Domänengetriebener Entwurf von ressourcenorientierten Microservices

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Pascal Giessler

aus Lahr/Schwarzwald

Tag der mündlichen Prüfung:	04.05.2018
Erster Gutachter:	Prof. Dr. Sebastian Abeck
Zweiter Gutachter:	Prof. Dr. Ralf Reussner

„You simply have to put one foot in front of the other and keep going. Put blinders on and plow right ahead.“
— George Lucas

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als akademischer Mitarbeiter in der Forschungsgruppe Cooperation & Management (C&M) der Fakultät für Informatik am Karlsruher Institut für Technologie (KIT), der hauptberuflichen Tätigkeit als Software-Ingenieur bei der iteratec GmbH sowie meiner letztlichen Tätigkeit als Entwicklungsleiter in der Digitalagentur SYNDIKAT7 GmbH (ehemals ZWEI14 Digital GmbH). An dieser Stelle möchte ich die Gelegenheit nutzen, mich bei allen Personen ganz herzlich zu bedanken, die mich während dieser Zeit unterstützt haben.

Zunächst möchte ich mich bei meinem Doktorvater, Herrn Prof. Dr. Sebastian Abeck, für die Betreuung ganz herzlich bedanken. Die geführten Diskussionen haben mir dabei geholfen meine Fragestellungen und so die Ausrichtung meiner Arbeit sowie die einzelnen Forschungsbeiträge entscheidend zu schärfen. Darüber hinaus konnte ich stets auf seine Unterstützung sowohl in fachlichen als auch persönlichen Fragestellungen setzen. Weiter bedanke ich mich für sein entgegengebrachtes Vertrauen bei der Betreuung von Studierenden sowie dem Halten von Vorlesungseinheiten. Daneben gilt mein Dank Herrn Prof. Dr. Ralf Reussner, der die Rolle des Korreferenten einnahm und mir dabei geholfen hat der Arbeit den letzten Feinschliff zu geben.

Darüber hinaus gilt mein Dank meinen akademischen Kollegen Benjamin Hippchen, Philip Hoyer, Michael Schneider und Roland Steinegger bei C&M. Ohne das harmonische Zusammenarbeiten und das freundschaftliche Verhältnis auch außerhalb des Instituts wäre das effiziente und zielgerichtete wissenschaftliche Arbeiten in dieser Form nicht möglich gewesen.

An dieser Stelle möchte ich mich auch bei allen Studierenden bedanken, die ich in dieser Zeit betreut habe. In Zusammenarbeit wurden neue Ideen generiert und deren Tauglichkeit erprobt, die letztlich als Grundlage für einige Ansätze dieser Arbeit dienten. Hier möchte ich mich insbesondere bei meinen Masteranden bedanken: Pascal Burkhardt, Danny Forscher, Giullian Franke, Manuel Gerster, Dimitrij Sarancin und Nils Sommer.

Weiter gilt mein Dank meinen ehemaligen Arbeitskollegen bei der iteratec GmbH. Hervorheben möchte ich an dieser Stelle Heiko Schrader, Geschäftsführer der iteratec GmbH in Stuttgart, der mir in dieser Zeit zahlreiche Einblicke in verschiedene Bereiche der Industrie ermöglicht hat. Ein besonderer Dank gilt Dr. Michael Gebhart, der während meiner gesamten Promotionszeit und auch nach Ausscheiden bei der iteratec GmbH als Freund und ehemaliger Betreuer meiner Masterarbeit stets ein offenes Ohr für mich hatte und mir wertvolle Ratschläge gegeben hat. Auch das gemeinsame wissenschaftliche Publizieren hat mir stets viel Vergnügen bereitet.

Daneben möchte ich mich bei meinen Arbeitskollegen bei der SYNDIKAT7 GmbH für ihr Verständnis vor allem in der Schlussphase meiner Arbeit bedanken. Besonders hervorheben möchte ich hier Mario Beiser, der mir als Freund, Mentor und Geschäftsführer bei der SYNDIKAT7 GmbH stets den Rücken gestärkt und mich unterstützt hat sowie mir den notwendigen Freiraum zur Verwirklichung meiner Ziele gegeben hat.

Abschließend gilt mein besonderer Dank meiner Familie insbesondere meiner Eltern, die mich durch das gesamte Studium begleitet haben. Daneben bedanke mich bei meiner Lebensgefährtin, die mir in dieser Zeit zur Seite stand und mir unglaubliche Kraft sowie Verständnis entgegengebracht hat. Auch gilt mein Dank der Familie Rein, Schies, Schnell und Wilbois. Sie alle haben mir die notwendige Kraft und Durchhaltevermögen gegeben und mich bedingungslos in jeder Situation unterstützt.

Pascal Giessler
Karlsruhe, Juni 2018

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Heutzutage sollen Informationssysteme in Unternehmen immer häufiger geschäftsrelevante Aktivitäten durch Automatismen unterstützen oder bei Bedarf alle benötigten Informationen zu einer Aktivität verfügbar machen [Kr10, Ge11, DL12]. Infolge dieser steigenden Abhängigkeit müssen sich die Informationssysteme immer schneller den Bedürfnissen eines Unternehmens bzw. dessen geschäftlichen Anforderungen anpassen, damit die Unternehmen weiterhin wettbewerbsfähig bleiben [Ge11]. Diese Ausrichtung der Informationssysteme auf die geschäftlichen Aktivitäten eines Unternehmens (engl. IT/business alignment) bringt allerdings einige Herausforderungen mit sich [DL12, Ha16]. So ist bspw. unklar, ob sich die im Einsatz befindlichen Informationssysteme schnell genug an neue Bedürfnisse eines Unternehmens anpassen lassen und ob die erbrachte Leistung durch eben diese Informationssysteme den Bedürfnissen eines Unternehmens gerecht wird [DL12]. Diese Bedürfnisse werden als fachliche Anforderungen an das Informationssystem gestellt. Veränderte Bedürfnisse können in einem Unternehmen viele Ursachen haben. So ist heutzutage ein Unternehmen zahlreichen Faktoren ausgesetzt, die seine Geschäftsaktivitäten beeinflussen – etwa veränderliche Marktverhältnisse, neue gesetzliche Regelungen, die Integration bestehender Systeme, neue Plattformen, sich wandelndes Kundenverhalten oder neu verfügbare Technologien [Kr10, Ge11, DL12]. Eine hohe Flexibilität und Wirtschaftlichkeit bei der Anpassung von Informationssystemen ist daher erstrebenswert.

Um diesen Anforderungen gerecht zu werden, setzten bisher viele Unternehmen bei der Gestaltung ihrer Informationssysteme zunehmend auf serviceorientierte Architekturen (engl. Service-Oriented Architecture, SOA) [Jo07, Ge11]. Innerhalb der serviceorientierten Architektur finden sich wiederum sogenannte Services, welche eine immaterielle Dienstleistung über eine wohldefinierte Schnittstelle (kurz: Web-Application Programming Interface (Web-API)) einem Service-Konsumenten bereitstellen und damit die geschäftlichen Aktivitäten durch Informationstechnologie (IT) abbilden [Ge11]. Mittels einer Komposition von Services konnten mehrere geschäftliche Aktivitäten miteinander verknüpft und damit ein kompletter Geschäftsprozess abgebildet werden [Ge11]. Trotz umfangreicher Forschung und Literatur [Er05, Jo07, Ge11, DL12] sowie neu entwickelten Werkzeugen [Ge14], welche die Umsetzung einer serviceorientierten Architektur (SOA) unterstützen sollten, gestaltete sich die Realisierung für Unternehmen aufgrund der Komplexität des SOA-Architekturstils als schwierig. Auch konnten Unternehmen die Vorzüge durch Anwendung der Serviceorientierung nur schwer quantifizieren und qualifizieren. Dies hatte zur Folge, dass sich viele Unternehmen von SOAs lossagten [Ne15, Ri15a].

In den letzten Jahren wurde ein neuer Ansatz für die Umsetzung von SOAs als Reaktion auf kaum noch wartbare Software-Monolithen und Herausforderungen in Bezug auf die Skalierbarkeit dieser Software-Systeme entwickelt [BG+16]. Gleichzeitig haben sich in diesem Zeitraum viele neue Technologien im Bereich der Virtualisierung und Infrastrukturautomatisierung hervorgetan, die diesen Ansatz erst ermöglicht haben [Ne15]. Feingranulare und autonome Services, sogenannte Microservices, sollten helfen, neue fachliche Anforderungen im Vergleich zum ursprünglichen Ansatz der SOA deutlich schneller umzusetzen und gleichzeitig das Software-System wesentlich modularer zu gestalten [Ne15]. Diese Microservices weisen einen engen Bezug zu der abzubildenden Geschäftsdomäne auf, weshalb dem domänengetriebenen Ansatz nach Evans [Ev03] zur Strukturierung von Software-Systemen in diesem Kontext eine hohe Bedeutung zu Teil wurde. Anders als das ursprüngliche Konzept einer SOA entstammt dieser Ansatz aus der Praxis, wodurch sich seine Tragfähigkeit bereits durch umgesetzte und lauffähige Projekte demonstrieren ließ [Ne15, Ri15a]. Beispiele sind etwa Netflix, Amazon oder Zalando, deren Informationssysteme zunehmend auf Microservices setzen [Ne15, Ti15, Gi16]. Aber auch andere Unternehmen aus unterschiedlichen Branchen investieren gegenwärtig viel Zeit und Geld in die Modernisierung ihrer Informationssysteme mithilfe von Microservices.

Der systematische und nachvollziehbare Entwurf derartiger Microservices stellt daher eine wesentliche Aufgabe dar, um zukunftssichere Microservice-Architekturen zu konzipieren und so eine hohe Flexibilität der Informationssysteme zu gewährleisten. Dieser Umstand wurde bereits von Gebhart [Ge11] im Kontext serviceorientierter Architekturen erkannt, da der Entwurf letztlich die Spezifikation für die Implementierung des jeweiligen Service darstellt. In diesem Zusammenhang verweist er auf die Arbeiten von Erl [Er08] und Perepletchikov et al. [PR+08]. Zudem betont er die Notwendigkeit für ein nachvollziehbares Vorgehen, damit sich systematisch aus den Artefakten der Analysephase die passenden Entwürfe für Services ableiten lassen. Gleichzeitig sind auch die Bedürfnisse der Konsumenten der Services bzw. Microservices zu berücksichtigen, welche die angebotene Dienstleistung verwenden bzw. integrieren, um neue Anwendungsfälle zu bedienen. Für die Integration von Microservices dient deren veräußerte Web-API, die in geeigneter Form spezifiziert werden muss, um eine reibungslose Kommunikation zu ermöglichen. Deswegen wurden Web-APIs in den letzten Jahren als wichtiger Vermögenswert (engl. business asset) eines Unternehmens und als Antrieb der digitalen Transformation erkannt, was sich u. a. in den steigenden Einnahmen durch die Bereitstellung von öffentlichen Web-APIs zeigt [JB+11, IS15, GG+16a]. Ein weiteres Indiz für diese Entwicklung ist die steigende Nachfrage nach API-Management-Systemen zur zielgerichteten Verwaltung. So werden nach Forrester Research die jährlichen Ausgaben für API-Management allein von Unternehmen in den Vereinigten Staaten (engl. United States, kurz: US) bis 2020 voraussichtlich auf 660 Millionen US-Dollar steigen [HY+15]. Aus diesem Grund ist der Entwurf einer Web-API ein wichtiger Bestandteil des Entwurfs eines Service bzw. Microservice und sollte unter Berücksichtigung von unterschiedlichen Qualitätseigenschaften konzipiert werden, wie bspw. im Hinblick auf dessen Langlebigkeit oder auch Benutzbarkeit.

Als Architekturstil für Services bzw. Microservices im Hinblick auf deren Veräußerung von Dienstleistungen über eine Web-API hat sich REST über die Jahre zunehmender Beliebtheit erfreut. Allerdings zeigen Untersuchungen, dass nicht immer alle Randbedingungen von REST beim Entwurf berücksichtigt werden. Insbesondere der Hypermedia-Aspekt, der dem Forschungsbereich des semantischen Web zuzuordnen ist, erscheint diesbezüglich als Herausforderung [RS+12, BM14]. Das hat vielfältige Ursachen und führte dazu, dass viele Unternehmen eher einen ressourcenorientierten Stil statt eines Hypermedia-getriebenen Ansatzes verfolgen. Die Ressourcenorientierung kann hier als Vorstufe des zuvor genannten Ansatzes angesehen werden.

Obwohl Literatur zu Microservices mittlerweile die grundlegenden Konzepte beschreibt, fehlt bisher ein systematisches und nachvollziehbares Vorgehen für den Entwurf von Microservices auf Grundlage eines Domänenmodells nach Evans [Ev03] sowie dessen Überführung auf die Implementierungsebene, der auch den Entwurf einer Web-API in adäquater Weise betrachtet. Evans stellt in diesem Zuge kein Bezug zu Microservices oder Web-APIs her, während Vernon [Ve13] zumindest ein Bezug zum Architekturstil REST und dessen Eignung aufzeigt: „a system designed conforming to REST principles fulfills the promise of loose coupling“ [Ve13, S. 138]. Allerdings liefern weder Evans noch Vernon eine einheitliche Modellierung für den domänengetriebenen Entwurf anhand dessen letztlich weitere nachvollziehbare Verfeinerungen im Bezug auf die Web-API durchgeführt werden können und schließlich eine systematische Überführung auf die Implementierungsebene erfolgen kann. Weiter kann nur durch die Berücksichtigung von Qualitätseigenschaften beim Entwurf gewährleistet werden, dass der resultierende Microservice auch einen hohen Grad an Wiederverwendbarkeit aufweist. Dies wiederum liefert die Voraussetzung für eine schrittweise digitale Transformation der Geschäftsdomäne in eine Microservice-Architektur, während gleichzeitig die Gefahr von sogenannten Insellösungen reduziert wird.

1.2 Einordnung der Arbeit

Die Arbeit lässt sich in den Bereich der Softwaretechnik mit Schwerpunkt auf dem Entwurf von Software (engl. Software Design) einordnen, welcher ein Teilbereich der Informatik darstellt. Als Entwurf von Software gilt ein Vorgehen, bei dem Anforderungen zur Beschreibung der internen Struktur einer Software analysiert werden. Das Resultat ist eine Beschreibung der sogenannten Software-Architektur, die als Grundlage für die Umsetzung der Software dienen soll [BF14]. Die Software-Architektur beschreibt „how software is decomposed and organized into components – and the interfaces between those components“ [BF14, S. 50]. Eine Komponente wiederum ist „eine Kompositionseinheit mit vertraglich spezifizierten Schnittstellen, die nur explizite Abhängigkeiten zu ihrem Kontext hat. Eine Software-Komponente kann unabhängig eingesetzt werden und sie kann durch Dritte komponiert werden“ [VA+09, S. 162] zitiert nach [Sz98].

Die in dieser Arbeit betrachteten Microservices können als betriebene Software-Komponenten mit

bestimmten Eigenschaften angesehen werden. So zeichnen sie sich u. a. durch ihre unabhängige Verteilung, die Möglichkeit, durch verschiedene Entwicklungsteams entwickelt zu werden, oder durch Technologieheterogenität aus [Ne15, NM+16]. Der Fokus dieser Arbeit liegt in der Mikroarchitektur eines oder mehrerer Microservices insbesondere derer Web-APIs, welche die vertraglich spezifizierten Schnittstellen darstellen. Eine Web-API veräußert dabei die immateriellen Dienstleistungen eines Microservice über ein Netzwerk nach dem Blackbox-Prinzip, sodass für Außenstehende die Implementierung gemäß des *Information Hiding*-Prinzips verborgen bleibt [Pa72]. Die einzelnen Microservices ergeben sich durch die Aufteilung der zugrundeliegenden Geschäftsdomäne in logische und organisatorische Einheiten im Zuge von IT/Business Alignment. Hier wird der Ansatz von Evans [Ev03] genutzt, der zentral für diese Arbeit ist.

Die Zielgruppe der Ergebnisse dieser Arbeit sind Software-Architekten und letztendlich die Auftraggeber, die bei dem Entwurf von ressourcenorientierten Microservices durch Anwendung des domänengetriebenen Entwurfs nach Evans [Ev03] unterstützt werden sollen. Bei Software-Architekten handelt es sich meist um erfahrene Software-Ingenieure, die vor allem Software entwerfen und den Auftraggeber bei der Entwicklung von Software beraten. Die, durch diese Arbeit, erbrachte Unterstützung beim Entwurf soll Fehler reduzieren, Entwurfsentscheidungen auf dieser konzeptionellen Ebene durch Wissen und Erfahrungen vereinfachen und die Wiederverwendbarkeit des Microservice verbessern. Ein weiteres Ziel ist es, die so gewonnenen Entwurfsartefakte als Vorgabe für die Implementierung des Microservice zu nutzen und so Implementierungsfehler zu minimieren.

1.3 Betrachtetes Szenario

Die in dieser Arbeit bearbeiteten Problemstellungen und der Rahmen dieser Arbeit werden beispielhaft durch das Szenario in Abbildung 1.1 veranschaulicht.

Ausgangssituation

In diesem Szenario sind Nutzeranforderungen, also grundsätzlich das Bedürfnis nach informationstechnischer Unterstützung, die Ausgangssituation für den domänengetriebenen Entwurf von mindestens einem ressourcenorientierten Microservice. Sie umfassen anwendungs- als auch geschäftsbezogene Anforderungen. Diese Arbeit fokussiert die geschäftsbezogenen Anforderungen, um auf sie gestützt die Geschäftsdomäne bzw. einen Ausschnitt davon (nachfolgend als „Geschäftsausschnitt“ bezeichnet) zu definieren, der für die Umsetzung der Nutzeranforderungen benötigt wird. Die erforderlichen Informationen werden schließlich in einem sogenannten Domänenmodell festgehalten und in Zusammenarbeit mit einem oder mehrerer Domänenexperten sowie durch Sichtung etwaiger Artefakte mit weiterführenden und ergänzenden Informationen weiter verfeinert. So sind bspw. die Begrifflichkeiten des Geschäftsausschnitts sowie die gewünschten Funktionalitäten des zu entwickelnden

Software-Systems aus den geschäftsbezogenen Anforderungen wichtige Informationen, die im Domänenmodell angemessen formalisiert und verwendet werden. Das Domänenmodell repräsentiert damit die fachlichen Anforderungen an das resultierende System und legt den Ausschnitt für die durch die IT mittels Microservices umzusetzenden Geschäftsdomäne fest. Konkret handelt es sich bei einem Domänenmodell laut Evans [Ev03] um: „not just the knowledge in a domain expert's head; it is a rigorously organized and selective abstraction of that knowledge“ [Ev03, S. 3]. Dies bedeutet, dass darin lediglich das benötigte Wissen zum Verständnis und zur Abbildung eines Geschäftsausschnitts strukturiert in einem Modell abgebildet ist. Zudem lässt sich das Domänenmodell in weitere Teildomänen unterteilen, um etwa organisatorische Einheiten und deren Zugehörigkeit wiederzugeben. Geschäftliche Entitäten werden durch sogenannte Domänenobjekte abgebildet, die sich mit weiteren Domänenobjekten in Beziehung setzen lassen. Gleichzeitig wird mit der Modellierung ein einheitliches Vokabular geschaffen, das die Kommunikation zwischen Domänenexperten, Software-Architekten und -Entwicklern vereinfachen soll [Ge11, Ev03]. Durch diese festgelegten Begrifflichkeiten und deren Semantik können laut Gebhart Missverständnisse und Unklarheiten deutlich reduziert werden. Evans bezeichnet dieses Vokabular selbst als „Ubiquitous Language“ [Ev03].

Domänengetriebener Entwurf von ressourcenorientierten Microservices

Ausgehend von dem im Vorfeld beschriebenen Domänenmodell werden nun die entsprechenden Microservices durch Befolgen eines ressourcenorientierten Architekturstils entworfen. Laut Gebhart [Ge11] aus [Jo08] kann der Entwurf auf drei verschiedene Arten erfolgen: (1) *Top-down*, (2) *Bottom-up* und (3) *Middle-out*. Bei einem *Top-down*-Ansatz werden Microservices ohne Berücksichtigung schon existierender Microservices entwickelt. Erst nach der Veröffentlichung bzw. Inbetriebnahme von jenen findet eine Betrachtung der existierenden statt. Daher ist denkbar, dass die offerierte immaterielle Dienstleistung (nachfolgend als „Dienstleistung“ bezeichnet) anderer Microservices erneut implementiert werden würde. Dies beeinträchtigt wiederum den Grad der Wiederverwendbarkeit, fördert jedoch andererseits die lose Kopplung und die Autonomie eines Microservice. Beim *Bottom-up*-Ansatz hingegen werden ausgehend von bestehenden Microservices die neuen entworfen, welche die veräußerte Dienstleistung anderer Microservices wiederverwenden. Dieser Ansatz birgt das Risiko, dass Microservices im Hinblick auf bereitgestellte Dienstleistungen zu groß und komplex werden, was letztlich die Wartbarkeit verschlechtert. Die Kombination beider Ansätze resultiert in dem sogenannten *Middle-out*-Ansatz.

Die vorliegende Arbeit verfolgt den letztgenannten Ansatz, da bestehende Microservices berücksichtigt werden. So reiht sie sich in die Arbeit von Gebhart ein. Allerdings wird hier angenommen, dass bereits ein sogenannter Geschäftsanalyst diese Betrachtung durchgeführt hat. Falls ein existierender Microservice bereits Teile der geschäftsbezogenen Anforderungen erbringt, wird dieser als eigenständiger sogenannter Bounded Context im Domänenmodell betrachtet. Der Bounded Context, so die Annahme in dieser Arbeit, legt die Zuständigkeit eines Microservice in einer Geschäftsdomäne

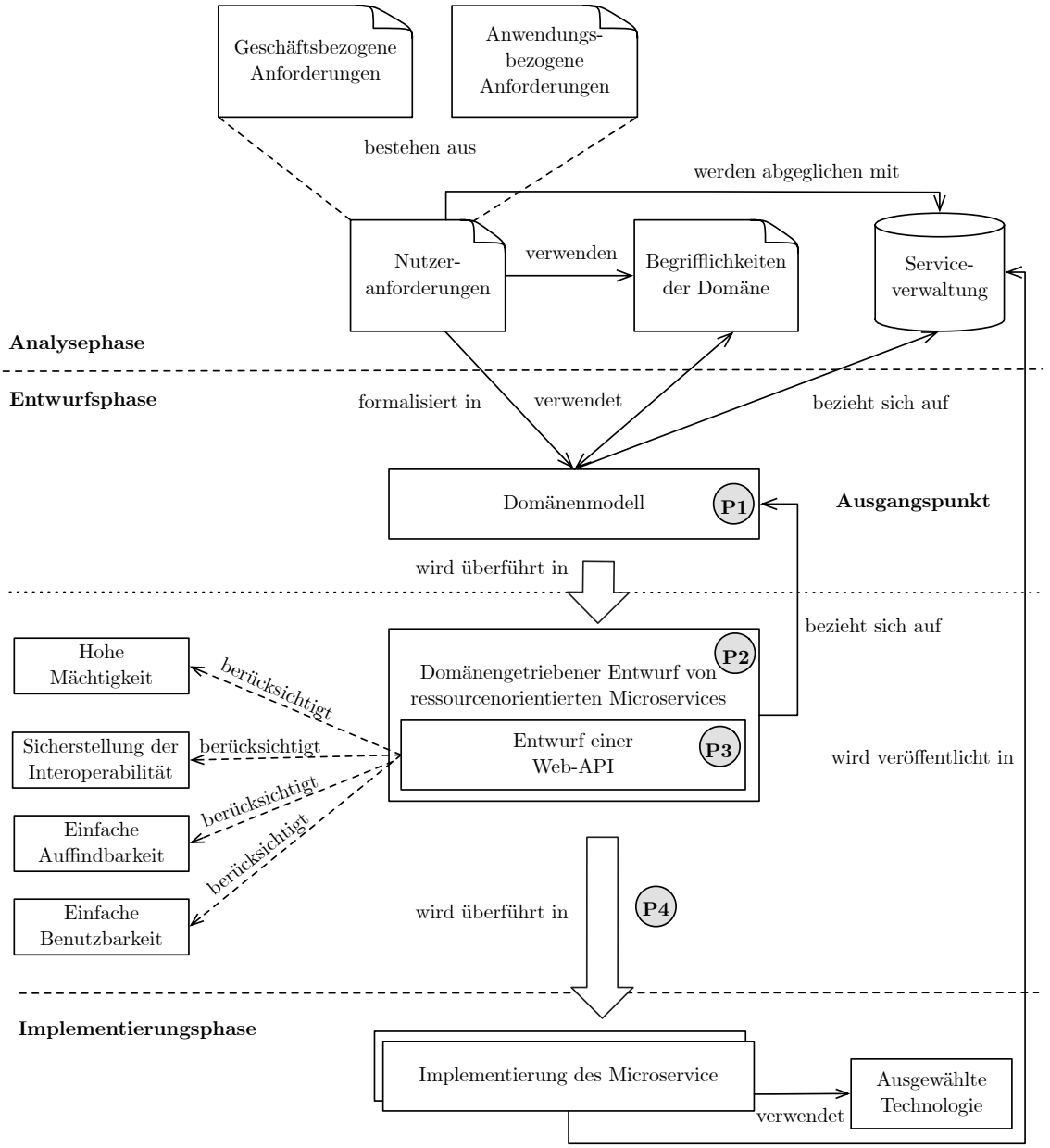


Abbildung 1.1: Betrachtetes Szenario

fest. Das Domänenmodell enthält somit als Ausgangspunkt der Arbeit alle Anforderungen für den abzubildenden Geschäftsausschnitt auf Basis der Nutzeranforderungen.

Beim Entwurf von ressourcenorientierten Microservices muss der Software-Architekt das Domänenmodell und die darin ermittelten Bounded Contexts in jeweils ein ressourcenorientiertes Modell (kurz: Ressourcenmodell) überführen, da jeder Bounded Context einem Microservice entspricht. Dabei soll der Entwurf der Web-API eine hohe Wiederverwendbarkeit ermöglichen, indem bspw. sichergestellt

ist, dass die Web-API keine Vorgaben in Bezug auf die verwendeten Technologien macht. Denn diese sind durch das jeweilige Entwicklungsteam des Microservice selbst zu bestimmen. Gleichzeitig ist es im Hinblick auf die zukünftige Erweiterbarkeit eines Microservice erforderlich, die Web-API gegenüber Änderungen interoperabel zu entwerfen, damit abhängige Microservices weiterhin mit der veräußerten Web-API kompatibel sind. Zudem muss auch sichergestellt sein, dass ein Entwicklungsteam den Entwurf des Microservice zur Überführung auf die Implementierungsebene nutzen kann. Dabei sollte erst zu diesem Zeitpunkt eine Verknüpfung mit einer entsprechenden Technologie erfolgen, sodass die letztliche Technologieentscheidung dem Entwicklungsteam unterliegt.

Schließlich muss es bei einem *Middle-out*-Ansatz möglich sein, die immaterielle Dienstleistung eines Microservice über ein wohldefiniertes Format zu beschreiben und zu veräußern, sodass der Geschäftsanalyst bei seiner Untersuchung bestehende Microservices berücksichtigen kann. Die Veräußerung der immateriellen Dienstleistung eines Microservice in eine sogenannte Serviceverwaltung (engl. service registry) ist nicht Bestandteil dieser Arbeit und wird daher nur am Rande betrachtet. Doch liefert diese Arbeit die notwendigen Vorarbeiten für die Auffindbarkeit von existierenden Microservices im Kontext einer Microservice-Landschaft (engl. microservice landscape).

1.4 Problemstellungen

Die Problemstellungen der Arbeit, die sich aus dem beschriebenen Szenario ergeben, werden nun skizziert. Sie sind gleichzeitig die Motivation für die Überlegungen und Ziele dieser Arbeit. Die Problemstellungen sind in Abbildung 1.1 als P1 bis P4 gekennzeichnet.

P1: Anforderungen an das Domänenmodell

Um, wie in Abbildung 1.1 dargestellt, ausgehend von einem Domänenmodell und den darin enthaltenen Bounded Contexts einen Entwurf von ressourcenorientierten Microservices zu ermöglichen, sind zuerst die Vorgaben und Anforderungen an die Modellierung eines Bounded Context zu definieren. Das ist die Grundlage für eine systematische und nachvollziehbare Überführung mithilfe einer zu entwickelnden Überführungsvorschrift, womit gleichermaßen der Bezug zum domänengetriebenen Entwurfsansatz nach Evans [Ev03] hergestellt werden kann. Deswegen sind die strukturellen und die verhaltensspezifischen Aspekte eines Bounded Context adäquat zu formalisieren, um darauf gestützt ressourcenorientierte Microservices zu entwerfen. Ohne eine derartige Formalisierung müssten beliebig viele Randbedingungen betrachtet werden, und ein systematisches wie auch nachvollziehbares Vorgehen wäre ausgeschlossen. Obgleich ein Bounded Context möglichst lose gekoppelt von anderen Bounded Contexts agieren sollte, ist eine Verknüpfung nicht gänzlich auszuschließen. Dies ist bspw. dann gegeben, wenn sich verschiedene Bounded Contexts bestimmte Domänenobjekte teilen. Deshalb muss auch dieser Fall geeignet modelliert werden.

P2: Überführung des Domänenmodells in einen ressourcenorientierten Entwurf

Die erste Problemstellung hat die Anforderungen an das Domänenmodell bzw. die enthaltenen Bounded Contexts formuliert. Nun folgt deren Überführung in einen ressourcenorientierten Entwurf, was letztlich die Web-API darstellt. Jeder Bounded Context repräsentiert einen Geschäftsausschnitt durch eine Ansammlung sogenannter Domänenobjekte und -services (fortan als „Elemente des Bounded Context“ bezeichnet) im Sinn des domänengetriebenen Entwurfsansatzes nach Evans [Ev03]. Unklar ist hier, wie diese Konzepte sich systematisch und nachvollziehbar mittels Ressourcen im Zuge der Anwendung eines ressourcenorientierten Architekturstils abbilden lassen. Hierzu ist zunächst zu klären, welche Arten von Ressourcen zu unterscheiden und wie diese auf semantischer Ebene mit den Elementen eines Bounded Context harmonisierbar sind. Anschließend gilt es, Transformationsregeln abzuleiten, mit deren Hilfe die einzelnen Elemente systematisch auf Ressourcen abgebildet werden können. Doch müssen nicht alle Elemente eines Bounded Context zwangsläufig über eine Web-API veräußert werden. Deshalb ist immer eine Verknüpfung zu den Nutzeranforderungen erforderlich. Die überführten Ressourcen müssen schließlich wiederum in einem Modellierungsartefakt manifestiert werden, das für die weiteren Entwurfsschritte genutzt werden kann.

