Komponente mit ausgeliefert werden können. Durch die Einbindung der OpenAPI-Beschreibungen haben Nutzer aber dennoch eine Möglichkeit, alle zur Verfügung stehenden Ressourcen-URIs und Aktionen darauf einzusehen.

Das Hinzufügen einer Abstraktionsebene hat auch Auswirkungen auf die Performanz der gesamten SOA-Umgebung und damit auf die Antwortzeiten bei Anfragen. Durch die zusätzliche Transformation der Nachrichten sind kleinere Performanz-Einbußen möglich, allerdings sind die Transformationen nicht sehr rechenintensiv. Andererseits könnte der neu eingeführte Cache der R2SMA sogar zu einer Verbesserung der Performanz und Antwortzeiten

führen. Dies gilt insbesondere für sich häufig wiederholende Anfragen oder das partielle Abrufen von Listenelementen. In solchen Fällen müssen Anfragen dann nicht erst an die Backend-Services weitergeleitet werden, sondern können direkt aus dem Cache beantwortet werden.

Zusammengefasst erlaubt die vorgestellte REST-to-SOAP Middleware Architecture Unternehmen die Weiternutzung ihrer existierenden Webservice-Infrastruktur und eröffnet gleichzeitig die Möglichkeit diese Services über REST-Schnittstellen einer neuen Zielgruppe zur Verfügung zu stellen und für moderne, REST-basierte Anwendungsfälle einzusetzen. Durch den Einsatz der Proxy-Komponenten sind dabei keinerlei Änderungen an den existierenden Services notwendig, wie in Abschnitt 5.1 gefordert. Damit haben Unternehmen die Möglichkeit kostengünstig neue Geschäftsfelder zu erschließen, um beispielsweise externen Entwicklern Zugriff auf Services zu ermöglichen und diesen nutzungsabhängig abzurechnen. Durch die Einbindung in die unternehmensweite Sicherheitsarchitektur kann zugleich sichergestellt werden, dass diese Zugriffe gesichert ablaufen.

Durch die Nutzung eines Caches wird dabei die Last auf den Backend-Systemen möglichst gering gehalten, gleichzeitig aber weiterhin ein hohes Sicherheitsniveau erhalten. Ebenfalls ermöglicht werden REST-Funktionalitäten wie das partielle Abrufen von Listen. Entsprechend sind, wenn überhaupt, nur minimale Investitionen in zusätzliche Hardware notwendig, um mit der gesteigerten Anzahl an Anfragen umzugehen. Die Integration des SOA Repositorys ermöglicht die zentrale Verwaltung von REST-APIs und klassischen Webservices, inklusive Versionierbarkeit, und bildet die Informationsgrundlage für die teilautomatische Generierung der REST-APIs.

Kapitel 6

Unterstützung des SOA-Testings

Das folgende Kapitel setzt sich mit dem Testen serviceorientierter Architekturen (SOA) auseinander und stellt ein Konzept zur Generierung von Testzeitplänen auf Basis dokumentierter Abhängigkeiten zwischen Services vor. Teile des Konzeptes wurden in einer im Rahmen dieser Forschungsarbeit konzipierten und betreuten Masterarbeit implementiert [Kra16b].

Abschnitt 6.1 gibt einen Überblick über das Testen in SOA-Verbünden. Abschnitt 6.2 stellt ein Konzept zur Erzeugung von Testzeitplänen und der Ermittlung der effizientesten Teststrategie vor. Daran anschließend werden in Abschnitt 6.3 Möglichkeiten zur Optimierung dieser Testzeitpläne vorgestellt. Abschnitt 6.4 stellt die zugehörige prototypische Implementierung vor. Eine Zusammenfassung und ein Ausblick schließen das Kapitel in Abschnitt 6.5 ab.

6.1 Testen in SOA-Verbünden

Im Rahmen einer SOA müssen bei der Umsetzung und Ausbringung von Änderungen funktionale und nicht-funktionale Anforderungen an die geänderten Komponenten getestet werden. Die besonderen Herausforderungen beim Testen einer SOA werden unter anderem von Canfora et al. [CD06; CD09], Bertolino und Polini [BP09] sowie Kalamegam und Godandapani [KG12] diskutiert. Zusammengefasst ergeben sich insbesondere die folgenden drei Herausforderungen:

- **Evolution und Dynamik:** Die Flexibilität, die eine SOA durch das Binding zur Laufzeit und den leichten Austausch von Schnittstellen bietet, kann sich beim Testen in einen Nachteil umkehren, da die Service-Consumer nicht bekannt sind und sie damit auch nicht zu Tests aufgefordert werden können.
- **Abhängigkeit vom Service-Provider und fehlende Informationen:** Service-Consumer sind stark abhängig von den Service-Providern und darauf angewiesen, dass diese Services weiterhin anbieten und auch Änderungen kommunizieren.
- **Umgang mit einer Vielzahl an Stakeholdern:** Im Rahmen einer SOA sind viele unterschiedliche Stakeholder beteiligt, die häufig konträre Standpunkte in technischen oder organisatorischen Fragen vertreten.

Für den in dieser Forschungsarbeit untersuchten Fall einer Unternehmens-SOA mit Verbundreleases gelten die genannten Herausforderungen teilweise mit Einschränkung, da ein Teil der Probleme dort nicht existiert: Die *Evolution und Dynamik* einer SOA ist im Unternehmensumfeld weniger stark ausgeprägt, da Services zwar dynamisch gebunden werden, allerdings müssen im Unternehmensumfeld vorab entsprechende Service Level Agreements geschlossen werden. Damit sind alle Consumer-Provider-Beziehungen bekannt. Die Herausforderung hier ist es, diese Beziehungen und damit einhergehende Abhängigkeiten so zu dokumentieren, dass diese nutzbar bei der Durchführung von Tests sind. Ähnliches gilt für die *Abhängigkeit vom Service-Provider*.

Diese ist im Unternehmensumfeld ebenfalls nicht so stark ausgeprägt. Da Services häufig im Unternehmen selbst entwickelt werden und mit externen Service-Anbietern entsprechende Service Level Agreements bestehen, ist ein Ansprechpartner des Service-Anbieters meist schnell verfügbar und Änderungen können an die Consumer kommuniziert werden. Allerdings ist es hierfür notwendig, wie bereits mehrmals beschrieben, eine umfangreiche und aktuelle Dokumentation der Consumer-Provider-Beziehungen verfügbar zu haben.

Bei einem Verbundrelease ergeben sich noch zusätzliche Herausforderungen, die in der oben genannten Literatur nicht zur Sprache kommen. In einem Verbund nutzen viele Systeme einerseits Services die von anderen Systemen angeboten werden. Andererseits bieten Sie wiederum selbst Services an, die von anderen Systemen des Verbunds genutzt werden. Abbildung 6.1 stellt diese Abhängigkeiten in einem SOA-Verbund an einem vereinfachten Beispiel dar. Dabei ist nicht erkennbar, ob zur Bereitstellung eines eigenen Services Daten eines fremden Services genutzt werden. In einem solchen Fall könnten sich Fehler oder fehlerhafte Daten über mehrere Services hinweg weiterverbreiten und an völlig anderer Stelle im Verbund zutage treten. Dort lässt sich dann nur noch schwer, beziehungsweise nur nach aufwändiger Fehlersuche nachvollziehen, wo der Fehler ursprünglich entstanden ist. So kann im Beispiel in Abbildung 6.1 Sys_2 zur Bereitstellung von $S_{2.2}$ Daten nutzen, die von $S_{4.1}$ aufgrund eines Software-Bugs fehlerhaft geliefert wurden. Sys_1 erkennt diesen Fehler zwar beim Testen von $S_{2.2}$, allerdings kann erst nach einer aufwändigen Fehlersuche festgestellt werden, dass der Bug von $S_{4.1}$ verursacht wurde. Wären die Abhängigkeiten der Service-Nutzung in die Testreihenfolge mit aufgenommen worden, hätte Sys_2 zunächst $S_{4.1}$ getestet und wäre bereits in diesem Test auf den Bug gestoßen.

Um also eine solche Fortpflanzung von Fehlern zu verhindern, ist es wichtig Fehler an der Quelle zu identifizieren. Dazu muss beim Testen die Abhängigkeitskette in die Testreihenfolge mit einbezogen werden. Eine manuelle Erstellung einer solchen abhängigkeitsbasierten Testreihenfolge ist aufgrund der Komplexität der Abhängigkeiten (siehe Abschnitt 2.3) in einem SOA-Verbund sehr umfangreich und sollte daher automatisiert erfolgen. Eine

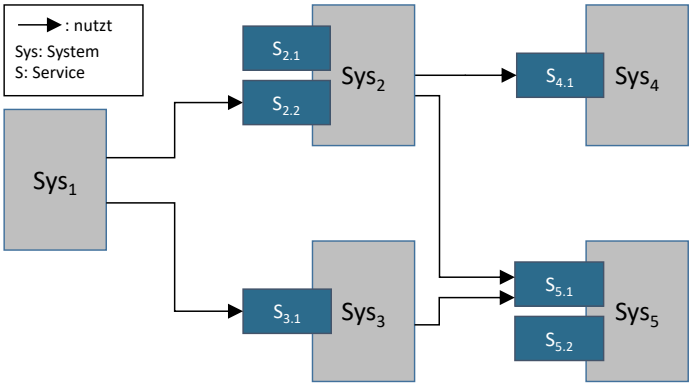


Abbildung 6.1: Abhängigkeiten zwischen Systemen und Services in einem SOA-Verbund

automatische Erstellung eines Testzeitplans ist allerdings nur möglich, wenn alle Abhängigkeiten zwischen den Services bekannt sind. Dementsprechend ist eine umfangreiche und aktuelle Dokumentation hier essentiell. Ein so generierter Testzeitplan kann dann im Anschluss von einem Test-Manager weiter verfeinert und angepasst werden.

6.2 Erzeugung von Testzeitplänen

Die Erzeugung eines Testzeitplanes wird insbesondere von zwei Faktoren beeinflusst. Dabei handelt es sich zum einen um die bereits genannten Abhängigkeiten zwischen den Verbund-Services, welche die Testreihenfolge beeinflussen. Zum anderen haben die Randbedingungen des Tests, also Veröffentlichungszeitpunkte, Testaufwand und verfügbare Testressourcen einen Einfluss. In dieser Hinsicht gleicht die Erstellung eines Testzeitplans klassischen Zeitplan-Problemen (Englisch: Timetabling Problem) [GW02]. Unter dem Begriff wird eine Klasse von Problemen verstanden, die sich damit auseinandersetzen, wie sich eine definierte Menge an Terminen auf begrenzte Ressourcen verteilen lassen. Dabei kann es sich beispielsweise um die Erstellung eines Stundenplans handeln, bei dem eine bestimmte Anzahl

an Unterrichtsstunden verteilt werden muss. Dabei darf es nicht zu Überschneidungen in der Raumbelegung kommen, ebenso wenig dürfen geltende Einschränkungen verletzt werden. So können Lehrkräfte beispielsweise im gleichen Zeitraum maximal eine Lehrveranstaltung durchführen. In den folgenden Abschnitten wird ein Konzept zur Erzeugung von Testzeitplänen vorgestellt. Dabei werden zunächst die notwendigen Rahmenbedingungen diskutiert und im Anschluss die Generierung der Testzeitpläne.

6.2.1 Ermittlung der Abhängigkeiten

Zur automatischen Generierung von Testzeitplänen müssen dementsprechend zunächst verschiedene Metadaten und Informationen erfasst worden sein. Wichtig sind dabei insbesondere die genannten Abhängigkeiten und Beziehungen zwischen den Systemen und Services eines SOA-Verbundes. Diese müssen aktuell und korrekt dokumentiert sein, damit alle Consumer-Systeme in den Test mit einbezogen werden können. Dementsprechend muss besonders für Tests von neuen oder geänderten Services die Dokumentation bereits vor der offiziellen Veröffentlichung des Services aktuell sein, es müssen also auch zukünftige Consumer-Provider-Beziehungen erfasst werden.

Das hier vorgestellte Verfahren setzt dazu auf die in Kapitel 4.3 vorgestellte SOA Governance Ontology auf und stellt diese Abhängigkeiten in einem Graphen dar. Abbildung 6.2 zeigt beispielhaft den Abhängigkeitsgraphen zum SOA-Verbund aus Abbildung 6.1. Zur Vereinfachung der Darstellung und besseren Übersicht wurde in dieser und den folgenden Abbildungen auf die Darstellung der Services verzichtet. Die Darstellung ist also zu verstehen als: Sys_A nutzt [einen Service, den] Sys_B [anbietet]. Am dargestellten Beispiel ist gut erkennbar, dass es in einem solchen Teilgraphen drei unterschiedliche Arten von Systemen geben kann: reine Provider-Systeme (orange), Systeme, die sowohl Provider- als auch Consumer-System sind (blau) und reine Consumer-Systeme (grün). Für die Tests bedeutet dies, dass reine Consumer-Systeme nur testen, Provider-Consumer-Systeme sowohl testen als auch getestet werden und reine Provider-Systeme nur getestet werden.

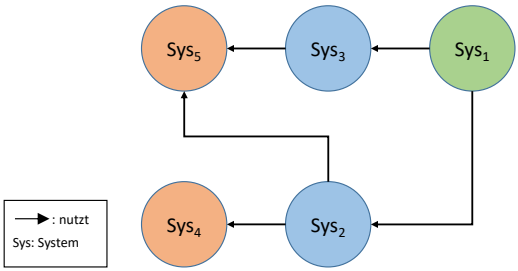
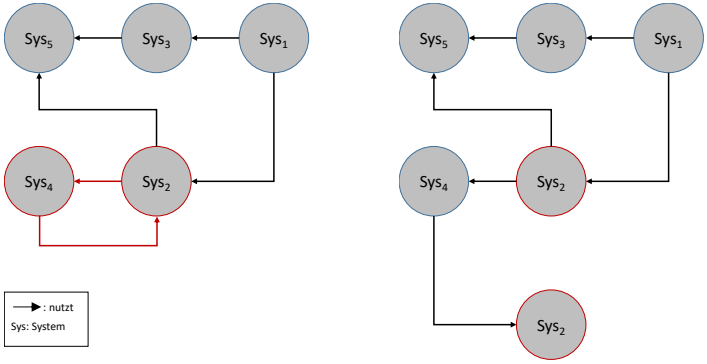


Abbildung 6.2: Abhängigkeiten zwischen Systemen in einem SOA-Verbund

Zur Erstellung des Testzeitplans werden nun jene Teilgraphen betrachtet, in denen alle geänderten Services mit ihren jeweiligen Consumer-Systemen enthalten sind. Diese Teilgraphen werden im Anschluss sortiert, traversiert und mit den gespeicherten Metadaten in Zusammenhang gesetzt. Daraus ergibt sich zunächst die Testreihenfolge und in einem weiteren Schritt der genaue Testzeitplan entsprechend der definierten Randbedingungen und Metadaten.

Da zwischen Systemen und Services beliebige Consumer-Provider-Beziehungen bestehen dürfen und auch die SOA Governance Ontology eine entsprechende Modellierung zulässt, muss damit gerechnet werden, dass es in den zur Erstellung des Testzeitplans betrachteten Teilgraphen zu Zyklen kommt. Da sich aus einem Graph mit Zyklus kein Zeitplan ableiten lässt, ist es in einem solchen Spezialfall notwendig, den Zyklus aufzulösen. Abbildung 6.3 zeigt die Auflösung eines Zyklus und den daraus resultierenden Graphen mit dupliziertem Knoten (rot hervorgehoben). Die Auflösung des Zyklus wird erreicht, indem einer der Knoten, welcher Teil des Zyklus ist, dupliziert wird. So entsteht ein zyklenfreier Graph in dem alle ursprünglichen Relationen enthalten sind. Damit können alle Systeme, die Teil des Zyklus waren, dennoch korrekt getestet werden. Zyklen können beispielsweise mittels Tiefensuche gefunden werden.



(a) Graph mit zyklischer Abhängigkeit (b) Aufgelöster Zyklus

Abbildung 6.3: Auflösen eines Zyklus

6.2.2 Bestimmung der Teststrategie

Wie aus den vorhergehenden Abschnitten zu entnehmen ist, ist das Testen eines SOA-Verbunds und einzelner Services darin eine komplexe Angelegenheit. In diesem Abschnitt wird daher eine Methode zur Vereinfachung dieser Testaufwände vorgestellt. Die Methode betrachtet zur Bewertung dabei einerseits die Kritikalität eines Services beziehungsweise des dahinter liegenden Systems und andererseits das Risiko einer durchgeführten Änderung.

Eine Änderung kann grundsätzlich unterschiedlich aufwändig getestet werden. Mittels *Funktionstest* wird die gesamte Funktionalität eines Services abgesichert, also auch die von der Änderung theoretisch nicht betroffenen Teile einem Test unterzogen. Weniger umfangreich ist ein *Regressionstest*, bei dem existierende Testfälle erneut ausgeführt werden um sicherzustellen, dass diese auch für die neue Version des Services die selben Ergebnisse liefern wie bei vorherigen Testläufen [Lig09]. Ein *Akzeptanztest* wird abschließend durchgeführt um sicherzustellen, dass ein Service die geforderten Funktionalitäten abdeckt.