P3: Entscheidungsunterstützung beim Entwurf

Das Modellierungsartefakt stellt eine Ansammlung miteinander verknüpfter Ressourcen dar. Allerdings lässt sich daraus nicht ohne weiteres eine Web-API mit hoher Wiederverwendbarkeit ableiten. So ist bspw. unklar, wie die Ressourcen letztlich adressiert werden oder wie mit diesen zu interagieren ist. Bevor sich das Modellierungsartefakt derart verfeinern lässt, sind entscheidungsunterstützende Artefakte zu identifizieren und ihr Einfluss auf die Qualitätsteilmerkmale der Wiederverwendbarkeit zu untersuchen. So bieten bspw. Muster oder bewährte Methoden etwaige Lösungsansätze für wiederkehrende Fragen oder Probleme beim Entwurf. Es kann jedoch nicht vorausgesetzt werden, dass jeder Software-Architekt trotz der Wichtigkeit einer qualitativ hochwertigen Web-API schon jetzt über entsprechendes Wissen und Erfahrungen in diesem Bereich verfügt. Das erleben Unternehmen [MM+16, FS15], die deswegen unternehmensintern Expertengruppen aufstellen. Nach Identifikation und Untersuchung derartiger entscheidungsunterstützender Artefakte müssen sich diese systematisch und nachvollziehbar auf das zugrundeliegende Modellierungsartefakt anwenden lassen. In diesem Sinne sind wiederum geeignete Transformationsregeln auf Basis der identifizierten entscheidungsunterstützenden Artefakte zu entwerfen. Das Ergebnis der angewandten Transformationsregeln ist wiederum auf ein geeignetes Modellierungsartefakt zu übertragen und dabei zu prüfen, ob ein einziges ausreicht oder ob weitere Modellierungsartefakte für den letztlichen Entwurf einer Web-API benötigt werden. Weiter kann nicht davon ausgegangen werden, dass stets auf „grüner Wiese“ ein Entwurf durchgeführt wird. So müssen auch etwaige Änderungen an existierenden Modellierungsartefakten in adäquater Weise begegnet werden, sodass existierende Service-Konsumenten mit der resultierenden Web-API möglichst weiterhin interoperabel sind. Demnach muss auch ein Vorgehen existieren, um

Service-Konsumenten über etwaige Änderungen zu unterrichten, die eine Anpassung auf deren Seiten zur Sicherstellung der Interoperabilität erfordern. Der Entwurf sollte zudem technologie-agnostisch sein, damit das jeweilige Entwicklungsteam die Technologie selbstständig wählen kann.

P4: Überführung des Entwurfs auf die Implementierungsebene

Die Entwurfsartefakte sollen schließlich systematisch auf die Implementierungsebene überführt werden. Die grundsätzliche Architektur eines Microservice ist zwar durch die hexagonale Architektur nach Vernon [Ve13] im Sinne des domänengetriebenen Entwurfsansatzes vorgegeben. Ungeklärt ist jedoch die Überführung einzelner Elemente des Entwurfs auf die letztliche Architektur. Vorher muss allerdings das Entwurfsartefakt der Web-API mit einem Anwendungsschichtprotokoll verknüpft werden. Erst dann kann eine wohldefinierte Web-API-Spezifikation für eine reibungslose Kommunikation zwischen Microservice und letztlichen Service-Konsumenten (nachfolgend als „Interaktionsteilnehmer“ bezeichnet) abgeleitet werden. Diese Web-API-Spezifikation, die auf den vorangehenden Entwurfsentscheidungen im Rahmen von *P3* basiert, ist der Vertrag, an dem sich alle Interaktionsteilnehmer orientieren müssen. Deshalb lässt sich diese Spezifikation zur Parallelisierung von Entwicklungsvorhaben nutzen. Nichtsdestominder lässt sich trotz systematischer Prozeduren nicht gewährleisten, dass die Entwurfsartefakte für die Web-API fehlerfrei sind. Deshalb ist zu untersuchen, wie sich auch nachträgliche Änderungen an der Web-API vor der eigentlichen Umsetzung reduzieren lassen. Sollten Änderungen nicht erforderlich sein, kann das Artefakt implementiert werden. Hierzu sind wie für *P3* angemessene Regeln zu entwerfen, mit deren Hilfe die einzelnen Elemente der Entwurfsartefakte auf die Implementierungsebene abgebildet werden können. Diese Regeln müssen dabei von einer konkreten Technologie und Ausführungsplattform abstrahieren, sodass die Konzepte sich mit geringem Aufwand auf unterschiedliche Technologien anwenden lassen.

1.5 Zielsetzungen und Beiträge dieser Arbeit

Nun werden die Ziele dieser Arbeit auf Basis der bisher identifizierten Probleme erläutert. Es ist das Ziel, einen domänengetriebenen Entwurf für ressourcenorientierte Microservices mit hoher Wiederverwendbarkeit sowie die Verknüpfung des Entwurfs mit der letztlichen Zielarchitektur und anschließender Überführung auf die Implementierungsebene zu entwickeln. Die Web-API stellt dabei ein Schwerpunkt dar, da eine reibungslose Kommunikation nur möglich ist, wenn sich alle Interaktionsteilnehmer daran orientieren.

B1: Modellierungsartefakte für den domänengetriebenen Entwurf

Der erste Beitrag dieser Arbeit widmet sich der Modellierung von Bounded Contexts in einem Domänenmodell, um auf dieser Basis den Entwurf eines ressourcenorientierten Microservice oder mehrerer zu beginnen. Dieser Beitrag fokussiert vor allem Problemstellung *P1* und liefert so Vorgaben

für die Modellierungsartefakte. Hierzu werden zunächst Anforderungen in Bezug auf Syntax und Semantik unter Berücksichtigung der strukturellen und verhaltensspezifischen Aspekte des gewählten Geschäftsausschnitts betrachtet. Das Verfahren verwendet Ansätze der modellgetriebenen Softwareentwicklung unter Berücksichtigung des domänengetriebenen Ansatzes nach Evans [Ev03]. So wird sichergestellt, dass die modellierten Bounded Contexts in einem Domänenmodell die notwendigen Bedingungen für eine nachvollziehbare und systematische Überführung erfüllen und ein Bezug zum domänengetriebenen Entwurfsansatz besteht. Durch die Festlegung von implizitem Verhalten im Zuge des Lebenszyklus der einzelnen Domänenobjekte innerhalb eines Bounded Context wird schließlich der Aufwand bei der Modellierung reduziert.

B2: Qualitätsgestützter Entwurfsprozess von ressourcenorientierten Microservices

Ein systematisches und nachvollziehbares Vorgehen unterstützt den Software-Architekten bei der Überführung aus dem zugrundeliegenden Modellierungsartefakt (vgl.*B1*). Durch dieses Vorgehen lässt sich die grundlegende Fragestellung bearbeiten, wie sich die Modellelemente eines ressourcenorientierten Entwurfs aus Modellelementen eines Bounded Context im Hinblick auf hohe Wiederverwendbarkeit ableiten und verfeinern lassen. Dieser Beitrag fokussiert daher die Problemstellungen *P2* und *P3*. Das entwickelte Vorgehen ergibt einen Entwurfsprozess, der bestehende Vorgehensmodelle berücksichtigt und sich so für bereits etablierte Entwicklungsprozesse verwenden lässt. Der Entwurfsprozess besteht aus mehreren Schritten und erweitert bestehende Ansätze um zusätzliche Randbedingungen sowie Konzepte der modellgetriebenen Softwareentwicklung und insbesondere der Modell-zu-Modell-Transformation (M2M)-Transformation. Zudem wird jede Entscheidung beim Entwurf mit Qualitätsteilmerkmalen der Wiederverwendbarkeit verknüpft und so deren Einfluss auf den letztlichen Entwurf eines ressourcenorientierten Microservice dokumentiert. So fördert bspw. eine konsistente Benennung die Auffindbarkeit der immateriellen Dienstleistung eines ressourcenorientierten Microservice [Ge11], die sich wiederum aus den Begrifflichkeiten des Bounded Context ergibt. Hohe Auffindbarkeit korreliert wiederum positiv mit der Wiederverwendbarkeit.

B3: Überführung des Entwurfs auf die Implementierungsebene

Nachdem die vorherigen Beiträge die Problemstellungen *P1*, *P2* und *P3* adressiert haben, soll dieser Beitrag schließlich *P4* behandeln. Der vorangehende Entwurf bringt zwei grundlegende Entwurfsartefakte mit sich: den abzubildenden Geschäftsausschnitt sowie die letztliche Web-API zur Interaktion mit dem Geschäftsausschnitt. Hier soll nun gezeigt werden, wie sich diese Artefakte systematisch auf die Implementierungsebene bzw. die hexagonale Architektur nach Vernon [Ve13] überführen lassen. Hierzu wird erst die technologie-agnostische Web-API mit einem Anwendungsschichtprotokoll verknüpft und eine wohldefinierte Spezifikation abgeleitet, die von Maschinen interpretiert und im Zuge der Auffindbarkeit in Bezug auf eine Serviceverwaltung dienlich ist. Um das Risiko nachträglicher Änderungen an der Web-API vor der Implementierung des ressourcenorientierten Microservice zu

reduzieren, wird zudem ein manueller Review-Prozess eingeführt, der sich auf etablierte Ansätze aus der Industrie stützt. Der Beitrag wird durch die Abbildung der Entwurfsartefakte auf der Implementierungsebene abgeschlossen, was die Verortung des Entwurfs auf Implementierungsebene ermöglicht, damit Auswirkungen von Änderungen bereits hier erkennbar sind. Gleichzeitig ermöglicht die Verortung zielgerichtete Änderungen der Implementierung, was auch die Wartbarkeit verbessert.

Validierung der Beiträge

Zur Validierung der Beiträge dieser Arbeit und der damit zu erzielenden Vorzüge werden verschiedene Validierungsansätze angewandt und die Ergebnisse kritisch diskutiert. Neben der Demonstration der Beiträge an einem durchgehenden Beispiel im universitären Kontext werden die folgenden drei Studien im industriellen Kontext durchgeführt, was die externe Validität und damit die Tragfähigkeit dieser Arbeit belegt.

1. **Entwurf eines Health-Microservice:** Die Firma *milon* mit Hauptsitz in Emersacker/Deutschland ist ein führender Hersteller für elektronisch gesteuerte Zirkelsysteme mit Schwerpunkt auf der körperlichen Fitness. Für die Hauptentwicklung der plattformübergreifenden Softwarelösung *milon CARE* ist die *milon care GmbH* in München verantwortlich. *milon CARE* ist eine Web-Plattform für die komplette Steuerung der unterstützten Fitnessgeräte, die Dokumentation der Trainingsergebnisse, Analyse von Trainierenden sowie die Erstellung individueller Trainingspläne. Als Erweiterung von *milon CARE* soll ein Health-Service nach den erzielten Beiträgen dieser Arbeit als Microservice entworfen und entwickelt werden. Dieser soll bestehende Plattformen von Drittherstellern vereinheitlichen und eine herstellerneutrale Sicht ermöglichen.

2. **Entwurf und Implementierung einer Microservice-Architektur zur Steuerung und Verwaltung von Trainingsgeräten:** Die Five Konzept GmbH ist ein führender Anbieter für Beweglichkeitstrainings in Europa und bietet als einziger im Premiumsegment ein digitales Training mit visueller Unterstützung an. Für dessen Steuerung soll zukünftig ein System basierend auf Microservices nach den Beiträgen dieser Arbeit entworfen und entwickelt werden, damit mehr Flexibilität im Hinblick auf künftige Anforderungen existiert. Bislang wurde hierfür ein System eines Drittherstellers eingesetzt.

3. **Entwurf und Implementierung einer Anwendung zur Erfassung von Störungen:** Störungen im Prozessablauf von Produktionswerken können Unternehmen erhebliche Kosten verursachen. Deshalb soll für einen Automobilkonzern eine Microservice-Architektur entwickelt werden, die sich den Erfordernissen der einzelnen Produktionswerke anpassen lässt und Störungen erfasst, sodass bei Problemen eine weitere Ansteuerung zur Beseitigung erfolgen kann sowie präventive Maßnahmen schneller möglich werden.

Die Beiträge dieser Arbeit verfolgen das Ziel, etwaige Fehler schon beim Entwurf eines ressourcenorientierten Microservice zu minimieren, da nachträgliche Änderungen am Entwurf und letztlich der Implementierung in der Regel mit hohen Kosten verbunden sind. Gleichzeitig soll das für Entwurfsentscheidungen erforderliche Wissen in Bezug auf die Web-API mithilfe von aufbereiteten Informationen reduziert und dessen zielgerichtetes Anwenden durch ein systematisches und nachvollziehbares Vorgehen erleichtert werden. Dadurch sollen letztlich Web-APIs entstehen, die u.a. stabiler im Hinblick auf etwaige Änderungen sind sowie mächtiger im Sinne der Anzahl möglicher abzubildender Anwendungsfälle, was letztlich in einer höheren Wiederverwendbarkeit resultiert. Die durch die Beiträge entstandenen Modellierungsartefakte sollen schließlich als Vorgabe für das entsprechende Entwicklungsteam dienen, um Implementierungsfehler bei der Umsetzung zu reduzieren. Die dahingehende Nachvollziehbarkeit soll auch in geringeren Wartungskosten und in einer Beschleunigung des Return On Investment (ROI) resultieren.

1.6 Prämissen der Arbeit

Um die Problemdomäne dieser Arbeit einzugrenzen, werden nun die Prämissen vorgestellt, auf die sich die Beiträge dieser Arbeit stützen. Gleichzeitig dienen die Prämissen zur Auswahl relevanter Arbeiten im Forschungsgebiet eines domänengetriebenen Entwurfs von ressourcenorientierten Microservices.

Prämisse 1: Betrachtung von ressourcenorientierten Microservices

In der vorliegenden Arbeit werden ausschließlich ressourcenorientierte Microservices mit Ressourcen als deren fundamentale Bausteine betrachtet, aus denen sich die veräußernde Web-API eines Microservice zusammensetzt [WP+10, S. 4]. Im Gegensatz zu REST wird hier nicht die explizite Verwendung von Hypermedia gefordert, da dies ein weiteres Forschungsgebiet im Bereich des semantischen Web wäre [LG11, LG12, La14]. Zudem wird der Hypermedia-Aspekt von vielen als RESTful bezeichneten Services nicht betrachtet [AS+11, RS+12, BM14], sondern ausschließlich die Ressourcenorientierung auf der zweiten Ebene des Richardson-Maturity-Modells (RMM).

Prämisse 2: Vorhandensein eines Domänenmodells

Der Entwurf von einem oder mehreren Microservices setzt voraus, dass vorher die Nutzeranforderungen systematisch analysiert und damit der Rahmen für die Betrachtung der Geschäftsdomäne gelegt wurde. Aus der zu betrachtenden Geschäftsdomäne ergeben sich deren weitere Aufteilungen gemäß der strategischen Modellierung nach Evans [Ev03]. Die sich dadurch ergebenden Bounded Contexts stellen dabei ideale Kandidaten für Microservices dar, da diese u. a. eine hohe Kohäsion und in der Regel eine lose Kopplung aufweisen. Die Identifizierung von Bounded Contexts ist nicht Bestandteil dieser Arbeit. Stattdessen wird davon ausgegangen, dass die Aufteilung in Bounded Contexts bereits

erfolgt ist und auf Grundlage dessen der Entwurf der ressourcenorientierten Microservices eingeleitet werden kann. Die einzelnen Bounded Contexts sind wiederum in einem Domänenmodell verortet, welches als zentrales Modellierungsartefakt dem Entwurf zugrunde liegt.

Prämisse 3: Fokus auf die abzubildende Geschäftsdomäne

Nutzeranforderungen enthalten u. a. anwendungs- und geschäftsbezogene Anforderungen. Letztere sind der Rahmen für die betrachtete Geschäftsdomäne und stehen im Fokus dieser Arbeit. Dagegen werden anwendungsbezogene Anforderungen (können auch als anwendungsspezifische Anforderungen bezeichnet werden) beim Entwurf von einem oder mehreren ressourcenorientierten Microservices nur beiläufig betrachtet. Denn derartige Anforderungen sind schon eine Ausrichtung auf bestimmte Anwendungsfälle, die so die Wiederverwendbarkeit der abzubildenden Bounded Contexts und letztlich der Microservices gefährden. Der Vollständigkeit wegen werden die anwendungsbezogenen Anforderungen durch deren Verortung in einer Microservice-Architektur am Rande betrachtet, woran sich weitere Arbeiten auf diesem Gebiet orientieren und anlehnen können.

Prämisse 4: Eingeschränkte Betrachtung der strukturellen und verhaltensspezifischen Aspekte

Der Entwurf eines ressourcenorientierten Microservice muss die strukturellen und die verhaltensspezifischen Aspekte der zu betrachtenden Geschäftsdomäne innerhalb der ermittelten Bounded Contexts abbilden. Ein Bounded Context repräsentiert in dieser Arbeit gemäß der Prämisse 2 einen ressourcenorientierten Microservice. Die strukturellen Aspekte fokussieren dabei die grundsätzliche Strukturierung mithilfe von Domänenobjekten, während die verhaltensspezifischen Aspekte das Verhalten der einzelnen Domänenobjekte bzw. der Domänenservices über die Zeit abbilden [Ev03, Ve13]. Diese Arbeit betrachtet nur die strukturellen und verhaltensspezifischen Aspekte, die sich aus den modellierten Bounded Contexts und damit dem zugrundeliegenden Domänenmodell ergeben bzw. ableiten lassen. Die Vollständigkeit der Modellierung dieser Bounded Contexts ist keine notwendige Voraussetzung für die erarbeiteten Beiträge dieser Arbeit.

Prämisse 5: Unterstützung bei Entwurfsentscheidungen im Zuge der Wiederverwendbarkeit

Der domänengetriebene Entwurf von ressourcenorientierten Microservices soll Software-Architekten bei Entwurfsentscheidungen unterstützen. Diese fokussieren dabei die zu veräußernde Web-API, die bei Microservice-Architekturen sehr wichtig ist. Eine Web-API korreliert positiv mit der Wiederverwendbarkeit eines Microservice, da sie die immaterielle Dienstleistung eines Bounded Context bereitstellt und letztlich für die Integration in Service-Konsumenten verwendet wird. In dieser Arbeit

werden die Benutzbarkeit, Mächtigkeit, Auffindbarkeit und Interoperabilität einer Web-API als Qualitätsteilmerkmale der Wiederverwendbarkeit betrachtet. Für diese Betrachtung werden ausschließlich Informationen herangezogen, die zum Zeitpunkt des Entwurfs vorliegen bzw. aus zugrundeliegenden Artefakten ableitbar sind. Laufzeitinformationen und die sich dadurch ergebenden Gewinne an neuen Informationen schenkt diese Arbeit keine Beachtung.

1.7 Struktur und Aufbau der Arbeit

Die vorliegende Arbeit ist in acht Kapitel unterteilt, die Abbildung 1.2 veranschaulicht: In **Kapitel 1** wird zunächst in das Thema eingeführt und das Vorhaben in die Forschungslandschaft integriert. Anschließend veranschaulicht die Untersuchung eines Szenarios die zu bearbeitenden Problemstellungen, um die Zielsetzungen dieser Arbeit zu formulieren. **Kapitel 2** legt das Fundament für die nachfolgenden Kapitel, indem grundlegende Begriffe und Konzepte im Kontext des domänengetriebenen Entwurfs von ressourcenorientierten Microservices erläutert werden. **Kapitel 3** skizziert den Forschungsstand. Es werden relevante Arbeiten hinsichtlich eines aufgestellten Anforderungskatalogs analysiert und ausgewertet. Der sich daraus ergebende Handlungsbedarf bildet die Grundlage für die Beiträge in dieser Arbeit.

Kapitel 4 bildet den ersten Beitrag und fokussiert den domänengetriebenen Entwurf eines oder mehrerer ressourcenorientierter Microservices. Nach einem Überblick über diesen Entwurfsprozess werden die einzelnen Prozessaktivitäten erläutert. Anschließend wird sich der Modellierung eines gewählten und abzubildenden Geschäftsausschnitts gewidmet und entsprechende Modellierungsartefakte auf Basis von Unified Modeling Language (UML) entwickelt. Darauf stützt sich schließlich ein Szenario, um die Modellierung und weitere Prozessaktivitäten zu illustrieren und zu demonstrieren. Mit **Kapitel 5** wird der nächste Beitrag geliefert, der sich dem ressourcenorientierten Entwurf von Microservices widmet und die im Vorfeld eingeführten Prozessaktivitäten im Detail erläutert sowie an dem zuvor erwähnten Szenario illustriert. Die einzelnen Prozessaktivitäten fokussieren jeweils einen spezifischen Bereich der Ressourcenorientierung und werden durch Muster sowie festzulegende Konventionen gestützt. Daneben ist die Beschreibung der Prozessaktivitäten auf jeweils ein zugrundeliegendes Modellierungsartefakt ausgerichtet, um so die Nachvollziehbarkeit über Prozessaktivitätsgrenzen hinweg zu gewährleisten. Nachdem bisher der Bezug zu einer Technologie vermieden wurde, widmet sich **Kapitel 6** der Überführung auf die Implementierungsebene – der nächste Beitrag dieser Arbeit. Hierzu werden die Modellierungsartefakte des Entwurfs auf ein Anwendungsschichtprotokoll bezogen und schließlich eine Web-API-Spezifikation abgeleitet, die ein Vertrag zwischen Interaktionsteilnehmern darstellt und eine reibungslose Interaktion ermöglichen soll. Darauf aufbauend wird gezeigt, wie sich die Modellierungsartefakte des Entwurfs auf der Implementierungsebene unter Berücksichtigung der hexagonalen Architektur widerspiegeln und so die Implementierung anleiten. Die Betrachtung der Makroarchitektur verortet schließlich die resultierenden Microservices in einer Microservice-Architektur.

Darauf aufbauend wird mit **Kapitel 7** die Validierung der einzelnen Beiträge und damit gleichermaßen die Tragfähigkeit erbracht. Es werden mehrere Validierungsarten vorgestellt und Ansätze für die Validierung eines Entwurfsprozesses abgeleitet. Diese Validierungsansätze im universitären und im industriellen Kontext werden detailliert beschrieben und ihre Ergebnisse in Bezug auf die Bedrohung der Validität diskutiert. **Kapitel 8** beschließt die Arbeit mit einer Zusammenfassung der Beiträge und gibt einen Ausblick auf mögliche weiterführende Arbeiten in diesem Feld.

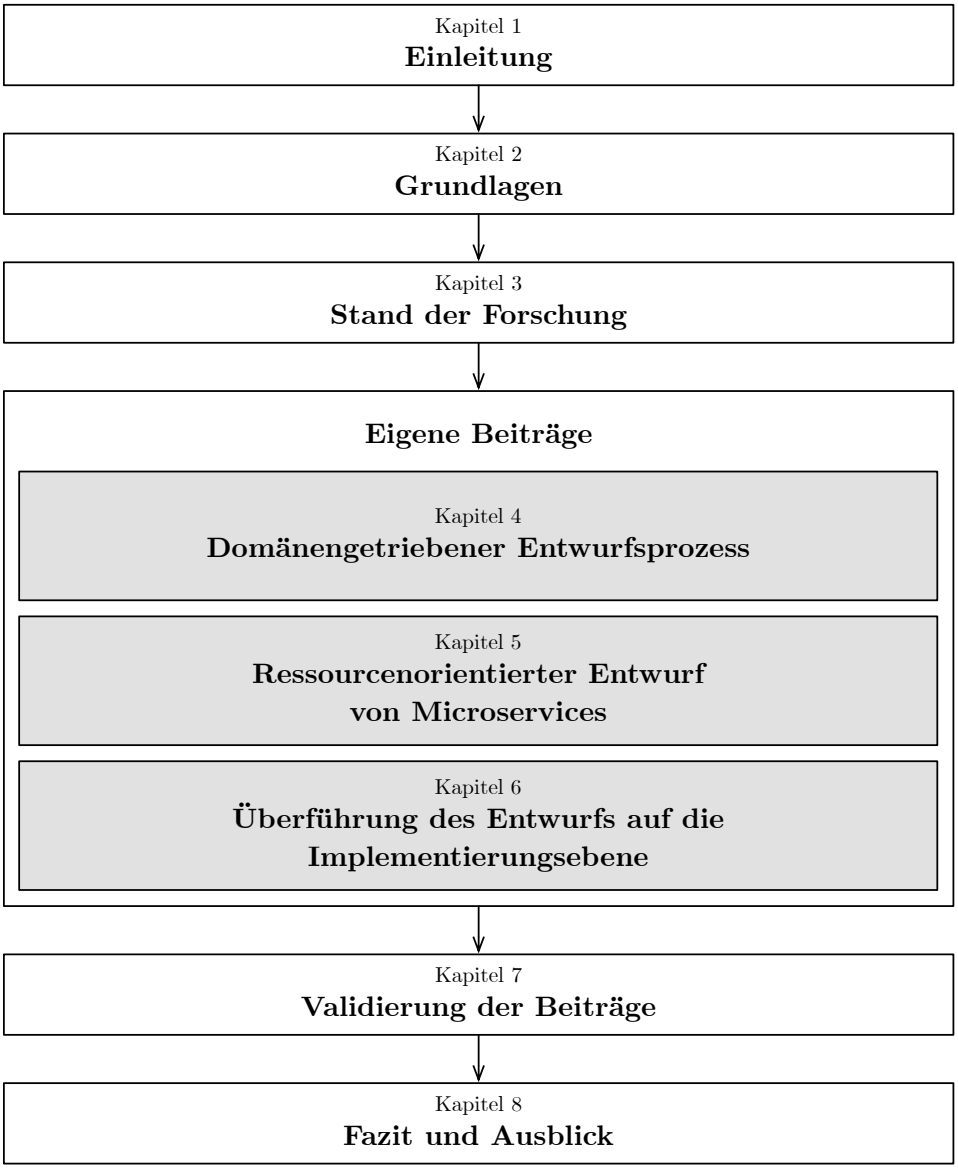


Abbildung 1.2: Aufbau und Struktur der vorliegenden Arbeit

2 Grundlagen

Dieses Kapitel liefert die Grundlagen für das Verständnis dieser Arbeit. Dazu werden zunächst die tragenden Begriffe „Microservice“ und „Microservice-Architektur“ definiert und der dieser Arbeit zugrundeliegende domänengetriebene Entwurfsansatz erläutert. Darauf aufbauend werden die zu betrachtenden architekturellen Sichten vorgestellt. Anschließend wird die Web-API als Schnittstelle des Microservice beleuchtet. Zudem werden Ansätze zur Spezifikation von Web-APIs präsentiert, die u. a. als Grundlage für die Generierung von Programmcode genutzt werden können. Der Architekturstil REST und die damit einhergehenden Randbedingungen sowie mögliche Ausprägungen des Architekturstils schließen das Kapitel ab.

2.1 Microservice und Microservice-Architekturen

Grundlage für die in dieser Arbeit diskutierten Problemstellungen (vgl. Abschnitt 1.4) stellen auf Microservices gestützte Architekturen (kurz: Microservice-Architekturen) dar, um Software-Systeme in Unternehmen in geeigneter Form zu strukturieren. Deshalb werden nun Microservices und die auf ihnen errichtete Microservice-Architektur definiert.

2.1.1 Begrifflichkeit eines Microservice

Microservices sind zum Zeitpunkt des Schreibens dieser Arbeit nicht einheitlich definiert [NM+16]. Doch lassen sich Gemeinsamkeiten aus den vorhandenen Definitionen ableiten, die eine allgemeinere Definition gestatten.

Bevor allerdings der Microservice erörtert werden kann, ist zunächst der Begriff des Service zu erläutern. Das ist laut Gebhart [Ge11] und dem Standard der International Organization for Standardization (ISO) [DIN-9000:2015-11] das Ergebnis mindestens einer Tätigkeit zwischen Unternehmen und Kunden über eine Schnittstelle mit einem in der Regel immateriellen Ergebnis. Ein Service wird demnach stets über eine Schnittstelle bereitgestellt, und sein Konsument (Kunde) kann lediglich über diese mit dem Unternehmen interagieren (auch als *Supplier/Customer*-Modell bekannt). Letzterer erblickt demnach nur das von außen sichtbare Verhalten, was dem Blackbox-Prinzip als Ausprägung des *Information Hiding* entspricht [Pa72, VA+09]. Die zur Erbringung erforderlichen Prozesse bleiben ihm daher verborgen, weshalb eine Verwendung des Service ohne dieses Wissen möglich sein muss. So ist die Schnittstelle ein wichtiges Element eines Service und damit auch eines Microservice [Ge11].

Um nun die grundlegende Fragestellung zu klären, wie nun ein Microservice definiert wird, folgt nachfolgend eine Vorstellung existierender Definitionen:

> *„A microservice is a cohesive, independent process interacting via messages“*
>
> – Dragoni et al. [DG+17, S. 2]

> *„Microservices [are] small and lightweight services that are purposely built to perform a very cohesive business function, and is an evolution of the traditional service oriented architecture style“*
>
> – Alshuqayran et al. [AA+16, S. 44]

> *„A microservice is an independent deployable component of bounded scope that supports interoperability through messaged-based communication“*
>
> – Nadareishvili [NM+16, S. 6]

> *„Microservices are relatively small, autonomous services that work collaboratively together. Each of the services implements a single business requirement“*
>
> – De Santis et al. [DSF+16, S. 16]

> *„Microservices are small, autonomous services that work together“*
>
> – Newman [Ne15, S. 2]

> *„Microservices [dienen zur Modularisierung eines großes Software-System] und beeinflussen die Organisation [sowie] die Software-Entwicklungsprozesse“*
>
> – Wolff [Wo15, S. 2]

Hieraus geht hervor, dass die Autonomie, die Interoperabilität und der beschränkte Umfang (engl. bounded scope) im Sinne einer kohäsiven Bündelung von Funktionalitäten wesentliche Qualitätsmerkmale eines Microservice darstellen. Eine Erläuterung der einzelnen Qualitätsmerkmale ist Anhang A.3 auf Seite 208 zu entnehmen. Die zuletzt zitierte Definition von Wolff verdeutlicht zudem, dass Microservices nicht nur auf technischer Ebene angewendet werden, sondern auch die Organisation beeinflussen. Auf diese Erkenntnisse gestützt, lässt sich im Kontext einer abzubildenden Geschäftsdomäne ein Microservice so fassen.

> **Definition (Microservice)** *Ein Microservice ist ein autonomer Service, der genau einen festgelegten minimalen Ausschnitt der Geschäftsdomäne mit hoher Kohäsion abbildet und eine nachrichten-basierte Kommunikation über eine wohldefinierte Schnittstelle zur Interaktion bereitstellt.*

Üblicherweise sollte sich der Ausschnitt einer Geschäftsdomäne an den jeweiligen Kommunikationsstrukturen der Organisation anlehnen. Diese Empfehlung geht zurück auf das Gesetz nach Conway [Co68], laut welchem Organisationen nur Systeme entwerfen können, die ihre Kommunikationsstrukturen in geeigneter Form widerspiegeln. Demnach beeinflussen letztere wesentlich die Modularisierung eines Systems und damit auch die genannten Qualitätsmerkmale eines Microservice [Wo15]. Diese Arbeit wird die Begriffe des Service und des Microservice synonym verwenden. Die Größe eines Microservice richtet sich an den abzubildenden Ausschnitt der Geschäftsdomäne, sollte sich allerdings durch ein kleines Entwicklungsteam in adäquater Zeit umsetzen lassen. Eine fixe Anzahl an Teammitgliedern kann an dieser Stelle nicht geliefert werden, lediglich Richtwerte wie bspw. von Amazon: „Two Pizza Team (i.e. the whole team can be fed by two pizzas), meaning no more than a dozen people“ [Fo14][Wi17]. Wird dieser Richtwert überschritten, ist eine Aufteilung des gewählten Geschäftsausschnitts zu erwägen, wodurch getrennte Entwicklungsteams entstehen.

2.1.2 Microservice-Architektur

Eine Microservice-Architektur ist nach Dragoni et al. [DG+17], eine verteilte Anwendung bestehend aus Microservices. Andere wiederum bezeichnen eine Microservice-Architektur als Architekturstil oder als einen neuen Entwicklungsansatz [LF04, LN+15, DSF+16]. Auch hier ist also eine einheitliche Definition erforderlich. Dafür wird zunächst die Definition eines Architekturstils herangezogen. Dabei handelt es sich um eine: „Lösungsstruktur[...], die für ein Element einer Architektur durchgängig und mit weitgehendem Verzicht auf Ausnahmen angewandt“ [RH08, S. 91] wird. Für die genannten Definitionen einer Microservice-Architektur bedeutet das, dass sie durchaus als ein Architekturstil aufgefasst werden kann, wobei sie dann eher das Resultat des angewandten Architekturstils ist. Auf Grundlage dieser Betrachtungsweise ergeben sich folgende Definitionen:

> **Definition (Microservice-orientierter Stil)** *Ein microservice-orientierter Stil ist ein Architekturstil eines verteilten Software-Systems, bei dem die zugrundeliegenden Systembausteine ausschließlich Microservices sind.*

> **Definition (Microservice-Architektur)** *Eine Microservice-Architektur ist ein verteiltes Software-System, das durch Anwendung eines microservice-orientierten Stils entsteht.*

Wird das Konzept einer Microservice-Architektur mit der einer "klassischen" SOA verglichen, zeigen sich neben Parallelen [NM+16] auch wesentliche Unterschiede bspw. im Hinblick auf die Wiederverwendbarkeit oder die Orchestrierung [Ri15a, Wo15]. Weiter ist der microservice-orientierte Stil aus der Praxis durch vertieftes Wissen über Systeme und deren Architektur entstanden. Laut Newman sollte der microservice-orientierte Stil eher als Ausprägung eines serviceorientierten Stils aufgefasst werden: „So you should instead think of microservices as a specific approach for SOA in the same way that XP or Scrum are specific approaches for Agile software development“ [Ne15, S. 9]. Da der

Vergleich beider Architekturen für diese Arbeit nicht weiter erforderlich ist, wird dies hier nicht weiter vertieft. Auch unterbleibt hier ein direkter Vergleich zu monolithischen Architekturen. Es sei lediglich auf Anhang A.2 verwiesen, der einige Nachteile derartiger Architekturen aufführt.

2.2 Domänengetriebener Entwurf

Dieser Abschnitt beleuchtet das Konzept des domänengetriebenen Entwurfs, auf dem diese Arbeit basiert. So soll auch dessen Eignung für den Entwurf von Microservices insbesondere beim sogenannten Service-Schnitt und beim IT/Business-Alignment verdeutlicht werden.

2.2.1 Domäne, Domänenmodell und Bounded Context

Eine Domäne im Kontext einer Organisation repräsentiert das gesamte Wissen, jegliche Aktivitäten im Unternehmen, etwaige Einflussfaktoren sowie die umgebende Umwelt, in der sie operiert [Ev03]. Deshalb wird sie oft auch als Geschäftsdomäne (engl. business domain) bezeichnet [Ve13], woran sich diese Arbeit anlehnt. Gleichzeitig charakterisiert sich eine Domäne immer durch ihre Abgeschlossenheit, die den Rahmen für ihre Gültigkeit legt. Ähnlich wie der organisatorische Aufbau eines Unternehmens kann auch die Domäne in mehrere Teilbereiche aufgeteilt werden, in der nur eine Teilmenge der Domäne betrachtet wird. Bei Entwicklung von derartigen Software-Systemen sollten bei der Aufteilung die Kommunikationsstrukturen des Unternehmens im Sinne des Gesetzes nach Conway (engl. Conway's Law) berücksichtigt werden. Es besagt, dass „organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations“ [Co68, S. 31]. Ergänzend unterscheidet Evans [Ev03] noch zwischen Kern- und unterstützender Domäne, wobei erstere das zentrale Geschäft des Unternehmens verkörpern sollte, während unterstützende Domänen Mehrwert im Sinne einer logischen Ergänzung der Kerndomäne liefern: „The business creates a Supporting Subdomain because it is somewhat specialized“ [Ve13, S. 224]. Ergänzend existieren noch sogenannte Bounded Contexts, die das Wissen eines spezifischen Teilbereichs der Geschäftsdomäne kapseln und in der Regel eine hohe Kohäsion aufweisen [Ev03, SG$^+$17]. Üblicherweise handelt es sich dabei um organisatorische Einheiten oder Fachbereiche, die spezifisch den Erfolg des Unternehmens fördern. Aus der Kommunikation mit anderen organisatorischen Einheiten können schließlich etwaige Anforderungen für die bereitzustellende immaterielle Dienstleistung abgeleitet werden. Begriffe in einem Bounded Context haben nur dort ihre Gültigkeit bzw. eine eindeutige Semantik, wodurch der Abstimmungsaufwand mit anderen organisatorischen Einheiten deutlich reduziert wird. Identische Begriffe können also in unterschiedlichen Bounded Contexts unterschiedliche Semantik haben. Aus diesem Grund führt Evans die „Context Map“ ein, um zwischen verschiedenen Bounded Contexts zu wechseln.

Eine Domäne wird durch ein Domänenmodell abgebildet, das nach Evans alles enthält, was zu ihrem Verständnis beiträgt bzw. dafür benötigt wird. Darin verankert sind die identifizierten Bounded

Contexts, die wiederum untereinander in Beziehung stehen können. Der domänengetriebene Entwurf geht weiter als der traditionelle Software-Entwicklungsansatz, bei dem ein Domänenmodell lediglich durch ein formalisiertes UML-Modell, zumeist ein UML-Klassendiagramm, repräsentiert wird [SG+17, HG+17]. Um diese verschiedenen Konzepten zu unterscheiden, liefert [SG+17, HG+17] eine entsprechende Auftrennung.

2.2.2 Grundsätzliches Vorgehen

Bei einem domänengetriebenen Entwurf nach Evans [Ev03] wird das Domänenmodell auf Grundlage vorhandener Informationen der durch IT abzubildenden Domäne entworfen (vgl. Abbildung 2.1). Doch es steht nicht eine allumfassende Modellierung der Domäne im Fokus des Entwurfsansatzes, sondern immer nur der Ausschnitt, der zur Erfüllung des Bedürfnisses nach informationstechnischer Unterstützung benötigt wird. Als Informationsquellen für das Domänenmodell dienen hier sämtliche Informationen, die zum Verständnis der Domäne beitragen, wie bspw. Anforderungsspezifikationen oder etablierte Literatur im Kontext der Domäne. Mittels des „Knowledge Crunching“ wird aus der Menge der vorhandenen Informationen die zur Modellierung der Domäne notwendige Teilmenge identifiziert. Laut Evans kann das ein langwieriger Prozess sein, der zudem die Zusammenarbeit der Projektbeteiligten erfordert, also in diesem Fall alle Personen, die Interesse an der Modellierung und Abbildung der entsprechenden Domäne durch die IT und auch entsprechendes Wissen haben. Dazu zählen bspw. Geschäftsanalysten, Domänenexperten, Software-Architekten oder Software-Entwickler.

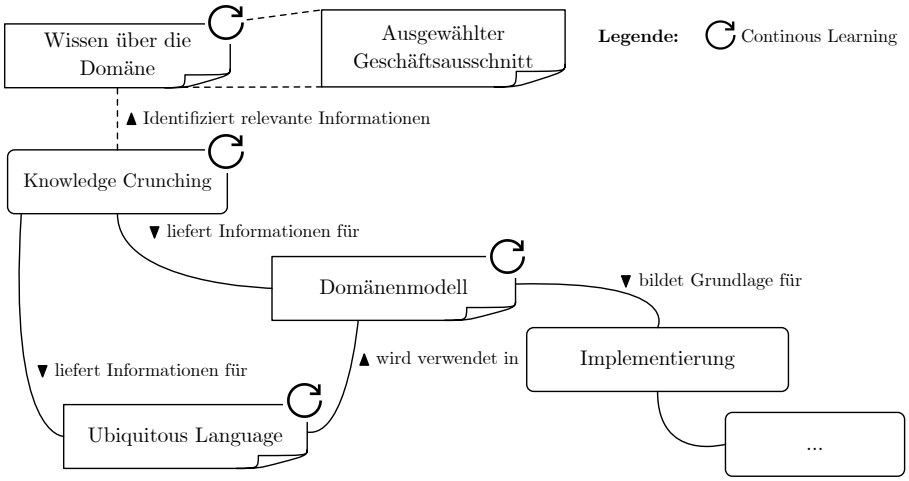


Abbildung 2.1: Grundsätzliches Vorgehen beim Domain Driven Design (DDD)

Eine wichtige Erkenntnis bei der Durchführung des domänengetriebenen Entwurfs war, dass kaum jemand vollständiges Verständnis über eine Domäne hat. Stattdessen entwickelt sich dieses im Verlauf der Zusammenarbeit der Projektbeteiligten. Deshalb hat Evans den Begriff des „Continuous

Learning“ eingeführt. Die Modellierung einer Domäne wird also als iterativer und inkrementeller Prozess aufgefasst, der während der gesamten Entwicklung weiter vorangetrieben werden sollte. Mit der Modellierung der Domäne wird auch eine gemeinsame Sprache für alle Projektbeteiligten, die „Ubiquitous Language“, geschaffen. Dies fördert die Kommunikation zwischen Projektbeteiligten, die eventuell durch unterschiedliche fachliche Hintergründe auch unterschiedliche Auffassungen haben. Zudem hilft ein gemeinsames, konsistent verwendetes Vokabular, Missverständnisse als Folge der natürlichen Sprache zu vermeiden und die Einarbeitung von neuen Projektbeteiligten zu erleichtern. Konsequent angewandte Ubiquitous Language findet sich im Domänenmodell und auch auf Ebene der Implementierung wieder, indem beide Ebenen miteinander verknüpft werden, sodass das Domänenmodell das zentrale Artefakt im Entwicklungsprozess wird. Evans bezeichnet die Verwendung eines einzigen Modells für die Analyse und den darauf gegründeten Entwurf als „Model-driven Design“, was gleichzeitig den Bezug zu Model Driven Software Development (MDSD) herstellt.

2.2.3 Domänenmodell

Ein Domänenmodell ist ein Entwurfsartefakt in der Softwareentwicklung, das fachliches Wissen über die abzubildende Domäne durch Anwendung des domänengetriebenen Entwurfs repräsentiert und Unterteilungen in Form von Bounded Contexts enthalten kann. Abhängig von der Ausprägung des Domänenmodells kann dieses auch schon um technische Details ergänzt worden sein, da es bei einem *Model-driven Design* auch Grundlage für die Implementierung ist. Also muss dieses Artefakt die Software-Architektur adäquat widerspiegeln. Ein Domänenmodell enthält sowohl statische als auch dynamische Informationen über die Domäne. Überträgt man dies auf Ebene der Software-Architektur, so kann dies mit den architekturellen Sichten gleichgesetzt werden (vgl. Abschnitt 2.2.4). Im Hinblick auf die Modellierung dieser Aspekte macht Evans allerdings keinerlei Vorgaben. Es lässt sich grundsätzlich jede Form der Modellierung nutzen, da das Domänenmodell auch in der Analysephase seine Anwendung findet [SG+17]. Deshalb darf der Begriff des Domänenmodells nicht mit dem Modellbegriff aus dem Bereich der MDSD gleichgesetzt werden, wo das Modell auf einer eindeutigen Syntax und Semantik in Form eines Meta-Modells beruht [SV+06].

Evans identifiziert beim *Model-driven Design* folgende drei Bausteine, die zum besseren Verständnis auch durch das Meta-Modell in Abbildung 2.2 illustriert werden: 1) Entity (werden oftmals auch als „Reference Object“ bezeichnet [Ev03, Ve13]), 2) Value Object und 3) Domänenservice (engl. Domain Service) nach Vernon [Ve13] bzw. Service nach Evans [Ev03].

Eine *Entity* und ebenso ein *Value Object* repräsentieren jeweils Objekte der Domäne (kurz: Domänenobjekte) mit zugehörigen Attributen und Operationen, wobei die Attribute die statischen Informationen und durch Operationen die dynamischen Informationen abgebildet werden. Zudem hat eine *Entity* ihre eigene eindeutige Identität. Der Rahmen, in dem die Eindeutigkeit einer *Entity* gewährleistet ist, wird

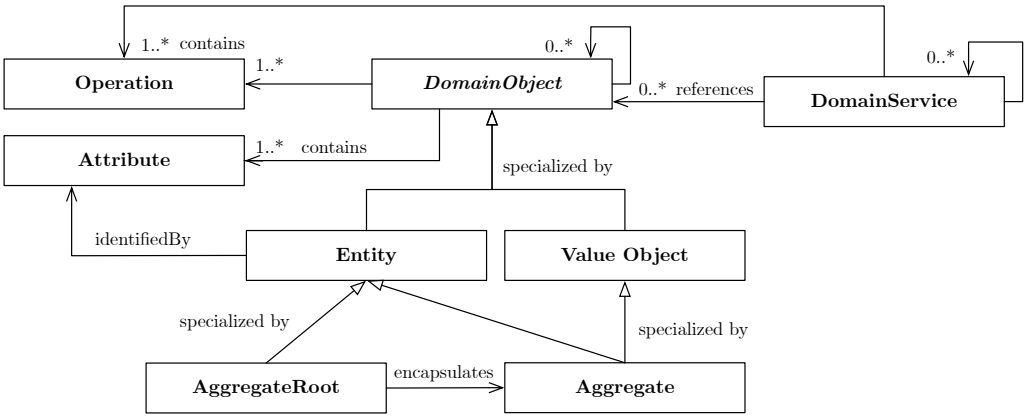


Abbildung 2.2: Vereinfachtes Meta-Modell der Bausteine des Model-driven Design nach [Ev03]

durch den jeweiligen Identifikator bestimmt – etwa ein Universally Unique IDentifier (UUID) gemäß [RFC4122-2005], der „unique across both space and time, with respect to the space of all UUIDs“ [RFC4122-2005, S. 2] ist. Ein *Value Object* ist im Gegensatz zur *Entity* ein unveränderliches (engl. immutable) Domänenobjekt ohne einen Identifikator. Bei einer Änderung wird das Domänenobjekt durch eine neue Version ersetzt, wohingegen bei einer *Entity* nur die entsprechenden Werte verändert werden, um so auch eine Form der Versionsverfolgung zu ermöglichen. Die Überprüfung auf Gleichheit eines *Value Object* erfolgt anhand der Attribut-Wertpaare und somit dessen Struktur, was auch als *strukturelle Gleichheit* bezeichnet wird.

Da sich nicht alle Operationen einem Domänenobjekt zuordnen lassen, wurden sogenannte *Domänenservices* eingeführt. Das sind Operationen, die meist mehrere Domänenobjekte adressieren und so bspw. Geschäftsprozesse abbilden können. Laut Evans hat ein *Domänenservice* folgende drei Eigenschaften: 1) Die Operation lässt sich weder einer *Entity* noch einem *Value Object* zuordnen, 2) sie bezieht sich auf existierende Domänenobjekte, und 3) sie ist zustandslos.

Neben diesen Bestandteilen existieren auch noch sogenannte *Aggregates*, über die sich mehrere Domänenobjekte kapseln und über eine sogenannte *Aggregate Root* zugänglich machen lassen. Als *Aggregate Root* kann bspw. eine *Entity* fungieren, die sich in einer Teil-Ganzes-Beziehung mit einem anderen Domänenobjekt befindet und die Hoheit über die Beziehung besitzt. Ein *Value Object* als *Aggregate Root* würde hingegen nicht funktionieren, da es nicht direkt über einen Identifikator adressiert werden kann.

2.2.4 Architekturelle Sichten

Ein Software-Architekt betrachtet die Architektur des domänengetriebenen Entwurfs aus verschiedenen Blickwinkeln, die in der Literatur als *architekturelle Sichten* (engl. architectural views) bezeichnet werden. Hierbei handelt es sich nach [RB+16a] entsprechend dem ISO-Standard 42010 [ISO-42010]

um ein Arbeitsprodukt, das die Architektur eines Systems aus der Sicht eines spezifischen Aspekts beschreibt. Bei dem Arbeitsprodukt kann es sich bspw. um ein formal beschriebenes Dokument oder auch um eine Beschreibung in natürlicher Sprache handeln. Der Begriff *Aspekt* drückt einen Interessenschwerpunkt sowie die Erwartung einer in den Entwicklungsprozess involvierten Person an das zugrundeliegende System aus [Gl07, RB+16a].

Jede architekturelle Sicht ist im Sinne der genutzten Terminologie in [RB+16a] aus [GB+12, Bu13] zur Unterscheidung der verschiedenen Modellierungsebenen eine Ausprägung eines sogenannten *Sichttyps*. Als ein Sichttyp gilt: „the set of meta-classes whose instances a view can display and compromises a definition of a concrete syntax plus a mapping to the abstract meta-model syntax“ [RB+16a, S. 42]. Demnach selektiert ein Sichttyp entsprechende Bestandteile eines zugrundeliegenden Meta-Modells und stellt diese der Sicht für dessen Instanziierung zur Verfügung. Zum besseren Verständnis setzt die Abbildung 2.3 die eingeführten Termini nochmals in Beziehung zueinander. Nun werden die für diese Arbeit relevanten Aspekte näher beleuchtet.

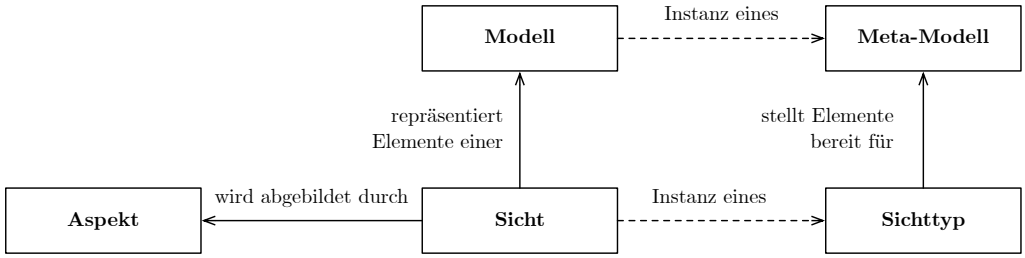


Abbildung 2.3: Beziehung zwischen Aspekt, Sicht und Sichttyp nach [RB+16a]

Strukturelle Aspekte

Die strukturellen Aspekte spezifizieren die Abhängigkeiten sowie die grundlegenden Bausteine eines Systems. Es lassen sich zwei Typen von Sichten unterscheiden – der system-spezifische und der system-unspezifische Sichttyp [RB+16a]. Letzterer beschreibt die im Kontext von Microservices verfügbaren und wiederverwendbaren Microservices als Teil eines Service-Verzeichnisses. Da die Wiederverwendung von Microservices die lose Kopplung und damit auch die Autonomie eines Microservice verschlechtert, ist eine derartige Entwurfsentscheidung sorgfältig zu überlegen [Ne15]. Im Bereich der objektorientierten Programmierung bringen Gamma et al. eine Definition, welche ebenfalls die strukturellen Aspekte beschreiben: „Structural patterns deal with composition of classes or objects“ [GH+94, S. 10]. Wird dies auf die Modellierung einer Domäne übertragen, so werden damit vorrangig die Domänenobjekte und damit die *Entities* und *Value Objects* adressiert. Aber auch *Domänenservices* können trotz ihres Schwerpunkts in der Abbildung von verhaltensspezifischen Aspekten zu den strukturellen Aspekten gezählt werden. Dies begründet sich darin, da ein *Domänenservice* auch eine entsprechende Manifestierung im letztlichen System bekommt.

Verhaltensspezifische Aspekte

Verhaltensspezifische Aspekte fokussieren das erwartete Verhalten eines Systems: „matter of interest is primarily the expected behavior of a system or system component in terms of its reaction to given input stimuli and the functions and data required for processing the stimuli and producing the reaction“ [Gl07, S. 25]. Reussner et al. [RB+16a] unterscheiden hier für Software-Komponenten zwei Varianten: (1) das Verhalten in einer Komponente und (2) das Verhalten zwischen mehreren Komponenten. Übertragen auf Microservices bedeutet dies bei der ersten Variante, einen Microservice unabhängig von anderen isoliert zu betrachten. Auch Gamma et al. liefern für die erste Variante eine Definition aus dem Bereich der Objektorientierung, bei der das innere Verhalten adressiert wird: „Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility“ [GH+94, S. 10]. Die zweite Variante fokussiert dagegen Interaktion und Kommunikation zwischen Microservices. Letztere wird im Rahmen dieser Arbeit implizit durch die Interaktionsmöglichkeiten mit einem Microservice behandelt.

2.3 Web-API

Dieser Abschnitt liefert eine einheitliche Definition einer Web-API und grenzt diese von anderen Arten von APIs ab. Gleichzeitig soll ihre zunehmende Bedeutung bei der Gestaltung heutiger Systeme verdeutlicht werden. Anschließend werden verschiedene, aus der Literatur abgeleitete Qualitätsmerkmale einer Web-API vorgestellt. Die Beschreibung des Architekturstils REST schafft das Grundverständnis für die Ressourcenorientierung, um dann die verschiedenen Ausprägungen von REST und insbesondere die Ressourcenorientierung zu erörtern. Abschließend werden Spezifikationsformate für ressourcenorientierte Web-APIs vorgestellt, mit deren Hilfe eine Web-API beim Entwurf strukturiert beschrieben werden kann.

2.3.1 Definition und Bedeutung

Eine API ist eine Schnittstelle zwischen mindestens zwei unterschiedlichen Software-Anwendungen [SR+04]. Die Schnittstelle repräsentiert dabei: „einen Vertrag, der für den Fall des vertragsgemäßen Aufrufs die ebenfalls vertragsgerechte Ausführung einer bestimmten Aktion garantiert“ [RH08, S. 42]. Ergänzend definiert die DIN EN ISO 9241-210:2010 [DIN-9241-210] eine Schnittstelle als: „alle Bestandteile eines interaktiven Systems (Software oder Hardware), die Informationen und Steuerelemente zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen“ [DIN-9241-210, S. 7]. Übertragen auf APIs wäre der Benutzer hier einem Software-Entwickler gleichzusetzen, während die Steuerelemente eine Ansammlung von Methoden sind, um mit dem System zu interagieren. Die Implementierung der Methoden bleibt allerdings für Außenstehende verborgen, sodass lediglich über den Rückgabewert ermittelt werden kann, ob eine Methode erfolgreich ausgeführt wurde oder nicht. Dieses Prinzip,

auch als *Information Hiding* bezeichnet, ermöglicht die unabhängige Implementierung [Me10]: „The main advantage of this approach is the possibility of separating modules into public (the API itself) and private (the implementation of the API) parts so changes to the private part can be performed without impacting the public one and therefore minimizing the dependencies between these two parts“ [SR+04, S. 64]. Daneben sollte eine API bestimmten Richtlinien unterliegen, wie bspw. eine *low barrier to Entry*, wodurch letztlich die Benutzbarkeit bzw. deren Erlernbarkeit adressiert wird (vgl. Abschnitt 2.3.2) [Ro09].

Eine Web-API ist eine besondere Form der API, die über ein Netzwerk bereitgestellt wird, mit der sich wiederum über ein festgelegtes Protokoll interagieren lässt. Andere APIs hingegen, die nicht über ein Netzwerk bereitgestellt werden, werden nach Li et al. [LX+13] als lokale APIs oder nach [EZ+14] als statisch verknüpfte (engl. static linked) APIs bezeichnet. Deshalb wird eine Web-API wie folgt definiert:

> **Definition (Web-API)** *Eine Web-API ist eine Schnittstelle, die immaterielle Dienstleistungen über das Netzwerk bereitstellt. Dabei erfolgt die Interaktion mit einer Web-API mittels eines zuvor festgelegten Protokolls, wie bspw. HyperText Markup Language (HTTP).*

In dieser Arbeit ist die Web-API ein zentraler Bestandteil eines Microservice. Deshalb wird nun zudem der Begriff des *Service-Konsumenten* eingeführt. Die Web-API verbindet Service-Konsument und Microservice bzw. Service.

> **Definition (Service-Konsument)** *Ein Service-Konsument kann ein weiterer Service, eine Anwendung oder ein System sein, das die bereitgestellte Dienstleistung eines Service über eine Web-API nutzt.*

Zum besseren Verständnis illustriert die Abbildung 2.4 die Beziehungen zwischen den eingeführten Begriffen sowie deren Bezug zu einem Microservice.

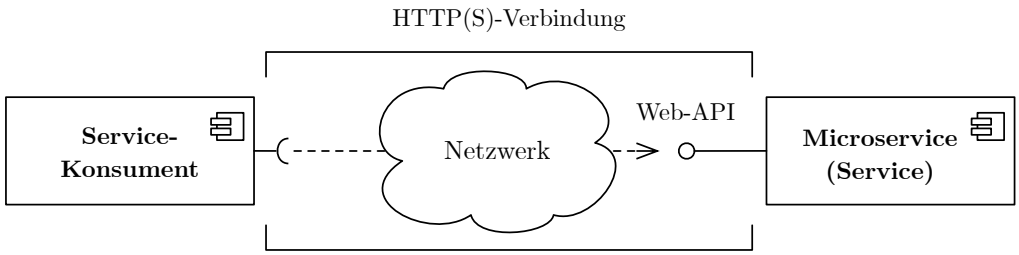


Abbildung 2.4: Zusammenhang zwischen Microservice, Web-API und Service-Konsument

2.3.2 Qualitätsmerkmale von Web-APIs

Für Web-APIs haben sich bestimmte Qualitätsmerkmale entwickelt, die mit dem Erfolg der Web-APIs und damit letztlich auch einer Microservice-Architektur korrelieren [SM07, GG+16b]. Vorzüge sind bspw. Wiederverwendbarkeit und Flexibilität. Ersteres korreliert wiederum positiv mit der Benutzbarkeit (nach der DIN EN ISO 9241-11:2016 [DIN-9241-11] auch als Gebrauchstauglichkeit bezeichnet) und der Mächtigkeit einer Web-API. Nur wenn diese auch einfach benutzbar bzw. erlernbar sowie mächtig genug ist, um verschiedene Anwendungsfälle der Service-Konsumenten zu bedienen, wird sie auch tatsächlich in Unternehmen eingesetzt und verschafft ihnen einen echten Mehrwert. Dafür muss sich die offerierte Web-API innerhalb und gegebenenfalls auch außerhalb des Unternehmens tatsächlich finden lassen. Deswegen kann die Auffindbarkeit als Befähiger (engl. enabler) für die Wiederverwendbarkeit angesehen werden [Ge11]. Im Lauf der Zeit kann sich die Web-API bspw. als Folge domänenspezifischer Anpassungen verändern. Dann muss stets sichergestellt sein, dass die Interoperabilität zwischen verschiedenen Service-Konsumenten und ressourcenorientierten Microservices nicht beeinträchtigt wird, um eine kontinuierliche Wiederverwendbarkeit zu gewährleisten. Indem entsprechende Veränderungen möglich sind, wird wiederum die Flexibilität gewährleistet, die ein Unternehmen heutzutage benötigt, um wettbewerbsfähig zu bleiben. Abbildung 2.5 fasst diese Einflüsse der Qualitätsmerkmale auf die Wiederverwendbarkeit und Flexibilität zusammen, die in den folgenden Unterabschnitten erläutert werden. Im Kontext dieser Arbeit werden diese Menge an Qualitätsmerkmalen auch als -teilmerkmale der Wiederverwendbarkeit bzw. Flexibilität aufgefasst.

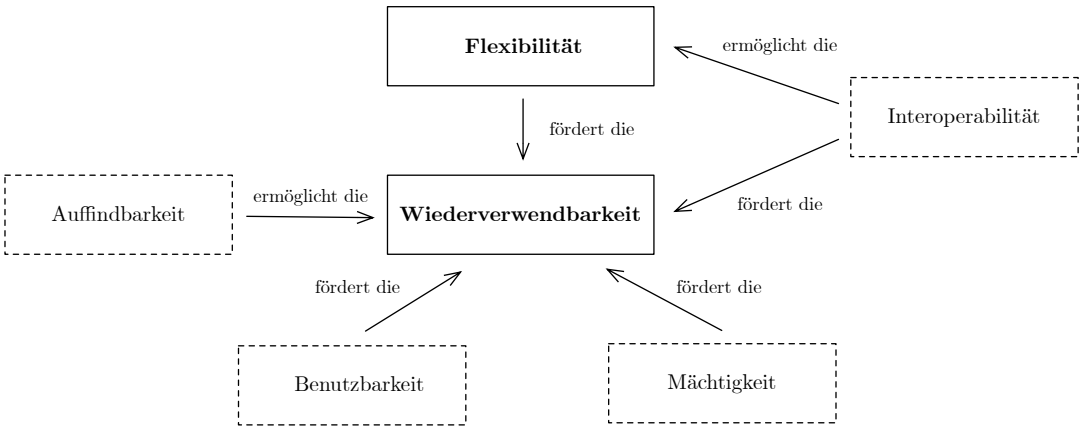


Abbildung 2.5: Qualitätsmerkmale von Web-APIs und deren Einfluss auf Wiederverwendbarkeit und Flexibilität

Benutzbarkeit

Als Benutzbarkeit gilt laut DIN EN ISO 9241-11:2016 [DIN-9241-11] das „Ausmaß, in dem ein System, ein Produkt oder eine Dienstleistung durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden können, um festgelegte Ziele effektiv, effizient und zufriedenstellend

zu erreichen“ [DIN-9241-11, S. 8]. Benutzbarkeit soll also das Erreichen von Zielen vereinfachen, was auch für eine API bzw. Web-API gilt. Die zunehmende Bedeutung in diesem Bereich belegt auch die steigende Anzahl von Usabilitytests von Microsoft bei der Konzeption von APIs [Cl04, Cl06]. Obgleich diese Untersuchungen keine Web-APIs im Speziellen adressieren, ist es naheliegend, diese Ergebnisse auch auf Web-APIs zu übertragen, sodass sie auch dort ihre Gültigkeit finden. So lässt sich auch bei Web-APIs ein zunehmender Trend von veröffentlichten Richtlinien zum Entwurf von Web-APIs bspw. von Zalando[1], Paypal[2] oder Microsoft[3] erkennen, die u. a. die Steigerung der Benutzbarkeit adressieren. In Hunter [Hu16] ist in diesem Zusammenhang die *Developer Experience (DX)* erwähnt, um das Benutzererlebnis des Software-Entwicklers zu beschreiben: „your [Web] API still won't be successfull if you don't pay close attention to developer experience once you've released it“ [Hu16, S. 109]. Basierend auf der Definition der *User Experience (UX)* betrifft die DX die „Wahrnehmungen und Reaktionen [eines Software-Entwicklers], die aus der tatsächlichen und/oder der erwarteten Benutzung eines Produkts, eines Systems oder einer Dienstleistung resultieren“ [DIN-9241-210, S. 7]. Zur DX oder auch UX gehört demnach auch die Erwartung eines Nutzers vor der eigentlichen Nutzung sowie deren Erfüllung. Diese Arbeit behandelt die Erwartung eines Nutzers, die der eigentlichen Nutzung vorausgeht, als Teil der Benutzbarkeit.