Je nachdem wie stark, beziehungsweise ob ein Consumer-System von

einer Service-Änderung betroffen ist, sollte demnach auch Auswirkungen auf die Teststrategie haben. Da allerdings häufig nicht bestimmt werden kann welche Auswirkungen zu befürchten sind, werden zur Absicherung meist von jedem Consumer ausführliche Funktionstests erwartet. Zur effizienteren Nutzung der vorhandenen Ressourcen und der Minimierung der Testaufwände bei gleichzeitig hoher Testqualität soll für jede Consumer-Provider-Testbeziehung die am besten geeignete Teststrategie gefunden werden. Gestützt auf eine ausführliche Dokumentation der Services, Systeme und ihrer Abhängigkeiten, lässt sich diese mit der im Folgenden geschilderten Methode ermitteln.

Die Teststrategie TS für ein System s nach einer Änderung c ergibt sich aus folgender Formel, die in den folgenden Abschnitten detailliert vorgestellt wird:

$$TS_s = ((C_s + A_s) + max(C_{consumer_n})) * R_c$$

mit

$$A_s = \begin{cases} 0 & \text{wenn } s \text{ nicht betroffen} \\ 1 & \text{wenn } s \text{ betroffen} \end{cases}$$

Dabei beschreibt C_s die individuelle Systemkritikalität, $C_{consumer_n}$ beschreibt die Systemkritikalität eines Consumer-Systems n des Services s.

Zur Festlegung der Systemkritikalität werden Systeme hinsichtlich ihrer Verfügbarkeitsanforderungen bewertet, also wie wichtig sie für die reibungslose Durchführung der Geschäftsprozesse sind. Hier wird von folgender Einteilung ausgegangen:

- Kategorie A: Missionskritisches System. Selbst kurze Ausfälle haben starke Auswirkungen auf den reibungslosen Ablauf wichtiger Geschäftsprozesse.
- Kategorie B: Hauptsystem. Kurze Ausfälle sind anderweitig überbrückbar und führen nicht zum Ausfall wichtiger Geschäftsprozesse.

- Kategorie C: Nebensystem. Auch längere Ausfälle können überbrückt werden, das System erfüllt keine Aufgabe oder liefert keine Daten, die für die Durchführung relevanter Geschäftsprozesse wichtig sind.

Für die Bestimmung der Teststrategie werden folgende Werte für die unterschiedlichen Level vergeben: $A = 3$, $B = 2$, $C = 1$

Sofern ein System direkt von einer Änderung betroffen ist, es also den geänderten Service nutzt, erhöht dies die individuelle Kritikalität um 1. Nicht betroffen ist ein System, wenn es zwar einen Service des geänderten Systems nutzt, allerdings nicht den Service, an dem die Änderung durchgeführt wurde. Ebenfalls mit einbezogen werden Consumer des testenden Systems. Dies ist notwendig, da ein System, das als *C*-System eingestuft ist, von Systemen der Kategorie *A* genutzt werden könnte, wodurch ein Ausfall des *C*-Systems unter Umständen Auswirkungen auf die *A*-Systeme haben kann. In solch einem Fall muss auch das *C*-System einen umfangreichen Test des geänderten Services durchführen, damit sichergestellt werden kann, dass es nicht selbst durch einen nicht entdeckten Fehler ausfällt. Hierzu wird die maximale Kritikalität der Consumer-Systeme von s herangezogen und analog zur individuellen Kritikalität bewertet.

Als letzter Faktor wir das Risiko der Änderung $R_c \in [1..2]$ mit einbezogen. Je umfangreicher die Änderung, desto höher der Wert für R_c. Das Hinzufügen eines weiteren Parameters zu einer Serviceschnittstelle ist beispielsweise deutlich weniger risikobehaftet als die komplette Neuimplementierung der Geschäftslogik eines Services.

Mit diesen Daten lässt sich nun die Teststrategie TS für einen Service s bestimmen:

$$TS_s \begin{cases} < 4 & \Longrightarrow \text{Regressionstest} \\ \geq 4 & \Longrightarrow \text{Funktionaler Test} \end{cases}$$

In jedem Fall müssen Consumer des geänderten Services einen Akzeptanztest durchführen. Zusammengefasst ergibt sich dadurch, dass nicht mehr alle Systeme einen funktionalen Test durchführen, ein deutlich verminderter

Gesamtaufwand beim Testen. Dies führt insgesamt zu einer effizienteren Teststrategie und gezielten Nutzung der Testressourcen.

6.2.3 Randbedingungen und Einschränkungen

Verschiedene Randbedingungen und Einschränkungen haben Einfluss auf die Erstellung des Testzeitplans. Diese reichen vom Umfang der Testaufwände hin zur Verfügbarkeit von Testern und Testressourcen.

6.2.3.1 Testaufwände

Zur Erstellung eines Testzeitplanes muss eine Abschätzung des voraussichtlichen Testaufwands für die zu testenden Services vorliegen. Dieser hängt von der Art der Tests ab, die durchgeführt werden und muss im Vorfeld in Abstimmung mit den zuständigen Test-Managern, welche die Durchführung der Tests verantworten, festgelegt werden. Diese sind auf Basis von Erfahrungswerten in der Lage, die Aufwände abzuschätzen.

6.2.3.2 Test-Ressourcen

Zur Durchführung eines Tests ist es erforderlich, die benötigten Testressourcen zur Verfügung zu haben. Dazu ist es notwendig, frühzeitig einen Testplan zu erstellen und diese benötigten Ressourcen einzuplanen. Insbesondere handelt es sich dabei um:

- **Personal:** In größeren Unternehmen gibt es häufig spezialisierte Testabteilungen, die als interne Dienstleister das Testen von Software übernehmen. Diese müssen dazu extra beauftragt werden und stehen im Allgemeinen nicht kurzfristig zur Verfügung. Ebenfalls sind auch andere Mitarbeiter, die Testaufgaben übernehmen, häufig nicht durchgängig verfügbar.
- **Infrastruktur:** Ziel ist es immer, Systeme und deren Services auf Hardware und Infrastruktur zu testen, die der Produktivumgebung möglichst nahe kommt. Bei komplexen hochverfügbaren Systemen ist die

Produktivinfrastruktur umfangreich, also muss dementsprechend auch die Testumgebung umfangreich sein und steht gegebenenfalls nicht immer zur Verfügung.

- **Testfälle:** Testfälle müssen bereits während der Entwicklung einer Software-Komponente parallel geschrieben werden und sollten nicht erst nach der Entwicklung entstehen. Dementsprechend ist es wichtig, dass entsprechende Entwicklungskapazitäten bereits während der Entwicklung dafür bereit stehen.

6.2.4 Generierung von Testzeitplänen

Auf Basis der bisher diskutierten Daten, Informationen und Methoden kann nun der tatsächliche Testzeitplan generiert werden. Dazu werden die ermittelten Teilgraphen traversiert und mit den Abhängigkeiten, der ermittelten Teststrategie, den Testaufwänden und Testressourcen in Verbindung gesetzt. Die Traversierung startet wie in Abbildung 6.4 dargestellt ausgehend von reinen Provider-Systemen (vgl. Abschnitt 6.2.1), also Knoten mit nur eingehenden Kanten. Die Testschritte sind in der Abbildung durchnummeriert. Diese Testreihenfolge ist notwendig, da sich Fehler umgekehrt zur Nutzungsbeziehung fortpflanzen.

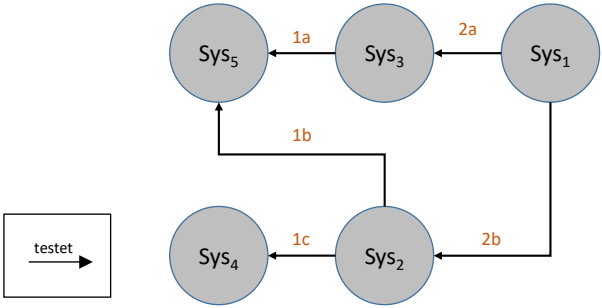


Abbildung 6.4: Traversierung des Testabhängigkeitsgraphen

Für den aktuell bearbeiteten Knoten ergibt sich auf Basis der gewählten Teststrategie und den zuvor definierten Testaufwänden der Endzeitpunkt

für die jeweilige Testperiode und der Startzeitpunkt für die Testperiode der Nachfolgeknoten. So wird weiter vorgegangen, bis alle Knoten abgearbeitet wurden.

Nach Abschluss der Generierung kann ein Testmanager den Testplan betrachten und manuell anpassen, beziehungsweise nach Veränderung einzelner Parameter oder Randbedingungen den Testzeitplan erneut generieren lassen. Ebenfalls können weitere Optimierungen des Testzeitplan durchgeführt werden, die im folgenden Abschnitt näher beschrieben werden.

6.3 Optimierung von Testzeitplänen

Ziel der Optimierung von Testzeitplänen ist es, die vorab generierten Testzeitpläne hinsichtlich weiterer Kriterien zu optimieren und möglichst individuell an die zur Verfügung stehende Zeit anzupassen. Dabei soll es möglich sein mehrere Optimierungen durchzuführen und im Anschluss die unterschiedlichen Testzeitpläne gegenüberzustellen.

6.3.1 Komplette Nutzung der Testperiode

Standardmäßig werden die Testperioden so knapp wie möglich geplant, um den Gesamttestzeitraum möglichst kurz zu halten. Wenn nun zwei Systeme A und B parallel den gleichen Service testen und System A einen deutlich höheren Testaufwand vorsieht als System B, führt dies dazu, dass der Gesamttestzeitraum auf Basis von System A definiert wird. Für System B wird ein deutlich kürzerer Zeitplan definiert, basierend auf dem niedrigeren Testaufwand. Da der von System A und B getestete Service selbst erst testen kann, wenn beide Tests abgeschlossen sind, hätte theoretisch auch System B die Möglichkeit, seine Tests über einen längeren Zeitraum zu verteilen.

Um dies im Testzeitplan widerspiegeln zu können, kann mittels dieser Optimierung der Testzeitraum für alle Systeme auf das Maximum aller Testperioden für einen bestimmten Service erhöht werden. Abbildung 6.5 verdeutlicht das Optimierungsverfahren. Durch die Expansion der Testperioden kann der Zeitplan ohne Auswirkung auf die Gesamttestdauer entzerrt

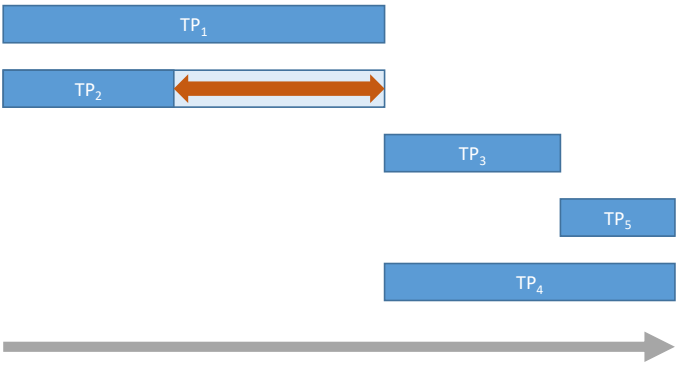


Abbildung 6.5: Komplette Nutzung einer Testperiode

werden. Tester haben so mehr Flexibilität ihre Tests verteilt über einen längeren Zeitraum durchzuführen.

6.3.2 Verschiebung der Testperiode

Das oben vorgestellte Vorgehen berechnet die Testperioden ausgehend von einem definierten Startdatum. Unter Umständen ist es nach Generierung eines Testplans aber notwendig, Gesamtstart oder -ende zu einem anderen Zeitpunkt zu verschieben. Dies kann beispielsweise der Fall sein, wenn aufgrund der Abhängigkeiten und Randbedingungen das Testende nach dem geplanten Veröffentlichungsdatum liegt. Ebenfalls kann sich eine Verschiebung anbieten, wenn zwischen Ende des Testzeitraums und Veröffentlichungsdatum mehr Zeit liegt denn als Puffer eingeplant werden muss.

6.3.3 Parallelisierung der Tests

Bei bestimmten Konstellation von geänderten Services kann es unter Umständen dazu kommen, dass einzelne Testpfade wie in Abbildung 6.6 gezeigt deutlich länger als andere sind. Bei gleichzeitig langer Testdauer kann dies zu sehr langen Testzeiträumen führen, die sich nicht mehr mit den Rahmenbedingungen für einen Test vereinbaren lassen. Hier kann es unter kriti-

scher Abwägung der Nebenwirkungen notwendig sein, bestimmte eigentlich sequentiell geplante Tests zu parallelisieren. Da dabei die Testreihenfolge aufgebrochen wird, könnte es zur eingangs genannten unentdeckten Fortpflanzung von Fehlern kommen, die durch die strenge Einhaltung der Testreihenfolge eigentlich verhindert werden sollte. Das Aufbrechen des Abhängigkeitsgraphen sollte also mit Bedacht eingesetzt werden und möglichst an einer Stelle erfolgen, die keine missionskritischen Systeme betrifft.

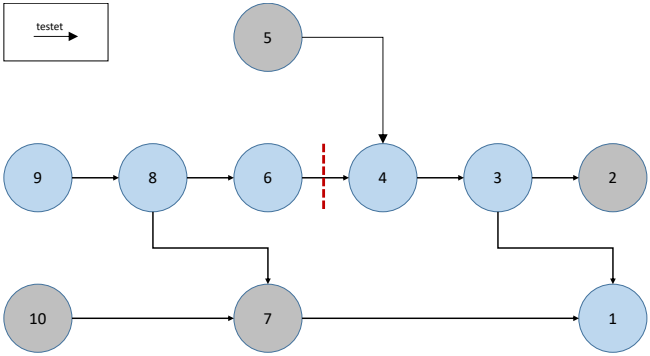


Abbildung 6.6: Graph der Testabhängigkeiten

Zur Umsetzung dieser Optimierung wird der Abhängigkeitsgraph durch des Entfernen einer Consumer-Provider-Beziehung aufgeteilt, wie in Abbildung 6.6 dargestellt. Blau hervorgehoben ist in der Abbildung der kritische Pfad, also der Pfad mit der höchsten Summe an Testaufwänden. Dieser Pfad wird aufgetrennt, mit dem Ziel die Gesamttestdauer zu verringern. Bedingung für die Auftrennung des Graphen ist, dass die Testzeit eines langen sequentiellen Pfades auch deutlich über der weiterer Pfade liegt. Andernfalls ergibt sich durch die Auftrennung kein Vorteil. Zu beachten ist auch, dass entsprechende Testressourcen, wie beispielsweise Testrechner verfügbar sein müssen, ansonsten kann der erwartete Zeitvorteil durch die mangelnde Ressourcenverfügbarkeit wieder zunichte gemacht werden.

Zur Auftrennung wird zunächst der kritische Pfad bestimmt, also der Pfad im Graphen mit mehreren Knoten und der längsten Testdauer. Für diesen Pfad wird dann der beste Schnittpunkt ermittelt. Zur Erreichung

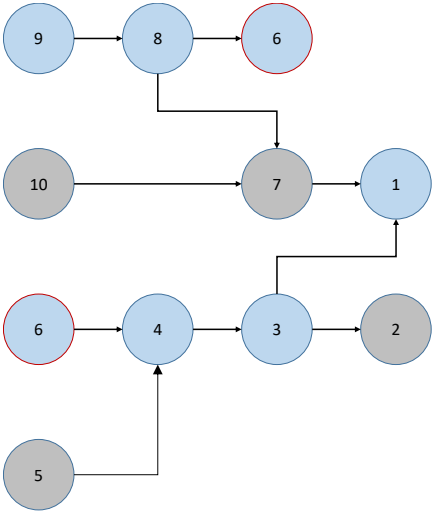


Abbildung 6.7: Aufgeteilter Graph. Untereinander angeordnete Systeme werden ungefähr gleichzeitig getestet.

des größtmöglichen Nutzens muss die Teilung möglichst nah am zeitlichen Mittelpunkt des kritischen Pfades durchgeführt werden (in Abbildung 6.6 mit einer roten gestrichelten Linie dargestellt). Dieser kann zunächst durch eine einfache Aufsummierung der einzelnen Testdauern gefunden werden. In einem zweiten Schritt muss der Schnittpunkt gegebenenfalls so verschoben werden, dass die Auftrennung zwischen C-Systemen, also unwichtigeren Nebensystemen, erfolgt. Dadurch kann das Risiko verringert werden, dass Fehler an wichtigen Systemen nicht oder nicht rechtzeitig entdeckt werden.

Abbildung 6.7 zeigt den geänderten Graphen nach der Auftrennung. Die Systeme des ursprünglichen kritischen Pfades sind auch hier wieder blau dargestellt. Gut zu erkennen ist, dass die beiden Teilstränge nun parallel getestet werden. Da auch die entfernte Abhängigkeit getestet werden muss, ist das System, vor dem getrennt wurde, im resultierenden Abhängigkeitsgraphen doppelt vorhanden - einmal als testendes System, einmal als zu testendes System (*System 6*, rot umrandet in Abbildung 6.7).