Stylos und Myers [SM07] untergliedern die Benutzbarkeit in sechs Qualitätsteilmerkmale, während Clarke [Cl04] etwaige Einflussfaktoren für die Benutzbarkeit einer API aus Sicht eines Software-Entwicklers liefert. Diese Faktoren lassen sich nur schwer auf die Eigenheiten einer Web-API übertragen und erfordern zudem Untersuchungen zur Laufzeit. Auch die ISO/IEC 25010:2011 [ISO-25010] unterteilt die Benutzbarkeit in verschiedene Qualitätsteilmerkmale, ohne das auf APIs zu beziehen. Deshalb wird für die folgenden Betrachtungen zur Benutzbarkeit einer Web-API ausschließlich die Arbeit von [SM07] fokussiert.

Stylos und Myers definieren sechs Qualitätsteilmerkmale: (1) Erlernbarkeit (engl. learnability), (2) Produktivität (engl. productivity), (3) Prävention von Fehlern (engl. error prevention), (4) Einfachheit (engl. simplicity), (5) Konsistenz (engl. consistency) und (6) Einhaltung von mentalen Modellen (engl. matching mental models). Erlernbarkeit (1) betrifft die Einstiegshürde, um eine API zu verstehen und zu verwenden. Das Kriterium der Produktivität (2) erfasst, wie die API von Software-Entwicklern verwendet wird. Dies erfordert allerdings Wissen zur Laufzeit, weshalb dieses Qualitätsteilmerkmal in dieser Arbeit nicht weiter beachtet wird. Das Qualitätsteilmerkmal der Prävention von Fehlern (3) soll vor allem helfen, diese im Vorfeld zu vermeiden. Das verlangt eine ausführliche Dokumentation zur Nutzung einer API. Die Einfachheit einer API (4) kann entweder durch Einbeziehung von Laufzeitinformationen oder durch entsprechende Reviews erfolgen. Beides erfordert, dass ein Erstentwurf durchgeführt wurde, bevor das Qualitätsteilmerkmal untersucht werden kann. Bei der Konsistenz (5)

[1] https://zalando.github.io/restful-api-guidelines/index.html (Letzter Zugriff: 04.02.2018)
[2] https://github.com/paypal/api-standards/blob/master/api-style-guide.md (Letzter Zugriff: 04.02.2018)
[3] https://github.com/Microsoft/api-guidelines (Letzter Zugriff: 04.02.2018)

wird sich auf die zu veräußernden Informationen bezogen – von der Bezeichnung der Endpunkte bis hin zu den übermittelten Nachrichten im Kontext von Web-APIs. Das letzte Qualitätsteilmerkmal (6) richtet sich an die Erwartungen der Software-Entwickler bei der Verwendung einer veräußerten Dienstleistung und hängt eng mit der Erlernbarkeit zusammen.

Auffindbarkeit

Das Auffinden von offerierten Dienstleistungen ist ein wesentliches Ziel von serviceorientierten Architekturen [Ge11, GG+16b]. Dementsprechend sollte etwa die Benennung logisch nachvollziehbar sein und sich auf den abgebildeten Geschäftsausschnitt beziehen lassen, um Rückschlüsse zu erlauben, sodass der entsprechende Microservice gefunden werden kann.

Interoperabilität

Die Interoperabilität wurde bereits als ein Qualitätsmerkmal eines Microservice in Abschnitt A.3 beschrieben, wo die Web-API im Mittelpunkt stand. Bei der Interoperabilität sind etwaige Veränderungen der Web-API bspw. durch neue Anforderungen oder eventuell erforderliche Fehlerkorrekturen zu betrachten. Das kann negative Folgen für die Service-Konsumenten haben und so die Interoperabilität verschlechtern [EZ+14].

Mächtigkeit

Stylos und Myers definieren die Mächtigkeit einer Web-API durch fünf Qualitätsteilmerkmale: (1) die Ausdrucksmächtigkeit (engl. expressiveness), (2) die Erweiterbarkeit (engl. extensibility), (3) die Evolvierbarkeit (engl. evolvability), (4) die Performanz (engl. performance) und die (5) Robustheit (engl. robustness). Die Ausdrucksmächtigkeit (1) erfasst, dass viele verschiedene Anwendungsfälle adressiert werden, die sich mithilfe einer Web-API realisieren lassen. Als Erweiterbarkeit (2) gilt das Potential für nutzungsspezifische Anpassungen bzw. Erweiterungen einer API durch Konsumenten: „[C]reate convenient user-specific components“ [SM07, S. 53]. Bei einer Web-API unterliegt diese allerdings zumeist nicht der Kontrolle der Service-Konsumenten, weshalb dieses Qualitätsteilmerkmal nicht weiter betrachtet wird. Evolvierbarkeit (3) betrifft Erweiterungen der Web-API im Zuge von Veränderungen des abgebildeten Geschäftsausschnitts. Dieses Qualitätsteilmerkmal wird im Rahmen der Interoperabilität betrachtet. Schließlich richtet sich die Performanz (4) an die effiziente Ressourcennutzung und die Robustheit (5) an die Abwesenheit von Fehlern in der Implementierung. Letztere Qualitätsteilmerkmale erfordern Informationen zur Laufzeit bzw. die Implementierung, die zum Zeitpunkt des Entwurfs nicht vorliegen. Sie werden deshalb nicht weiter betrachtet. Eine Ausnahme in Bezug auf Performanz sind Angaben wie etwa die benötigte Bandbreite, die auch ohne Laufzeitinformationen möglich sind.

2.3.3 Qualitätsaspekte/-indikatoren von Web-APIs

Die zu berücksichtigenden Qualitätsmerkmale einer Web-API sind also Benutzbarkeit, Auffindbarkeit, Interoperabilität und Mächtigkeit. Die grundlegenden Bedeutungen dieser Qualitätsmerkmale wurden ebenso wie ihr Einfluss auf die Wiederverwendbarkeit und Flexibilität eben beleuchtet. Um diese Qualitätsmerkmale beim Entwurf zu berücksichtigen, sind sie so weit zu verfeinern, dass sie klar zeigen, was beim Entwurf eines Ressourcenmodells und damit der Web-API zu beachten ist. Die Begriff der API ist im Nachfolgenden mit einer Web-API gleichzusetzen.

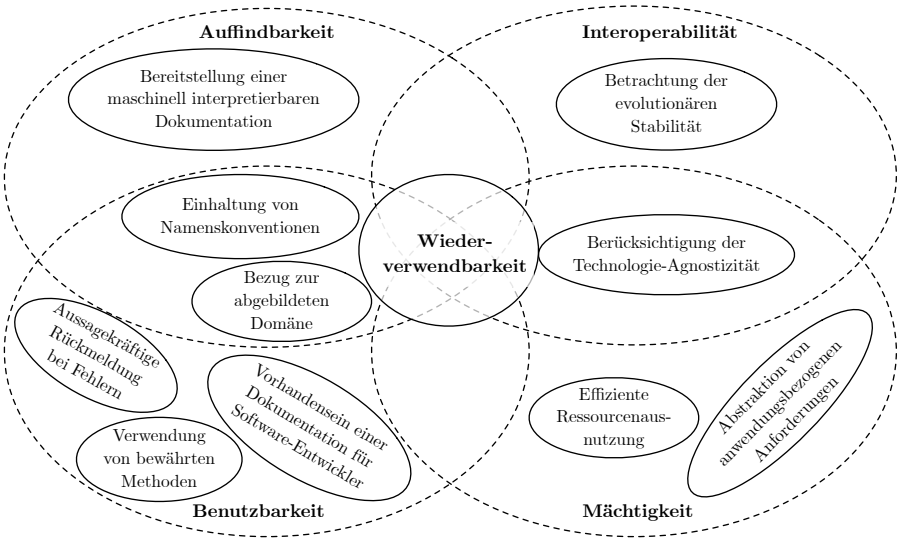


Abbildung 2.6: Betrachtete Qualitätsaspekte einer Web-API im Rahmen der Benutzbarkeit, Mächtigkeit, Auffindbarkeit und Interoperabilität

Gebhart [Ge11] liefert, gestützt auf Balzert [Ba08], einen Ansatz zur Verfeinerung von Qualitätsmerkmalen, die sogenannte "Qualitätsindikatoren" hervorbringt. Doch diese Verfeinerung liegt nicht im Fokus dieser Arbeit und würde unter Anwendung des zuvor erwähnten Ansatzes zu weit in die Tiefe führen. Deshalb wird nun lediglich aufgeführt, was zur Verbesserung derer Ausprägung beiträgt. Vollständigkeit ist hier zwar nur schwer möglich, wegen der Flexibilität dieses Ansatzes allerdings auch nicht erforderlich. Bezogen auf Gebhart [Ge11] oder auch [GG+16b] lassen sich die nun genannten und in dieser Arbeit als Qualitätsaspekte bezeichneten Faktoren auch als eine Form von Qualitätsindikatoren auffassen, die allerdings keine Quantifizierung ermöglichen. Es werden nur die Aspekte im Hinblick auf die Prämissen in Abschnitt 1.6 näher betrachtet, die beim Entwurf berücksichtigt werden können.

Abbildung 2.6 liefert eine Übersicht über die nun erläuterten Qualitätsaspekte, die beim Entwurfsprozess einer Web-API berücksichtigt werden müssen und den Rahmen für die Qualitätsbetrachtung einer Web-API schaffen. Es ist zu beachten, dass ein Qualitätsaspekt sich mehr als einem Qualitätsmerkmal zuordnen lässt.

Benutzbarkeit

Stylos und Myers [SM07] untergliedern die Benutzbarkeit zwar in die sechs schon vorgestellten Qualitätsteilmerkmale (vgl. Abschnitt 2.3.2), liefern jedoch keine weiteren Informationen, wie sie zu erreichen oder zu messen sind. Auf ihre Arbeit und weiterführende Literatur gestützt, wurden folgende Qualitätsaspekte im Kontext der Ressourcenorientierung (vgl. Abschnitt 2.3.4) abgeleitet:

1. **Einhaltung von Namenskonventionen:** Durch Einhaltung aufgestellter Konventionen zur Bezeichnung von Ressourcen, Repräsentationen und Uniform Resource Identifiers (URIs) steigt die Konsistenz, was sich positiv auf die Benutzbarkeit sowie die Auffindbarkeit auswirkt [SM07, Ge11, GG+16b]. Ergänzend definiert Clarke [Cl06] die Konsistenz als den Grad, in dem „an API can be inferred once part of it is learned“ [Cl04]. Höhere Konsistenz verringert zudem den Dokumentationsaufwand für eine API: „An inconsistent API requires more documentation [...], because it must frequently describe and explain the inconsistencies“ [Wa14, S. 2].

2. **Vorhandensein einer Dokumentation für Software-Entwickler:** Die Dokumentation einer API enthält sowohl Informationen zu deren Nutzung als auch Meta-Informationen, wie bspw. den Namen der API oder einen einleitenden Beschreibungstext, der den Nutzungskontext bzw. den Bounded Context erhellt. Die Dokumentation sollte die Beschreibung der korrekten Nutzung einer Web-API fokussieren, was die kognitive Einstiegshürde verringert, etwaige Fehler bei der Nutzung im Vorfeld vermeiden hilft und schließlich auch die Erlernbarkeit verbessert [SM07]. Die Dokumentation sollte alle notwendigen Informationen enthalten, die einen Software-Entwickler bei der Integration unterstützt. Den Einfluss von Dokumentationen auf die Nutzung untersucht die Studie in [Ro09]. Hunter [Hu16] unterscheidet diesbezüglich drei Dokumentationstypen: 1) Referenzdokumentation, 2) Arbeitsabläufe und 3) Anleitungen.

3. **Bezug zur abgebildeten Domäne:** Als Bezug zur Domäne (engl. domain correspondence) beschreibt Clarke [Cl04, Cl06] den Grad, inwieweit die einzelnen Bestandteile einer API den Objekten einer Domäne sich zuweisen lassen. Übertragen auf die Ressourcenorientierung entsprechen die zuvor erwähnten Komponenten den Ressourcen. So soll das Verständnis für Software-Entwickler verbessert werden, welche diese Funktionalitäten verwenden möchten.

4. **Verwendung von bewährten Methoden:** Als *bewährte Methoden* werden im Lauf der Zeit etablierte Lösungen für wiederkehrende Probleme bezeichnet. Ihre Verwendung senkt u. a. die Gefahr von *Cognitive Friction*, also den Konflikt zwischen Erwartung und tatsächlichem Verhalten der Technologie [Co99]. Hunter [Hu16] vereinheitlicht das mit den Entwurfsprinzipien *Don't surprise your users* und *Copy successful APIs*. Bei Stylos und Myers [SM07] entspricht dies den *matching mental models*. Als Korrespondenz mit einem mentalen Modell gilt, dass ein Service-Konsument die Funktionsweise einer Web-API intuitiv versteht. Das

mentale Modell hängt u. a. von bisherigen Erfahrungen, einem gesunden Urteilsvermögen sowie den bereitgestellten Informationen ab [Ba16].

5. **Aussagekräftige Rückmeldung bei Fehlern:** Fehler bei der Nutzung einer Web-API lassen sich nie grundsätzlich ausschließen, sondern nur mithilfe bereitgestellter Informationen zur Nutzung reduzieren. Allerdings lassen sich Fehler bei dieser Nutzung anders als im Fall von lokalen APIs nicht per Debugger oder ähnliches analysieren, da die zugrundeliegende Implementierung nicht der eigenen Kontrolle unterliegt und zumeist nicht eingesehen werden kann. Deshalb ist die Analysierbarkeit etwaiger Fehler durch aussagekräftige Rückmeldungen vom Service ein wichtiger Qualitätsaspekt. Clarke fasst diesen Aspekt unter dem Begriff *Penetrability* [Cl04, Cl06]. Die Fehleranalyse fokussiert hier die Analysemöglichkeiten des Service-Konsumenten, mit deren Hilfe dieser die Ursache des Fehlers ohne Kommunikation mit dem Service-Anbieter finden und ggf. selbst beheben kann, sofern der Fehler nicht auf den Service-Anbieter zurückgeht.

Mächtigkeit

Die Mächtigkeit korreliert im Zuge der Wiederverwendbarkeit und im Rahmen dieser Arbeit positiv mit der Anzahl an realisierbaren Anwendungsfälle. Letztere stehen dabei immer in Bezug zu funktionalen und nicht-funktionalen Anforderungen. Funktionale Anforderungen fokussieren die zu erbringenden immateriellen Dienstleistungen und lassen sich durch die Frage „Was wird bereitgestellt?“ erfassen. Die nicht-funktionalen Anforderungen betreffen hingegen deren Bereitstellung über ein geteiltes Netzwerk. Die Frage „Wie effizient ist die Bereitstellung?“ liefert hierfür entsprechende Einblicke.

1. **Abstraktion von anwendungsbezogenen Anforderungen**: Eine Abstraktion von konkreten Anwendungsfällen kann die Mächtigkeit in Bezug auf die realisierbaren Anwendungsfälle deutlich steigern [ZAL-API-GUIDE], da dann die Ausrichtung nicht eine spezifische Anwendung betrifft, sondern lediglich der abzubildende Ausschnitt der Geschäftsdomäne betrachtet wird. Um dennoch anwendungsbezogene Anforderungen umzusetzen sollte stattdessen ein dedizierter Microservice für eine Anwendung oder eine Anwendungsgruppe umgesetzt werden sofern die Anforderung eine Umsetzung auf Service-Seite bedingt. Hierzu liefert bspw. das BFF-Muster einen entsprechenden Ansatz [Ne15].

2. **Effiziente Ressourcenausnutzung:** Wie andere APIs auch, sollten Web-APIs möglichst schnell auf eine Anfrage des Service-Konsumenten mit einer passenden Antwort reagieren und die notwendigen Anfragen für das Durchführen eines Anliegens minimal halten. Hier hängt die Antwortzeit von verschiedenen Faktoren durch das geteilte Medium ab und ebenso von der bereitgestellten Web-API sowie der zugrundeliegenden Implementierung. Um die Übertragungsmenge und so auch die Antwortzeit zu verkürzen, muss die Web-API eine Selektion

und Transformation von Informationen ermöglichen, welche der Service-Konsument effizient weiterverarbeiten kann. Die Informationsmenge orientiert sich demnach an seinen Bedürfnissen. Die Einflussfaktoren des geteilten Mediums werden dagegen nicht weiter beleuchtet, da hierzu Laufzeitinformationen bzw. betriebliche Informationen notwendig sind.

Interoperabilität

Die Interoperabilität fokussiert im Gegensatz zur Benutzbarkeit einer Web-API die technische Sichtweise und damit die Integrierbarkeit durch Service-Konsumenten. Eine Web-API sollte möglichst keine Vorgaben hinsichtlich der vom Service-Konsumenten verwendeten Technologien machen. Daneben ist bei Änderungen an der Web-API infolge interner oder externer Faktoren darauf zu achten, dass existierende Service-Konsumenten auch weiterhin mit ihr interagieren können, ohne dass die Interoperabilität leidet. Dieser Qualitätsaspekt ist anders als bei lokalen APIs zu berücksichtigen, da die Evolution einer Web-API nicht Kontrolle der Service-Konsumenten unterliegt: „In the statically linked API context, developers could choose to stay with an older version of e.g. libxml, which meets their needs, yet, with web service APIs the provider can at any time unplug a specific version (and functionality), thus forcing an upgrade“ [EZ+14, S. 84].

1. **Berücksichtigung der Technologie-Agnostizität:** Die Frage der Agnostizität betrifft die Kontextunabhängigkeit [Pa11]. Kontextunabhängigkeit in Bezug auf eine Technologie stellt sicher, dass der Entwurf einer Web-API nicht von einer konkreten Technologie abhängt. So wird sichergestellt, dass weder für die Implementierung durch ein Entwicklungsteam noch für die Integration in einem Service-Konsumenten eine konkrete Technologie verlangt wird. Letzteres hat demnach auch einen Einfluss auf die *Mächtigkeit* einer API.

2. **Betrachtung der evolutionären Stabilität:** Die Begrifflichkeit der *evolutionären Stabilität* bezeichnet das Anliegen, dass Service-Konsumenten auch nach der Änderung (bzw. Evolution) an einer Web-API möglichst weiterhin funktionsfähig bleiben und die Web-API demnach weiterhin als stabil betrachtet werden kann. Die Umfrage von Espinha et al. zeigt, dass Software-Entwickler hier entsprechende Probleme sehen: „it takes them far more time maintaining the integration than it does integrating with a web API in the beginning“ [EZ+14, S. 86].

Auffindbarkeit

Die Auffindbarkeit von bestehenden Funktionalitäten eines Microservice, die über eine Web-API veräußert werden, ist eine Grundvoraussetzung für deren Wiederverwendbarkeit. Im Kontext einer SOA identifizierte Gebhart [Ge11] hierfür drei Qualitätsindikatoren, die zur Entwurfszeit bestimmt werden können und auch bei REST-basierten Services ihre Anwendung finden [GG+16b]. Zu den Qualitätsindikatoren zählen die: 1) die Einhaltung von Namenskonventionen, 2) die fachliche Benennung der Funktionalitäten und 3) der Informationsumfang. Werden die Qualitätsindikatoren mit

den zuvor beschriebenen Qualitätsaspekten der Benutzbarkeit in Einklang gebracht, zeigen sich die Qualitätsindikatoren der Auffindbarkeit als eine Teilmenge. Lediglich die fehlende Bereitstellung einer Dokumentation kann die Auffindbarkeit erschweren, weshalb ein weiterer Qualitätsaspekt der Auffindbarkeit zugeordnet wird.

1. **Bereitstellung einer maschinell interpretierbaren Dokumentation:** Während Benutzbarkeit der Dokumentation einer Web-API sich an Software-Entwickler von Service-Konsumenten richtet, ist für die Auffindbarkeit eine maschinell interpretierbare Dokumentation mit festgelegter Syntax und Semantik entscheidend. Nur so kann eine effiziente Auffindbarkeit von offerierten Funktionalitäten eines Microservice bspw. in Form eines API-Entwicklerportals (vgl. [De17, S. 175f.]) sichergestellt werden. Ergänzend lassen sich auch zusätzliche semantische Technologien und Sprachen einsetzen, wie bspw. Resource Description Framework (RDF) [W3C-RDF] und SPARQL [W3C-SPAQRL].

2.3.4 REST

REpresentational State Transfer (REST) ist ein Architekturstil für verteilte und interoperable Hypermedia-Systeme [Fi00, GG+16c]. Bei einem Architekturstil handelt es sich um eine Anzahl von Randbedingungen (engl. constraints) für Architekturelemente, um bestimmte Systemeigenschaften zu erzielen, wie bspw. eine hohe Erweiterbarkeit oder Skalierbarkeit (vgl. auch Abschnitt 2.1.2) [RA+13]. Im Sinne von Fielding [Fi00] lässt sich der Architekturstil so definieren:

> **Definition (REST)** *REpresentation State Transfer (REST) ist ein Architekturstil, der aus einer Vielzahl von netzwerkbasierten Architekturstilen abgeleitet wurde, und um weitere Randbedingungen im Zuge einer einheitlichen Schnittstelle (engl. uniform interface) angereichert wurde.*

Da REST für diese Arbeit ein integraler Bestandteil darstellt, erläutern die nachfolgenden Absätze nun die mit REST einhergehenden Randbedingungen für eine einheitliche Schnittstelle sowie die verschiedenen Ausprägungen von REST-basierten Anwendungen. Bei der Beschreibung von letzteren wird vor allem die Ressourcenorientierung beleuchtet, die auch im Fokus dieser Arbeit liegt.

Bevor die einzelnen Randbedingungen detailliert betrachtet werden, sind für das korrekte Verständnis die Begriffe *Ressource* und *Repräsentation* im Kontext von REST zu definieren.

> **Definition (Ressource)** *Eine Ressource ist eine eindeutig adressierbare und konzeptionelle Zuweisung zu einer oder mehreren Entitäten, wobei sie nicht vom Zeitpunkt der Instanziierungen abhängt. Als Entität lässt sich grundsätzlich alles fassen, womit ein Service-Konsument interagieren möchte [RFC2396-1998, Fi00, WP+10, RA+13].*

Definition (Repräsentation) *Eine Repräsentation ist eine Zustandsbeschreibung einer Ressource, die kein bestimmten Datenformat verlangt [Fi00, WP+10, RA+13].*

Randbedingungen für eine einheitliche Schnittstelle

Fielding [Fi00] definiert für die einheitliche Schnittstelle vier Randbedingungen [RA+13]: (1) Identifizierung von Ressourcen, (2) Manipulation von Ressourcen durch Repräsentationen, (3) selbstbeschreibungsfähige Nachrichten und (4) Hypermedia As The Engine Of Appliation State (HATEOAS).

Die Identifizierung von Ressourcen (1) erfolgt durch mindestens eine URIs [WP+10, RA+13]. Diese wird als: „compact string of characters for identifying an abstract or physical resource“ [RFC2396-1998, S. 1] definiert. Eine URI setzt sich aus einem Schema und einer schemaspezifischen Struktur zusammen, wobei das Schema festlegt wie die schemaspezifische Struktur aufgebaut und zu interpretieren ist [RFC2396-1998]. Um herauszufinden, ob mehrere URIs die gleiche Ressource adressieren, ist allerdings weiteres Wissen notwendig [WP+10].

Sobald eine Ressource identifiziert ist, ist eine Interaktion mit ihr möglich. Um den Zustand einer Ressource zu manipulieren (2), übermittelt der Service-Nutzer den gewünschten Zustand mittels einer Repräsentation an die Ressource, die diese Repräsentation empfängt und dann den eigenen Zustand verändert [RA+13]. Grundsätzlich können mehrere Repräsentationen für eine Ressource existieren, um bspw. unterschiedliche Anwendungsfälle abbilden zu können [WP+10].

Die nächste Bedingung (3) von Fielding [Fi00] erfordert, dass alle notwendigen Informationen zum Verständnis der Antwort bzw. Anfrage in der Nachricht selbst enthalten sein sollten oder zumindest mit dieser irgendwie verknüpft sind. Eine Nachricht ist in diesem Fall durch den Nachrichtenkopf mit etwaigen Meta-Informationen sowie den Nachrichtenrumpf mit einer entsprechenden Repräsentation definiert [RA+13].

Abschließend wird mit HATEOAS die automatische Anpassung des Service-Konsumenten auf Änderungen seitens der Web-API [RA+13] adressiert. Dies ist vergleichbar mit einer herkömmlichen Webseite, bei der allerdings der Nutzer entscheidet, welche URIs er als nächstes aufrufen will [WP+10, RA+13]. Diese Randbedingung wird in dieser Arbeit im Sinne der aufgestellten Prämissen (vgl. Abschnitt 1.6) nicht näher behandelt.

Ausprägungen von REST

Mit der Zeit sind verschiedene Ausprägungen im Kontext von REST entstanden, wie bspw. Pragmatic REST nach Mulloy [Mu12]. Sie lassen sich nach RMM [Fo10, WP+10] klassifizieren, das mehrere hierarchisch angeordnete Ebenen umfasst. Jede Ebene hat bestimmte Eigenschaften von REST, während die Ebenen darunter jeweils eine Untermenge der oberen Ebenen sind. Als Übersicht zeigt Abbildung 2.7 die einzelnen Ebenen von RMM in leicht überarbeiteter Form.

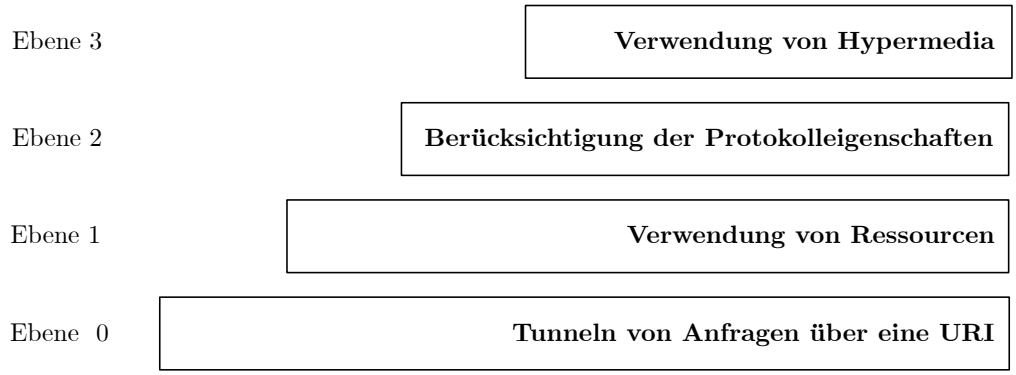


Abbildung 2.7: Modifiziertes Richardon Maturity Model (RMM) in Anlehnung an [WP+10]

Auf *Ebene 0* sind die Services klassifiziert, die nur eine URI aufweisen und bei denen die gesamte Interaktion über den Nachrichtenrumpf des entsprechenden Anwendungsschichtprotokolls gesteuert wird. Bekannte Beispiele sind Services nach den WS*-Spezifikation mittels Simple Object Access Protocol (SOAP) unter Verwendung von HTTP. Sämtliche Anfragen werden dabei in Web Service Description Language (WSDL) definiert und in Form von Extensible Markup Language (XML) im Nachrichtenrumpf instanziiert. Auf *Ebene 0* gestützt, wird auf *Ebene 1* die URI in mehrere URIs aufgeteilt sowie das Konzept der Ressourcen eingeführt. Jede URI verweist so auf eine Ressource, die dann als Endpunkt für die Interaktion fungiert. Mit *Ebene 2* wird schließlich die korrekte Verwendung des Anwendungsschichtprotokolls fokussiert. So sollten bspw. für eine Leseoperation nur idempotente und seiteneffektfreie Methoden verwendet werden, bspw. bei HTTP die GET-Methode. Die dritte und damit letzte Ebene (*Ebene 3*) setzt die Verwendung von Hypermedia voraus. Dann sollten zum einen Hyperlinks zwischen Ressourcen zur Navigation genutzt werden und zum anderen die gesamte Semantik, die zur Interpretation der Ressourcen und Hyperlinks notwendig ist, in den Repräsentationen der Ressourcen enthalten sein. Nur, wenn der Service die Prinzipien von Hypermedia berücksichtigt und verwendet, kann und sollte er als RESTful Service bezeichnet werden [Fi08].

Auf Grundlage dieses RMM lässt sich die Ressourcenorientierung grundsätzlich auf Ebenen 1, 2 als auch 3 verorten. Allerdings liefert *Ebene 3* bereits die Grundlage für hypermedia-getriebene Web-APIs, weshalb die Ressourcenorientierung hier nicht mehr im Fokus steht. Die *Ebene 1* führt zwar das Konzept der Ressourcenorientierung ein, setzt allerdings keine korrekte Verwendung des ausgewählten Anwendungsschichtprotokolls voraus. Deshalb wird für diese Arbeit die Ressourcenorientierung durch Einhaltung der *Ebene 2* des RMM bestimmt, wie die folgende Definition verdeutlicht:

> **Definition (Ressourcenorientierung)** *Die Ressourcenorientierung ist auf Ebene 2 des Richardson-Maturity-Modells (RMM) anzusiedeln und bildet damit gleichzeitig die Vorstufe zu hypermedia-getriebenen Web-APIs.*

2.3.5 Spezifikation von ressourcenorientierten Web-APIs

Für die Spezifikation von ressourcenorientierten Web-APIs existieren mittlerweile einige Spezifikationssprachen, wie bspw. Web Application Description Language (WADL)[4], WSDL[5] in der Version 2, OpenAPI (OAI)[6] (ursprünglich Swagger), RESTful API Modeling Language (RAML)[7] oder API Blueprint[8]. Grundsätzlich weisen alle Spezifikationssprachen eine Überlappung im Hinblick auf ihre Mächtigkeit zur Spezifikation von ressourcenorientierten Web-APIs auf und sind zumeist mit vertretbarem Aufwand ineinander überführbar. Für die Wahl einer Spezifikationssprache sind offenbar andere Kriterien eher sinnvoll, wie bspw. die Lizenz, die Beabsichtigung der Kommerzialisierung, die Unterstützung durch Werkzeuge, die Entwicklung über die letzten Jahre oder die Verbreitung im industriellen Kontext. Ähnlich deutet das auch Surwase in [Su16] beim Vergleich verschiedener Spezifikationssprachen für REST-basierte Web-APIs: „Developer should choose the Specification based on tool that available and ease of development it affords“ [Su16, S. 637].