6.3.4 Bewertung und Vergleich von Testzeitplänen

Die Anwendung der unterschiedlichen Optimierungen, aber auch die Anpassung der Randbedingungen und Einschränkungen vor der Generierung eines Testzeitplans wirken sich direkt auf die erstellten Testzeitpläne aus. Um mehrere erstellte Testzeitpläne des gleichen Verbundtests vergleichen zu können, ist es notwendig diese anhand bestimmter, quantifizierbarer Kenngrößen bewerten zu können. Hierzu werden die folgenden Kenngrößen definiert:

- Länge Testzeitraum
- Anzahl Testphasen im kritischen Pfad
- Grad der Parallelität
- Ausnutzung der Testressourcen

Ein Vergleich dieser Kenngrößen für mehrere Testzeitpläne kann die Entscheidung für einen der Pläne unterstützen. Zur Individualisierung dieser Entscheidung können die Kenngrößen gewichtet werden. Bei sowieso hoher Verfügbarkeit der Testressourcen kann es beispielsweise wichtiger sein, dass die Tests möglichst schnell durchgeführt werden, wohingegen bei einer flexiblen Zeitplanung oder komplexen Tests mehr Wert auf Ressourcen-Sparsamkeit gelegt wird.

6.4 Implementierung und Anwendungsfall

Das vorgestellte Konzept wurde in Teilen im SOA Governance Repository (SGR) implementiert ([Kra16b]) und wird im Folgenden kurz vorgestellt.

Zur Erstellung eines Testzeitplans muss der Nutzer zunächst ein Verbundrelease erstellen und diesem eine Menge von geänderten Serviceversionen zuordnen. Im Anschluss kann dann ein Testzeitplan generiert und optimiert werden. Für die Generierung wird auf die im SGR gespeicherten Metadaten zu Services, Serviceversionen, Abhängigkeiten und Testaufwänden zurückgegriffen. Komplexere Einschränkungen durch die limitierte Verfügbarkeit von

Testressourcen wie Tester oder Testumgebungen wurden nicht umgesetzt. Dazu wäre der Einsatz von komplexeren Solvern, also spezieller mathematischer Software, wie beispielsweise OptaPlanner [1] notwendig. Ebenfalls gibt es Ansätze zur Lösung von Zeitplan-Problemen mit genetischen Algorithmen [CDM92; AT08].

Dem Nutzer wird dann der Testzeitplans grafisch dargestellt, wie Abbildung 6.8 beispielhaft zeigt. Außerdem implementiert wurden die oben vorgestellten Optimierungen, somit ist es möglich die generierten Testzeitpläne im Anschluss auszudehnen, zu verschieben und zu parallelisieren.

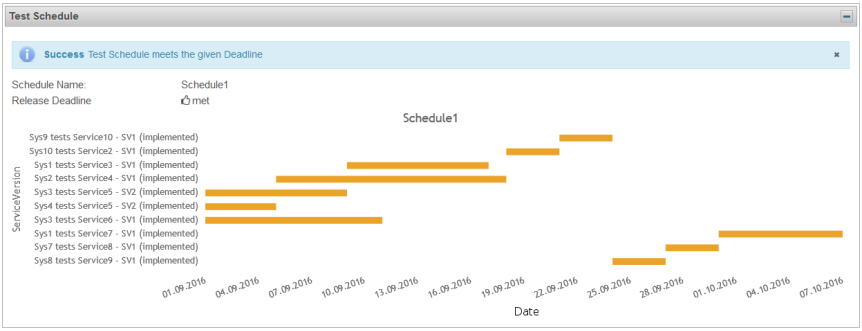


Abbildung 6.8: Darstellung eines Testzeitplans im SGR

Es wurden Versuche mit einem anonymisierten realen Datensatz des Anwendungspartners durchgeführt. Dazu wurde aus dem Datensatz zufällig eine Menge an Services als geändert markiert und in ein Verbundrelease aufgenommen. Für diese Verbundreleases wurden dann mit zufällig erzeugten Aufwandsschätzungen Testzeitpläne generiert und die implementierten Optimierungsverfahren angewendet. In wiederholten Messungen benötigte die Generierung der Testzeitpläne im Durchschnitt knapp unter 300 Millisekunden. Die durchgeführten Optimierungen benötigen jeweils im Durchschnitt circa drei Sekunden.

[1] http://www.optaplanner.org

6.5 Zusammenfassung und Ausblick

Die in diesem Abschnitt vorgestellten Konzepte und Modelle unterstützen Tester eines SOA-Systemverbunds bei der zielgerichteten und effizienten Testplanung und -durchführung. Durch die auf Basis existierender Abhängigkeiten erstellten Testreihenfolge lässt sich die Fortpflanzung von Fehlern über mehrere Systeme hinweg eindämmen, da die Fehler direkt an der Quelle entdeckt werden können. Durch die aus dem Verbund abgeleitete Teststrategie werden Services und Systeme nur so umfangreich wie wirklich notwendig getestet. Systeme, die von einer Änderung nicht betroffen sind oder bei denen ein kurzfristiger Ausfall weniger Auswirkungen hätte, werden dementsprechend weniger umfangreich abgesichert.

Eine Testreihe mit anonymisierten Daten des SOA-Verbundes des Kooperationspartners zeigte zuverlässig valide Ergebnisse in der Generierung und Optimierung der Testzeitpläne. Insbesondere die vorgestellten Optimierungen erlauben eine individuelle Anpassung der Zeitpläne an die Randbedingungen einzelner Releases.

Kapitel 7

SOA Governance Repository

Die Herausforderungen bei der Einführung und der Verwaltung einer serviceorientierte Architektur (SOA) wurden in den bisherigen Kapiteln bereits ausführlich diskutiert. Durch die Einführung von Governance-Prozessen lassen sich bereits viele Fallstricke umgehen, zur wirklich effizienten Steuerung ist allerdings aufgrund der vorliegenden Komplexität eine Softwareunterstützung notwendig. Diese dient insbesondere dazu, die unterschiedlichen Stakeholder möglichst gut bei ihrer täglichen Arbeit in den SOA-Prozessen zu unterstützen. Mit dem im Folgenden vorgestellten SOA Governance Repository (SGR) wurde im Rahmen dieser Arbeit der Prototyp eines SOA Governance Informationssystems entwickelt, das dabei helfen soll, die Geschwindigkeit und Agilität von SOA-Prozessen zu verbessern, indem Stakeholder bei der Durchführung ihrer Arbeit bestmöglich unterstützt werden.

In Abschnitt 7.1 werden zunächst die Anforderungen sowie Randbedingungen zur Nutzung semantischer Technologien vorgestellt. Anschließend wird in Abschnitt 7.2 die Architektur und Implementierung des SGR vorgestellt.

Abschnitt 7.3 beleuchtet die Nutzung semantischer Technologien im SGR. In Abschnitt 7.4 wird ein Vergleich des SGR mit kommerziellen Produkten durchgeführt, bevor Abschnitt 7.5 das Kapitel mit einer Zusammenfassung und einem Ausblick abschließt.

7.1 Architekturentscheidungen und Randbedingungen

Dieser Abschnitt diskutiert Architekturentscheidungen und Anforderungen, welche die Implementierung des SGR erfüllen soll. Ziel ist es, ein zentrales SOA Governance Informationssystem zu entwickeln, das sowohl von technischen als auch von fachlichen Anwendern genutzt und auf ihre jeweiligen Bedürfnisse individuell zugeschnitten werden kann, beispielsweise durch die Erstellung personalisierter Sichten. Um dies zu erreichen, muss eine Implementierung die folgenden Anforderungen A_1–A_5 erfüllen.

A_1: **Zentrales Verwaltungssystem für SOA-Artefakte:** Das SGR soll als zentrales System der SOA-Landschaft dienen und für alle SOA-Artefakte als primäre Anlaufstelle dienen. Dazu ist es notwendig, dass das SGR einen konsistenten, aktuellen und vollständigen Zugriff auf alle SOA-Artefakte ermöglicht.

A_2: **Interaktives System:** Das SGR soll Stakeholder einer SOA mit einem Werkzeug ausstatten, das es ihnen erlaubt die SOA-Landschaft ihres Unternehmens interaktiv zu erkunden. Dadurch wird Stakeholdern eine bessere Übersicht über die System- und Servicelandschaft ermöglicht. Grafisch lässt sich auch die vorhandene Komplexität besser darstellen und handhaben. Eine ausführliche Informationsbereitstellung kann unter Umständen auch die erneute Entwicklung bereits existierender Funktionalität verhindert werden, da existierende Services leichter auffindbar sind.

A_3: **Personalisierbares (Management-)Informations-System:** Zusätzlich zu den bereits genannten interaktiven Funktionen soll das SGR auch Funktionalitäten bieten, die es den Stakeholdern erlauben den aktuellen Zustand des SOA-Verbundes einzusehen und zu beurteilen.

Die Software soll dazu anpassbar, also personalisierbar sein. Dies ist notwendig, da Stakeholder einer SOA eine Vielzahl an unterschiedlichen Rollen und damit Aufgaben haben, zu deren Erfüllung teils sehr unterschiedliche Informationen benötigt werden.

A_4: **Interoperabilität mit anderen Systemen:** Zur optimalen und effizienten Unterstützung von Governance-Prozessen, aber auch zur Etablierung eines Corporate Semantic Web ist eine der wichtigsten Anforderungen die Ermöglichung des Datenaustausches zwischen Anwendungen. Der Datenaustausch muss dabei sowohl zwischen semantisch unterstützen Anwendungen, also auch zwischen semantisch unterstützten und klassischen Anwendungen sowie weiterhin zwischen klassischen Anwendungen untereinander ermöglicht werden. Mit dem Ziel, das SGR als zentrales Informationssystem einer SOA zu etablieren, ist es unabdingbar, dass Schnittstellen angeboten werden, die es anderen Anwendungen erlauben mit dem SGR zu interagieren und Daten auszutauschen. Genauso muss das Repository auch die Daten aus anderen Anwendungen abfragen und weiterverarbeiten können.

A_5: **Informationsgewinnung:** Das SGR und die Informationen, die damit verwaltet werden, haben nur einen Nutzen, wenn sie in den Governance-Prozessen aktiv eingesetzt werden und Stakeholder dabei unterstützt werden effizient ihre Aufgaben zu erledigen. Dies ist insbesondere nur dann zu erreichen, wenn die Nutzer das Gefühl haben einen Mehrwert und eine gute Unterstützung der Prozesse durch das SGR zu erhalten. Dazu muss es möglich sein, aus den im SGR gespeicherten Informationen neue Erkenntnisse zu erzielen.

Da sich die genannten Anforderungen mit semantischen Technologien (vgl. Abschnitt 2.1.2) sehr gut umsetzen lassen, wurde die Entscheidung getroffen, das SGR auf dieser Basis prototypisch zu implementieren. Die folgenden Abschnitte geben einen Überblick über die Eigenschaften semantischer Anwendungen und notwendige Abwägungen zur Implementierung von Unternehmensanwendungen mit semantischen Technologien.

7.1.1 Eigenschaften semantischer Anwendungen

Semantische Anwendungen bringen diverse Vorteile mit sich, die sich auch die Implementierung des SGR zunutze machen soll. Diese werden im Folgenden diskutiert.

Offenheit, Erweiterbarkeit und Flexibilität Ressource Description Framework (RDF) definiert Tripel, bestehend aus *Subjekt*, *Prädikat* und *Objekt*. Eine Menge dieser Tripel ergibt, wenn untereinander in Beziehung gesetzt, einen Graphen [W3C14a]. Als solcher ist ein RDF-Modell niemals wirklich vollständig, sondern kann leicht erweitert oder mit anderen Graphen in Verbindung gebracht werden. Dies ist das zentrale Konzept des Semantic Web – die Schaffung von Verbindungen zwischen Informationen, so dass es „[...] über Anwendungs-, Unternehmens- und Gemeinschafts-Grenzen hinweg geteilt und wiederverwendet werden kann [...]“ [W3C13b].

Dadurch ergibt sich auch, dass ein RDF-Graph als Datenmodell nicht durch ein definiertes Schema begrenzt ist, wie es in einem relationalen Datenbanksystem der Fall ist. Durch diese Schemalosigkeit kann das Datenmodell semantischer Applikationen auch zur Laufzeit angepasst und weiterentwickelt werden, indem beispielsweise neue Datentypen oder Relationen zwischen existierenden Daten eingeführt werden. Ebenso kann das Datenmodell erweitert werden, wenn sich neue Anwendungsszenarien für die Applikation ergeben. Auch in der Entwicklung von Unternehmensanwendungen ergeben sich Vorteile, ganz besonders, wenn iterative oder agile Softwareentwicklungs-Methoden zum Einsatz kommen. So kann das zugrundeliegende Graph-Datenmodell mit jeder Iteration der Software erweitert werden, ohne dass sich große Auswirkungen auf bereits umgesetzte Bereiche der Anwendung ergeben. Entsprechende Softwareentwicklungs-Methoden kommen bei umfangreichen Unternehmensanwendungen häufig zum Einsatz, da sich dadurch schnell Kernfunktionalität umsetzen lässt, die dann in nachfolgenden Iterationen erweitert wird.

Ähnlichkeit zu objektorientierten Modellen Zwischen einem semantischen RDF-Modell und einem objektorientierten Modell existieren gewisse Gemeinsamkeiten. So bestehen beide Modelle aus Klassen, Eigenschaften und Instanzen. Klassen lassen sich in beiden Fällen mittels Vererbung zu einer Klassenhierarchie verbinden und können durch Eigenschaften beschrieben werden. Diese Eigenschaften haben als Wert entweder andere Klassen oder primitive Datentypen, bzw. Literale [KOTW06]. Hier finden sich allerdings auch kleinere Unterschiede – in RDF ist eine Eigenschaft alleinstehend und kann an mehr als einer Klasse gebunden sein. In einem objektorientierten Modell hingegen ist eine Eigenschaft immer Teil genau einer Klasse. Diese Gemeinsamkeiten erlauben ein leichteres Matching zwischen RDF-Modell und objektorientierter Anwendung und es kommt somit nicht zu einem *object-relational impedance mismatch*, wie es beim Einsatz relationaler Datenbanken der Fall ist.

Wiederverwendbarkeit und Interoperabilität Die Offenheit eines semantischen RDF-Datenmodells bzw. einer Ontologie bedeutet auch, dass sie für andere Zwecke wiederverwendet werden kann, falls sie nicht zu domänenspezifisch ist. Eine allgemein gehaltene Ontologie, wie etwa *friend of a friend* (vergleiche Abschnitt 4.3), kann in jeder Domäne eingesetzt werden, die Personen und Beziehungen zwischen ihnen beschreiben möchte. Dahingegen kann eine Ontologie eines Automobilherstellers in unterschiedlichen Anwendungen des Herstellers und eventuell noch von anderen Automobilherstellern genutzt werden. Für einen Einsatz in der Pharmabranche wird sie allerdings ungeeignet sein. Ontologien werden häufig von einem Gremium aus unterschiedlichen Domänenexperten entworfen. Dadurch ergibt sich automatisch ein gewisser Grad an Abstraktion, da sonst keine Einigung erzielt werden kann. Dies erlaubt dann dementsprechend auch eine bessere Wiederverwendung der Ontologie.

Suchen und semantisches Reasoning Der Einsatz von Ontologien und RDF-Datenmodellen in Unternehmensanwendungen ermöglicht neue An-

sätze zum Durchsuchen von Daten und Auffinden neuer Informationen. Im vorliegenden Anwendungsfall könnte beispielsweise die „Wichtigkeit“ eines Mitarbeiters für den SOA-Verbund ermittelt werden, indem die ausgehenden Kanten des Objekts, das ihn repräsentiert, zu anderen SOA-Artefakten gezählt werden. Ebenso können Reasoning-Verfahren genutzt werden, um implizite Unterklassen-Beziehungen abzuleiten oder ein Datenmodell auf Konsistenz zu überprüfen.

7.1.2 Abwägungen zur Implementierung semantischer Anwendungen

Bei der Implementierung einer semantischen Anwendung im Unternehmensumfeld sollten vorab einige Punkte bedacht werden.

Performanz Traditionelle relationale Datenbanksysteme wurden über Jahrzehnte hinweg unzählige Male optimiert und können Lese- und Schreiboperationen selbst für komplexeste Anfragen immer noch performant handhaben. Tripel- bzw. Graph-Datenbanken allgemein haben durch ihr vergleichsweise junges Alter noch deutlich mehr Optimierungspotential, da sie noch nicht alle Anfragen optimal verarbeiten können. Aktuelle Benchmarks zeigen jedoch, dass Graphdatenbanken bereits heute auch bei mehreren Billionen Tripeln angemessene Antwortzeiten erreichen können [Ora14; VMS12].

Für den Anwendungsfall des SOA Governance Repository sind an dieser Stelle keine Probleme zu erwarten, da selbst bei einem umfangreichen SOA-Verbund die Anzahl der Tripel deutlich niedriger liegt. Bei einem Testdatensatz mit knapp 500 gespeicherten Artefakten müssen weit unter 10000 Tripel gespeichert werden.

Objekt-Tripel-Mapping Bei der Implementierung von Anwendungen auf Basis von Tripel-Speichern müssen die Tripel in der verwendeten Programmiersprache abgebildet werden. Wie oben beschrieben ist dies durch die Ähnlichkeit zwischen objektorientiertem und RDF-Modell gut realisierbar. Dennoch ist es notwendig, die Flexibilität und Erweiterung des Datenmodells mit der Anwendung in Einklang zu bringen. Möglichkeiten um dies zu

realisieren werden in [KOTW06] und [QSM09] vorgestellt. Knublauch et al. empfehlen den Einsatz von Quellcodegeneratoren, um aus Ontologien ein objektorientiertes Klassenmodell zu erzeugen [KOTW06]. Quasthoff et al. beschreiben die Abbildung eines RDF-Modells auf ein objektorientiertes Modell und stellen eine zugehörige Implementierung vor [QSM09].