Für diese Arbeit wurde OAI gewählt wegen ihrer Entwicklung in den letzten Jahren und der aktiven Beteiligung von führenden Technologie-Unternehmen, darunter Microsoft, Google, IBM, PayPal oder SAP. Die OAI verfolgt das Ziel, „to define a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic“ [OAI-REPO].

Die OAI unterstützt als Auszeichnungssprache die JavaScript Object Notation (JSON) und die Yet Another Multicolumn Layout (YAML), wobei beiden Sprache ineinander transformiert werden können [Hu16]. Für diese Arbeit fiel die Wahl auf YAML, da so auf eine überflüssige Klammerung verzichtet und so die Lesbarkeit erhöht wird. Eine beispielhafte Darstellung einer OAI-Spezifikation ist dem Quelltext 2.1 zu entnehmen. Darin ist die Version der Spezifikation ersichtlich und ebenso Meta-Informationen zu einer Web-API. Der Hauptbestandteil ist allerdings den einzelnen Endpunkten (*paths*) sowie Definition der Repräsentationen (*schemas*) vorbehalten.

```
1 openapi: 3.0.0
2 info:
3   version: 0.1.1
4   title: DisruptionService
5   description:
6   contact:
```

[4] https://www.w3.org/Submission/wadl/ (Letzter Zugriff: 07.01.2018)
[5] https://www.w3.org/TR/wsdl20/ (Letzter Zugriff: 07.01.2018)
[6] https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md (Letzter Zugriff: 07.01.2018)
[7] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/ (Letzter Zugriff: 07.01.2018)
[8] https://github.com/apiaryio/api-blueprint (Letzter Zugriff: 07.01.2018)

```
7       name: Name of the development team
8   paths:
9     /v1/causers:
10       post:
11        description: "Add new causer"
12        requestBody:
13          required: true
14          content:
15            application/vnd.sample.v1.5.hal+json:
16              schema:
17                 $ref: "#/components/schemas/causer"
18        responses:
19          204:
20            description: "The causer was successfully added"
21            content:
22              application/vnd.sample.v1.5.hal+json:
23                schema:
24                  $ref: "#/components/schemas/created_resource"
25          400:
26            description: "Not all invariants were fulfilled"
27            content:
28              application/vnd.sample.v1.5.hal+json:
29                schema:
30                  $ref: "#/components/schemas/error"
31  components:
32    schemas:
33      error:
34        type: "object"
35        properties:
36          type:
37            type: "string"
38          # ...
```

Quelltext 2.1: Beispielhafter Ausschnitt einer OAI-Spezifikation (Version 3.0)

Mittlerweile findet die OAI-Spezifikaton eine breite Unterstützung bei kostenlosen und proprietären Entwicklungswerkzeugen. Gleichzeitig existieren auch eigens entwickelte Werkzeuge, die allerdings noch den Namen des ursprünglichen Produktes halten. Dazu zählt bspw. Swagger UI, das eine interaktive Visualisierung der OAI-Spezifikation bereitstellt. Dadurch können Software-Entwickler von potentiellen Service-Konsumenten die Web-API und damit den zugrundeliegenden Service ausprobieren ohne dass eine Form von Implementierung erforderlich ist.

3 Stand der Forschung

Für den Entwurf von ressourcenorientierten Microservices basierend auf dem domänengetriebenen Entwurfsansatz bestehen mehrere Anforderungen. Sie dienen dazu, verfügbare Arbeiten zum Thema und ebenso die in dieser Arbeit erzielten Ergebnisse zu bewerten. Deshalb werden die Anforderungen nun erläutert und in einem Anforderungskatalog zusammengefasst. Auf diesen gestützt wird der Forschungsstand vorgestellt und analysiert. Dieses Kapitel schließt mit der Skizzierung des so ermittelten Handlungsbedarfs, der auch die Motivation für die eigenen Beiträge liefert.

3.1 Anforderungen an einen domänengetriebenen Entwurf von ressourcenorientierten Microservices

Die Anforderungen an einen domänengetriebenen Entwurf von ressourcenorientierten Microservices wurden vorrangig aus den zu bearbeitenden Problemstellungen sowie den letztlichen Beiträgen und Vorzügen dieser Arbeit abgeleitet (siehe Abschnitt 1.5). Die Arbeit von Gebhart [Ge11] bietet einige Orientierung, da sie ebenfalls vorrangig die Mikroarchitektur von Services im Kontext einer SOA beleuchtet, jedoch den Schwerpunkt auf die Ermittlung von Qualitätsindikatoren sowie der Überarbeitung eines Service-Entwurfs legt. Zur besseren Übersicht werden die Anforderungen für diese Arbeit am Ende des Abschnitts in einen sogenannten Anforderungskatalog überführt.

3.1.1 Beschreibung der Anforderungen

Zur Bewertung der bestehenden Arbeiten und der eigenen Beiträge im Hinblick auf den Entwurf von Microservices wurden sechs Anforderungen (*A1* bis *A6*) identifiziert. Diese Anforderungen lassen sich in Beziehungen zu den Beiträgen dieser Arbeit und damit zu den zugrundeliegenden Problemstellungen setzen. *A1, A2, A5* und *A6* befassen sich vor allem mit dem ganzheitlichen Entwurfsprozess und der Modellierung, während es bei *A3* und *A4* vorrangig um den Entwurf einer ressourcenorientierten Web-API geht.

A1: Nachvollziehbarkeit

Die Nachvollziehbarkeit (engl. traceability) ist die erste Anforderung, die im domänengetriebenen Entwurf von ressourcenorientierten Microservices berücksichtigt werden muss. Bei diesem Kriterium handelt es sich, so Schwarz [Sc12, S. 12], um die Fähigkeit, die Beziehung zwischen zwei

Entitäten, die ein Software-System bilden oder bei dessen Entwicklung relevant sind, zu verfolgen und zu verstehen. Diese Betrachtungsweise ermöglicht eine systematische Ableitung des Entwurfs aus Analyseartefakten sowie deren Überführung auf die Implementierungsebene. Die Bedeutung der Nachvollziehbarkeit wurde insbesondere für modellgetriebene Ansätze erkannt [SV+06, Ge11, Sc12]. Um eine Nachvollziehbarkeit für den Entwurf sicherzustellen, ist zunächst zu ermitteln, welche Informationen aus dem zugrundeliegenden Domänenmodell bzw. den Bounded Contexts benötigt werden, und zu klären, wie diese Informationen geeignet formalisiert werden, damit eine systematische Überführung in ein ressourcenorientiertes Modell möglich ist. Daneben muss sichergestellt sein, dass eine entsprechende Zuweisungsvorschrift existiert, welche die konkrete Überführung der formalisierten Informationen auf die Entwurfsebene beschreibt. Dieser Entwurf soll letztlich als Grundlage für die Abbildung auf die Implementierungsebene dienen, weshalb auch hier die Nachvollziehbarkeit zur verwendeten Technologie gegeben sein muss. Dies stellt sicher, dass der resultierende Entwurf konkret genug für die Umsetzung des Microservice ist. Wird diese Betrachtungsweise mit den verschiedenen Dimensionen der Nachvollziehbarkeit aus [Sc12] in Einklang gebracht, wird hiermit vorrangig die „forward-traceability“ und „inter-level traceability“ anvisiert.

A2: Betrachtung von strukturellen und verhaltensspezifischen Aspekten

Ressourcenorientierte Microservices veräußern ihre immaterielle Dienstleistung über eine Web-API, die aus Ressourcen und einem beschränkten Set an Operationen zum Erstellen, Lesen, Modifizieren und Löschen (kurz: Create-Read-Update-Delete (CRUD)) besteht. Die zu veräußernde Dienstleistung ergibt sich aus dem abzubildenden Geschäftssauschnitt bzw. dem Bounded Context. Die Ressourcen sollen die in dem jeweiligen Bounded Context enthaltenen Domänenobjekte in geeigneter Form auf die technische Ebene im Sinne von Vogel et al. [VA+09] abbilden. Dabei müssen sich sowohl die strukturellen als auch die verhaltensspezifischen Aspekte eines Bounded Context in geeigneter Form überführen lassen, da ansonsten wichtige Informationen fehlen. So müssen bspw. auch etwaige Invarianten sowie Vor- und Nachbedingungen durch die letztlich genutzten Ressourcen bzw. deren Operationen abgebildet werden. Dieses Vorgehen deckt sich auch mit der Aussage von Porres und Rauf: „It is possible to create web services with a complex application state that still follow the REST architectural style“ [PR11, S. 19]. Zu beachten ist hier, dass damit nur die strukturellen und verhaltensspezifischen Aspekte basierend auf dem Bounded Context adressiert werden.

A3: Optionale Verwendung des Hypermedia-Aspekts

Zwar wird der Hypermedia-Aspekt im Sinne der Prämissen in Abschnitt 1.6 nicht explizit in dieser Arbeit betrachtet. Doch soll der Entwurf von ressourcenorientierten Microservices weder die spätere Verwendung von Hypermedia einschränken noch soll deren Verwendung verpflichtend sein. Die Ressourcenorientierung, die diese Arbeit fokussiert, bildet die Voraussetzung für den Einsatz von Hypermedia entsprechend des RMM (vgl. Abschnitt 2.3.4). Das soll sicherstellen, dass der Entwurf auch

für hypermedia-getriebene Microservices mit einer entsprechenden Erweiterung des Lösungsansatzes genutzt werden kann. Die letztliche Entscheidung im Hinblick auf die Verwendung von Hypermedia obliegt dabei dem jeweiligen Software-Architekten und wird in dieser Arbeit nicht weiter beleuchtet, da die semantische Erweiterung ein weiteres Forschungsfeld wäre.

A4: Unterstützung bei Entwurfsentscheidungen

Die Unterstützung eines Software-Architekten beim Entwurf von ressourcenorientierten Microservices ist die vierte Anforderung. So muss sichergestellt werden, dass der Software-Architekt durch fundierte Informationen und angereichertes Expertenwissen zum domänengetriebenen Entwurf von ressourcenorientierten Microservices nachvollziehbare Entscheidungen treffen kann. Deswegen muss klar sein, wie Entwurfsentscheidungen die endgültige Qualität des Entwurfs für den Microservice beeinflussen. Hier ist zudem zu beachten, dass Entwurfsentscheidungen neben Vorzügen auch Nachteile haben oder abhängig von der abzubildenden Domäne und dem Anwendungsfall als optional zu betrachten sind [Ge11]. Die Betrachtung der Qualität eines Entwurfs ist nach Vogel et al. [VA+09] der technischen Architektur zuzuordnen. In dieser Arbeit werden unter Berücksichtigung der Prämisse P5 Benutzbarkeit, Mächtigkeit, Auffindbarkeit und Interoperabilität als Qualitätsteilmerkmale der Wiederverwendbarkeit betrachtet (siehe Abschnitt 2.3.2), womit vorrangig die resultierende Web-API adressiert wird. Die tatsächliche Unterstützung beim Entwurf in Bezug auf die zu betrachtenden Qualitätsmerkmale kann vielfältig sein: eine einfache Kontrollliste (engl. checklist) [Vi08, GG+16c] oder auch ein Entscheidungsunterstützungssystem (engl. decision support system).

A5: Technologie- und Plattformunabhängigkeit

Die modellierten Entwurfsartefakte, die bei der Anwendung des Vorgehens entstehen, müssen zum einen unabhängig von einer konkreten Implementierungstechnologie und zum anderen unabhängig von einer konkreten Plattform sein. Eine Plattform besteht laut Vogel et al. [VA+09] aus Software- und ggf. Hardware-Bausteinen zusammen, die zur Ausführung von Software-Bausteinen dienen. In diesem Kontext ist ein Software-Baustein einem Microservice gleichzusetzen. Gründe die Technologieunabhängigkeit sind zum einen, dass abstraktere Konzepte im Vergleich zu konkreten Technologien stabiler sind [SV+06] und zum anderen keine Einschränkungen im Hinblick auf die zu verwendeten Implementierungstechnologie aufgelegt werden. Das jeweilige Entwicklungsteam entscheidet, welche Technologien zur Umsetzung eingesetzt werden. Weiter müssen die Entwurfsartefakte unabhängig von der späteren System-Umgebung sein. Deshalb darf der Entwurf bspw. keine Angaben zum Laufzeitverhalten enthalten. Nur dann lässt er sich mit verschiedenen Technologien abbilden und auf unterschiedlichen System-Umgebungen betreiben.

A6: Abbildung des Entwurfs auf die Implementierungsebene

Durch die Abbildung auf die Implementierungsebene sowie die Verknüpfung mit der Zielarchitektur sollen die Entwurfsartefakte die Umsetzung befördern. Dabei muss der technologie- und plattformagnostische Entwurf eines Microservice mit der zu wählenden Technologie verknüpft und schließlich auf eine Software-Architektur abgebildet werden können. Nur das stellt sicher, dass sich der letztliche Entwurf auch adäquat auf der Implementierungsebene widerspiegelt. Andernfalls können Diskrepanzen zwischen Entwurf und Implementierung den Wartungsaufwand und die Fehleranfälligkeit durch nicht betrachtete Randfälle erhöhen.

3.1.2 Anforderungskatalog

Der Anforderungskatalog fasst die genannten Anforderungen (siehe Abschnitt 3.1.1) zur Bewertung relevanter Arbeiten und der eigenen Beiträge in dieser Arbeit in der Tabelle 3.1 zusammen.

Nummer	Beschreibung
A1	Nachvollziehbarkeit
A2	Betrachtung von strukturellen und verhaltensspezifischen Aspekten
A3	Optionale Verwendung des Hypermedia-Aspekts
A4	Unterstützung bei Entwurfsentscheidungen
A5	Technologie- und Plattformunabhängigkeit
A6	Abbildung des Entwurfs auf die Implementierungsebene

Tabelle 3.1: Anforderungskatalog für die Bewertung relevanter Arbeiten sowie der eigenen Beiträge

3.2 Bewertung bestehender Arbeiten

Dieser Abschnitt bewertet relevante Forschungsarbeiten, die einen ähnlichen Ansatz verfolgen oder Schwerpunkte auf einzelne Anforderungen des Anforderungskatalogs in Abschnitt 3.1.2 setzen. Für die folgende Evaluation mithilfe des Anforderungskatalogs wurden Arbeiten mit den meisten Überdeckungen zu den Beiträgen dieser Arbeit ausgewählt. Zudem wurden die Anzahl an Referenzierungen, die Aktualität, also der Zeitpunkt der Veröffentlichung, sowie der Veröffentlichungsort berücksichtigt.

3.2.1 Rossi: UML-based Model-Driven REST API Development

In [Ro16] wird ein modellgetriebener Ansatz für die Entwicklung von REST-basierten Web-APIs präsentiert. Die Notwendigkeit des Beitrags begründet der Autor mit dem Fehlen von expliziten Modellierungsartefakten bei der Entwicklung von REST-basierten Web-APIs und dem Fehlen von

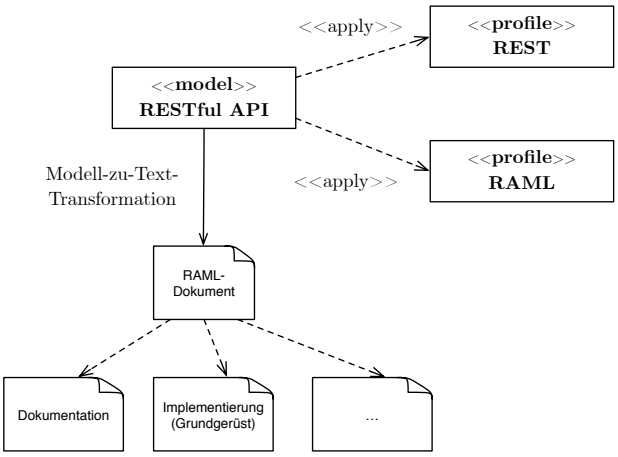


Abbildung 3.1: Modellgetriebener Ansatz für die Entwicklung von APIs basierend auf REST

standardisierten Methoden zum Auffinden und Verstehen der bereitgestellten immateriellen Dienstleistung eines REST-basierten Service. Seiner Ansicht nach beeinträchtigt das die Entwicklung: „on the provider side no modeling artifact is available to guide the implementation of the API, on the consumer side there is no formal documentation on how to use the API. As a consequence also the concept of a contract among the parties cannot be addressed“ [Ro16, S. 194]. Die Relevanz des Problems belegt er mit dem gesteigerten Aufkommen von neuen, im Gegensatz zu UML, nicht standardisierten Modellierungsbeschreibungen für Web-APIs im Kontext von REST. Dabei stellt er gleichzeitig klar, dass diese Modellierungsbeschreibungen lediglich weitere Domain Specific Languages (DSLs) sind, die in entsprechende Werkzeugketten (engl. toolchains) im Sinne von Continuous Integration (CI) integriert werden müssen.

Der in der Arbeit vorgestellte UML-basierende Lösungsansatz fußt auf den folgenden fünf Randbedingungen (R1 - R5), die der Autor im Vorfeld eigenständig definiert hat:

(R1) Generierung von Quellcode für die Implementierung sowie optional von einer Web-API-Dokumentation,

(R2) Betrachtung der Plattform- und Technologieunabhängigkeit,

(R3) Vermeidung von neuen DSLs, wie es bspw. in [Sc14] der Fall ist,

(R4) Fokus auf die strukturelle Modellierung und

(R5) keine Abhängigkeit von proprietären Werkzeugen.

Hinsichtlich des Aufbaus setzt sich der Lösungsansatz zum einen aus der Modellierung der Web-API mittels eines eigens entwickelten UML-Profils für REST (kurz: REST-Profil) und zum anderen aus einer Modell-zu-Text-Transformation in ein RAML-Dokument zusammen, bei der im Vorfeld ein

weiteres UML-Profil (kurz: RAML-Profil) zum Einsatz kommt (siehe Abbildung 3.1). RAML ist „a concise, expressive language for describing RESTful APIs. Built on broadly used standards such as YAML and JSON, RAML is a non-proprietary, vendor-neutral open spec“ [Hu16, S. 138]. Der Vergleich von RAML mit anderen DSLs, wie bspw. OpenAPI [OAI-WEB], wird in dieser Arbeit nicht weiter betrachtet. Daneben beinhaltet das resultierende RAML-Dokument lediglich die strukturellen Aspekte und muss für die Implementierung um weitere Details ergänzt werden. Es ist „a skeleton containing structural information and it has to be enriched with additional details before refining it in successive artifacts“ [Ro16, S. 196]. Allerdings ist die Veränderung eines generierten Artefakts bei MDSD als Folge der notwendigen Erweiterung immer problematisch, sobald sich das Quell-Artefakt bzw. in diesem Fall das zugrundeliegende UML-Modell verändert.

Deswegen verweist der Autor hier auf einen Mechanismus bei RAML, der eine Erweiterung ohne Veränderung des ursprünglichen Artefakts erlaubt. Gleichzeitig betont er jedoch, dass dies nicht die von ihm bevorzugte Variante ist: „this is a viable solution for most scenarios we still would like to support development strategies in which adding as much information as possible in the initial UML model is the preferred option“ [Ro16, S. 197]. Aus diesem Grund hat er ein neues UML-Profil (RAML-Profil) entworfen, das die Eigenheiten von RAML auf Modellierungsebene abbildet. Die Ausdrucksmächtigkeit der Modellierung im Vergleich zur RAML-Spezifikation ist allerdings begrenzt: „there are valid RAML documents that cannot be represented by the profile“ [Ro16, S. 196]. Die Wahl dieser Einschränkung begründet der Autor mit dem Risiko, ansonsten zu viele Details zu modellieren, was zu komplizierten und schwer verständlichen Modellen führt. Abschließend erwähnt er, dass das RAML-Dokument für weitere darauf aufbauende Transformationen genutzt werden kann.

Bewertung

Der von Rossi [Ro16] vorgestellte Ansatz ermöglicht die Modellierung von REST-basierten Web-APIs mithilfe von UML und deren Überführung in RAML. Eine systematische und nachvollziehbare Ableitung der Web-API aus einem zugrundeliegenden Domänenmodell oder einem Analyseartefakt steht nicht im Fokus der Arbeit. Stattdessen wird dies vorausgesetzt. Weiter wird die Überführung des Modells in die Implementierung nicht näher betrachtet, sondern dies lediglich als Ausblick für weiterführende Arbeiten gegeben. Die Nachvollziehbarkeit ist daher nicht erfüllt (A1). Daneben werden nur die strukturellen Aspekte eines Systems betrachtet, während die verhaltensspezifischen Aspekte lediglich als mögliche zukünftige Erweiterungen angesehen werden (A2): „behavioral aspects could be added with future extensions but are not a primary concern at this point“ [Ro16, S. 195]. Beim Entwurf wird mithilfe des UML-Profils eine Ressourcenorientierung nach RMM auf Ebene 2 sichergestellt. Zu Hypermedia fehlt hingegen eine Aussage, weshalb diesbezüglich noch geprüft werden müsste, ob der Ansatz die Erweiterung um Hypermedia ermöglicht. Die Anforderung ist daher als teilweise erfüllt anzusehen (A3). Hier könnte die Arbeit von Schreibmann [Sc14] entsprechende Ansätze liefern, die einen modellgetriebenen Ansatz vorsieht, mit dem sich ohne Hintergrundwissen

eine RESTful Web-API entwerfen lässt. Das Hintergrundwissen bezieht sich bei dieser Arbeit auf die Randbedingungen, die der Architekturstil REST vorgibt (vgl. [Fi00]). Eine Unterstützung beim Entwurf bietet der Ansatz nicht, sodass der Software-Architekt entsprechendes Wissen besitzen muss, um fundierte Entwurfsentscheidungen im Hinblick auf die zu erfüllenden Qualitätsteilmerkmale der Wiederverwendbarkeit zu treffen (A4). Durch die zu berücksichtigende Randbedingung (R2) des Lösungsansatzes wurde keine Aussage zur Technologie oder zur verwendeten Plattform getroffen. Lediglich HTTP wurde als gegeben erachtet, was allerdings heutzutage von nahezu allen Technologien unterstützt wird und daher keine Einschränkung ist (A5). Auf die Überführung des Entwurfs auf die Implementierungsebene verzichtet der Lösungsansatz von Rossi, da dies nicht zu seiner Untersuchung gehört. Die Möglichkeit mittels RAML wird lediglich angedeutet (A6).

3.2.2 Pautasso et al.: A Pattern Language for RESTful Conversations

In [PI+16] werden Muster in Form einer Mustersprache (engl. pattern language) für die Kommunikation zwischen einem Service-Konsumenten und einem RESTful Web Service bzw. dessen Web-API (nachfolgend als RESTful-Konversation bezeichnet) vorgestellt, um einen systematischen Wissensaustausch zur Verbesserung der Qualität einer Web-API zu ermöglichen und deren Benutzbarkeit und Mächtigkeit zu verbessern. Durch die Nutzung von Mustern beim Entwurf von RESTful Web-APIs wollen die Autoren Software-Architekten beim Entwurf unterstützen und zudem Software-Entwicklern von Service-Konsumenten das Verständnis der Web-API erleichtern: „the designer of the API chooses the corresponding conversation patterns which form the basis for the API design. These patterns will thus help the client developer to understand API's features and to build upon them clients that will perform the complex conversation to achieve their goals“ [PI+16, S. 4]. Neben dem zu erreichenden Ziel begründen die Autoren die Notwendigkeit ihrer Arbeit mit der steigenden Anzahl von neuen, auf REST-basierenden Web-APIs, die im Zeitraum von 2005 bis 2016 registriert wurden. Weiter sehen sie eine direkte Beziehung zwischen dem Erfolg eines Web Service und dessen Web-API: „As a good user interface design is important for the success of an app, so is a good API for the success of a Web service“ [PI+16, S. 1].

Die vorgestellte Mustersprache umfasst zehn häufig wiederkehrende RESTful-Konversationen, wobei hier die Absicht der Interaktion nicht betrachtet wird. Die Absicht bezieht sich in diesem Fall auf das gewünschte Ergebnis, die der Service-Konsument durch die Interaktion erreichen möchte (vgl. Message Intent in [HW03, S.140]). Zu den betrachteten Mustern im Rahmen dieser Arbeit zählen: (M1) Einmaliges POST (engl. POST Once Exactly (POE)), (M2) Erzeugen mittels POST-PUT (engl. POST-PUT Creation), (M3) Langlebige Anfragen (engl. Long Running Request), (M4) Server-seitige Weiterleitung mittels Status Codes (engl. Server-side Redirection with Status Codes), (M5) Client-seitige Navigation durch das Folgen von Hyperlinks (engl. Client-side Navigation following Hyperlinks), (M6) Traversierung von Listen-Ressourcen (engl. Resource Collection Traversal), (M7) (Partielle) Modifikation von Ressourcen (engl. (Partial) Resource Editing), (M8) Bedingte

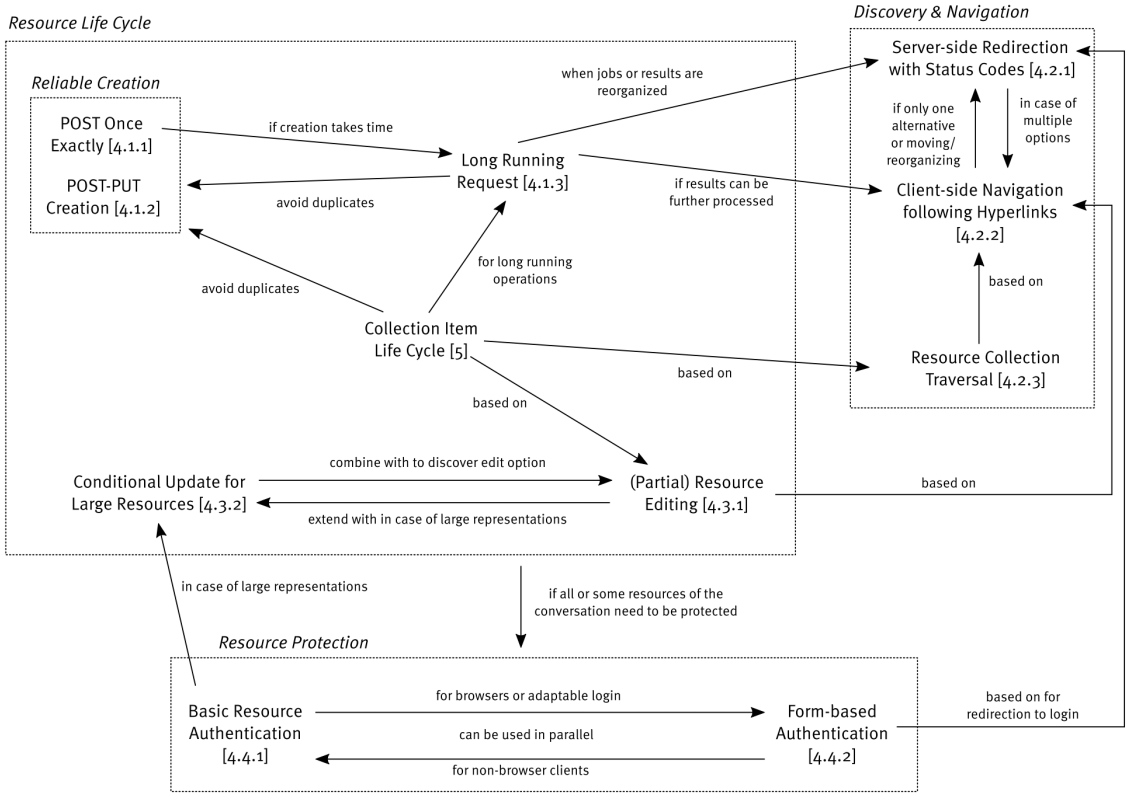


Abbildung 3.2: Übersicht über die Mustersprache für RESTful-Konversationen aus [PI+16].

Modifikation von Ressourcen (engl. Conditional Update for Large Resources), (M9) Grundlegende Ressourcen-Authentifizierung (engl. Basic Resource Authentication) und (M10) Formular-basierte Authentifizierung (engl. Form-based Authentication). Einige der aufgeführten und betrachteten Konversationen wurden aus der Vorarbeit in [HL+15] übernommen. Für die Beschreibung der einzelnen Muster wurde sich an der Arbeit von Meszaros und Doble [MD97] orientiert, die eine Mustersprache zum Formulieren von Mustern entworfen haben. Zur besseren Übersichtlichkeit gruppiert die Abbildung 3.2 die zuvor genannten Muster und setzt diese in Beziehung zueinander.

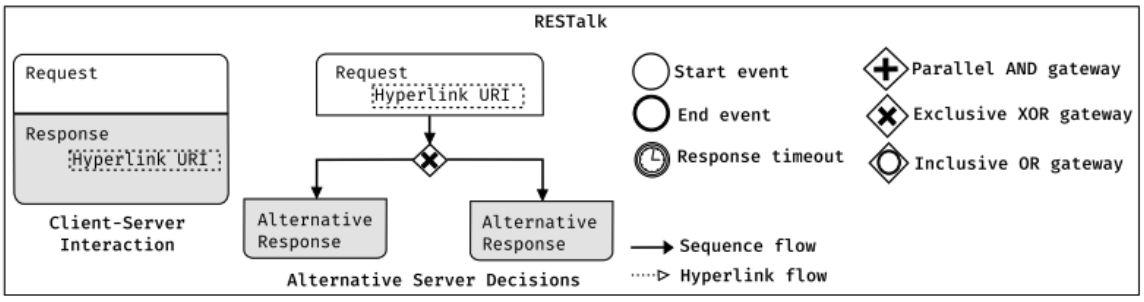


Abbildung 3.3: Übersicht über die häufigsten Modellierungselemente von RESTalk [PI+16]

Für die visuelle Modellierung der Muster wurde die Domain Specific Modeling Language (DSML) RESTalk aus [PI+15] genutzt. Diese DSML wurde auf Grundlage der Business Process Model and Notation (BPMN)-Choreographie entworfen und durch entsprechende Konstrukte zur Abbildung von REST-Konversationen angepasst. Zudem wurde die domänenspezifische Modellierungssprache mithilfe von explorativen Studien überprüft und auf Grundlage der Rückmeldungen iterativ und inkrementell im Hinblick auf die Verständlichkeit und Ausdrucksmächtigkeit verbessert [Iv16, IP+16]. Die häufigsten Modellierungselemente von RESTalk zeigt Abbildung 3.3. Im Ausblick versprechen die Autoren, RESTalk um die Abbildung von Konversationen mit mehreren Teilnehmern (engl. multi-party conversations) zu erweitern.

Bewertung

Pautasso et al. stellen mit [PI+16] eine Mustersprache für RESTful-Konversationen vor, die sie neben der eigentlichen Musterbeschreibung mithilfe der DSML RESTalk zur besseren Verständlichkeit modelliert haben. Es fehlt allerdings ein systematisches und nachvollziehbares Vorgehen zum Entwurf eines Microservice mithilfe der vorgestellten Muster (A1). Die Verwendung eines RESTful Web Service beruht auf einer Konversation zwischen Service-Konsument und Web Service, die durch die Muster beschrieben wird. Die strukturellen und verhaltensspezifischen Aspekte werden nur am Rande aufgeführt, sofern sie für das Verständnis des Musters vonnöten sind. Die Anforderung (A2) wird daher als nicht erfüllt angesehen. Bei dem Entwurf der Mustersprache wurde neben der Ressourcenorientierung auch Hypermedia berücksichtigt, sodass diese Anforderung (A3) erfüllt ist. Im Hinblick auf den Entwurf der Web-API, die letztlich die Implementierung leiten soll, können Muster die Software-Architekten unterstützen, indem neben der adressierten Problemdomäne die entsprechende Lösung sowie deren Konsequenzen und ebenso etwaige Lösungsalternativen präsentiert werden. Allerdings wird mit den Mustern hauptsächlich die Benutzbarkeit und Mächtigkeit einer Web-API betrachtet, nicht jedoch die Auffindbarkeit und Interoperabilität (A4). Die präsentierten Muster enthalten keine Angaben zu einer Plattform oder einer spezifischen Implementierungstechnologie, sodass diese Entscheidung von dem Entwicklungsteam selbstständig getroffen werden kann (A5). Im Hinblick auf die Überführung des Entwurfs auf die Implementierungsebene wird keinerlei Aussage getroffen, da dies nicht im Fokus dieser Arbeit steht (A6).

3.2.3 Ed-Douibi et al.: EMF-REST: Generation of RESTful APIs from Models

In [EdI+16] wird ein modellgetriebener Ansatz mit dem Namen *EMF-REST* basierend auf dem Eclipse Modeling Framework (EMF) zur Generierung von RESTful Web Services präsentiert. EMF ist ein Framework für die modellgetriebene Software-Entwicklung mit Ecore, einer Meta-Meta-Modell-Implementierung zur Modellierung von Meta-Modellen, als zentralem Baustein [SV+06]. Der Ansatz verfolgt das Ziel Web-APIs mittels sogenannter EMF-Datenmodelle unter Berücksichtigung der Randbedingungen von REST [Fi00] sowie bekannter Bibliotheken und Standards zu generieren.

Die Autoren begründen ihre Arbeit mit steigendem Bedarf an RESTful Web Services sowie dem fehlenden Standard für den Entwurf derartiger Web Services. Letzteres führt auch dazu, dass der Entwurf eine Herausforderung beim Entwicklungsprozess ist: „developing high-quality REST APIs for non-trivial applications may become a hard and time-consuming task“ [EdI+16, S. 1446].

Für *EMF-REST* wurden zunächst die Prinzipien von EMF und REST mittels einer Zuweisungsvorschrift (engl. mapping) in Einklang gebracht. So wurden bspw. die URIs von EMF zur Adressierung von Modellen in einen Uniform Resource Locator (URL) oder das XML Metadata Interchange (XMI)-Format für die EMF-Modelle in JSON und XML überführt. Anschließend wurden Validierungsmöglichkeiten der zu generierenden Web-API vorgestellt, bei der mittels Object Constraint Language (OCL) entsprechende Invarianten auf Modellebene definiert wurden. Außerdem erörtert der Ansatz Sicherheitsaspekte wie Verschlüsselung, Authentifizierung und Autorisierung. Eine Zuweisungsvorschrift für die Validierungsmöglichkeiten und Sicherheitsaspekte fehlt allerdings.

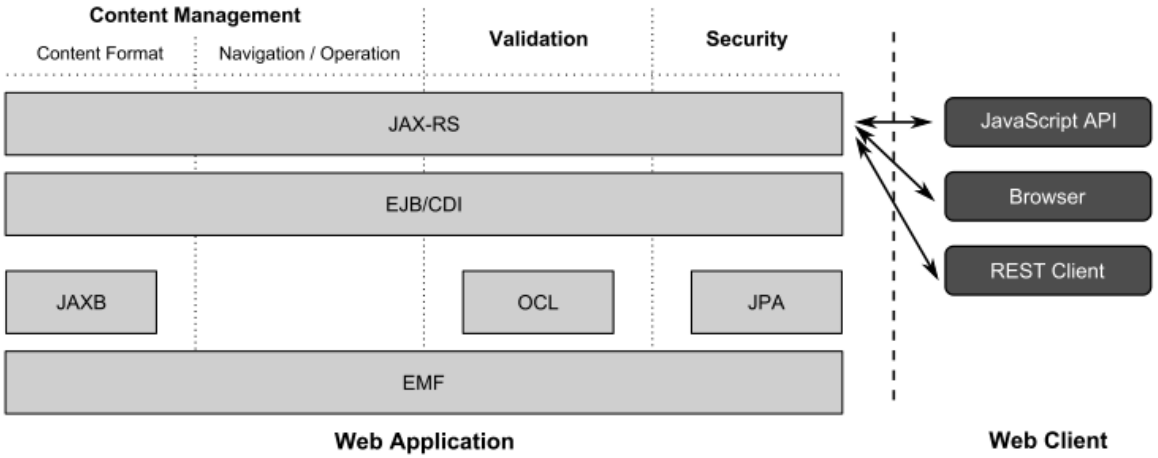


Abbildung 3.4: Architektur eines generierten Service mittels *EMF-REST* [EdI+16]

Nachdem zuerst die grundlegenden Konzepte des Ansatzes eingeführt wurden, stellt der nächste Abschnitt die Architektur einer mittels *EMF-REST* erzeugten Anwendung vor (siehe Abbildung 3.4). Die Architektur besteht aus mehreren Schichten, die jeweils unterschiedliche Funktionalitäten bedienen. Auf der untersten Ebene befindet sich das EMF als Grundlage für die notwendige Modellierung der EMF-Datenmodelle schafft. Mit Java Architecture for XML Binding (JAXB) wird auf der nächsten Ebene das Marshalling von XML/JSON in Java und umgekehrt geschaffen, während OCL für die Validierung der Modelle und Java Persistence API (JPA) für die Speicherung der Sicherheitsaspekte genutzt wird. Darauf aufbauend werden mit Enterprise Java Bean (EJB) die EMF-Datenmodelle geladen und über Java API for Representational State Transferg (JAX-RS) unter Verwendung der Context Dependency Injection (CDI) die Ressourcen über eine RESTful Web-API verfügbar gemacht. Neben der Generierung der eigentlichen Web-API auf Basis der EMF-Datenmodelle, sehen die Autoren die Erzeugung einer administrativen Oberfläche zur Konfiguration von Sicherheitsaspekten

und auch eine JavaScript-API vor, um die Entwicklung von Service-Konsumenten zu vereinfachen. Als Ausblick nennen sie die Entwicklung einer einfachen DSL zur Konfiguration der generierten Web-API, die Untersuchung der Sicherheit und Skalierbarkeit sowie die Vorzüge des Ansatzes durch die Verwendung von client-seitigen Modellierungsumgebungen.

Bewertung

Die Autoren stellen in ihrer Arbeit einen modellgetriebenen Ansatz zur Generierung von RESTful Web-APIs auf Grundlage eines Datenmodells vor. Die Nachvollziehbarkeit (A1) kann durch den durchgängigen Ansatz beginnend vom Datenmodell bis hin zur Generierung des RESTful Web Service als erfüllt angesehen werden. So wurden für die einzelnen Modelltransformationen entsprechende Zuweisungsvorschriften definiert, um eine systematische und nachvollziehbare Ableitung zu ermöglichen. Der Ansatz fokussiert allerdings ausschließlich die strukturellen Aspekte und geht dabei nicht auf das Verhalten ein, das sich bspw. aus einem abzubildenden Geschäftsausschnitts zwangsweise ergibt (A2). Weiter wurde der Ansatz unter Berücksichtigung der Randbedingungen von REST erstellt, sodass es in Bezug auf Hypermedia keine Einschränkungen gibt (A3). Für die Unterstützung bei Entwurfsentscheidungen liefert dieser Ansatz keine Hinweise (A4). So werden zwar bewährte Methoden nach [Mu12] berücksichtigt, deren Auswirkung oder die Entscheidung auf Anwendung findet jedoch keine Betrachtung. Stattdessen wird angenommen, dass alle bewährten Methoden den resultierenden Web Service bzw. die Web-API verbessern. Obwohl die präsentierte Software-Architektur (siehe Abbildung 3.4) auf EMF als Fundament setzt, kann die Technologieunabhängigkeit durch die vermehrte Verwendung von Java-spezifischen Technologien als nicht erfüllt angesehen werden. Anders sieht dies bei der Plattformunabhängigkeit aus, da hier keinerlei Angaben getroffen werden. Die Anforderung (A5) ist daher als teilweise erfüllt anzusehen. Eine Überführung des Entwurfs auf die Implementierungsebene ist durch den präsentierten Ansatz automatisch gegeben, was allerdings darauf beruht, dass eine komplexe Domänenlogik nicht vorliegt, sonderlich lediglich auf CRUD-Operationen gesetzt wird. Dennoch ist die Anforderung (A6) als erfüllt zu bewerten.

3.2.4 Haupt et al.: A Conversation based Approach for Modeling REST APIs

In [HL+15] wird ein Ansatz zur Modellierung von REST-basierten Web-APIs mittels Konversationen und ein darauf basierendes Meta-Modell präsentiert. Diese Konversationen abstrahieren die Interaktion zwischen Service-Konsument und Service und bilden diese auf Entwurfsebene ab, was gleichzeitig die Grundlage für die Mustersprache in [PI+16] (siehe Abschnitt 3.2.2) darstellt. Eine Konversation ist in deren Arbeit als eine zustandslose Interaktion zwischen Service-Konsument und den veräußerten Ressourcen definiert, die ein REST-basierender Service über HTTP bereitstellt. Deshalb wird dies auch als RESTful-Konversation bezeichnet. Die vorgestellten Konversationen sind eine Untermenge aus [PI+16] und wurden im Gegensatz dazu mit einem UML-Sequenzdiagramm und nicht in RESTalk modelliert.

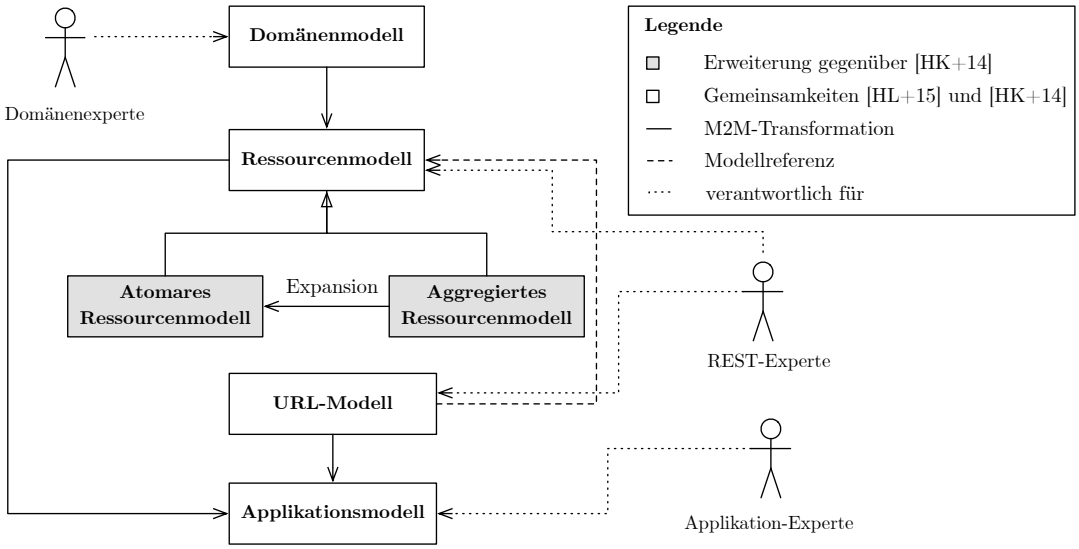


Abbildung 3.5: Meta-Modell für den modellgetriebenen Ansatz aus [HK+14, HL+15]

Für den Entwurf und die Realisierung von RESTful Web-APIs mit Konversationen erweitern die Autoren einen modellgetriebenen Ansatz in [HK+14] zur Einhaltung der Randbedingungen von REST. Dieser besteht aus mehreren Modellen (siehe Abbildung 3.5) zusammen, die nun nachfolgend beschrieben werden:

1. **Domänenmodell:**
   Das resultierende Domänenmodell bildet die Applikationsdomäne ab und dient zur Definition der Web-API. Weiter beinhaltet dieses Modell keinerlei Bezug zum Architekturstil REST. Allerdings kann das Domänenmodell auch als optional für die nachfolgende Modellierung angesehen werden: „Alternatively, a service designer can also start modeling the resource model without providing a domain model“ [HL+15, S. 5]. Stattdessen wird lediglich die Notwendigkeit von Anforderungen für die abzubildende Applikationsdomäne vorausgesetzt, die anschließend manuell in das nächste Modell bzw. das Ressourcenmodell überführt werden müssen. Diese Form der Überführung wird allerdings nicht näher beschrieben.

2. **Ressourcenmodell:**
   Das Ressourcenmodell ist das Domänenmodell, das in einen ressourcenorientierten Stil überführt wurde. Allerdings ist dies nicht immer trivial und vollständig, da die Meta-Modelle nicht zwangsläufig kompatibel zueinander sein müssen. So verfolgt das Domänenmodell meist einen objektorientierten Stil [Ev03]. Dieser Umstand wird auch als „impedance mismatch“ bezeichnet [IB+09, HK+14]. Weiter wird beim Ressourcenmodell zwischen einem atomaren und einem aggregierten Ressourcenmodell (engl. atomic resource model bzw. composite resource model) unterschieden, die nachfolgend erläutert werden:

a) **Atomares Ressourcenmodell:**
Ein atomares Ressourcenmodell erlaubt die Beschreibung einer RESTful Web-API mithilfe der grundlegenden Bestandteile von REST, wie bspw. der Ressourcen oder der Repräsentationen. Aus diesem Modell lassen sich entsprechende formale Service-Beschreibungen ableiten, wie bspw. WADL [W3C-WADL]. Im Gegensatz zu [HK+14] wurde das atomare Ressourcenmodell im Rahmen dieser Arbeit um ein interaktionszentriertes Meta-Modell (engl. interaction centric meta model) erweitert, bei dem den modellierten Ressourcen die zu unterstützenden Interaktionen hinzugefügt wurden. Dieser Ansatz führt allerdings schnell zu unverständlichen und komplexen Modellen, was als Quelle für mögliche Modellierungsfehler erkannt wurde. Deshalb musste der Modellierungsansatz auf einer höheren Abstraktionsebene stattfinden, was schließlich ein konversations-zentriertes bzw. aggregiertes Meta-Modell ergab.

b) **Aggregiertes Ressourcenmodell:**
Ein aggregiertes Ressourcenmodell erlaubt eine grob-granulare Modellierung als das atomarer Ressourcenmodell, bei dem dessen Bestandteile wiederverwendet werden, um neue Modellierungskonstrukte zu formen: „[it] allow[s] aggregating (composing) multiple elements of the atomic resource model into new and coarser grained modeling constructs to enable modeling on a higher level of abstraction“ [HL+15, S. 174]. Das im vorigen Listenpunkt (a) erwähnte, konversations-zentrierte Meta-Modell ergänzt also logisch das aggregierte Ressourcenmodell [HK+14]. Durch Minimierung des Risikos für Modellierungsfehler kann sich ein Software-Architekt darauf konzentrieren, was eine RESTful Web-API unterstützen soll, und nicht, wie diese zu realisieren ist. Die resultierenden Modelle auf höherem Abstraktionsniveau lassen sich schließlich wieder in Modelle mit niedrigeren Abstraktionsniveaus transformieren. Diese Form der Transformation wird in der Arbeit als „Expansion“ bezeichnet, bei der ein Modellierungselement durch ein Set an zusammenhängenden Modellierungselementen ersetzt wird. Gleichzeitig definieren die Autoren für die Expansion bestimmte Vorbedingungen und mögliche Ausprägungen, da eine Expansion nicht in allen Nutzungsszenarien die gleiche Anwendung findet. Das aggregierte Ressourcenmodell lässt sich als Erweiterung des atomaren Ressourcenmodells auffassen, was die Autoren so verdeutlichen: „Parts of a REST API can be modeled using conversations whenever applicable, but it is always possible to resort to the basic elements of the interaction-based metamodel“ [HL+15, S. 174].

3. **URL-Modell:**
Im Sinne der Randbedingungen von REST muss jede Ressource über eine eindeutige URL verfügen. Während das Ressourcenmodell die Beziehungen zwischen den einzelnen Ressourcen beschreibt, ergänzt das URL-Modell diese um eine URL. Dies ist nach Ansicht der Autoren ein wichtiger Unterschied zu anderen Meta-Modell-Ansätzen für RESTful Web-APIs.

4. **Applikationsmodell:**
   Die bisher betrachteten Modelle lassen sich aus Sicht der modellgetriebenen Software-Entwicklung jeweils als Platform Independent Model (PIM) bezeichnen, da sie keine Angaben zur späteren Implementierung auf einer Plattform machen [SV+06, HK+14]. Das Applikationsmodell ist hingegen ein sogenanntes Platform Specific Model (PSM) mit den plattformspezifischen Aspekten zur Realisierung ein oder mehrerer PIMs für eine konkrete Plattform beinhaltet. Dabei wird das Ziel verfolgt, durch eine Modell-zu-Text-Transformation eine entsprechende Implementierung aus einem oder mehreren PSMs zu generieren. Obwohl eine vollautomatische Transformation wünschenswert wäre, ist dies im Allgemeinen nicht immer möglich: „there are parts of the application where developers have to manually add specific application logic" [HK+14, S. 133]. Die Herausforderungen bei zukünftigen Modell-Anpassungen und die daraus resultierende notwendige Zusammenführung der manuellen Anpassungen mit den „neuen" Modellen anders als von [Ro16, vgl. S. 196f.] nicht näher betrachtet, da das nicht im Fokus dieser Arbeit ist.

Neben den verschiedenen Modellen wird in [HK+14] zusätzlich ein Rollenkonzept mit drei Rollen eingeführt: (1) Domänen-Experte, (2) REST-Experte und (3) Applikation-Experte. Jede Rolle ist für ein oder mehrere Modelle und/oder weitere Artefakte verantwortlich. So trägt bspw. für das Domänenmodell allein der Domänen-Experte die Verantwortung.

Bewertung

Die Arbeit von Haupt et al. [HL+15] (basierend auf der Vorarbeit [HK+14]) präsentiert einen, auf Konversationen basierenden, Ansatz zur Modellierung von RESTful Web-APIs. Obwohl die Arbeiten ein ganzheitliches Vorgehen präsentieren – beginnend mit dem Domänenmodell über das Ressourcenmodell bis hin zum Applikationsmodell, der Grundlage für die Implementierung –, kann die Nachvollziehbarkeit (A1) nur als teilweise erfüllt angesehen werden. So ist bspw. unklar, wie systematisch aus einem Domänenmodell die Ressourcen samt ihren Repräsentationen und den benötigten Operationen abgeleitet werden können. Eine entsprechende Zuweisungsvorschrift fehlt. Während in [HK+14] lediglich die strukturellen Aspekte betrachtet, enthält [HL+15] eine Erweiterung um die verhaltensspezifischen Aspekte, die durch die Interaktion zwischen Service-Konsument und Service entstehen. Allerdings wird so lediglich das Verhalten der technischen Schnittstelle, nicht jedoch das sich aus dem zugrundeliegenden Domänenmodell ergebende betrachtet. Die Anforderung (A2) ist daher nur teilweise erfüllt, da lediglich die strukturellen Aspekte betrachtet werden. Weiter berücksichtigt der Ansatz die Einhaltung aller Randbedingungen von REST. Dabei zeigt insbesondere das URL-Modell die Betrachtung von Hypermedia (A3). Im Hinblick auf die Entscheidungsunterstützung beim Entwurf liefern die Autoren eine entsprechende Unterstützung mithilfe von Konversationen. So wird bspw. beschrieben, wie verteilte atomare Transaktionen durchgeführt werden oder wie der Zugriff auf Ressourcen erfolgen sollte. Allerdings sind die Auswirkungen bei Anwendung dieser

Konversationen unklar, sodass ein Software-Architekt entsprechendes Wissen über die getroffenen Entscheidungen besitzen muss. Die Anforderung (A4) ist daher teilweise erfüllt. Durch das Verfolgen eines modellgetriebenen Ansatzes mit PIMs (Ressourcenmodell, URL-Modell) und PSM (Applikationsmodell) ist der präsentierte Ansatz grundsätzlich als technologie- und plattformunabhängig einzustufen (A5). In der gesamten Arbeit findet sich kein Bezug zur Ableitung der Implementierung aus dem zugrundeliegenden Entwurf, obwohl die Autoren eine manuelle Adaption des generierten Quellcodes durch Software-Entwickler für notwendig halten. Eine vollautomatisierte Generierung wie in [EdI+16] ist demnach nur eingeschränkt möglich.

3.2.5 Rathod et al.: Structural and Behavioral modeling of RESTful Web Service Interface using UML

Rathod et al. präsentieren mit [RP+13] einen Ansatz zur Modellierung der strukturellen und verhaltensspezifischen Aspekte eines RESTful Web Service bzw. dessen Schnittstelle mithilfe von UML. Die verhaltensspezifischen Aspekte charakterisieren sich laut den Autoren durch das Verhalten der Ressourcen auf eingehende Anfragen von Service-Konsumenten sowie dem daraus resultierenden neuen Anwendungszustand. Als strukturelle Aspekte werden hingegen die Grundelemente bezeichnet, aus denen ein RESTful Web Service besteht wie etwa die Ressourcen und deren Operationen.

Die Modellierung der strukturellen Aspekte eines RESTful Web Service basiert auf einem UML-Klassendiagramm. Dabei werden anders als bei Rossi [Ro16] acht verschiedene Arten von Ressourcen unterschieden (die Arbeit erwähnt sieben Arten, was allerdings an einer falschen Aufzählung liegt [RP+13, vgl. S. 29]): (1) Primär-Ressource, (2) Sub-Ressource, (3) Listen-Ressource, (4) Filter-Ressource, (5) Projektion-Ressource, (6) Aggregation-Ressource, (7) Paginierung-Ressource und (8) Aktivität-Ressource. Für die Zuweisung von Ressourcen zu Klassen wurden Richtlinien definiert, ohne allerdings auf die zuvor aufgeführten Ressourcenarten einzugehen.

Die Modellierung der verhaltensspezifischen Aspekte erfolgt nach dem Ansatz von Rathod et al. mithilfe eines Zustandsdiagramms, das allerdings nur die HTTP-Methoden GET, PUT, POST, DELETE betrachtet. Eine Begründung für die Auswahl dieses Diagrammtyps fehlt. Allerdings wird in einer früheren Arbeit die Wahl eines Zustandsdiagramms zur Modellierung des Verhaltens so begründet: „[it] is suitable for representing the behavior of a web service as it provides interface specifications with information on conditions under which methods can be invoked and the expected output from them“ [PR11, S. 3]. Auch die Publikation von Rauf et al. [RI+08] mit Fokus auf der Komposition von Services zeigt, dass Zustands- oder Aktivitätsdiagramme vermehrt für die verhaltensspezifische Modellierung verwendet werden. Laut Rathod et al. müssen aus der Verhaltensmodellierung eines RESTful Web Service drei Bestandteile hervorgehen bzw. ableitbar sein: (1) die korrekte Reihenfolge der Operation für eine erfolgreiche Ausführung, (2) die Bedingungen, unter der die Operationen

aufgerufen werden können, und (3) das erwartete Ergebnis nach Durchführung der Operationen. Deshalb wurde das Verhalten eines RESTful Web Service als traversierender Graph von Zustandsknoten modelliert, bei dem jeder Zustandsknoten mit Transaktionen verknüpft ist, die durch Anfragen vom Service-Konsumenten ausgelöst und bei denen vor der eigentlichen Anfragenverarbeitung etwaige Vorbedingungen (engl. pre-conditions) überprüft werden. Diese Vorbedingungen werden im Rahmen der Arbeit auch als *Guard Clause* bezeichnet. Das ist nach Beck ein Muster zum Ausdrücken von lokalen Ausnahmen durch eine frühe Rückmeldung [Be07, S. 63]. Die Verarbeitung der Anfrage ist genau dann erfolgreich, wenn das Ergebnis die definierten Nachbedingungen (engl. post-conditions) erfüllt. Die Auswertung der Vor- und Nachbedingungen erfolgt anhand einer sogenannten Invariante, die einen booleschen Ausdruck repräsentiert.

Bewertung

Der Ansatz von Rathod et al. ermöglicht die Modellierung einer RESTful Web-API, bei der sowohl die strukturellen als auch die verhaltensspezifischen Aspekte berücksichtigt werden. Eine systematische und nachvollziehbare Ableitung der Web-API aus einem Domänenmodell oder einem Analyseartefakt ist nur teilweise gegeben, da lediglich ein beispielhaftes Szenario zur Demonstration der Modellierung herangezogen wird. Auch die Modellierung der strukturellen und verhaltensspezifischen Aspekte weißt Lücken auf. So ist bspw. unklar, wie sich die verschiedenen aufgeführten Ressourcenarten korrekt abbilden ließen oder wie die Vor- und Nachbedingungen systematisch abzuleiten und zu formalisieren wären. Eine nachvollziehbare Überführung auf die Implementierungsebene wurde in dem Ansatz ebenfalls nur oberflächlich beschrieben (A1). Die zweite Anforderung (A2) ist hingegen erfüllt, da sowohl die strukturellen als auch die verhaltensspezifischen Aspekte betrachtet werden. Die Arbeit betrachtet einen RESTful Web Service, sodass zumindest die Ressourcenorientierung berücksichtigt wird. Ein direkter Bezug zu Hypermedia fehlt in der Arbeit allerdings, weshalb dahingehend eine Überprüfung erfolgen müsste. Die Anforderung (A3) ist daher nur teilweise erfüllt. Auf eine Unterstützung beim Entwurf wird in dem präsentierten Ansatz nicht eingegangen. Also muss der Software-Architekt entsprechendes Wissen haben, um fundierte Entscheidungen zur Wiederverwendbarkeit treffen zu können (A4). Die Agnostizität im Hinblick auf die Wahl einer Plattform und einer Technologie kann als gegeben angesehen werden, da diesbezüglich keine Vorgaben gemacht werden (A5). Die letzte Anforderung (A6) wird in der Publikation nicht näher erörtert. Allerdings findet sich ein entsprechender Bezug in der Vorarbeit. Dort heißt es, die Modelle könnten genutzt werden, um Testfälle zu generieren, was zumindest ansatzweise eine Verknüpfung zur Implementierung erahnen lässt [RI+08]. Nichtsdestominder lässt sich diese Anforderung nicht als erfüllt betrachten.

3.2.6 Schreier: Modeling RESTful Applications

Schreier präsentiert ein Meta-Modell zur Modellierung von RESTful Web Services als Folge der fehlenden Unterstützung in den frühen Phasen des Software-Entwicklungsprozesses – insbesondere

in der Analyse- und Entwurfsphase [Sc11]. Mithilfe dieses Meta-Modells möchte der Autor eine Terminologie für den Entwurf und die Implementierung von RESTful Web Services schaffen, die gleichzeitig als Basis für weiterführende Lösungen dienen soll. Daneben hilft das Meta-Modell zur Abgrenzung von bestehenden Ansätzen, da: „Most publications use their own informal model, tailored to their needs, for describing the solution. A metamodel for designing and implementing RESTful [web services] is still missing.“ [Sc11, S. 2]. Das Meta-Modell ist in strukturelle und verhaltensspezifische Aspekte unterteilt und wurde mithilfe von Ecore wegen dessen Einfachheit und der verfügbaren Werkzeugunterstützung entworfen. Die Anwendung des Meta-Modells zur Modellierung von RESTful Web Services wird anhand einer beispielhaften Anwendung illustriert.

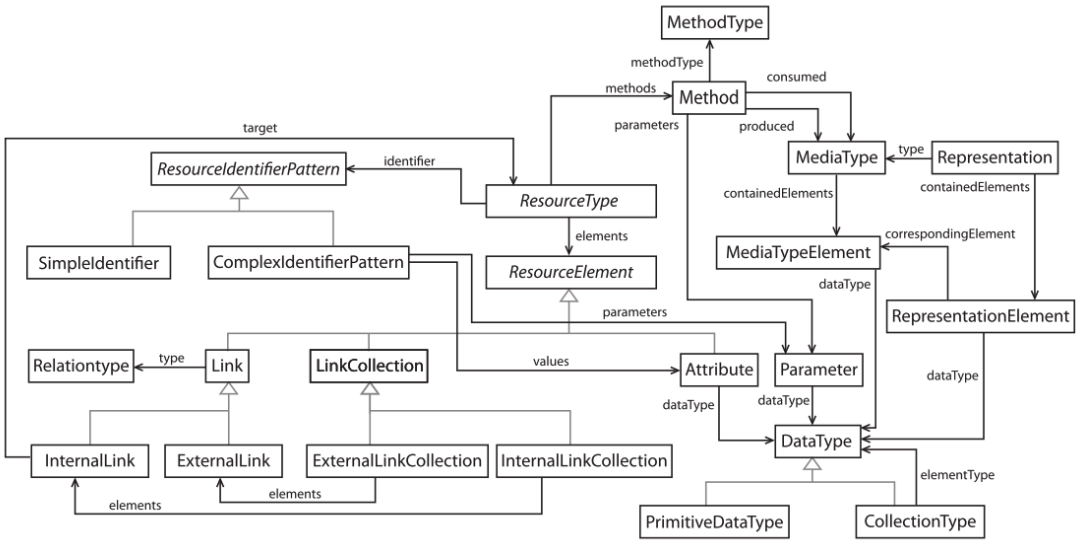


Abbildung 3.6: Meta-Modell zur Modellierung der strukturellen Aspekte eines RESTful Web Service aus [Sc11]

Für den Entwurf des Meta-Modells zur Abbildung der Struktur von RESTful Web Services wurden zunächst die strukturellen Modellierungselemente identifiziert. Dazu zählen neben Ressourcen auch Ressourcenarten zur Gruppierung von Ressourcen. Die aufgeführten Ressourcenarten decken sich mit denen aus [RP+13], wobei keine explizite Referenzierung zwischen den Arbeiten existiert. Ausgehend von den Ressourcenarten wurden weitere Modellierungselemente mittels einer rekursiven Verfeinerung ermittelt, woraus sich schließlich die Ressourcenattribute, Datentypen, Methoden, Methodentypen, Methodenparameter, Verweise, Beziehungsarten, Medientypen und Repräsentationen ergeben haben. Die beiden letzteren sind nicht Bestandteil des Meta-Modells: „The concrete appearance of a Representation or MediaType are not covered by the metamodel at this stage but could be added later on, e. g., using templates“ [Sc11, S. 3]. Das resultierende Meta-Modell für die strukturellen Aspekte ist der Abbildung 3.6 zu entnehmen.