Für die Umsetzung des SOA Governance Repository verspricht ein hybrider Ansatz den besten Erfolg. Damit wäre es möglich, die Teile der Anwendung, an deren Datenmodell voraussichtlich nur minimale Änderungen zu erwarten sind, eng an das Datenmodell zu knüpfen. Dies hat allerdings den Nachteil, dass Änderungen auch immer parallel in der Anwendung umgesetzt werden müssen. Andererseits kann nur so eine Benutzeroberfläche realisiert werden, die den Nutzer angemessen führt und Informationen ansprechend darstellt. An anderen Stellen können dann definierte Erweiterungspunkte geschaffen und das Datenmodell dynamisch gehandhabt werden.

Entwicklerkompetenzen Zu guter Letzt ist auch der menschliche Faktor nicht zu vernachlässigen, da dieser starken Einfluss auf die Umsetzbarkeit haben kann. In Unternehmen sind noch häufig keinerlei Kompetenzen auf dem Gebiet semantischer Anwendungen vorhanden. Diese müssen zunächst entwickelt werden, entweder durch die Fortbildung der Mitarbeiter oder dem Einkaufen dieser Kompetenzen in Form neuer Mitarbeiter bzw. externer Dienstleister. Ohne das Vorhandensein dieser Kompetenzen ist ein Entwicklerteam nicht in der Lage semantische Anwendungen mit dem gleichen Level an Zuverlässigkeit und Robustheit zu entwickeln, wie es das bei konventionellen Anwendungen kann.

7.2 Prototypische Implementierung des SGR

Dieser Abschnitt präsentiert zunächst die logische Architektur des SOA Governance Repository und stellt dann die technische Realisierung des Prototypen vor.

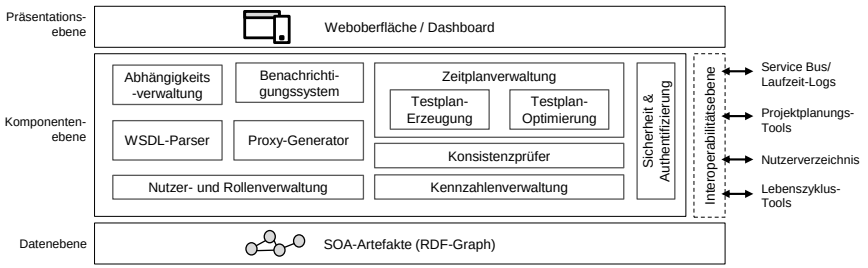


Abbildung 7.1: Logische Architektur des SOA Governance Repository in Anlehnung an [KM16]

7.2.1 Logische Architektur

Der folgende Abschnitt gibt einen Überblick über die logische Architektur der prototypischen Implementierung des SGR. Das SGR ist ebenfalls ein integraler Bestandteil der in Abschnitt 2.3 vorgestellten Stuttgart IT Architecture for Manufacturing (SITAM) zur Realisierung einer datengetriebenen Fabrik.

Abbildung 7.1 zeigt die logische Architektur des SGR mit vier Ebenen: *Datenebene*, *Komponentenebene*, *Interoperabilitätsebene* und *Präsentationsebene*, die in den folgenden Abschnitten im Detail vorgestellt werden.

7.2.1.1 Datenebene

Die Datenebene bietet eine Abstraktion der im RDF-Tripel-Speicher gespeicherten Daten des SGR. Als Basis für die Speicherung der Daten kommt die in Abschnitt 4.3.3 vorgestellte SOA Governance Ontology zum Einsatz. Alle im Tripel-Speicher abgelegten SOA-Artefakte sind Instanzen der in der Ontologie beschriebenen Klassen. Artefakte sind insbesondere Services und Serviceversionen, zugehörige Business Objects und ihre jeweiligen Stakeholder.

7.2.1.2 Komponentenebene

Die Komponentenebene enthält alle logischen funktionalen Komponenten des SGR. Die folgenden Abschnitte stellen die wichtigsten Komponenten und ihre Funktionalitäten kurz vor und verweisen gegebenenfalls auf die vorherigen Kapitel, in denen die prototypische Implementierung der jeweiligen Komponente vorgestellt wurde.

Nutzer- und Rollenverwaltung Wie bereits diskutiert sind eine Vielzahl von Stakeholdern in die Nutzung einer SOA involviert. Diese Personen benötigen Zugriff auf die vom SGR zur Verfügung gestellten Informationen entsprechend ihrer Aufgaben und Verantwortlichkeiten, also entsprechend der Rollen, die sie innehaben. Um dies zu realisieren ist eine feingranulare Nutzer- und Rollenverwaltung notwendig. Die Rollenverwaltung des SGR unterstützt dazu zwei unterschiedliche Arten von Rollen:

- Allgemeine Rollen geben Nutzern vollständige Rechte auf bestimmte Funktionen das SGR,
- Spezifische Rollen sind an gewisse Artefakte (wie beispielsweise Services) gebunden und geben Nutzern Rechte speziell auf diese.

Mit dieser Unterscheidung ist es zum Beispiel möglich einen bestimmten Nutzer als Eigner eines bestimmten Services festzulegen. Die mit der Rolle Eigner einhergehenden Rechte hat der Nutzer dann aber ausschließlich für den bestimmten Service, aber nicht für die anderen im SGR verwalteten Services.

Kennzahlenverwaltung Das SGR stellt Nutzern ein rollenbasiertes Dashboard zur Verfügung, das aktuelle Kennzahlen des SOA-Verbunds bereitstellt und als Informationsplattform dient. Die Kennzahlenverwaltung verarbeitet die dafür notwendigen Daten und bereitet sie für die Darstellung auf. Als Datengrundlagen kommen einerseits die im Repository gespeicherten Informationen zum Einsatz, andererseits können über die Interoperabilitäts-

ebene auch Daten von anderen Systemen abgerufen werden, beispielsweise Laufzeitinformationen eines Enterprise Service Bus.

WSDL-Parser Der WSDL-Parser dient zum Einlesen kompletter WSDL-Dateien (Web Service Description Language) von Services. Dies ermöglicht zum einen ein leichteres Anlegen neuer Services oder Serviceversionen, da nicht mehr alle Daten von Hand eingegeben werden müssen. Zusätzlich ist so eine deutlich umfangreichere Dokumentation der Services möglich, da neben den allgemeinen Informationen wie Service-Name, -Beschreibung und -Endpunkt auch noch Schnittstelleninformationen und Datenmodell im Repository zur Verfügung stehen. Die WSDL-Datei wird dabei nicht einfach im Dateisystem abgelegt, sondern als Tripel in der Datenbank des SGR gespeichert.

Proxy-Generator Im Proxy-Generator finden sich die in Kapitel 5 detailliert vorgestellten Funktionen zur Generierung von REST-Proxys. Der Proxy-Generator dient dazu, für klassische, SOAP-basierte, Webservices einen REST-Proxy zu generieren, der die Nutzung des Services für leichtgewichtigere Anwendungsfälle ermöglicht. Ebenso wird mithilfe dieser Komponente ein Beschreibungsdokument für den REST-Service erstellt. Diese Komponente setzt prototypisch die in Kapitel 5 entwickelten Konzepte um.

Abhängigkeitsverwaltung Die Abhängigkeitsverwaltung dient zur Verwaltung der Consumer-Provider-Beziehungen. Dies schließt insbesondere die Verwaltung von geschlossenen Service-Level-Agreements ein. Ebenfalls bietet die Komponente die Möglichkeit, die Abhängigkeiten zwischen Services mittels einer Servicelandkarte grafisch darzustellen.

Benachrichtigungssystem Das SGR bietet ein feingranulares Benachrichtigungssystem, das in der Lage ist, zielgerichtete Benachrichtigungen an Benutzer zu senden. Um Stakeholder in ihrer täglichen Arbeit zu unterstützen, ist es notwendig, sie zeitnah und präzise über anstehende Änderungen

in der SOA-Landschaft zu benachrichtigen. Damit diese Benachrichtigungen von den Stakeholdern auch gelesen werden und nicht direkt weggefiltert oder gelöscht werden müssen nur solche Benachrichtigungen versendet werden, die den jeweiligen Nutzer auch tatsächlich interessieren. Durch das feingranulare Rollensystem des SGR lassen sich die Aufgabenbereiche der Nutzer entsprechend detailliert festlegen. Auf dieser Basis können Benachrichtigungen dann zielgerichtet nur an die Nutzer versendet werden, für die die Benachrichtigung interessant ist. Wenn also beispielsweise eine neue Serviceversion geplant ist, können die Verantwortlichen der Systeme, die eine alte Version des Services nutzen, automatisch benachrichtigt werden, sobald die neue Serviceversion im SGR eingepflegt wird. Zusätzlich könnte das SGR auch an einen Verzeichnisdienst im Unternehmen angebunden werden, so dass auch Mitarbeiter, die das Unternehmen verlassen oder die Position wechseln, automatisch erkannt werden. Entsprechend kann dann bei den ehemaligen Vorgesetzten um Zuweisung eines Ersatz-Ansprechpartners gebeten werden, wodurch verhindert wird, dass das System auf einen veralteten Datenbestand zurückgreift und Benachrichtigungen nicht bei den korrekten Stakeholdern ankommen.

Zeitplanverwaltung Die Zeitplanverwaltung übernimmt die Verwaltung aller Zeitpläne in Zusammenhang mit SOA-Artefakten. Das schließt beispielsweise die Dokumentation geplanter Lebenszyklus-Änderungen für Serviceversionen mit ein.

Da mit steigender Anzahl an Services und Serviceversionen auch die Anzahl der Abhängigkeiten steigt, wird mit der Zeit auch die Planung von Testzeitplänen für Verbundreleases immer wichtiger. Die Subkomponenten Testplan-Erzeugung und Testplan-Optimierung bieten hierbei Unterstützung. Mittels der Testplan-Erzeugung lassen sich semiautomatisch Testzeitpläne für Verbundreleases unter Beachtung der Abhängigkeiten zwischen Services generieren, die Testplan-Optimierung erlaubt eine Optimierung dieser Zeitpläne mit den in Kapitel 6 vorgestellten Optimierungsverfahren.

Konsistenzprüfer Die Konsistenzprüfung stellt mittels semantischem Reasoning sicher, dass die im SGR gespeicherten Daten vollständig sind und setzt die in Abschnitt 7.3 vorgestellten Verfahren um.

7.2.1.3 Interoperabilitätsebene

Die Interoperabilitätsebene dient dem Datenaustausch mit anderen Systemen und bindet das SGR in eine unternehmensweite Tool-Landschaft ein. Es bietet dazu beispielsweise eine Schnittstelle, über die Daten abgerufen werden können. In der prototypischen Implementierung der *Stuttgart IT Architecture for Manufacturing* werden über diese Schnittstelle Endpunkte der Services abgerufen, das SGR dient in diesem Fall also als Service Registry.

7.2.1.4 Präsentationsebene

Das SGR hat eine webbasierte Benutzerschnittstelle mit responsivem Design. Dadurch passt sich die Oberfläche an die Auflösung des genutzten Endgeräts an und das SGR ist somit sowohl am PC als auch an mobilen Geräten leicht bedienbar. Startansicht für jeden Nutzer des SGR ist das rollenbasierte Dashboard. Dieses kann ein Nutzer im Rahmen seiner Rollen und Rechte frei anpassen und sich die für ihn notwendigen Informationen anzeigen lassen. Manager können so beispielweise mit allgemeinen Kennwerten und Statistiken einen Überblick über den gesamten SOA-Verbund behalten, wohingegen der Eigner eines bestimmten Services detailliert die Anzahl der Aufrufe seines Services überwachen kann. Zudem ist über die Benutzerschnittstelle die Verwaltung aller weiteren SOA-Artefakte möglich. Nutzer werden dazu mittels Assistenten durch die Eingaben geführt.

7.2.2 Technische Umsetzung

Die Architektur des SGR basiert auf dem MVVM-Muster (*Model View ViewModel*, [Gar11]), das eine Abwandlung des bekannteren Model-View-Controller-Musters ist. MVVM führt eine zusätzliche Trennung der *View*-Ebene von der Präsentationslogik ein. Dadurch lässt sich die Benutzeroberfläche getrennt

von der Geschäftslogik entwickeln. So lässt sich beispielsweise für ein mobiles Endgerät eine abgespeckte Variante der Benutzerschnittstelle entwickeln, die nur darauf ausgelegt ist bestimmte, überschaubare Aufgaben zu erfüllen.

Das SGR selber ist eine Java EE-Anwendung (Java Platform, Enterprise Edition), die auf jedem Application-Server mit entsprechender Unterstützung ausgerollt werden kann. Im Rahmen dieser Forschungsarbeit kamen der *WSO2 Application Server*[1] und *Apache Tomcat*[2] zum Einsatz.

Die Abstraktion von der Datenbankebene wurde mittels *Empire RDF*, einer Implementierung der *Java Persistence API* für RDF-Tripel-Speicher realisiert. Als Datenbank wird der *RDF4J Memory Store* des Eclipse RD4J Frameworks[3] eingesetzt. Diese ist speziell auf kleine Datensets bis zu 10 Millionen Tripeln optimiert und nutzt zum schnellen Zugriff den Hauptspeicher. Die Persistierung der Daten erfolgt auf einem Festplattenspeicher.

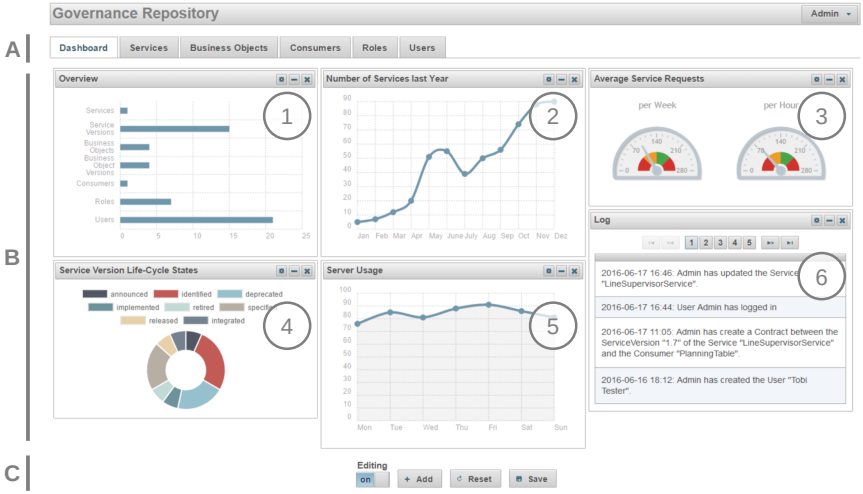


Abbildung 7.2: Screenshot des SGR-Dashboards

[1] http://wso2.com/products/application-server/
[2] http://tomcat.apache.org
[3] http://rdf4j.org

Für die Entwicklung der Benutzeroberfläche (*Model* bzw. *ViewModel*), kam PrimeFaces[1] zum Einsatz. Dabei handelt es sich um eine umfangreiche Komponentenbibliothek für *Java Server Faces*.

Abbildung 7.2 zeigt einen Screenshot des Prototypen mit geöffneter Dashboard-Ansicht. Mittels der Navigationsleiste (A) können Nutzer auf die Hauptbereiche des SGR zugreifen. Das Dashboard (B) selbst zeigt

(1) eine Übersicht über die SOA-Artefakte, die im SGR dokumentiert sind,

(2) die Entwicklung der Anzahl an Services im vergangen Jahr,

(3) die durchschnittliche Anzahl an Service-Anfragen,

(4) eine detaillierte Übersicht über die Lebenszyklusstatus der dokumentierten Services,

(5) aktuelle Server-Statistiken,

(6) Auszüge aus den Log-Dateien des SGR.

Nutzer können diese Ansicht mittels eines Editier-Modus (C) personalisieren, indem sie Kacheln hinzufügen, entfernen oder umsortieren.

7.3 Semantische Informationsgewinnung im SGR

Aufbauend auf der in Kapitel 5 vorgestellten SOA Governance Ontology und den in Abschnitt 7.1 diskutierten Architekturentscheidungen und Eigenschaften semantischer Unternehmensanwendungen, soll in den folgenden Abschnitten gezeigt werden, wie durch das semantische Datenmodell des SGR mittels Reasoning (vgl. Abschnitt 2.1.2) zusätzliches Wissen verfügbar gemacht werden kann.

Durch die Nutzung der SOA Governance Ontology ist es möglich, das SGR auf unterschiedliche Arten mit semantischen Technologien zu unterstützen:

- Nutzung von domänenspezifischen Taxonomien zur Kategorisierung von Services.

[1] http://www.primetek.com.tr

- Verschlagwortung von Services zur besseren Auffindbarkeit.
- Inferieren von neuem Wissen zur Laufzeit auf Basis der dokumentierten Metadaten und Verschlagwortung
- Validierung der Daten durch syntaktische und semantische Regeln.

In den folgenden Abschnitten werden diese Möglichkeiten detailliert vorgestellt.