Die Modellierung des Verhaltens eines RESTful Web Service wurde auf Grundlage eines Zustandsau-

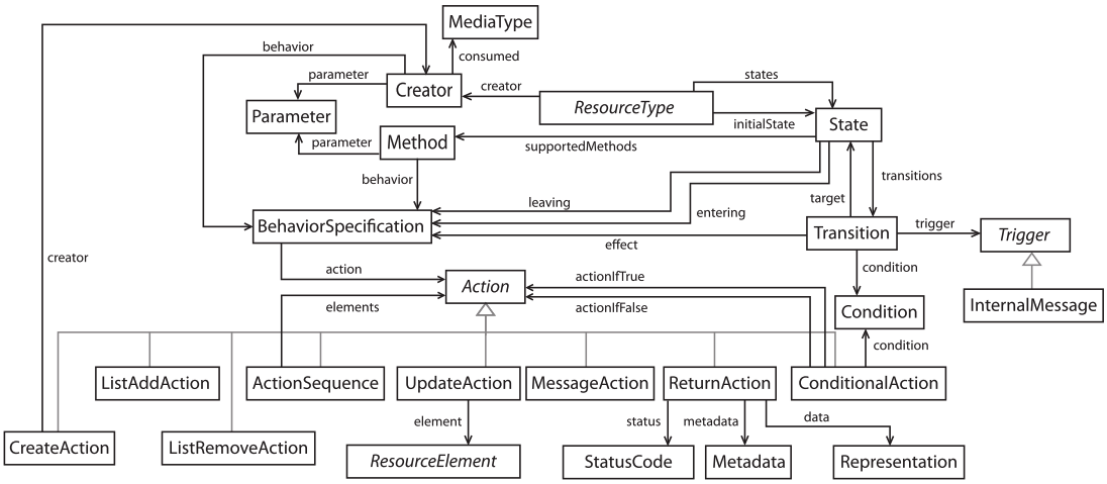


Abbildung 3.7: Erweiterung des Meta-Modells zur Modellierung der verhaltensspezifischen Aspekte eines RESTful Web Service aus [Sc11]

tomaten entworfen, um neben dem aktuellen auch mögliche Zustandsübergänge als Folge bestimmter Anfragen durch die Service-Konsumenten zu modellieren. Jede Ressourcenart kann zahlreiche Zustände definieren, wobei ein Zustand als *initial* deklariert wird. Als dieser initiale Zustand wird derjenige nach Erzeugung der jeweiligen Ressource gedeutet. Jeder modellierte Zustand kann beliebig viele ausgehende Zustandsübergänge besitzen, die wiederum einem oder mehreren Auslösern zugeordnet werden. Die Modellierung von Bedingungen ist im Meta-Modell noch nicht vorgesehen, aber für künftige Versionen geplant. Zurzeit können diese nur über entsprechende textuelle Repräsentationen ergänzt werden. Weiter ist für jede modellierte Methode eine Verhaltensspezifikation über die erlaubten Aktionen und deren Reihenfolge zu definieren. So kann bspw. die Methode *HTTP-GET* insofern spezifiziert werden, dass keine Veränderung des Ressourcenzustands erlaubt ist. Abbildung 3.7 zeigt das Meta-Modell für die verhaltensspezifischen Aspekte.

Bewertung

Der Ansatz von Schreier präsentiert ein Meta-Modell für die Modellierung von RESTful Web Services mit dem Ziel einer einheitlichen Modellierung auf hohem Abstraktionsniveau ohne eine Einschränkung hinsichtlich der zu verwendenden Technologie für die Implementierung. Die Arbeit fokussiert die formale Modellierung auf Entwurfsebene, weshalb die Ableitung aus Analyseartefakten oder die Überführung eines Modells auf die Implementierungsebene nicht betrachtet wird (A1). Mit diesem Meta-Modell lassen sich sowohl die strukturellen als auch die verhaltensspezifischen Aspekte eines RESTful Web Service modellieren, aber lediglich aus technischer Sicht. Deshalb ist die Anforderung (A2) als nicht erfüllt zu werten. Inwieweit die darin enthaltenen Modellierungselemente ausreichen, um alle möglichen Szenarien abzubilden, wäre allerdings noch zu prüfen: „Testing the metamodel with various scenarios in addition to using the vocabulary defined by it are suggested as validation

techniques [...]. The lessons learned thereby will help to improve the metamodel incrementally“ [Sc11, S. 7]. Die Randbedingungen nach RMM auf Ebene 2 (A3) werden berücksichtigt, da das Meta-Modell die notwendigen Modellierungselemente hat und zusätzlich den Hypermedia-Aspekt betrachtet. Bei der Modellierung des RESTful Web Service findet keine Unterstützung für den Software-Architekten statt, sodass dieser entsprechendes Wissen haben muss, um den Entwurf bspw. im Hinblick auf die evolutionäre Stabilität zu entwerfen (A4). Zudem wird die Technologie- und Plattformunabhängigkeit (A5) erfüllt, da keine Vorgaben in Bezug auf die zu verwendende Technologie und Plattform gemacht werden. Zur Überprüfung des Entwurfs auf der Implementierungsebene sind keinerlei Angaben zu finden (A6).

3.2.7 Laitkorpi et al.: Towards a Model-Driven Process for Designing RESTful Web Services

In [LS+09] wird ein modellgetriebener Prozess für den Entwurf von RESTful Web Services vorgestellt, auf den viele Publikationen verweisen [Sc11, SS12, RP+13, HK+14]. Laut den Autoren gab es zum Zeitpunkt des Verfassens dieser Arbeit keinen vergleichbaren Ansatz: „at the time of the writing no systematic approach for model-driven development of ReSTful Web services has been reported“ [LS+09, S. 179].

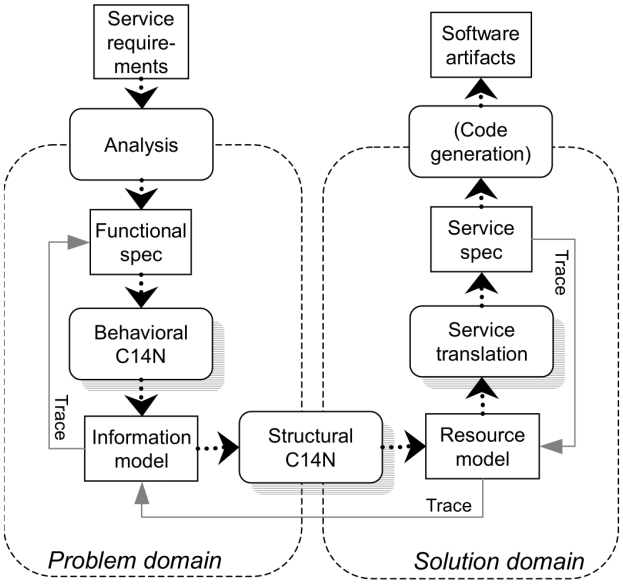
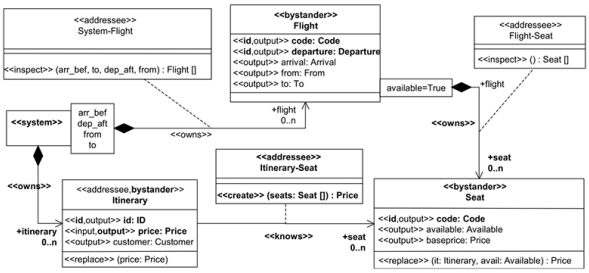


Abbildung 3.8: Gesamtprozess für den Entwurf von RESTful Web Services aus [LS+09]

Der in der Arbeit dargelegte Prozess besteht aus mehreren Stufen mit Modelltransformationen und basiert auf Erfahrungen im Nokia Research Center. Diese Erfahrungen beziehen sich hier auf die Ableitung von RESTful Web-APIs aus funktionalen Spezifikationen. Eine funktionale Spezifikation

INTERACTION 7+9: addSeats	
Concern	**Answer encoded as model elements**
listener	`Itinerary-Seat` association class representing `Itinerary` $\xrightarrow{\ll knows \gg}$ $*$ `Seat`
bystanders	≪system≫, `Itinerary`, `Seat`, `Flight`
relationships	≪system≫ $\xrightarrow{\ll owns \gg}$ $*$ `Itinerary`
	`Itinerary` $\xrightarrow{\ll knows \gg}$ $*$ `Seat`
	`Flight` $\xrightarrow{\ll owns \gg}$ $*$ `Seat`
qualifiers	≪id≫ attributes: `Itinerary::id`, `Seat::code` (values such as "K12"), `Flight::code` (values such as "AY204"), `Flight::departure`
intention	≪stateChange≫ (not shown in the model)
effect	≪create≫
content	≪input≫: `Seat` ([] indicates plural)
	≪input≫ mapped to concept attributes: `Seat::code`, `Seat::flight`
	≪output≫: `price`
	≪output≫ mapped to concept attributes: `Itinerary::price`; because `Itinerary` is not the same as the ≪addressee≫, this interaction must be split into two: `Itinerary-Seat::≪create≫()` and `Itinerary::≪inspect≫()`. Only the former is exemplified here.

(a) Informationsmodellfragment



(b) Informationsmodell

Abbildung 3.9: Informationsmodellfragment und Informationsmodell

beschreibt, was der RESTful Web Service bereitstellen muss und wie ein Service-Konsument vermutlich mit ihm interagieren wird. Neben der steigenden Verbreitung von REST begründen die Autoren die Notwendigkeit ihrer Arbeit mit der häufigen Abweichung existierender RESTful Web Services von den Randbedingungen von REST. Vor allem erkennen sie Abweichungen in der Entwurfsphase: „We claim that major deviations from the ReST principles arise during the service design when required functionality is mapped to concrete service API elements“ [LS+09, S. 173]. Als Grund hierfür sehen sie die Beschreibungsform der funktionalen Spezifikation, die sich eher für einen objektorientierten Entwurf mit einer methodenorientierten anstelle einer ressourcenorientierten Schnittstelle eignet: „impedance mismatch that requires developers to gradually transition between these two mindsets “[LS+09, S. 173]. Der vorgestellte mehrstufige Prozess wird auch als API-Design-Prozess bezeichnet und besteht aus folgenden fünf Schritten (siehe Abbildung 3.8).

1. **Analyse:**
   Die Arbeit basiert auf der Prämisse, dass alle Anforderungen und Anwendungsfälle bereits in die Form einer funktionalen Spezifikation überführt wurden. Vollständigkeit der funktionalen Spezifikation ist entscheidend für das Ergebnis des Gesamtprozesses: „Regarding the quality of the overall process outcome, it is important that the functional specification explicitly covers all the functionality“ [LS+09, S. 175]. Auf Basis der funktionalen Spezifikation werden die Interaktionen zwischen Service und Service-Konsument als UML-Sequenzdiagramm dargestellt.

2. **Verhalten-Normalisierung (engl. behavioral canonicalization):**
   Im Zuge der Verhalten-Normalisierung wird jede abgebildete Interaktion als Manipulierung eines Zustands betrachtet, woraus sich implizit die zugrundeliegenden Zustände und deren

Zustandsinformationen ergeben. Für deren Formalisierung wurde ein UML-Klassendiagramm verwendet. Die *Speech Act*-Theorie nach Bergholtz et al. [BJ+04] wurde verwendet, um die abgeleiteten Interaktionen und also auch das UML-Klassendiagramm mit weiteren Informationen anzureichern. Hierfür wurde ein darauf basierendes UML-Profil entwickelt, damit das UML-Klassendiagramm u. a. erkennen lässt, welche Zustände bei einer Interaktion involviert sind. Die Anreicherung wurde zunächst als ein sogenanntes Informationsmodellfragment festgehalten und später in ein sogenanntes Informationsmodell überführt (siehe Abbildung 3.9). Letzteres dient als Ausgangsartefakt für die nächste Stufe und beinhaltet damit alle Zustände, Zustandsübergänge und angereicherte Interaktionen.

3. **Struktur-Normalisierung (engl. structural canonicaliation):**
   Mit der Struktur-Normalisierung werden die Zustände und Zustandsinformationen aus dem im Vorfeld erstellten Informationsmodells extrahiert und durch ein weiteres UML-Profil erweitert. Das hierfür entwickelte UML-Profil umfasst die Eigentümerschaft von Ressourcen, deren Hierarchie sowie die verfügbaren Operationen. Die Anwendung des UML-Profils bildet damit eine logische Ergänzung, wie folgende Aussage zeigt: „Once the structural canonicalization is completed, the resulting resource model represents an externally referenceable layer on top of the information model“ [LS+09, S. 178]. Das Ergebnis dieser Phase ist ein sogenanntes Ressourcenmodell.

4. **Service-Übersetzung (engl. service translation):**
   In der Stufe der Service-Übersetzung wird dieses Ressourcenmodell genutzt und mit der Ziel-Architektur verbunden, um ein Artefakt zu erzeugen, das sich im Hinblick auf die bevorstehende Implementierung als Spezifikation nutzen lässt. Als Spezifikationsformat wurde im Kontext dieser Arbeit beispielhaft auf WADL gesetzt.

5. **Quellcode-Generierung:**
   Der letzte Prozessschritt ist die Generierung der Implementierung auf Grundlage der zuvor abgeleiteten Spezifikation. Auf diesen Prozessschritt geht die Arbeit allerdings nicht näher ein.

Abschließend wird der entwickelte Ansatz im Hinblick auf die zu erwartenden Vorzüge, die Evaluationskriterien und die künftigen Bestrebungen zu seiner Weiterentwicklung erörtert.

Bewertung

Der modellgetriebene Ansatz von Laitkorpi et al. [LS+09] beschreibt einen mehrstufigen Prozess zum Entwurf von RESTful Web Services beginnend mit den Service-Anforderungen in Form einer funktionalen Spezifikation bis hin zu einer Service-Spezifikation, die als Grundlage für die Implementierung dient. Die Nachvollziehbarkeit ist trotz der existierenden Zuweisungsvorschriften zwischen den einzelnen Modellen nur teilweise erfüllt, da bspw. unklar ist, wie sich die verhaltensspezifischen Aspekte aus

der zugrundeliegenden funktionalen Spezifikation systematisch ableiten lassen. Deshalb wollen die Autoren ein Werkzeug entwickeln: „We plan to use a supplementary questionnaire as a tool for guiding the API designer to construct a profile-compliant informationmodel“ [LS+09, S. 175] (A1). Daneben muss geprüft werden, inwieweit sich die vorgestellten Konzepte mit einem zugrundeliegenden Domänenmodell vereinen lassen. In Bezug auf die strukturellen und verhaltensspezifischen Aspekte (A2) wird die Anforderung erfüllt, da dies durch die Verhalten- und die Struktur-Normalisierung abgedeckt wird. Zur Unterstützung von Hypermedia im Kontext von REST treffen die Autoren keine explizite Aussage. Allerdings lässt der Ansatz keinerlei Einschränkungen diesbezüglich erkennen, weshalb diese Anforderung als erfüllt angesehen werden kann (A3). Eine Unterstützung bei Entwurfsentscheidungen findet in der Arbeit lediglich im Rahmen der Reflexion und Ausblicks des Ansatzes statt. Die Anforderung (A4) ist daher nicht erfüllt. Der Prozess ist technologie- und plattformunabhängig, da der Entwurf diesbezüglich keine weiteren Angaben enthält (A5). Auch das Entwicklungsteam kann die Technologie zur Implementierung durch die Entscheidung für das technologie-agnostische Spezifikationsformat WADL frei wählen. Für die Implementierung auf Grundlage des Entwurfs bzw. der Service-Spezifikation fehlt eine konkrete Beschreibung, weshalb die Anforderung (A6) als nicht erfüllt zu bewerten ist.

3.3 Zusammenfassung und resultierender Handlungsbedarf

Auf Grundlage dieser Auswertung von bestehenden Arbeiten (siehe Abschnitt 3.2) fasst die Tabelle 3.2 die Ergebnisse zusammen. Die Tabelle ist wie folgt zu interpretieren: Wird eine Anforderung vollständig von einer Arbeit erfüllt, so wird das mit ● angezeigt. Eine teilweise Erfüllung der Anforderung wird dagegen mit ◖ vermerkt. Falls eine Anforderung nicht erfüllt bzw. nicht berücksichtigt wird, erfolgt eine Kennzeichnung mit ○. Zu beachten gilt, dass *DDD* im Stand der Forschung nicht explizit betrachtet wurde, da der Fokus dieser Arbeit auf dem Entwurf ressourcenorientierter Microservices insbesondere ihrer Web-APIs basierend auf Konzepten von DDD aus [Ev03, Ve13] erfolgt (vgl. Abschnitt 2.2). Die Konzepte werden daher nur zum Zeitpunkt des Entwurfs eines ressourcenorientierten Microservice sowie dessen Überführung auf die Implementierungsebene angewandt.

Die Tabelle 3.2 zeigt, dass keine der analysierten Arbeiten die Anforderungen für einen domänengetriebenen Entwurf für ressourcenorientierte Microservices in Abschnitt 3.1.2 erfüllt. Dieses Ergebnis bildet gleichzeitig die Motivation für die eigenen Beiträge in dieser Arbeit, die sich jedoch auch auf die Konzepte der hier ausgewerteten Arbeiten stützt.

Für einen domänengetriebenen Entwurf von ressourcenorientierten Microservices ist zunächst ein nachvollziehbares Vorgehen erforderlich, um eine systematische Ableitung aus dem Domänenmodell über den Entwurf bis hin zur Implementierung des Microservice zu ermöglichen. Hierfür liefert der mehrstufige Prozess von Laitkorpi et al. [LS+09] wichtige Ansätze zur Modellierung von strukturellen und verhaltensspezifischen Aspekten der abzubildenden fachlichen Domäne. Dieser Ansatz verlangt

	A1: Nachvollziehbarkeit	A2: Betrachtung von strukturellen und verhaltensspezifischen Aspekten	A3: Optionale Verwendung des Hypermedia-Aspekts	A4: Unterstützung bei Entwurfsentscheidungen	A5: Technologie- und Plattformunabhängigkeit	A6: Abbildung des Entwurfs auf die Implementierungsebene
[Ro16] UML-based Model-Driven REST API Development	○	◖	◖	○	●	○
[PI+16] A Pattern Language for RESTful Conversations	○	○	●	◖	●	○
[EdI+16] EMF-REST: Generation of RESTful APIs from Model	●	◖	●	○	◖	●
[HL+15] A conversation based approach for modeling REST APIs	◖	◖	●	◖	●	○
[RP+13] Structural and behavioral modeling of RESTful web service interface using UML	◖	●	◖	○	●	○
[Sc11] Modeling RESTful Applications	○	○	●	○	●	○
[LS+09] Towards a Model-Driven Process for Designing RESTful Web Services	◖	●	●	○	●	○

Tabelle 3.2: Ergebnis der Bewertung relevanter Arbeiten

allerdings eine funktionale Spezifikation der Anforderungen der abzubildenden Applikationsdomäne. Die Arbeit von Haupt et al. [HL+15] beruht dagegen auf einem Domänenmodell und kann daher als logische Erweiterung angesehen werden, wobei der Ansatz nur die strukturellen Aspekte betrachtet. Weiter findet sich in keinem der untersuchten Entwurfsansätze eine Einbeziehung von existierenden Services zur Entwurfszeit wider, was demnach einem *Top-down*-Ansatz gleicht (vgl. Abschnitt 1.3) und die Wiederverwendbarkeit innerhalb einer Service-Landschaft maßgeblich verschlechtert. Daher ist dieser Umstand bereits zu Beginn des Entwurfs in adäquater Weise zu berücksichtigen.

Mit der Verknüpfung der fachlichen mit der technischen Ebene durch Ausprägung des Architekturstils REST ist die Unterstützung des Software-Architekten bei dem Entwurf im Hinblick auf die Qualitätsteilmerkmale der Wiederverwendbarkeit entscheidend. An dieser Stelle kann sich verstärkt den Ansätzen [HL+15, PI+16] bedient werden, die auf technischer Ebene den Software-Architekten bei der Realisierung unterstützen. Nach Angaben der Autoren haben sich hier Muster zur Entwurfsun-

terstützung besonders bewährt, die allerdings für die zu berücksichtigten Qualitätsteilmerkmale der Wiederverwendbarkeit aufbereitet werden müssen. Für den Entwurf des ressourcenorientierten Microservice und Vorgabe für die letztliche Implementierung liefert Rossi [Ro16] einen Ansatz, bei dem mit einem UML-Profil der Entwurf in ein Spezifikationsformat für Web-APIs überführt wird, um eine Implementierung des entsprechenden Service einzuleiten. Gleichzeitig ermöglicht diese Spezifikation die Generierung einer Dokumentation für Software-Entwickler von Service-Konsumenten im Hinblick auf die Nutzung der Web-API und damit des Microservice, was wiederum die Wiederverwendbarkeit fördert. Einen ähnlichen Ansatz liefern Laitkorpi et al. mit dem Spezifikationsformat WADL [LS+09]. Beide Ansätze setzen allerdings voraus, dass die Ressourcen aus dem zugrundeliegenden Artefakt zur Beschreibung des abzubildenden Geschäftsausschnitts identifiziert werden können. Hierfür liefern Rathod et al. einen Ansatz [RP+13] in Form von Heuristiken. Für die Modellierung der architekturellen Sichten des ressourcenorientierten Entwurfs kann die Arbeit von Schreier [Sc11] herangezogen werden, das ein entsprechendes Meta-Modell beinhaltet. Weiter kann aus [EdI+16] der grundsätzliche Automatisierungsgedanke bei der Entwicklung von ressourcenorientierten Microservices entnommen werden, der um verhaltensspezifische Aspekte erweitert werden müsste.

Nach der Spezifikation des ressourcenorientierten Microservice erfolgt die Überführung des Entwurfs auf die Implementierungsebene, wo der Entwurf zunächst mit der letztlichen Zielarchitektur bzw. der Technologie verknüpft werden muss. Dieser Aspekt wurde in [EdI+16] anhand von Java Enterprise Edition (Java EE) aufgezeigt. Anschließend ist eine systematische Überführung auf die verschiedenen Schichten einer Software-Architektur erforderlich, was keiner der untersuchten Ansätze im Detail aufzeigt. Für die Auffindbarkeit muss abschließend sichergestellt sein, dass die Spezifikation die veräußerte Dienstleistung des abgebildeten fachlichen Geschäftsausschnitts in adäquater Weise repräsentiert. Nur so kann eine Betrachtung von bestehenden Services zur Entwurfszeit erfolgen, um letztlich eine hohe Wiederverwendbarkeit sicherzustellen.

4 Domänengetriebener Entwurfsprozess

Viele Unternehmen setzen bei der Gestaltung ihrer Informationssysteme zunehmend auf eine serviceorientierte Architektur, um mehr Flexibilität und Wirtschaftlichkeit bei Anpassungen und Erweiterungen zu erzielen. Hierfür hat sich besonders der microservice-orientierte Stil für verteilte Systeme bewährt, wie zahlreiche Unternehmen bereits in der Praxis demonstriert haben [Ne15, Ti15, Gi16]. Demnach kann die Tragfähigkeit einer Microservice-Architektur zumindest in den jeweiligen Geschäftsdomänen als nachgewiesen gelten. Der Entwurf einer Microservice-Architektur erfordert zunächst den Entwurf der grundlegenden Systembausteine, bei denen es sich im Sinn der Definition in Abschnitt 2.1.2 um Microservices handelt. Die einzelnen Microservices ergeben sich wiederum aus einer geeigneten Aufteilung der Geschäftsdomäne in sogenannte Bounded Contexts (vgl. Abschnitt 2.2), für die u. a. der domänengetriebene Entwurfsansatz von Evans [Ev03] angewandt wird. Jeder Microservice repräsentiert demnach einen Ausschnitt der Geschäftsdomäne, der möglichst autonom ist und eine Kohäsion aufweist (vgl. Kohäsion und Autonomie in Anhang A.3). Die Gesamtheit der identifizierten Bounded Contexts dient dem Entwurf von ressourcenorientierten Microservices als Grundlage und ist in einem Domänenmodell verankert.

Dieses Kapitel erörtert daher die Frage, wie ein Bounded Context sich modellieren lässt, um eine systematische und nachvollziehbare Ableitung eines ressourcenorientierten Microservice zu ermöglichen. Bevor eine Beschäftigung mit dieser Frage möglich ist, ist zu klären, wie sich der domänengetriebene Entwurfsansatz in bekannte und etablierte Entwicklungsprozesse einordnen lässt, sodass er auch bei bestehenden Entwicklungsprozessen verwendet werden kann. Auf diese Einordnung gestützt, lässt sich die Forschungsfrage untersuchen, welche Modellierungsartefakte für einen ressourcenorientierten Microservice vonnöten sind und wie diese im Hinblick auf die Abbildung des Wissens einer (Geschäfts-)Domäne geartet sein müssen. Dabei soll sich die Abbildung von Wissen im Zusammenhang mit dem Bounded Context auf die Konzepte von Evans lehnen, damit ein direkter Bezug zu dessen domänengetriebenem Entwurfsansatz erkennbar ist.

Das Kapitel ist so strukturiert: Abschnitt 4.1 bringt einen Überblick über den domänengetriebenen Entwurfsprozess, bei dem dieser in einen etablierten Entwicklungsprozess eingeordnet wird und die dabei benötigten Rollen erläutert werden. Abschnitt 4.2 fokussiert die Modellierung des Domänenmodells im Rahmen eines oder mehrerer Bounded Contexts als Ausgangspunkt für den darauf aufbauenden Entwurfsprozess. Für die Modellierung der einzelnen Bounded Contexts wird zwischen deren strukturellen und verhaltensspezifischen Aspekten unterschieden, während stets der Bezug zu den Konzepten von Evans hergestellt wird. Abschnitt 4.3 stellt dann ein Szenario vor, um

den Entwurf eines Microservice zu demonstrieren und damit zugleich zur Validierung beizutragen. Abschnitt 4.4 beendet das Kapitel. Dort werden die zuvor definierten Modellierungsartefakte sowohl intern aufeinander als auch insgesamt auf die letztendliche Architektur eines Microservice bezogen.

4.1 Überblick über den domänengetriebenen Entwurfsprozess

Der domänengetriebene Entwurfsprozess steht im Zentrum dieser Arbeit, da er die verschiedenen Entwurfsaktivitäten miteinander verknüpft und so ein geordneter Rahmen für den Entwurf eines ressourcenorientierten Microservice ist. Deshalb sind nun zuerst der Entwurfsprozess in einen etablierten Entwicklungsprozess einzuordnen und die einzelnen Rollen innerhalb des Prozesses zu erläutern. Der Abschnitt gibt am Ende eine Übersicht über die einzelnen Entwurfsaktivitäten, deren detaillierte Beschreibungen Kapitel 5 erbringt.

4.1.1 Einordnung in einen etablierten Entwicklungsprozess

Das in Abbildung 4.1 veranschaulichte Vorgehen orientiert sich am Ansatz von Gebhart [Ge11], der sich wiederum auf etablierte Entwicklungsprozesse stützt, wie bspw. die allgemein beschriebenen Methodik in Bruegge et al. [BD09] oder den Rational Unified Process (RUP) [IBM-RUP-98] bzw. RUP/SOMA [WA+07]. Das Vorgehen wird dabei mit Prinzipien des domänengetriebenen Entwurfsansatzes nach Evans [Ev03] angereichert, wonach die so gewonnenen Entwurfsartefakte den Rahmen für die Implementierung legen sollen. Dies führt dazu, dass sich der Entwurf letztlich auf der Implementierungsebene in einem anderem Abstraktionsniveau widerspiegelt und so eine gewisse Nachvollziehbarkeit nach [Sc12] gegeben ist [Ev03, Ve13]. Das in dieser Arbeit präsentierte Vorgehen wurde bereits in [SG+17] zu Teilen veröffentlicht und gliedert sich im Wesentlichen in drei Prozessphasen, wie Abbildung 4.1 zeigt. Aus dieser lassen sich, mit Ausnahme der einzelnen Aktivitäten (how) der Prozessphasen, alle wesentlichen Bestandteile ableiten aus denen sich auch ein Prozess zusammensetzen lässt: „who is doing what, how, and when“ [IBM-RUP-98, S. 7]. Die verschiedenen Rollen des Entwicklungsprozesses sowie insbesondere die Aktivitäten im Entwurfsprozess werden in den Abschnitten 4.1.2 und 4.1.3 für ein ganzheitliches Verständnis des Entwicklungsprozesses behandelt.