7.3.1 Taxonomie und Verschlagwortung

Zur Ermöglichung einer semantischen Suche wurde das Datenmodell des SGR, das auf der SOA Governance Ontology basiert, um Schlagwörter und eine Taxonomie erweitert. Dadurch ist es möglich, Services zu annotieren und in eine hierarchische Struktur einzuordnen. Abbildung 7.3 zeigt diese Erweiterung der Ontologie. Ausgegraute Klassen sind Teil der ursprünglichen Ontologie. Hierbei wird die *Modular Unified Tagging Ontology* (MUTO, [LDA11]) wiederverwendet. Diese wurde nach Analyse existierender Tagging-Ontologien entwickelt mit dem erklärten Ziel der Wiederverwendbarkeit in beliebigen Anwendungsdomänen.

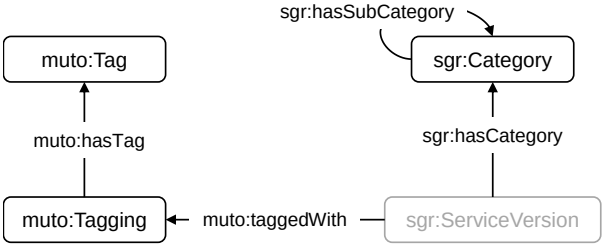


Abbildung 7.3: Erweiterung der SOA Governance Ontology

Die Einführung dieser Erweiterung erlaubt die Umsetzung einer semantischen Suche im SGR, über die Nutzer die Möglichkeit haben Services auf Basis definierter Schlüsselwörter zu finden. Ebenso ist eine Suche über die Kategorisierung der Taxonomie möglich, hierfür können Nutzer sich alle Services anzeigen lassen, die einer bestimmten Kategorie zugeordnet sind.

Die eingesetzte Taxonomie kann dabei im Vorfeld auf den jeweiligen Anwendungsbereich zugeschnitten werden. In der prototypischen Implementierung kommt eine Taxonomie aus dem Automobilbereich zum Einsatz.

Ebenfalls wird durch die Einführung der Kategorisierung und Verschlagwortung die Portfolioverwaltung des SOA-Verbunds erweitert, indem die Ähnlichkeit von Services bestimmt werden kann. Dies ist möglich durch den direkten Vergleich der Schlüsselwörter oder der Einordnung in die Taxonomie eines beliebigen Paars zweier Services. Sollten diese komplett oder in Teilen übereinstimmen, ist davon auszugehen, dass die Services gleiche oder zumindest sehr ähnliche Funktionalitäten anbieten. Darauf aufbauend können sie Consumern dann als Alternativen angezeigt werden oder durch entsprechende Konsolidierung der Funktionalitäten in einen einzelnen Service überführt werden, was sich positiv auf die Wiederverwendung auswirkt und die Wartung und Weiterentwicklung des Services vereinfacht. Durch die Wiederverwendung einer existierenden Ontologie können zudem Anwendungen, die diese Ontologie verstehen, die Informationen ebenfalls auswerten.

7.3.2 Semantisches Reasoning

Um neues Wissen zur Laufzeit zu inferieren und Daten zu validieren kommt semantisches Reasoning zum Einsatz. Semantische Regeln lassen sich dabei beispielsweise mittels der *SPARQL Inferencing Notation* (SPIN, [W3C11b]) definieren. SPIN bietet die Möglichkeit Resource Description Framework Schema (RDFS)- und Web Ontology Language (OWL)-Klassen mit SPARQL Protocol and RDF Query Language (SPARQL)-Anfragen zu verbinden. Neben semantischen Regeln lassen sich so auch Beschränkungen (englisch: *constraints*) definieren. SPIN-Regeln und -Beschränkungen können dann von einer SPIN Reasoning-Engine ausgeführt werden.

Das im vorherigen Kapitel genannte Beispiel der Ähnlichkeitsbewertung von Services kann mithilfe einer SPIN-Regel wie in Listing 7.1 gezeigt beschrieben werden. Die Regel definiert eine Relation *gr:isSimilarTo* zwischen zwei Serviceversionen, welche mit den gleichen Tags versehen sind. Im Kopf-

teil der Regel wird die Klasse definiert, auf welche die Regel angewendet wird (Zeile 2). Dann folgen Typ und Beschreibung der Regel (Zeilen 4-6) vor der Definition benötigter Präfixe (Zeilen 7-10). Mittels der *CONSTRUCT*-Klausel wird die neu inferierte Relation zwischen zwei Serviceversionen definiert (Zeilen 11-14). Diese neue Relation wird für jedes zutreffende Tripel aus der *WHERE*-Klausel hinzugefügt (Zeilen 15-25).

```
1  # Infer isSimilarTo-Relations for ServiceVersions with
       same Tags
2  sgr:ServiceVersion rdf:type rdfs:Class;
3  spin:rule [
4    rdf:type sp:Construct;
5    rdfs:comment "Infer isSimilarTo-Relations for
         ServiceVersions with same Tags"^^xsd:string ;
6    sp:text """
7    PREFIX sgr: <http://sgr#>
8    PREFIX sgrspin: <http://sgrspin#>
9    PREFIX gr: <http://purl.org/goodrelations/v1#>
10   PREFIX muto: <http://purl.org/muto/core#>
11   CONSTRUCT {
12     ?this gr:isSimilarTo ?otherserviceversion .
13     ?otherserviceversion gr:isSimilarTo ?this .
14   }
15   WHERE {
16     ?this muto:taggedWith ?tagging .
17     ?tagging rdf:type muto:Tagging .
18     ?tagging muto:hasTag ?tag .
19     ?tag rdf:type muto:Tag .
20     ?otherserviceversion muto:taggedWith ?svtagging .
21     ?otherserviceversion rdf:type sgr:ServiceVersion .
22     ?svtagging rdf:type muto:Tagging .
23     ?svtagging muto:hasTag ?tag .
24     FILTER (?otherserviceversion != ?this) .
25   }
26 """ ;
27 ] .
```

Listing 7.1: Ähnlichkeitsbeziehung zwischen Services

Durch den Einsatz dieser SPIN-Regeln ist es möglich auch ohne die Implementierung neuer Applikationslogik, das heißt ohne ein neues Softwarere-

lease zu erstellen, auf Datenbank- bzw. Tripel-Speicher-Ebene Logik umzusetzen. Im SGR sind eine Reihe von impliziten Relationen und Beschränkungen mittels SPIN-Regeln implementiert. Das Erstellen weiterer SPIN-Regeln und -Beschränkungen ist über die Benutzeroberfläche möglich. Damit können beispielsweise neue Konsistenz-Regeln definiert werden, wenn sich beispielsweise Prozesse ändern und dafür bestimmte Daten notwendig sind. Soll also eine bessere Erreichbarkeit der Serviceeigner bei Supportanfragen sichergestellt werden, ließe sich eine SPIN-Beschränkung definieren, dass jedem Eigner neben einer Mail-Adresse auch eine Telefonnummer zugewiesen sein muss.

7.4 Bewertung und Evaluierung

Dieser Abschnitt bietet eine Evaluierung des SGR mit Bezug auf die eingangs in Abschnitt 7.1 definierten Anforderungen und einen Vergleich mit am Markt erhältlichen, vergleichbaren Tools.

7.4.1 Evaluierung des Prototypen

Unterstützung von Governance-Prozessen Durch die Möglichkeit alle relevanten SOA-Artefakte, wie Services und -Versionen, Business Objects *plus*, Consumer-Provider-Beziehungen und Stakeholder sowie die Beziehungen zwischen ihnen zu dokumentieren, dient das SGR als zentrale Governance-Plattform für serviceorientierte Architekturen in Unternehmen. Dementsprechend erfüllt das System die unter A_1 (vgl. Abschnitt 7.1) definierten Anforderungen an ein zentrales Verwaltungssystem für SOA-Artefakte. Das RDF-Datenmodell lässt sich zudem flexibel und leicht erweitern, falls in Zukunft weitere Artefakte in das Repository aufgenommen und verwaltet werden müssen. Dabei kommt es zu keinen Auswirkungen auf bereits gespeicherte Artefakte. Eine Erweiterung des Datenmodells könnte beispielsweise bei der Verwaltung von Consumer-Provider-Verträgen notwendig sein, wenn weitere Leistungskennzahlen oder Vertragseigenschaften definiert werden sollen.

Usability & Performanz Das SGR unterstützt Nutzer mittels eines Assistenten, der sie schrittweise durch die Erstellung neuer Artefakte führt. Dadurch wird eine intuitive Nutzung der Software ohne umfangreiche Einweisung ermöglicht. Fehlende Usability und User Experience sind häufig ein Problem etablierter Unternehmenssoftware, meist aufgrund eines historisch gewachsenen Funktionsumfangs und fehlender Einbeziehung von Usability-Spezialisten in der (Weiter-)entwicklung. Damit ein Zugriff auf das Repository für Nutzer überall und jederzeit möglich ist, wurde bei der Umsetzung auf responsives Webdesign gesetzt. Dadurch ist eine automatische Anpassung der Benutzeroberfläche an praktisch jedes Endgerät gegeben. Eine grafische Darstellung der Services und ihrer Abhängigkeiten unterstützt Nutzer zusätzlich bei der Arbeit und erlaubt eine interaktive Erkundung der Service-Landschaft. Das SGR erfüllt somit die Anforderung A_2. Die Personalisierung des Informationsangebots nach Anforderung A_3 erfüllt das SGR durch die Bereitstellung des rollenbasierten Dashboards. Dieses erlaubt den Nutzern eine individuelle Anpassung an ihre Bedürfnisse. Das Dashboard bietet dazu aggregierte sowie detaillierte Informations-Visualisierungen zur Unterstützung der Nutzer in ihrer täglichen Arbeit. Das SGR enthält zudem ein Benachrichtigungssystem, das Stakeholder zeitnah mit den für sie relevanten Informationen versorgt.

Datenzugriff und Informationsgewinnung Das SGR bietet über eine Interoperabilitätsebene Application Programming Interfaces (APIs) für den Zugriff auf Daten und Informationen im System. Ebenso ist darüber eine Zusammenarbeit und ein Datenaustausch mit anderen Systemen möglich. Es können dadurch beispielsweise Informationen aus anderen Systemen ausgelesen werden, um eine redundante Datenhaltung und -eingabe zu verhindern. Dies erfüllt Anforderung A_4. Durch den Einsatz der SOA Governance Ontology als Grundlage für die Datenhaltung ist die Basis für eine semantische Informationsgewinnung gegeben und Anforderung A_5 erfüllt. In diesem Zusammenhang wurden bereits in Kapitel 7.3 die implementierten Beispiele für semantisches Reasoning vorgestellt.

7.4.2 Vergleich mit am Markt erhältlichen Systemen

Das Marktforschungsunternehmen Gartner untersucht in regelmäßigen Abständen Marktanalysen von Software-Tools und Anbietern deren Ergebnisse als sogenannte *Magic Quadrants* veröffentlicht werden. Ein Magic Quadrant teilt die Anbieter in vier Kategorien ein: Anführer (*Leader*), Visionäre (*Visionary*), Herausforderer (*Challengers*) und Nischenanbieter (*Niche players*). Im Folgenden werden die Anbieter, die im *Magic Quadrant for Application Services Governance* 2015 [Mal15] und im *Magic Quadrant for Full Life Cycle API Management* 2018 [MO18] als Anführer klassifiziert wurden dem SGR gegenübergestellt. Die Auswahl wurde auf Produkte eingeschränkt, die einen ähnlichen Funktionsumfang wie das SGR haben. Teilweise müssen mehrere Produkte eines Herstellers betrachtetet werden, da die entsprechenden Funktionalitäten über mehrere Lösungen verteilt sind. Der folgende Vergleich beschränkt sich dabei auf die Verwaltung von Services und APIs, die integraler Bestandteil des SGR sind. Die API-Management-Plattformen der Hersteller bieten viele zusätzliche Funktionalitäten, wie z.B. Authentifizierung oder Orchestrierung, die klassisch mit einem Enterprise Service Bus realisiert wurden. Diese sind nicht Bestandteil des SGR und werden daher nur nachrangig betrachtet. Ebenfalls ist zu beachten, dass alle Hersteller APIs primär als REST-Schnittstellen verstehen und dementsprechend der Hauptfokus der Produkte auf ebensolchen Services liegt.

Zunächst werden die Anbieter und Produkte kurz vorgestellt und im Anschluss anhand verschiedener Kriterien in ihrem Funktionsumfang bewertet und dem SGR gegenübergestellt.

7.4.2.1 Marktüberblick

Apigee: *Apigee Edge*[1] Apigee bietet eine umfangreiche API Management-Plattform und wurde von den Gartner-Analysten bereits mehrfach als Marktführer eingeordnet. Apigee wurde Ende 2016 von Google übernommen und wird seitdem über Google Cloud[2] angeboten, kann aber auch als vor Ort

[1] https://apigee.com/api-management/
[2] https://cloud.google.com

installierte Lösung eingesetzt werden. Die Plattform bietet Sicherheitsfunktionalität, wie auch Monitoring-Möglichkeiten für die angebotenen APIs. Ebenso können externe Entwickler direkt über die Plattform Zugänge anfordern und Nutzungsquota buchen.

CA Technologies: *CA API Management*[1] CA Technologies bietet ein umfangreiches Portfolio an API Management Software und weitere begleitende Produkte, die sowohl in der Cloud als auch vor Ort betrieben werden können. Ebenfalls eingeschlossen sind Sicherheitsfunktionalitäten, Monitoring und auch Gateways für mobile Anwendungen. Über ein Entwicklerportal können auch externe API-Entwickler mit eingebunden werden.

IBM: *IBM API Connect*[2] IBM bietet eine Lösung zur Erstellung, Verwaltung und Überwachung von APIs. Ebenfalls werden Funktionalitäten zur sicheren Kommunikation mit den APIs und zur Monetarisierung dieser angeboten. Die Plattform kann zudem mit dem *WebSphere Service Registry and Repository* integriert werden, wodurch sich weitere Verwaltungsmöglichkeiten für Services eröffnen.

Mulesoft: *Anypoint Platform*[3] Die Anypoint Platform des Anbieters Mulesoft, der im Mai 2018 von salesforce.com übernommen wurde, bietet Funktionalität zur Entwicklung, Nutzung und Verwaltung von APIs. Dabei können sowohl SOAP als als auch REST APIs angebunden werden. Der Betrieb der Plattform ist ebenfalls vor Ort und in der Cloud möglich, ebenso in einer hybriden Installation.

Software AG: *webMethods API Management Platform*[4] Software AG hat eine umfangreiche API Management Plattform im Produktportfolio, die eine gute Anbindung an weitere Produkte des Herstellers bietet. Auch dieser

[1] https://www.ca.com/de/products/ca-api-gateway.html
[2] https://www.ibm.com/de-de/marketplace/api-management
[3] https://www.mulesoft.com/platform/enterprise-integration
[4] http://www1.softwareag.com/de/products/webmethods_integration/api/

Hersteller bietet im Rahmen seiner Plattform Sicherheitsfunktionalitäten, Monitoring und ein Entwicklerportal an.

7.4.2.2 Vergleichskriterien

Im Folgenden werden Kriterien beschrieben, die für die vergleichende Betrachtung des SGR mit den am Markt verfügbaren Systemen wesentlich sind. Dies erleichtert die Auswahl eines geeigneten Systems und zeigt auf, wie die im SGR implementierten Inhalte den momentanen Stand der Technik sinnvoll ergänzen.

Artefaktverwaltung Dieses Kriterium bewertet, inwiefern mit der Software grundlegende SOA-Artefakte verwaltet werden können. Die Dokumentation dieser Metadaten ist unerlässlich für eine effiziente Unterstützung der SOA-Governance-Prozesse. Die Bewertung erfolgt hier gestaffelt nach dem Umfang der dokumentierbaren Artefakte *Services*, *Consumersysteme*, *Stakeholder*, und *Consumer-Provider-Beziehungen*.

Interoperabilität Zur effizienten Unterstützung der Governance-Prozesse ist auch eine Anbindung an andere Tools notwendig, um unter anderem die bereits geschilderte redundante Datenhaltung zu vermeiden. Die Bewertung erfolgt hier anhand des Funktionsumfangs eines Datenaustausches. Ist dieser in beide Richtungen möglich und werden dabei Standards unterstützt wird das Kriterium als erfüllt angesehen. Entsprechend abgestuft erfolgt die Bewertung, wenn keine Standards unterstützt werden oder der Datenaustausch nur in eine Richtung möglich ist.

Benachrichtigungssystem Dieses Kriterium bewertet die Unterstützung der Stakeholder durch ein intelligentes Benachrichtigungssystem, das zielgerichtet relevante Informationen bereitstellt. Die Benachrichtigungen sollen also zielgerichtet an Stakeholder mit bestimmten Rollen adressiert werden können. Abschläge in der Bewertung gibt es für das Fehlen der Möglichkeit

Benachrichtigungen gar nicht oder nur eingeschränkt an bestimmte Rollen zu adressieren.

Dashboard Für die Bewertung des Zustandes einer SOA und der darin angebotenen und verwalteten Services ist eine grafische Darstellung bestimmter Kenngrößen sinnvoll und hilfreich. Ein solches Monitoring wird meist über sogenannte Dashboards realisiert. Dieses Kriterium bewertet das Vorhandensein und den Umfang eines Monitoring-Dashboards und ob dieses sich individuell vom Nutzer auf seine Informationsbedürfnisse anpassen lässt. Abzüge in der Bewertung gibt es für die fehlende Individualisierbarkeit.

Unterstützung von SOAP-basierten Services Das Kriterium bewertet ob das untersuchte System neben REST-Services auch die Verwaltung von SOAP-basierten Services unterstützt. Ebenso wird bewertet, ob eine Möglichkeit existiert, vorhandene SOAP-basierte Services als REST-API anzubieten. Eine Abwertung erfolgt, wenn diese Funktionen nur teilweise oder gar nicht unterstützt werden.

Abdeckung des kompletten Lebenszyklus Zur effizienten Verwaltung und Unterstützung der Anwender ist es notwendig, Services auch in frühen Phasen, also bereits in der Entwurfs- und Entwicklungsphase in eine strukturierte Governance zu überführen. Dieses Kriterium bewertet daher, ob die untersuchte Software es erlaubt, Services und APIs bereits in diesen frühen Phasen zu verwalten und ob eine Versionsverwaltung der APIs möglich ist. Abzüge erfolgen entsprechend für das Fehlen einer solchen Unterstützung.

7.4.2.3 Vergleich

Die untersuchten kommerziellen Produkte zeigen durchgehend einen sehr hohen Reifegrad und eine gute Unterstützung für das API-Management, was die Einordnung der Gartner-Analysten bestätigt. Tabelle 7.1 zeigt eine Darstellung der Bewertung anhand der vorgestellten Kriterien.

Tabelle 7.1: Vergleich des SGR mit kommerziellen Produkten

	Apigee	CA	IBM	Mulesoft	Software AG	SGR
Artefaktverwaltung	◕	◕	◕	◕	◕	●
Interoperabilität	○	○	○	○	○	◑
Benachrichtigungssystem	◑	◔	◑	◑	◑	◕
Dashboard	●	●	◕	◕	●	●
SOAP-basierte Services	◕	◔	◔	◑	◕	◕
Lebenszyklusverwaltung	◔	●	◕	◑	●	●

Alle untersuchten Produkte unterstützen die Dokumentation der wichtigsten SOA-Artefakte. Leichte Schwächen zeigen sich auf Seite der Consumer- und Stakeholder-Dokumentation, die nur sehr rudimentär vorhanden ist. So können Consumer immer nur als Personen dokumentiert werden, über die Anwendungen, die eine API nutzen, ist nichts bekannt. Im Unternehmensumfeld wird aber meist auch eine Dokumentation der Anwendung, in der ein Service zum Einsatz kommt, gefordert und ein Service Level Agreement abgeschlossen. Zusätzliche Stakeholder können zu einzelnen APIs in keinem der Produkte dokumentiert werden. Es gibt höchstens einen Ansprechpartner je Service. Das SGR kann im Gegensatz dazu für jeden Service eine beliebige Anzahl Stakeholder je Service mit unterschiedlichen Rollen dokumentieren, wodurch unterschiedliche Ansprechpartner beispielsweise für Testbelange definiert werden können. Die Rechteverwaltung der untersuchten Produkte erlaubt eine feingranulare, freie Vergabe von Rechten. Teilweise können auch neue Rollen definiert werden.

Die untersuchten Produkte bieten zwar teilweise eine Interoperabilität mit anderen Produkten des Herstellers und eine Möglichkeit API-Beschreibungen in standardisierten Sprachen wie WSDL oder OpenAPI zu importieren und exportieren. Ein wie im SGR vorgesehener Datenaustausch mit Fremdprodukten und anderen Lebenszyklus-Werkzeugen über standardisierte Schnittstellen wie *Open Services for Lifecycle Collaboration*[1] ist in keinem der Produkte möglich.

[1] https://www.open-services.net/

Die Benachrichtigungssysteme der Anwendungen gestalten sich allesamt sehr rudimentär. Eine wirkliche Interaktion mit den Stakeholdern der Consumer-Systeme, die essentiell ist, um diese frühzeitig über anstehende Änderungen oder Wartungsarbeiten zu informieren, ist nicht vorgesehen. CA Technologies ermöglicht es zwar Nachrichten an Entwickler zu schreiben, allerdings werden keine automatischen Benachrichtigungen bei vordefinierten Ereignissen unterstützt. Apigee bietet zwar ein Benachrichtigungssystem auf API-Ebene, allerdings sind in diesem nur Nachrichten mit Bezug zur Monetarisierung vorgesehen. Auch IBM und Mulesoft bieten Benachrichtigungen, allerdings sind diese nur sehr grobgranular definierbar und umfassen nur eine sehr eingeschränkte Menge an Events. Die Software AG bietet automatische Benachrichtigungen an Consumer und Entwickler, wenn bei der Nutzung der Schnittstellen Fehler auftreten. Das SGR bietet im Gegensatz zu diesen Systemen ein umfangreiches Benachrichtigungskonzept, in dem Consumer-Stakeholder über alle Änderungen an Services informiert werden, die sie nutzen. Ebenfalls erhalten Sie frühzeitig Informationen, wenn neue Serviceversionen geplant sind. Zur Sicherstellung, dass diese Informationen auch immer die korrekten Stakeholder erreichen, ist das SGR an ein Nutzerverzeichnis angeschlossen und erkennt so, wenn Stakeholder die Abteilung wechseln oder das Unternehmen verlassen.

Alle Anwendungen bieten zum Monitoring der Services und APIs einen Analysebereich mit Dashboard. Bei CA Technologies und Software AG sind diese sehr gut anpassbar. Die anderen Hersteller bieten vordefinierte Leistungskennzahlen, die allerdings nicht in einer individualisierten Ansicht dargestellt werden können. Bezüglich dieses Kriteriums existieren nur sehr geringe Unterschiede zum SGR, das ebenfalls ein auf den Nutzer und seine Rollen angepasstes Dashboard bietet.

In allen untersuchten Anwendungen können neben REST-APIs auch SOAP-basierte Webservices verwaltet werden. Diese unterscheiden sich hauptsächlich durch den Typ, abgesehen davon lassen sich die gleichen Metadaten dokumentieren wie für REST-APIs. Apigee, Mulesoft und Software AG bieten auch die Möglichkeit einen REST-Proxy für SOAP-basierte Services zu erstellen. In Mulesoft muss dieser manuell erstellt werden, die anderen beiden

Hersteller bieten eine Unterstützung direkt in der Anwendung an. Erweiterte Enterprise-Funktionalitäten wie Caching der Anfragen und Propagation von Authentifizierungsinformationen bieten diese nicht an. Da diese im SGR nur konzeptuell vorgesehen sind, erhält auch dieses einen Abzug in der Bewertung.

Alle Hersteller bieten die Möglichkeit Services zu aktivieren und zu deaktivieren und diese zu versionieren. Apigee bietet darüber hinaus keine Lebenszyklusverwaltung der APIs an. Mulesoft bietet die Möglichkeit an, APIs als veraltet zu markieren, um so Consumer vor einer anstehenden Abschaltung der API zu warnen. Die anderen Hersteller und auch das SGR, bieten die Möglichkeit, feingranular Lebenszyklusstatus zuzuweisen. IBM verwaltet unterschiedliche Versionen einer API getrennt, es ist also nur am Namen zu erkennen, dass es sich um dieselbe API handelt. Die anderen Hersteller und das SGR verwalten Versionen zusammen, so dass klar erkennbar ist, dass es sich um ein und denselben Service in unterschiedlichen Versionen handelt.

7.5 Zusammenfassung und Ausblick

Mit dem SGR wurde auf Basis von Praxis-Anforderungen und Anwendungsfällen der Prototyp eines umfangreichen SOA Governance-Tools entwickelt, der in der Lage ist die tägliche Arbeit aller SOA-Stakeholder effizienter zu gestalten. Durch den Einsatz eines flexiblen, semantischen Datenmodells kann die Software auch auf sich ändernde Anforderungen angepasst werden. Die Entwicklung des SGR fußt auf den eingangs vorgestellten Eigenschaften von und Anforderungen an den Einsatz von semantischen Anwendungen im Unternehmensumfeld, dem sogenannten *Corporate Semantic Web*. Mittels semantischem Reasoning lassen sich aus den im SGR gespeicherten Informationen weitere Erkenntnisse ableiten und Relationen inferieren. Der Prototyp des SGR wurde mit im Unternehmensumfeld verbreiteten Technologien entwickelt und bietet umfangreiche Funktionalitäten zum Einsatz in diesem Feld. Im direkten Vergleich mit am Markt verfügbaren kommerziellen

Produkten bietet das SGR umfangreichere Dokumentationsmöglichkeiten für SOA-Artefakte und stellt die Interoperabilität mit anderen Lebenszyklus-Werkzeugen sicher. Das feingranulare Benachrichtigungssystem ermöglicht es, Stakeholder über für sie relevante Änderungen zu informieren. Zusammengefasst erfüllt das SGR die Anforderungen an ein zentrales Werkzeug zur effizienteren Ausführung von SOA-Governance-Prozessen in SOA-Verbünden.

Da es sich bei der Implementierung des SGR um einen Forschungsprototypen der in dieser Arbeit entwickelten Konzepte handelt, ist ein produktiver Einsatz der Software in einem Unternehmen nur schwer möglich, insbesondere da bei der Entwicklung kein besonderer Fokus etwa auf Robustheit gelegt wurde. Der entwickelte Prototyp zeigt dennoch, wie die entwickelten Konzepte auch in der Praxis umgesetzt werden können und Anwendung in einer Unternehmenssoftware finden können.

Kapitel 8

Zusammenfassung und Ausblick

Dieses Kapitel fasst zunächst die bearbeiteten Themen in Abschnitt 8.1 zusammen und gibt in Abschnitt 8.2 einen Ausblick auf weiterführende Forschungsthemen.

8.1 Zusammenfassung

Vor dem Hintergrund aktueller Entwicklungen, wie der durch Industrie 4.0 geförderten Digitalisierung in der Produktion, ist es notwendig einerseits eine durchgängige Integration aller IT-Systeme sicherzustellen und andererseits Prozesse und Konzepte zu entwickeln, um diese IT-Systeme effizient zu verwalten. Die benötigte Integration wird in Unternehmen häufig in Form serviceorientierter Architekturen (SOA) umgesetzt und die Weiterentwicklung der Softwaresysteme und Services in sogenannten SOA-Systemverbünden durchgeführt. An dieser Stelle setzt die vorliegende Arbeit an und betrachtet Aspekte des Change Management im Rahmen der SOA Governance.

Herausforderungen existieren dabei bei der Integration neuer Service-Consumer in den Systemverbund. Hierbei müssen häufig speziell für jede Anbindung Datenmodelltransformationen definiert werden, insbesondere wenn Zulieferer und Partnerunternehmen angebunden werden. Dies führt zu langwierigen Prozessen und steht dem Grundgedanken und der Anforderung einer schnellen und flexiblen IT entgegen. Auch die Weiterentwicklung von Systemen und Services stellt Unternehmen vor Herausforderungen, insbesondere bei der Absicherung von Änderungen. Um Stakeholder eines SOA-Systemverbundes effizient in ihrer Arbeit zu unterstützen, ist es notwendig, Dokumentationsprozesse so schlank wie möglich zu halten. Hierzu können neue Konzepte wie Semantic-Web-Technologien zum Einsatz kommen.

Aus dem übergeordneten Ziel der Entwicklung von Methoden und Verfahren zur Unterstützung von Änderungsprozessen und der Governance von SOA-Verbünden, ergeben sich drei Forschungsziele:

- Forschungsziel Z_1: Vereinfachung der Integration von neuen Consumern in eine SOA
- Forschungsziel Z_2: Einsatz von Semantic-Web-Technologien zur Unterstützung der SOA Governance
- Forschungsziel Z_3: Unterstützung des Testens von Systemverbünden

In Kapitel 2 wird ein Überblick über die Themenbereiche und den Kontext der Arbeit gegeben und der Stand der Forschung dargelegt. Dazu werden zunächst informationstechnische Grundlagen diskutiert. Nach einer Einführung in die SOA Governance wird anschließend die Arbeit in den Anwendungskontext großer SOA-Systemverbünde eingeordnet und deren spezielle Herausforderungen beschrieben.

Zur Realisierung eines einfachen, domänenübergreifenden Datenaustauschs wird in Kapitel 3 das Konzept der Business Objects *plus* (BO^+) vorgestellt. Dieses schafft semantisch einheitliche, in sich gekapselte Datenobjekte, die domänenübergreifend in Anwendungen und Serviceschnittstellen genutzt werden können. Dadurch wird die Anbindung von neuen Service-Consumern erleichtert, da diese intern bereits die selben Datenobjekte nutzen. Auch die

Anbindung von mehreren Services, beispielsweise bei Kooperationsprojekten oder der Integration neuer Zulieferer, wird vereinfacht. Da alle Services auf dieselben Datenobjekte zur Repräsentation gleicher Artefakte setzen, erübrigt sich die manuelle Definition von Transformationen für diese Services ganz oder teilweise. Das Konzept unterstützt auch die Abstimmung zwischen Fach- und IT-Bereichen eines Unternehmens, da aufgrund einer einheitlichen Semantik ein gleiches Verständnis der Artefakte existiert. Das entwickelte Konzept deckt zusätzlich auch organisatorische Gesichtspunkte ab und definiert Prozesse zur Einführung von BO^+ und zur Verwaltung ihrer Lebenszyklen. Das Kapitel liefert damit einen Beitrag zu Forschungsziel Z_1.

In Kapitel 4 werden nach einer umfangreichen Analyse Anforderungen an SOA-Governance-Prozesse in Unternehmen aufgestellt. Aus diesen wird ein Metamodell abgeleitet, das die relevanten SOA-Artefakte und ihre Beziehungen untereinander beschreibt. Zur Beurteilung der Unterstützung der Entwicklung eines SOA-Governance-Tools mittels semantischer Technologien wird auf Basis des Metamodells eine Ontologie, die *SOA Governance Ontology*, entwickelt. Diese liefert einen Beitrag zur Erreichung von Forschungsziel Z_2

Unternehmen setzten in der Vergangenheit meist auf klassische SOAP-basierte Webservices zur Realisierung ihrer Systemverbünde. Durch die im Vergleich zu anderen Kommunikationsprotokollen relative Schwergewichtigkeit des SOAP-Protokolls, sind diese allerdings nicht für moderne Anwendungsfälle, wie beispielsweise die Anbindung mobiler Endgeräte, geeignet. Hierfür wird in Kapitel 5 eine Architektur definiert, die REST-to-SOAP Middleware Architecture. Diese liefert einen Beitrag zur Erfüllung von Forschungsziel Z_1 und ermöglicht es Unternehmen, ihre vorhandenen SOAP-basierten Services über eine Proxy-Komponente auch als Services zur Verfügung zu stellen, die dem Architekturparadigma *Representational State Transfer* (REST) folgen. Dadurch haben Unternehmen die Möglichkeit günstig neue Anwendungs- und Geschäftsfelder zu erschließen. Die Middleware übernimmt dazu die Bereitstellung von Proxy-Services, die Anfragen in das SOAP-Protokoll umwandeln und an die SOAP-basierten Services weiterleiten. Die Proxys-Services werden dabei semiautomatisch erstellt, so dass der Aufwand, einen Service über die Middleware anzubieten, gering ist.

Kapitel 6 widmet sich dem Testen von SOA-Systemverbünden und liefert Beiträge zu Forschungsziel Z_3. Betrachtet wird insbesondere die Generierung und Optimierung von Testzeitplänen. In einem Systemverbund wird eine Menge an Softwaresystemen gemeinsam weiterentwickelt und veröffentlicht. Da die Systeme eng verzahnt sind und viele Daten ausgetauscht werden, kann es sein, dass Softwarefehler in einem System auch die Datenverarbeitung in einem anderen System beeinflussen. Um eine solche Fortpflanzung von Fehlern über eine längere Kette von Systemen hinweg zu verhindern, ist es notwendig eine Reihenfolge für das Testen der Schnittstellen unter Beachtung der Provider-Consumer-Beziehungen einzuhalten. Damit wird eine frühe und zielgerichtete Erkennung von Fehlern ermöglicht und aufwändige Ursachenforschung über mehrere Systeme verhindert. Ebenfalls wird in diesem Kapitel eine Methode zur Bestimmung der Teststrategie für einzelne Services vorgestellt. Mit dieser lässt sich, anhand einer individuellen Einstufung der Systemkritikalität des Services und seiner Consumer-Systeme, sowie der Risikobewertung einer Serviceänderung, eine Empfehlung für die effizienteste Teststrategie aussprechen.

Abschließend beschäftigt sich Kapitel 7 zunächst mit dem Einsatz von semantischen Technologien im Unternehmensumfeld. Dazu werden Eigenschaften von und Abwägungen zum Einsatz solcher Technologien diskutiert. Daran anschließend werden Anwendungsfälle für die semantische Informationsgewinnung auf Basis der SOA Governance Ontology vorgestellt. Abschnitt 7.2 stellt das SOA Governance Repository (SGR) vor, den Prototypen eines zentralen SOA-Governance-Informationssystems. In das SGR sind die im Rahmen der Forschungsbeiträge dieser Arbeit entwickelten Konzepte und Methoden als prototypische Implementierungen eingeflossen. Die in Abschnitt 7.4 durchgeführte Evaluierung und vergleichende Bewertung mit kommerziellen Produkten zeigt, dass das SOA Governance Repository gut geeignet ist, um Stakeholder eines SOA-Systemverbundes effizient in ihrer Arbeit zu unterstützen. Somit wird auch das übergeordnete Ziel, die Entwicklung von Methoden und Verfahren zur Unterstützung der Änderungsprozesse und der Governance serviceorientierter Architekturen erfüllt.

8.2 Ausblick

Die Flexibilität des Datenmodells der prototypischen Implementierung lässt sich mit dem aktuellen Stand der Forschung nicht vollständig in der Benutzeroberfläche widerspiegeln. Das Datenmodell lässt sich zwar flexibel anpassen, allerdings muss die zugehörige Benutzeroberfläche manuell entwickelt werden, was bei Änderungen zu Entwicklungsaufwänden und neuen Softwarereleases führt. Der entwickelte Prototyp ist zwar an definierten Stellen, beispielsweise bei der Definition von Service Level Agreements, auf eine flexible Benutzeroberfläche ausgerichtet. Um dies aber auf die gesamte Software auszudehnen kann in einem zukünftigen Projekt untersucht werden, inwiefern die dynamische Erzeugung von Benutzeroberflächen aus einem RDF-Datenmodell (*Ressource Description Framework*) realisierbar ist. Ansätze dazu finden sich bei Butt et al. [BHLX13], die untersuchen, ob sich aus beliebigen RDF-Graphen einfache Web-Formulare zur Bearbeitung dieser Graphen erzeugen lassen.

In Unternehmen ist zwar primär das untersuchte Bereitstellen von SOAP-basierten Webservices als REST-Webservices von Bedeutung, allerdings bietet auch die Gegenrichtung Möglichkeiten zu weiterer Forschung. Die Generierung und Nutzung eines SOAP-Proxys für REST-Services, macht die Nutzung der umfangreichen SOAP-Funktionalitäten nutzbar. Dadurch können externe REST-Services auch in Anwendungsfällen genutzt werden, die hohe Zuverlässigkeit und Sicherheit voraussetzen.

Die Konzepte der vorliegenden Arbeit wurden bereits auch in anderem, verwandten, Forschungskontext eingebunden. So haben die Governance-Konzepte Einzug in die *Stuttgart IT Architecture for Manufacturing* (SITAM, vgl. [KGK+17], Abschnitt 2.3) gefunden und sind so auch Teil der *Sozialen Fabrik*, als deren Basis die SITAM dient [KHW+17]. Hier ergibt sich die Möglichkeit noch weitere der entwickelten Konzepte, wie beispielsweise die REST-to-SOAP Middleware Architecture zu integrieren. Diese könnte die Mobile Middleware der SITAM ergänzen.

Literaturverzeichnis

[ACKM04] G. Alonso, F. Casati, H. Kuno und V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Berlin und Heidelberg: Springer, 2004 (Zitiert auf S. 25).

[Afs07] M. Afshar. *SOA Governance: Framework and Best Practices*. Hrsg. von Oracle. 2007 (Zitiert auf S. 38).

[Agr15] N. Agrawal. *HowTo – REST API proxy to SOAP webservice*. 2015. URL: `https://blogs.mulesoft.com/dev/howto/rest-api-proxy-to-soap-webservice/` (Zitiert auf S. 107).

[Api18] Apigee. *Exposing a SOAP service as an API proxy*. 2018. URL: `https://docs.apigee.com/api-platform/develop/exposing-soap-service-api-proxy` (Zitiert auf S. 106).

[ARW08] S. Aier, C. Riege und R. Winter. „Unternehmensarchitektur – Literaturüberblick und Stand der Praxis“. In: *WIRTSCHAFTSINFORMATIK* 50.4 (2008), S. 292–304 (Zitiert auf S. 25).

[AT08] S. Abdullah und H. Turabieh. „Generating university course timetable using genetic algorithms and local search“. In: *Third International Conference on Convergence and Hybrid Information Technology ICCIT'08*. Bd. 1. 2008, S. 254–260 (Zitiert auf S. 137).

[AW10] P. Allen und L. Wilkes. „SOA Governance: Challenge or Opportunity“. In: *CBDI Journal* 03/2010 (2010) (Zitiert auf S. 35, 84, 86).

[BC12] B. Burnes und B. Cooke. „The past, present and future of organization development: Taking the long view“. In: *Human Relations* 65.11 (2012), S. 1395–1429 (Zitiert auf S. 36).

[Ben12] S. G. Bennett. *A Framework for SOA Governance: Oracle Practitioner Guide*. 2012. URL: http://www.oracle.com/technetwork/topics/entarch/oracle-pg-soa-governance-fmwrk-r3-2-1561703.pdf (Zitiert auf S. 81, 83, 86).

[BGL07] K. Bierhoff, M. Grechanik und E. S. Liongosari. „Architectural Mismatch in Service-Oriented Architectures“. In: *International Workshop on Systems Development in SOA Environments (SDSOA 2007)*. SDSOA '07. [Piscataway, N.J.]: IEEE, 2007 (Zitiert auf S. 45).

[BHL01] T. Berners-Lee, J. Hendler und O. Lassila. „The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities“. In: *Scientific American* 284.5 (2001), S. 34–43. URL: https://www.scientificamerican.com/article/the-semantic-web/ (Zitiert auf S. 30).

[BHLX13] A. S. Butt, A. Haller, S. Liu und L. Xie. „ActiveRaUL: A Web form-based User Interface to create and maintain RDF data“. In: *Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track)*. Hrsg. von E. Blomqvist und T. Groza. Bd. 1035. CEUR-WS.org, 2013, S. 117–120 (Zitiert auf S. 171).

[BHM+04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris und D. Orchard. *Web Services Architecture*. 2004. URL: https://www.w3.org/TR/ws-arch/ (Zitiert auf S. 27).

[Bie06] N. Bieberstein. *Service-oriented architecture (SOA) compass: Business value, planning, and enterprise roadmap*. Developer works. Upper Saddle River, NJ [u.a.]: IBM Press, 2006 (Zitiert auf S. 17, 38).

[BIL14] O. Bossert, C. Ip und J. Laartz. *A two-speed IT architecture for the digital enterprise*. 2014. URL: http://www.mckinsey.com/business-functions/business-technology/our-insights/a-two-speed-it-architecture-for-the-digital-enterprise (Zitiert auf S. 104).

[Bis14] T. Biske. *SOA architecture: Why you need API management*. 2014. URL: http://searchsoa.techtarget.com/answer/SOA-architecture-Why-you-need-API-management (Zitiert auf S. 38).

[Bit17] Bitkom. *Jedes dritte Unternehmen bietet Arbeit im Homeoffice an*. 2.02.2017. URL: https://www.bitkom.org/Presse/Presseinformation/Jedes-dritte-Untershynehmen-bietet-Arbeit-im-Homeshyoffice-an.html (Zitiert auf S. 17).

[BKM+08] B. Byrne, J. Kling, D. McCarty, G. Sauter und P. Worcester. *The value of applying the canonical modeling pattern in SOA*. 2008. URL: https://www.ibm.com/developerworks/data/library/techarticle/dm-0803sauter/ (Zitiert auf S. 47).

[BM14] Brickley, Dan und Miller, Libby. *FOAF Vocabulary Specification 0.99*. 2014. URL: http://xmlns.com/foaf/spec/ (Zitiert auf S. 70, 93).

[BMT06] W. A. Brown, G. Moore und W. Tegan. *SOA Governance: IBM's approach*. Somers, New York, 2006 (Zitiert auf S. 84, 86).

[BP09] A. Bertolino und A. Polini. „SOA Test Governance: Enabling Service Integration Testing across Organization and Technology Borders". In: *International Conference on Software Testing, Verification and Validation workshops, 2009*. Piscataway, NJ und Piscataway, NJ: IEEE, 2009, S. 277–286 (Zitiert auf S. 122).

[Bun17] Bundesministerium für Bildung und Forschung. *Industrie 4.0: Innovationen für die Produktion von morgen*. 2017. URL: https://www.bmbf.de/pub/Industrie_4.0.pdf (Zitiert auf S. 16).

[Bur10] S. Burbeck. *The Tao of e-business services: The evolution of Web applications into service-oriented components with Web services*. (Originalquelle nicht mehr verfügbar), 2010. URL: https://web.archive.org/web/20130829164021/http://www.ibm.com/developerworks/library/ws-tao/index.html (Zitiert auf S. 25).

[CCMW01] E. Christensen, F. Curbera, G. Meredith und S. Weerawarana. *Web Services Description Language (WSDL) 1.1*. 2001. URL: https://www.w3.org/TR/2001/NOTE-wsdl-20010315 (Zitiert auf S. 27).

[CD06] G. Canfora und M. Di Penta. „Testing services and service-centric systems: Challenges and opportunities". In: *IT Professional* 8.2 (2006), S. 10–17 (Zitiert auf S. 122).

[CD09] G. Canfora und M. Di Penta. „Service-Oriented Architectures Testing: A Survey“. In: *Software Engineering. ISSSE 2006, ISSSE 2007, ISSSE 2008. Lecture Notes in Computer Science* 5413 (2009), S. 78–105 (Zitiert auf S. 122).

[CDM92] A. Colorni, M. Dorigo und V. Maniezzo. „A genetic algorithm to solve the timetable problem“. In: *Politecnico di Milano, Milan, Italy TR* (1992), S. 90–060 (Zitiert auf S. 137).

[CH09] D. Cohn und R. Hull. „Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes“. In: *IEEE Data Engineering Bulletin* 32.3 (2009), S. 3–9 (Zitiert auf S. 48).

[DCM12] DCMI Usage Board. *DCMI Metadata Terms*. 2012. URL: `http://dublincore.org/documents/dcmi-terms/` (Zitiert auf S. 92).

[dGRZ10] T. de Giorgio, G. Ripa und M. Zuccalà. „An Approach to Enable Replacement of SOAP Services and REST Services in Lightweight Processes“. In: *Current Trends in Web Engineering: 10th International Conference on Web Engineering ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers*. Hrsg. von F. Daniel und F. M. Facca. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 338–346 (Zitiert auf S. 106).

[EHR15] B. Ege, B. Humm und A. Reibold, Hrsg. *Corporate Semantic Web: Wie semantische Anwendungen in Unternehmen Nutzen stiften*. Aufl. 2015. X.media.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015 (Zitiert auf S. 34).

[Erl08] T. Erl. *SOA: Principles of service design*. The Prentice Hall service-oriented computing series from Thomas Erl. Upper Saddle River, NJ: Prentice Hall, 2008 (Zitiert auf S. 17, 46).

[Erl09] T. Erl. *SOA design patterns*. 1st ed. Prentice Hall service-oriented computing series from Thomas Erl. Upper Saddle River, NJ und London: Prentice Hall, 2009 (Zitiert auf S. 45, 81).

[FD05] C. Feier und J. Domingue. *D3.1v0.1 WSMO Primer*. Hrsg. von C. Feier. 2005. URL: `http://www.wsmo.org/TR/d3/d3.1/v0.1/` (Zitiert auf S. 91).

[Fie00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. Irvine: University of California, 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (Zitiert auf S. 102).

[Fie08] R. T. Fielding. *REST APIs must be hypertext-driven*. 2008. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (Zitiert auf S. 30, 112).

[Fow10] M. Fowler. *Richardson Maturity Model: Steps Toward the Glory of REST*. 2010. URL: https://martinfowler.com/articles/richardsonMaturityModel.html (Zitiert auf S. 30, 119).

[FT02] R. T. Fielding und R. N. Taylor. „Principled design of the modern Web architecture“. In: *ACM Transactions on Internet Technology (TOIT)* 2.2 (2002), S. 115–150 (Zitiert auf S. 29).

[Gar07] Gartner. *Bad Technical Implementations and Lack of Governance Increase Risks of Failure in SOA Projects*. 2007. URL: http://www.gartner.com/newsroom/id/508397 (Zitiert auf S. 18).

[Gar11] R. Garofalo. *Building enterprise applications with Windows Presentation Foundation and the model view ViewModel Pattern*. Sebastopol, Calif.: O'Reilly Media, 2011 (Zitiert auf S. 150).

[GGH+13] O. Ganschar, S. Gerlach, M. Hämmerle, T. Krause und S. Schlund. *Produktionsarbeit der Zukunft - Industrie 4.0*. Stuttgart: Fraunhofer-Verlag, 2013 (Zitiert auf S. 17, 19).

[Gru93] T. R. Gruber. „A translation approach to portable ontology specifications“. In: *Knowledge acquisition* 5.2 (1993), S. 199–220 (Zitiert auf S. 32).

[GW02] M. Groebner und P. Wilke. „A General View on Timetabling Problems“. In: *Proceedings of the 4th international conference on the Practice and Theory of Automated Timetabling*. Hrsg. von E. Burke und P. DeCausmaecker. Bd. 1. KaHo St. Lieven, 2002, S. 221–227 (Zitiert auf S. 124).

[HD13] B. Hensle und M. Deb. *SOA Maturity Model - Guiding and Accelerating SOA Success*. 2013. URL: https://www.oracle.com/technetwork/topics/entarch/oracle-wp-soa-maturity-model-176717.pdf (Zitiert auf S. 83, 86).

[Hew08] Hewlett Packard Development Company. *The eight most important best practices in SOA governance*. 2008 (Zitiert auf S. 84, 86).

[HHS+16] T. Hoppe, B. Humm, U. Schade, T. Heuss, M. Hemmje, T. Vogel und B. Gernhardt. „Corporate Semantic Web – Applications, Technology, Methodology: Summary of the Dagstuhl Workshop“. In: *Informatik-Spektrum* 39.1 (2016), S. 57–63 (Zitiert auf S. 34, 35).

[HKRS08] P. Hitzler, M. Krötzsch, S. Rudolph und Y. Sure. *Semantic Web: Grundlagen*. 1. Aufl. EXamen.press. Berlin: Springer Berlin, 2008 (Zitiert auf S. 25).

[HPG06] K. Holley, J. Palistrant und S. Graham. *Effective SOA Governance*. 2006. URL: `ftp://ftp.software.ibm.com/software/rational/web/whitepapers/soagov-mgmt.pdf` (Zitiert auf S. 35, 84, 86).

[HS16] M. Hüffmeyer und U. Schreier. „RestACL: An Access Control Language for RESTful Services“. In: *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*. New York, New York: The Association for Computing Machinery, 2016 (Zitiert auf S. 74).

[HW03] G. Hohpe und B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. The Addison Wesley signature series. Boston und London: Addison-Wesley, 2003 (Zitiert auf S. 25, 47).

[Int05] Internet Engineering Task Force. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. URL: `https://www.rfc-editor.org/rfc/rfc3986.txt` (Zitiert auf S. 28).

[Int07] Internet Engineering Task Force. *The Dublin Core Metadata Element Set*. 2007. URL: `https://www.rfc-editor.org/rfc/rfc5013.txt` (Zitiert auf S. 70).

[Int09] International Organization for Standardization. *Information and documentation – The Dublin Core metadata element set*. Geneva, Switzerland, 2009 (Zitiert auf S. 92).

[Int11] Internet Engineering Task Force. *vCard Format Specification*. 2011. URL: `https://www.rfc-editor.org/rfc/rfc6350.txt` (Zitiert auf S. 93).

[Int14] Internet Engineering Task Force. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. 2014. URL: https://www.rfc-editor.org/rfc/rfc7231.txt (Zitiert auf S. 28).

[Int15] Internet Engineering Task Force. *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants*. 2015. URL: https://www.rfc-editor.org/rfc/rfc7522.txt (Zitiert auf S. 113).

[Int18] Internet Engineering Task Force. *JSON Schema: A Media Type for Describing JSON Documents*. 2018. URL: https://tools.ietf.org/html/draft-handrews-json-schema-01 (Zitiert auf S. 117).

[JBSR17] S. Jeschke, C. Brecher, H. Song und D. B. Rawat, Hrsg. *Industrial Internet of Things: Cybermanufacturing Systems*. Springer Series in Wireless Technology. Cham: Springer International Publishing, 2017 (Zitiert auf S. 16, 44).

[Jos07] N. M. Josuttis. *SOA in practice*. Beijing und Cambridge: O'Reilly, 2007 (Zitiert auf S. 44).

[KAS09] T. Kokko, J. Antikainen und T. Systä. „Adopting SOA - Experiences from Nine Finnish Organizations". In: *13th European Conference on Software Maintenance and Reengineering, 2009*. Hrsg. von R. Ferenc. Piscataway, NJ: IEEE, 2009, S. 129–138 (Zitiert auf S. 45, 65).

[Kas17] L. B. Kassner. „Product Life Cycle Analytics". Dissertation. Stuttgart: Universität Stuttgart, Graduate School advanced Manufacturing Engineering, 2017 (Zitiert auf S. 20).

[KG12] P. Kalamegam und Z. Godandapani. „A Survey on Testing SOA Built using Web Services". In: *International Journal of Software Engineering and Its Applications* 6.4 (2012), S. 91–104 (Zitiert auf S. 122).

[KGK+17] L. Kassner, C. Gröger, J. Königsberger, E. Hoos, C. Kiefer, C. Weber, S. Silcher und B. Mitschang. „The Stuttgart IT Architecture for Manufacturing". In: *Enterprise Information Systems: 18th International Conference, ICEIS 2016, Rome, Italy, April 25–28, 2016, Revised Selected Papers*. Hrsg. von S. Hammoudi, L. A. Maciaszek, M. M. Missikoff, O. Camp und J. Cordeiro. Cham: Springer International Publishing, 2017, S. 53–80 (Zitiert auf S. 20, 40, 171).

[KHW+17] L. Kassner, P. Hirmer, M. Wieland, F. Steimle, J. Königsberger und B. Mitschang. „The Social Factory: Connecting People, Machines and Data in Manufacturing for Context-Aware Exception Escalation“. In: *50th Hawaii International Conference on System Sciences*. AIS Electronic Library (AISeL), 2017, S. 1673–1682 (Zitiert auf S. 171).

[KM16] J. Königsberger und B. Mitschang. „A Semantically-enabled SOA Governance Repository“. In: *Proceedings of the 2016 IEEE 17th International Conference on Information Reuse and Integration*. Los Alamitos, California, Washington, Tokyo: IEEE Computer Society, 2016, 423–432 ©2016 IEEE (Zitiert auf S. 21–23, 75, 95–99, 146).

[KM17] J. Königsberger und B. Mitschang. „Business Objects plus (BO+): An Approach to Enhance Service Reuse and Integration in Cross-Domain SOA Compounds“. In: *Proceedings of the 2017 IEEE International Conference on Information Reuse and Integration*. Hrsg. von C. Zhang, B. Palanisamy, L. Khan und S. Sedigh Sarvestani. Los Alamitos, California, Washington, Tokyo: IEEE Computer Society, 2017, 49–58 ©2017 IEEE (Zitiert auf S. 21, 22, 43, 51, 52, 58, 63, 66, 67, 72).

[KM18] J. Königsberger und B. Mitschang. „R2SMA - A Middleware Architecture to Access Legacy Enterprise Web Services using Lightweight REST APIs“. In: *Proceedings of the 20th International Conference on Enterprise Information Systems*. Hrsg. von S. Hammoudi, M. Smialek, O. Camp und J. Filipe. Bd. 2. SciTePress, 2018, S. 704–711 (Zitiert auf S. 21, 22, 101, 108).

[KOTW06] H. Knublauch, D. Oberle, P. Tetlow und E. Wallace. *A Semantic Web Primer for Object-Oriented Software Developers*. 2006. URL: `https://www.w3.org/2001/sw/BestPractices/SE/ODSD/` (Zitiert auf S. 143, 145).

[Kra16a] P. Kraus. „Dokumentation und automatische Generierung von Service-Beschreibungen“. Bachelorarbeit. Stuttgart: University of Stuttgart, 2016 (Zitiert auf S. 115).

[Kra16b] D. Krauss. „Generierung und Optimierung von Testzeitplänen im Rahmen des SOA Change Managements“. Masterarbeit. Stuttgart: University of Stuttgart, 2016 (Zitiert auf S. 121, 136).

[KS11] A. Katzenbach und H.-P. Steiert. „Engineering-IT in der Automobilindustrie – Wege in die Zukunft“. In: *Informatik-Spektrum* 34.1 (2011), S. 7–19 (Zitiert auf S. 37).

[KSH08] O. Kohnke, T. Scheffler und C. Hock. „SOA-Governance: Ein Ansatz zum Management serviceorientierter Architekturen“. In: *Wirtschaftsinformatik* 8 (2008), S. 408–412 (Zitiert auf S. 83, 86).

[KSM14] J. Königsberger, S. Silcher und B. Mitschang. „SOA-GovMM: A Meta Model for a Comprehensive SOA Governance Repository“. In: *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*. Hrsg. von J. Joshi, E. Bertino, B. Thuraisingham und L. Liu. IEEE Systems, Man, and Cybernetics Society (SMC), 2014, 187–194 ©2014 IEEE (Zitiert auf S. 21, 22, 75, 78, 90).

[LDA11] Lohmann, Steffen, P. Diaz und I. Aedo. „MUTO: The Modular Unified Tagging Ontology“. In: *Proceedings of the 7th International Conference on Semantic Systems*. Hrsg. von C. Ghidini. New York, NY: ACM, 2011 (Zitiert auf S. 153).

[Lee08] E. A. Lee. „Cyber Physical Systems: Design Challenges“. In: *Proceedings of the 2008 11th IEEE Symposium on Object-Oriented Real-Time Distributed Computing*. Los Alamitos Calif.: IEEE Computer Society, 2008, S. 363–369 (Zitiert auf S. 44).

[Lho98] R. Lhotka. *Visual Basic 6.0 business objects*. Birmingham: Wrox Press, 1998 (Zitiert auf S. 48).

[Lig09] P. Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. 2. Aufl. Heidelberg: Spektrum, Akad. Verl., 2009 (Zitiert auf S. 127).

[Mal15] P. Malinverno. *Magic Quadrant for Application Services Governance*. 2015. URL: `https://www.gartner.com/doc/3025524/` (Zitiert auf S. 158).

[Mee08] M. Meehan. *SOA adoption marked by broad failure and wild success*. 2008. URL: `http://searchsoa.techtarget.com/news/1319609/SOA-adoption-marked-by-broad-failure-and-wild-success` (Zitiert auf S. 17, 18).

[Min12] J. Minguez. „A Service-oriented Integration Platform for Flexible Information Provisioning in the Real-time Factory“. Dissertation. Stuttgart: Universität Stuttgart, Graduate School of advanced Manufacturing Engineering, Germany, 2012. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIS-2012-04&mod=0&engl=0&inst=AS (Zitiert auf S. 19).

[MO18] P. Malinverno und M. O'Neill. *Magic Quadrant for Full Life Cycle API Management*. 2018. URL: https://www.gartner.com/doc/3873383/ (Zitiert auf S. 158).

[Nat05] National Information Standards Organization (U.S.) *Guidelines for the construction, format, and management of monolingual controlled vocabularies: An American national standard*. Bethesda, Md., 2005 (Zitiert auf S. 45).

[NB16] C. Nöllenheidt und S. Brenscheidt. *Arbeitswelt im Wandel: Zahlen - Daten - Fakten*. 2016. URL: https://www.baua.de/DE/Angebote/Publikationen/Praxis/A95.html (besucht am 31.01.2018) (Zitiert auf S. 17).

[NERS08] M. Niemann, J. Eckert, N. Repp und R. Steinmetz. „Towards a Generic Governance Model for Service-oriented Architectures“. In: *Proceedings of the Fourteenth Americas Conference on Information Systems*. Hrsg. von I. Benbasat und A. R. Montazemi. Madison: Omnipress, 2008 (Zitiert auf S. 83, 86).

[NF14] F. Nogatz und T. W. Frühwirth. „From XML Schema to JSON Schema: Translation with CHR“. In: *Proceedings of the Eleventh Workshop on Contraint Handling Rules*. Hrsg. von R. Haemmerlé und J. Sneyers. Bd. abs/1406.2125. 2014 (Zitiert auf S. 117).

[OAS07] OASIS. *Web Services Business Process Execution Language*. 2007. URL: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (Zitiert auf S. 28).

[OAS09] OASIS. *Web Services Reliable Messaging (WS-ReliableMessaging)*. 2009. URL: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html (Zitiert auf S. 28, 104).

[OAS18] OASIS. *OSLC Core*. 2018. URL: http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/oslc-core-v3.0-part1-overview.html (Zitiert auf S. 70).

[Obj15] Object Management Group. *Financial Industry Business Ontology Foundations*. 2015. URL: http://www.omg.org/spec/EDMC-FIBO/FND/1.0 (Zitiert auf S. 94).

[Obj17] Object Management Group. *Unified Modeling Language*. 12.2017 (Zitiert auf S. 58).

[Ope18] Open API Initiative. *The OpenAPI Specification*. 2018. URL: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md (Zitiert auf S. 29, 113, 117).

[Ora14] Oracle. *Oracle Spatial and Graph: Benchmarking a Trillion Edges RDF Graph*. 2014. URL: http://download.oracle.com/otndocs/tech/semantic_web/pdf/OracleSpatialGraph_RDFgraph_1_trillion_Benchmark.pdf (Zitiert auf S. 144).

[PCH+10] A. Paschke, G. Coskun, R. Heese, M. Luczak-Rösch, R. Oldakowski, R. Schäfermeier und O. Streibel. „Corporate Semantic Web: Towards the Deployment of Semantic Technologies in Enterprises“. In: *Canadian Semantic Web*. Hrsg. von W. Du und F. Ensan. SpringerLink: Springer e-Books. Boston, MA: Springer Science+Business Media, LLC, 2010, S. 105–131 (Zitiert auf S. 34).

[PML09] Y. Y. Peng, S. P. Ma und J. Lee. „REST2SOAP: A framework to integrate SOAP services and RESTful services“. In: *IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2009*. Piscataway, NJ: IEEE, 2009 (Zitiert auf S. 106).

[Pre02] P. Prescod. „Roots of the REST/SOAP Debate“. In: *Proceedings of the Extreme Markup Languages 2002*. 2002 (Zitiert auf S. 106).

[Pre13] Preston-Werner, Tom. *Semantic Versioning 2.0.0*. 2013. URL: http://semver.org/spec/v2.0.0.html (Zitiert auf S. 60).

[Pri09] PriceWaterhouseCoopers. *SOA Governance: More than just registries and repositories*. 2009 (Zitiert auf S. 84, 86).

[PSVS07] S. Poi, B. Sleight, J. Viezel und D. Snavely. *Enabling SOA through organizational change and governance*. 2007. URL: ftp://public.dhe.ibm.com/software/solutions/soa/gov/GBS_SOAgov_Org_Change.pdf (Zitiert auf S. 18, 45).

[Pv07] M. P. Papazoglou und W.-J. van den Heuvel. „Service oriented architectures: approaches, technologies and research issues“. In: *The VLDB Journal* 16.3 (2007), S. 389–415 (Zitiert auf S. 25).

[PZL08] C. Pautasso, O. Zimmermann und F. Leymann. „Restful Web Services vs. 'Big' Web Services: Making the Right Architectural Decision“. In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. New York, NY, USA: ACM, 2008, S. 805–814 (Zitiert auf S. 28–30, 106).

[QSM09] M. Quasthoff, H. Sack und C. Meinel. „How to simplify building semantic web applications“. In: *SWESE 2009: 5th International Workshop on Semantic Web Enabled Software Engineering*. Hrsg. von E. F. Kendall, J. Z. Pan, M. Sabbouh, L. Stojanovic und Y. Zhao. 2009, S. 45–57 (Zitiert auf S. 145).

[Ram13] M. Ramakrishnan. *SOA Lifecycle Governance: Right from the Start*. 2013. URL: http://www.oracle.com/us/products/middleware/soa/soa-lifecycle-governance-wp-1971496.pdf (Zitiert auf S. 35, 83, 86).

[Ran11] S. Rance. *ITIL service transition*. 2nd ed. / [Stuart Rance, author]. London: TSO, 2011 (Zitiert auf S. 37).

[RB08] I. Rieger und R. Bruns. „SOA-Governance und -Rollen: Sichern des Mehrwerts einer Service-orientierten Architektur“. In: *OBJEKTspektrum* (2008) (Zitiert auf S. 38, 81, 83–86).

[RLSB12] M. Rosen, B. Lublinsky, K. T. Smith und M. J. Balcer. *Applied SOA: service-oriented architecture and design strategies*. John Wiley & Sons, 2012 (Zitiert auf S. 26).

[RR07] L. Richardson und S. Ruby. *RESTful web services*. Beijing und Farnham: O'Reilly, 2007 (Zitiert auf S. 28).

[SGK+15] S. Silcher, E. Groß, J. Königsberger, J. Siegert, M. Lickefett, T. Bauernhansl und B. Mitschang. „Mobile Factory Layout Planning: How Apps Can Provide an Advantage in Factory Planning“. In: *wt Werkstattstechnik online* 105.3 (2015), S. 96–101 (Zitiert auf S. 17).

[Sil14] S. Silcher. „Adaptive und wandlungsfähige IT-Architektur für Produktionsunternehmen“. Dissertation. Stuttgart: Universität Stuttgart, 2014. URL: https://elib.uni-stuttgart.de/handle/11682/3403 (Zitiert auf S. 17, 19).

[Swa12] P. Swabey. „Divide & Conquer: The science of splitting large IT projects into manageable chunks“. In: *InformationAge magazine* June (2012) (Zitiert auf S. 17).

[Sys06] Systinet. *SOA Governance: Balancing Flexibility and Control Within an SOA*. 2006 (Zitiert auf S. 84, 86).

[The09a] The Open Group. *SOA Governance Framework*. Berkshire, 8.2009. URL: http://www.opengroup.org/soa/source-book/gov/index.htm (Zitiert auf S. 38, 84, 86).

[The09b] The Open Group. *SOA Source Book*. 1st ed. The Open Group series. Zaltbommel: Van Haren Publishing, 2009 (Zitiert auf S. 26).

[The14] The Open Group. *Service-Oriented Architecture Ontology*. 2014. URL: https://publications.opengroup.org/standards/soa/c144 (Zitiert auf S. 91).

[Til15] S. Tilkov. *Why You Should Avoid a Canonical Data Model*. 2015. URL: https://www.innoq.com/en/blog/thoughts-on-a-canonical-data-model/ (Zitiert auf S. 45, 47).

[Ver08] Verband der Automobilindustrie. *Standardbelieferungsformen der Logistik in der Automobilindustrie*. 2008. URL: https://www.vda.de/de/services/Publikationen/5010---standardbelieferungsformen.html (Zitiert auf S. 16).

[VMS12] M. Voigt, A. Mitschick und J. Schulz. „Yet Another Triple Store Benchmark? Practical Experiences with Real-World Data“. In: *2nd International Workshop on Semantic Digital Archives (SDA2012)*. Hrsg. von A. Mitschick, F. Loizides, L. Predoiu, A. Nürnberger und S. Ross. 2012, S. 85–94 (Zitiert auf S. 144).

[W3C04] W3C. *OWL-S: Semantic Markup for Web Services*. 2004. URL: https://www.w3.org/Submission/OWL-S/ (Zitiert auf S. 91).

[W3C07a] W3C. *https://www.w3.org/TR/soap12-part0/*. 2007. URL: https://www.w3.org/TR/soap12-part0/ (Zitiert auf S. 27).

[W3C07b] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 2007. URL: https://www.w3.org/TR/wsdl20/ (Zitiert auf S. 27).

[W3C07c] W3C. *Web Services Description Language (WSDL) Version 2.0: RDF Mapping*. 2007. URL: https://www.w3.org/TR/wsdl20-rdf/ (Zitiert auf S. 93).

[W3C07d] W3C. *Web Services Policy 1.5 - Framework*. 2007. URL: https://www.w3.org/TR/ws-policy/ (Zitiert auf S. 28).

[W3C09] W3C. *SKOS Simple Knowledge Organization System*. 2009. URL: https://www.w3.org/TR/skos-reference/ (Zitiert auf S. 94).

[W3C11a] W3C. *Notation3 (N3): A readable RDF syntax*. 2011. URL: https://www.w3.org/TeamSubmission/n3/ (Zitiert auf S. 31).

[W3C11b] W3C. *SPIN - Overview and Motivation*. 2011. URL: https://www.w3.org/Submission/spin-overview/ (Zitiert auf S. 154).

[W3C12] W3C. *OWL 2 Web Ontology Language*. 2012. URL: https://www.w3.org/TR/owl2-overview/ (Zitiert auf S. 33, 34).

[W3C13a] W3C. *SPARQL 1.1 Query Language*. 2013. URL: https://www.w3.org/TR/sparql11-query/ (Zitiert auf S. 32).

[W3C13b] W3C. *W3C Semantic Web Activity*. 2013. URL: https://www.w3.org/2001/sw/ (Zitiert auf S. 142).

[W3C14a] W3C. *RDF 1.1 Concepts and Abstract Syntax*. 2014. URL: https://www.w3.org/TR/rdf11-concepts/ (Zitiert auf S. 31, 69, 142).

[W3C14b] W3C. *RDF 1.1 XML Syntax*. 2014. URL: https://www.w3.org/TR/rdf-syntax-grammar/ (Zitiert auf S. 31).

[W3C14c] W3C. *RDF Schema 1.1*. 2014. URL: https://www.w3.org/TR/rdf-schema/ (Zitiert auf S. 32).

[W3C14d] W3C. *The Organization Ontology*. 2014. URL: https://www.w3.org/TR/vocab-org/ (Zitiert auf S. 93).

[W3C14e] W3C. *vCard Ontology - for describing People and Organizations*. 2014. URL: https://www.w3.org/TR/vcard-rdf/ (Zitiert auf S. 93).

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey und D. F. Ferguson. *Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more*. Upper Saddle River, N.J.: Prentice Hall PTR und London : Pearson Education [distributor], 2005 (Zitiert auf S. 25).

[Web10] Web Services-Interoperability Organization. *Basic Profile Version 2.0*. 2010. URL: http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html (Zitiert auf S. 28).

[WSCL13] E. Westkämper, D. Spath, C. Constantinescu und J. Lentes, Hrsg. *Digitale Produktion*. Aufl. 2013. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013 (Zitiert auf S. 16).

Alle Internetreferenzen wurden zuletzt am 16.12.2018 geprüft.

Abbildungsverzeichnis

Tabellenverzeichnis