Anforderungs- und Geschäftsanalyse

Der Entwicklungsprozess beginnt hier im Gegensatz zu Gebhart [Ge11] mit der Analyse der Nutzeranforderungen in Anlehnung an die *Requirements Elicitation und Analysis Phase* in Bruegge [BD09] sowie der sogenannten Disziplin *Requirements* in RUP [IBM-RUP-98] bzw. RUP/SOMA [WA+07]. Die Nutzeranforderungen ergeben sich aus konkreten Bedürfnissen des Unternehmens nach informationstechnischer Unterstützung bei bestehenden oder neuen Geschäftsprozessen. Einen entsprechenden Ansatz zur Formulierung etwaiger Nutzeranforderungen ist bspw. der Behaviour

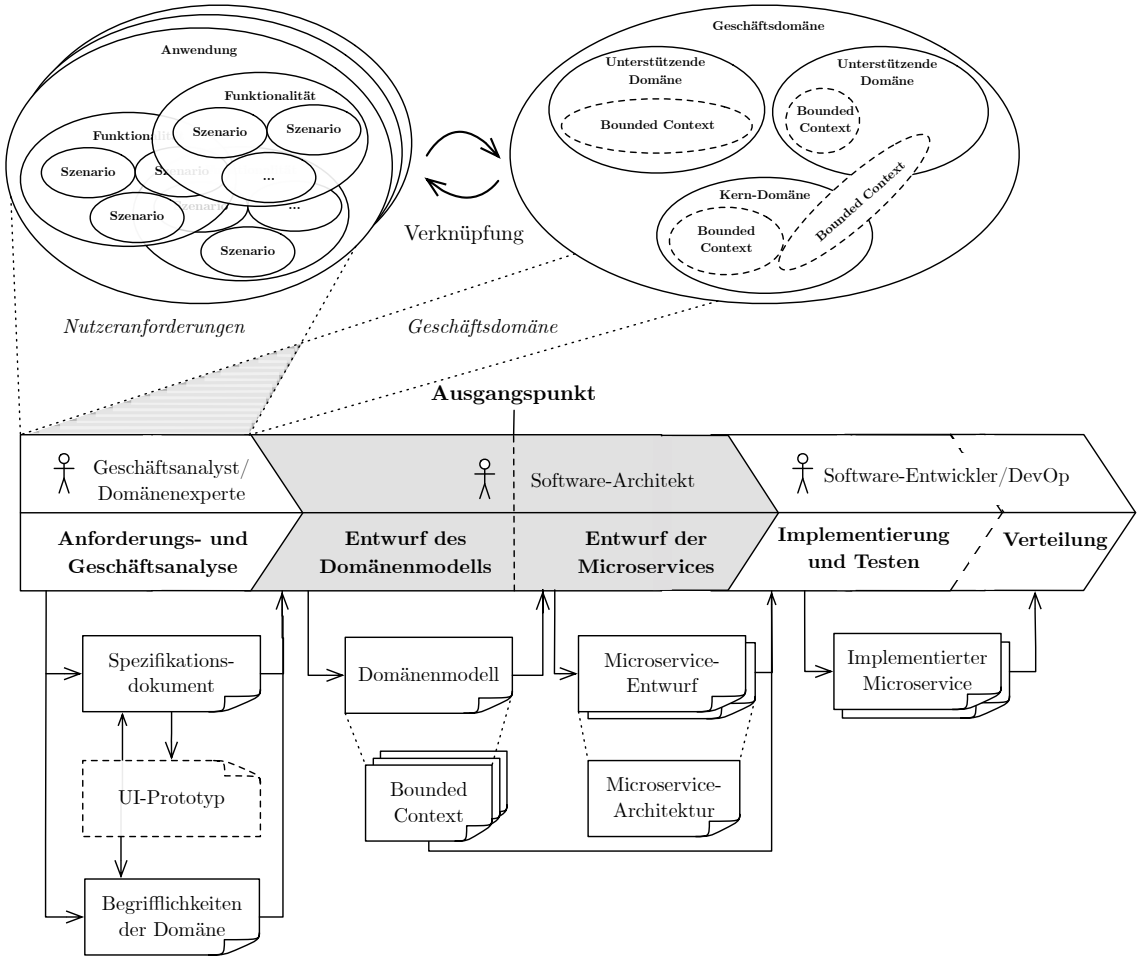


Abbildung 4.1: Domänengetriebenes Entwicklungsvorgehen für Microservices

Driven Development (BDD)-Ansatz, mit dem u. a. die Spezifikation einer Anwendung in textueller Form beschrieben wird und anschließend automatisch Tests generiert werden können [RW+15]. Die Nutzeranforderungen legen hier, anders als bei Gebhart, den Rahmen für die Geschäftsanalyse (vergleichbar mit der Disziplin *Business Modeling* in RUP bzw. RUP/SOMA) und damit die abzubildende Geschäftsdomäne fest. Dieses Vorgehen deckt sich auch mit dem grundlegenden Verständnis des domänengetriebenen Entwurfsansatzes, der nicht das Ziel verfolgt, ein allumfassendes Modell der Geschäftsdomäne abzubilden, sondern diese immer aus Sicht einer Anwendung betrachtet [Ve13]. Stattdessen werden eher mehrere Teildomänen sowie entsprechende Bounded Contexts gebildet, die insgesamt die Geschäftsdomäne adäquat abbilden, was eher einem verteilten Ansatz gleichkommt. Auch Bruegge nutzt zur Realisierung von Systemen eine Aufspaltung in der *System Design Activity*: „During system design, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams“ [BD09, S. 167]. Das gilt ebenso bei der Überführung eines Monolithen in eine Microservice-Architektur mit dem *Strangler Application*

Pattern in [Br17]. Als Monolith gilt laut Dragoni et al. [DG+17] eine Software-Anwendung, deren Einzelkomponenten sich nicht unabhängig voneinander ausführen lassen. Die mit einem Monolithen einhergehenden Einschränkungen sind beispielhaft Anhang A.2 zu entnehmen.

Die Teildomänen können entsprechend Abschnitt 2.2.1 gruppiert werden. Abhängig von ihrer Größe können sie nochmals weiter unterteilt werden, was bspw. dann gegeben ist, wenn mehrere Fachbereiche diesen Teildomänen untergeordnet sind. Ein Fachbereich ist in der Regel eine organisatorische Einheit, die tiefgehendes Wissen der Geschäftsprozesse eines spezifischen Ausschnitts der Geschäftsdomäne hat. Dabei erbringen ein oder mehrere Fachbereiche eine in sich abgeschlossene und kohäsive Dienstleistung. Derartige Dienstleistungen sind ideale Kandidaten für Bounded Contexts und letztlich auch Microservices im Sinne der Definition in Abschnitt 2.1.1. In den Fachbereichen haben wiederum die Domänenexperten bzw. Geschäftsanalysten das notwendige Domänenwissen. Diese Bounded Contexts müssen sich nicht zwangsläufig an organisatorischen Strukturen orientieren, sondern eher an den Kommunikationsrichtlinien im Unternehmen (vgl. Conway's Law in Abschnitt 2.2.1). Idealerweise sollte der Bounded Context möglichst eigenständig seine Dienstleistung erbringen [Ve13]. Allerdings kann das nicht immer sichergestellt werden, da Geschäftsprozesse nicht an die Grenzen eines Bounded Context gebunden sind. Evans sieht in diesem Fall das Konzept *Context Mapping* vor, was einer Überführungsfunktion von einem Bounded Context in einen anderen Bounded Context gleichkommt (vgl. [Ev03, Ve13]). So entstehende Abhängigkeiten zu bereits existierenden Microservices berücksichtigt der Entwurf, was auch dem Ansatz von Gebhart ähnelt und als *Middle-Out*-Ansatz bezeichnet wird.

Die Anforderungsanalyse und die darauf aufbauende Geschäftsanalyse ergeben ein Spezifikationsdokument mit sowohl geschäfts- als auch anwendungsbezogenem Wissen. Bei seiner Erstellung ist darauf zu achten, dass stets Begrifflichkeiten der abzubildenden Geschäftsdomäne verwendet werden. Die Gesamtheit dieser Nomenklatur wird auch als *Ubiquitous Language* bezeichnet (vgl. Abschnitt 2.2.2). Das soll, so Evans [Ev03], Mehrdeutigkeiten als Folge der natürlichen Sprachen vermeiden und die Kommunikation zwischen Domänenexperten/Geschäftsanalysten sowie dem Entwicklungsteam erleichtern.

Diese Arbeit betrachtet primär das domänenbezogene oder auch das geschäftsbezogene Wissen, das letztlich die Basis für den Entwurf des Domänenmodells und die ressourcenorientierten Microservices werden soll (vgl. Prämissen in Abschnitt 1.6). Das soll das hier bearbeitete Forschungsgebiet eingrenzen und zudem sicherstellen, dass die Wiederverwendbarkeit der Microservices nicht eingeschränkt wird. Die Anwendungsanforderungen werden nur am Rand betrachtet, um zu zeigen, wo sich diese letztlich in der Microservice-Architektur widerspiegeln. Das Aufbereiten des zuvor erwähnten domänenbezogenen Wissens, von Evans *Knowledge Crunching* genannt (vgl. Abschnitt 2.2.2), betrifft rein fachliche Informationen der abzubildenden Domäne und kann deshalb einem Computation Independent Model (CIM) aus dem Bereich der MDSD gleichgesetzt werden [St07]. Um die Vollständigkeit der Domäne zu überprüfen, lassen sich die Nutzeranforderungen aus der Anforderungsanalyse verwenden. Jede von ihnen muss sich mit dem Wissen der Domäne realisieren bzw. abbilden lassen. Für die

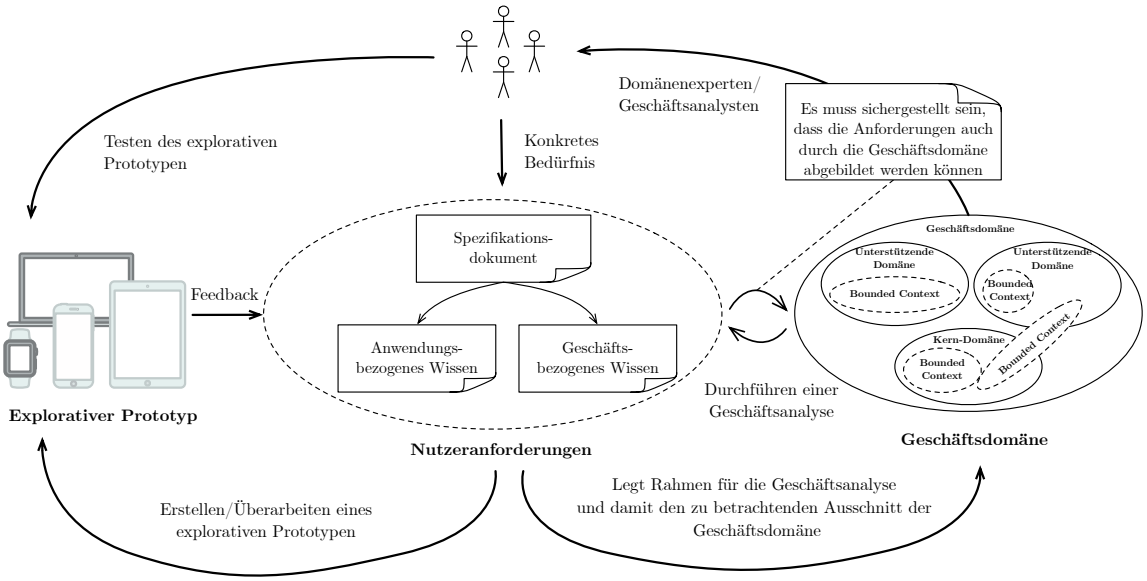


Abbildung 4.2: Übersichtliche Darstellung der Anforderungs- und Geschäftsanalyse

Vollständigkeit der Nutzeranforderungen kann wiederum ein sogenannter User Interface (UI)-Prototyp erstellt werden [SG+17]. Das ist der Entwurf einer Benutzerschnittstelle in Form eines explorativen Prototyps [Se06], mit dem der spätere Nutzer zur Erledigung seines Anliegens interagieren soll und der so die Vollständigkeit der Nutzeranforderungen verifizieren kann. Laut Arnowitz et al. [AA+10] ist ein Prototyp im Sinne der Definition in [MMJ92] eine Design-Idee zum Zweck der Evaluation, wie in Abbildung 4.2 dargestellt.

Entwurf des Domänenmodells

Auf diesen Prozessschritt folgt der Entwurf des Domänenmodells in Anlehnung an die *Analysis and Design Discipline* in RUP [IBM-RUP-98] bzw. RUP/SOMA [WA+07]. Hierzu werden für jeden identifizierten Bounded Context aus der Geschäftsanalyse die darin enthaltenen Domänenobjekte im Sinn von Evans extrahiert und aufeinander bezogen, damit sie die strukturellen Aspekte des jeweiligen Bounded Context repräsentieren. Domänenobjekte, die in mehr als einem Bounded Context existieren, werden als geteilte (engl. shared) Domänenobjekte gesondert behandelt. Danach werden die verhaltensspezifischen Aspekte dieser Geschäftsdomäne dem Spezifikationsdokument entnommen und mit den zuvor ermittelten Domänenobjekten verknüpft. Da diese Betrachtung zumeist Wissen im Hinblick auf die Modellierung und DDD voraussetzt und das resultierende Domänenmodell bereits Vorgaben hinsichtlich der Strukturierung der Implementierung enthält, wird der Entwurf des Domänenmodells hier anders als bei Gebhart [Ge11] nicht als Bestandteil der Geschäftsanalyse aufgefasst. Übertragen auf die Konzepte von MDSD und im Vergleich zur vorherigen Prozessphase ist dies allerdings weiterhin als CIM anzusehen, das trotz bestimmter Vorgaben in Bezug auf die

Modellierung nur formalisierte fachliche Informationen abbildet. Schließlich ist zu betonen, dass das entstehende Domänenmodell nach den Konzepten des *Model-driven Design* von Evans [Ev03] auch dank der besseren Verständigung der Beteiligten untereinander starken Bezug zu objektorientierten Paradigmen hat: „The fundamentals of object-oriented design seem to come naturally to most people“ [Ev03, S. 117].

Ausgangspunkt: Entwurf der Microservices

Im Anschluss an den Entwurf des Domänenmodells folgt der Entwurf der ressourcenorientierten Microservices in Anlehnung an die *Object Design Activity* in Bruegge [BD09]. Dieses ist der Kern dieser Arbeit und zugleich ihr Ausgangspunkt. Hier wird für jeden Bounded Context des Domänenmodells eine systematische und nachvollziehbare Ableitung des ressourcenorientierten Microservice-Entwurfs unter Berücksichtigung von Qualitätsteilmerkmalen der Wiederverwendbarkeit durchgeführt, wie in Abschnitt 2.3.2 skizziert. Die Web-API steht bei diesem Prozessschritt im Mittelpunkt, da sich nur über sie mit der letztlich abzubildenden Domäne interagieren lässt. Die Wiederverwendbarkeit betrifft vor allem die Service-Konsumenten und deren Software-Entwickler, damit die Integration minimal aufwändig ist. In Bezug auf die Qualitätsmerkmale eines Microservice geht es hier vorrangig um die Interoperabilität mit den Service-Konsumenten (vgl. Anhang A.3). Die übrigen Qualitätsmerkmale, also Autonomie und Kohäsion, werden als gegeben betrachtet, da ein Microservice seine Dienstleistung, sofern möglich, eigenständig und damit ohne zusätzliche Microservices erbringen sollte und Kohäsion bereits durch den Bounded Context gegeben ist.

Die Überführung eines objektorientierten Bounded Context in ein ressourcenorientiertes Modell (nachfolgend als Ressourcenmodell bezeichnet) ist keinesfalls eine triviale Aufgabe und verlangt in der Regel Abstriche (engl. trade-offs) [HK+14]. Dieses Problem wird in der Literatur auch als *impedance mismatch* bezeichnet. Es entsteht bspw. bei der Überführung von objektorientierten in relationale Datenstrukturen [PZ+08, HK+14, Ri15a]. Nach diesem Schritt wird das Ressourcenmodell sukzessiv durch weitere Informationen ergänzt – etwa durch eine Adressierung, mit deren Hilfe sich die Dienstleistung einer Domäne nutzen lässt. Bei diesen Verfeinerungen wird der Software-Architekt durch aufbereitetes Wissen unterstützt, damit er seine Entscheidung im Hinblick auf die Wiederverwendbarkeit des letztlichen Microservice fundiert treffen kann. Vor allem bei den offerierten Operationen ist das unverzichtbar. Als Ergebnis dieser Phase entstehen, abhängig von der Anzahl umzusetzender Bounded Contexts, mehrere Ressourcenmodelle, die unabhängig von einer konkreten Technologie sind und zusammen mit dem zugrundeliegenden Bounded Context die Implementierung des ressourcenorientierten Microservice anleiten sollen. Durch die Verknüpfung des fachlichen Wissens mit dem ressourcenorientierten Architekturstil sind die Ressourcenmodelle als PIMs anzusehen, da sie zwar technologische Informationen beinhalten, jedoch Freiheitsgrade hinsichtlich der Technologie zur Implementierung gestatten.

Implementierung, Testen und Verteilung der Microservices

Die nächste Phase ist die Verknüpfung des Entwurfs mit einer konkreten Technologie, weshalb diese nach den Konzepten von MDSD als PSM bezeichnet werden kann. Hierzu wird jedes Ressourcenmodell zunächst mit einem Anwendungsschichtprotokoll verknüpft, das als Grundlage für die Ableitung einer Web-API-Spezifikation dient. Die Web-API-Spezifikationen liefern schließlich den Vertrag zwischen Microservice und letztlichen Service-Konsumenten, sodass dann bereits eine Trennung von Entwicklungsteams ohne das Vorhandensein eines lauffähigen Microservice möglich ist. Die Wahl der Technologie zur Implementierung ist dann dem einzelnen Entwicklungsteam überlassen. Ergänzend zu dem Ansatz von Gebhart [Ge11] können an dieser Stelle bereits etwaige Blackbox-Tests unter Anwendung von Test Driven Development (TDD) ihre Anwendung finden, da die Web-API-Spezifikation die entsprechenden Antworten definiert und das Verhalten des Geschäftsausschnitts im Zuge des Entwurfs des Domänenmodells bzw. der Bounded Contexts in geeigneter Form modelliert wurde. Gleichzeitig finden im Rahmen dieser Phase auch eine Verteilung der Microservices im Zuge von CI/Continuous Delivery (CD) statt, sodass diese auch während der eigentlichen Implementierung im Betrieb getestet werden können. Bei CI handelt es sich um eine Praktik der Software-Entwicklung, bei der regelmäßig ein Software-Artefakt aus dem zugrundeliegenden Quellcode durch einen automatischen Bauprozess erzeugt wird, während CD das logisch erweitert und das resultierende Software-Artefakt auf entsprechende Laufzeitumgebungen verteilt [DM+07, HF10].

Eine strikte Trennung wie bei Gebhart, wo ein sogenannter Verteilungsmanager benötigt wird, unterbleibt hier. Stattdessen ist die Rolle des DevOp in einem Entwicklungsteam vorgesehen, der Kenntnisse über die Verteilung hat. Dabei ist grundsätzlich das Ziel, das Entwicklungsteam für den jeweiligen Microservice auch aus betrieblicher Sicht autonom agieren zu lassen. Hierfür verwenden Unternehmen entweder eigene *Platform as a Service (PaaS)-Lösungen*, wie bspw. STUPS[1] bei Zalando, oder setzen auf PaaS-Lösungen externer Anbieter, wie bspw. AWS Elastic Beanstalk[2] von Amazon. Beim Vergleich dieser Phase mit etablierten Entwicklungsprozessen ist diese den Disziplinen *Implementation, Testing und Deployment* in RUP [IBM-RUP-98] bzw. RUP/SOMA [WA+07] gleichermaßen zuzuordnen sowie Implementierung und Testen in [BD09]. In dieser Arbeit wird in diesem Zuge ausschließlich die nachvollziehbare Überführung des Microservice auf die Implementierungsebene mittels der zuvor erstellten Entwurfsartefakte beleuchtet.

4.1.2 Rollen im Entwicklungsprozess

Dieser Abschnitt beschreibt die verschiedenen Rollen in diesem Entwicklungsprozess im Hinblick auf deren Aufgabenspektrum bzw. Tätigkeitsbereiche. Fünf verschiedene Rollen wurden identifiziert (siehe Tabelle 4.1), die jeweils mindestens eine am Projekt beteiligte Person bekleiden sollte. Doch kann eine Person auch mehr als eine Rolle innehaben.

[1] https://docs.stups.io/en/latest/ (Letzter Zugriff: 29.10.2017)
[2] https://aws.amazon.com/de/elasticbeanstalk/ (Letzter Zugriff: 15.10.2017)

Rolle	Beschreibung
Domänenexperte	Ein Domänenexperte hat essentielles Wissen über einen Ausschnitt der Geschäftsdomäne [Ve13]. Er liefert das notwendige Wissen für den Entwurf eines Bounded Context, der einem Microservice zugrunde liegt.
Geschäftsanalyst	Ein Geschäftsanalyst hat im Gegensatz zum Domänenexperten eine übergeordnete Rolle. Seine Aufgabe ist es, gemeinsam mit jenem ein ganzheitliches Verständnis der Geschäftsdomäne zu entwickeln, um Lösungen vorzuschlagen, damit ein Unternehmen seine geschäftlichen Ziele erreicht [BBA09, Ge11].
Software-Architekt	Ein Software-Architekt soll einen Microservice im Sinn der zu beachtenden nicht-funktionalen Anforderungen entwerfen sowie nachvollziehbare und begründete Entwurfsentscheidungen treffen. Sein Entwurf soll als Grundlage für die Implementierung funktionieren.
Software-Entwickler	Ein Software-Entwickler soll auf Grundlage des Entwurfs und einer entsprechenden Technologieauswahl den Microservice implementieren und die implementierte Funktionalität anschließend auf Korrektheit überprüfen.
DevOp	Ein DevOp ist für die Verteilung (engl. deployment) und den Betrieb (engl. operation) des entwickelten Microservice im Entwicklungsteam verantwortlich. Dazu zählt bspw. die Konfiguration im Rahmen von CD, Monitoring oder auch die Skalierung des Microservice [KL+16].

Tabelle 4.1: Rollen im betrachteten Entwicklungsprozess

4.1.3 Aktivitäten im Entwurfsprozess

Der betrachtete Entwurfsprozess setzt auf einem zuvor erstellten Domänenmodell auf, das die Fachlichkeit des zu betrachteten Ausschnitts der Geschäftsdomäne widerspiegelt. Dabei werden neben den strukturellen auch die verhaltensspezifischen Aspekte betrachtet, die in geeigneter Form abgebildet werden müssen. Hierauf wird im darauffolgenden Abschnitt 4.2 näher eingegangen. Abbildung 4.3 veranschaulicht den Entwurfsprozess mitsamt den Entwurfsaktivitäten. Diese werden für jeden identifizierten Bounded Context durchgeführt. Diesen Schritt können unabhängige Entwicklungsteams durchführen. Um möglichst schnell sichtbare Ergebnisse im Hinblick auf die Bereitstellung eines Microservice zu erzielen, wurde der Entwurfsprozess hier so konzipiert, dass iteratives und inkrementelles Vorgehen möglich ist. Das gilt ebenso für den Entwurf des Domänenmodells in der vorigen Prozessphase.

Ausgangspunkt ist, bereits beschrieben, das vorhandene Domänenmodell. Mit der Überführung eines Bounded Context in einen ressourcenorientierten Architekturstil wird die modellierte Fachlichkeit um

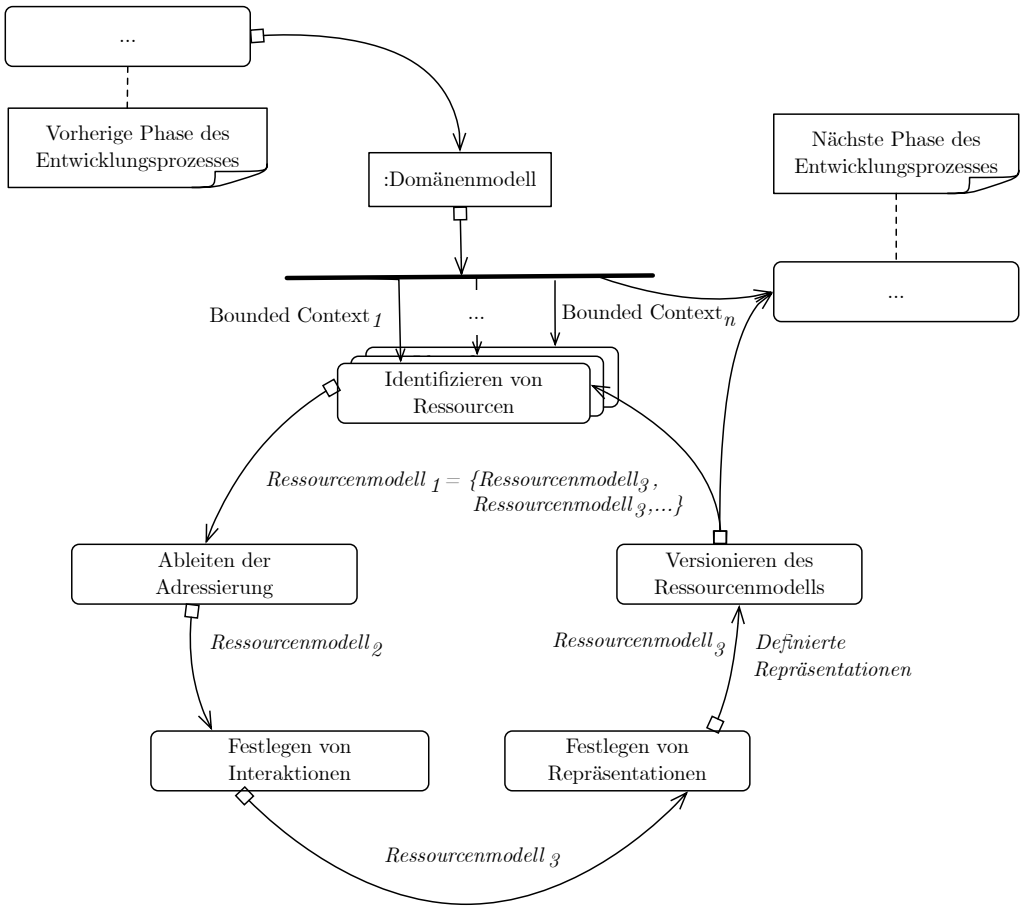


Abbildung 4.3: Übersicht über die Entwurfsaktivitäten

technologische Informationen angereichert, wie bspw. die Definition von Datenstrukturen in Form von Repräsentationen. Das Ergebnis dieser Überführung wird ähnlich den Ansätzen [LK+06, SL+08, RP+13, HK+14] als *Ressourcenmodell* bezeichnet. Wie bereits erwähnt, ist die Überführung keine triviale Aufgabe, da die zugrundeliegenden Meta-Modelle nicht zwangsläufig kompatibel zueinander sind. So verfolgt das Domänenmodell nach Evans [Ev03] einen objektorientierten Stil, während das Ressourcenmodell einen ressourcenorientierten Stil vorsieht. Dieser sogenannte *impedance mismatch* (vgl. Abschnitt 4.1) führt dazu, dass zwischen zwei verschiedenen Paradigmen gewechselt werden muss, was vor allem das Entwicklungsteam betrifft: „developers [have] to gradually transition between these two mindsets“ [LS+09, S. 173]. Aus diesem Grund wurde ähnlich zu dem Ansatz von Laitkorpi et al. ein mehrstufiger Prozess gewählt, der durch die einzelnen Aktivitäten repräsentiert wird. Diese Aktivitäten betreffen jeweils unterschiedliche Aspekte der Ressourcenorientierung und fügen so den initial überführten Ressourcenmodellen schrittweise weitere benötigte Informationen hinzu, sodass schließlich eine Web-API-Spezifikation ableitbar ist, welche die gewünschten Qualitätsaspekte der Wiederverwendbarkeit (vgl. Abschnitt 2.3.2) bestmöglich erfüllt.

(A1) **Identifizieren von Ressourcen:** Die im Bounded Context enthaltenen Domänenobjekte und Domänenservices werden im Kontext dieser Aktivität durch Heuristiken auf Ressourcen übertragen und in Beziehung zueinander gesetzt. Das stützt sich auf die Arbeiten von Haupt et al. [HK+14, HL+15] und Laitkorpi et al. [LS+09]. Neben den strukturellen werden auch die verhaltensspezifischen Aspekte der Domäne in geeigneter Form überführt, sodass sich das Verhalten auch im Ressourcenmodell widerspiegelt und letztlich über die Web-API veräußert werden kann.

Resultierendes Artefakt: Initiales Ressourcenmodell R_1

(A2) **Ableiten der Adressierung:** Hier werden die identifizierten Ressourcen mit einer eindeutigen URL verknüpft, über welche jene später erreichbar sein sollen. Das entspricht dem URL-Modell aus [HK+14, HL+15]. Für die Bildung der URL lassen sich bestehende linguistische Muster verwenden, um die Benutzbarkeit und letztlich auch die Wiederverwendbarkeit zu erhöhen.

Resultierendes Artefakt: Ressourcenmodell R_2 mit Informationen zur Adressierung

(A3) **Festlegen von Interaktionen:** Eine ressourcenorientierte Web-API ist eine wohldefinierte Schnittstelle mit einem beschränkten Set an Operationen. Auch die Interaktion mit einer Ressource unterliegt bestimmten Rahmenbedingungen, die sich durch das geteilte Medium und die Verteilung ergeben. Deshalb werden bei dieser Aktivität Muster vorgestellt, die das zu veräußernde Verhalten einer Domäne korrekt abbilden. Orientierung gaben hier die Mustersprache von [PI+16] sowie die Kontrollliste für den Entwurf von RESTful Web Services [GG+16c]. Diese wurden in Bezug auf die geforderten Qualitätsaspekte der Wiederverwendbarkeit erweitert und ausgerichtet.

Resultierendes Artefakt: Ressourcenmodell R_3 mit festgelegten Interaktionen

(A4) **Festlegen von Repräsentationen:** Nachdem die Interaktionen in geeigneter Form modelliert wurden, widmet sich diese Aktivität der Festlegung der Repräsentationen, die ein zentraler Bestandteil einer Interaktion sind. Es ist u. a. ein geeigneter Medientyp zu wählen und zu entscheiden, wie die Repräsentationen aus dem Ressourcenmodell schließlich abzuleiten und angemessen zu definieren sind, damit sie sich für eine Web-API-Spezifikation wiederverwenden lassen.

Resultierendes Artefakt: Repräsentationen $Repr(R)$ von Ressourcenmodell R_3

(A5) **Versionieren des Ressourcenmodells:** Nachdem das Ressourcenmodell mit den notwendigen Informationen zur Ableitung einer Web-API-Spezifikation angereichert wurde, wird es nun durch eine Versionskennung erweitert. Gleichzeitig wird mit dieser Aktivität ein Prozess

eingeführt, wie mit Änderungen an einem Ressourcenmodell zu verfahren ist, da etwaige Änderungen am Ressourcenmodell, bspw. infolge eines iterativen und inkrementellen Vorgehens, auch Änderungen an der Web-API nach sich ziehen können. Das wiederum kann Folgen auch auf die letztlichen Service-Konsumenten haben. Deshalb verlangen Änderungen bedachtsames Vorgehen [LX+13, EZ+14]. Im Zuge der Versionierung kann hier mehr als ein Ressourcenmodell existieren. Die Anzahl richtet sich an die zu unterstützenden Versionen eines Microservice für die Service-Konsumenten.

Resultierendes Artefakt: x Ressourcenmodelle R_3 mit Versionskennung V wobei $x \in \mathbb{N} \setminus 0$

4.2 Zugrundeliegende Modellierungsartefakte des Entwurfsprozesses

Dieser Abschnitt beschreibt den Ausgangspunkt für den Entwurf von ressourcenorientierten Microservices. Dabei handelt es sich um ein modelliertes Domänenmodell, dem Ansatz von Gebhart [Ge11] ähnlich, was den zu betrachteten Ausschnitt der Geschäftsdomäne repräsentiert (siehe Abbildung 4.5). Ein Domänenmodell kann nach Evans [Ev03] alles enthalten, was das Verständnis einer abzubildenden Geschäftsdomäne fördert. Diese Definition des Modells ist hier bzw. im Sinn von Evans nicht gleichzusetzen mit der Definition aus dem Bereich der MDSD, wo ein Modell einer eindeutigen Syntax und Semantik unterliegt. Das Domänenmodell enthält Teildomänen und Bounded Contexts. Letztere sind kohäsive Einheiten, die für die Geschäftsdomäne eine immaterielle Dienstleistung bereitstellen und deshalb ideale Kandidaten für ressourcenorientierte Microservices sind.

Deswegen werden nun entsprechende Modellierungen vorgestellt, die sowohl die strukturellen als auch die verhaltensspezifischen Aspekte eines Bounded Context adäquat abbilden können und damit den Ausgangspunkt des Entwurfsprozesses darstellen. Das allein erlaubt eine systematische und nachvollziehbare Ableitung für ressourcenorientierte Microservices. Dabei muss die Modellierung der strukturellen Aspekte derjenigen der verhaltensspezifischen Aspekte vorausgehen, wie auch in der Definition von Gamma et al. [GH+94] festgehalten ist (siehe Abschnitt 2.2.4). So müssen die Domänenobjekte bereits identifiziert sein und im Sinn der *Ubiquitous Language* (vgl. Abschnitt 2.2.2) einen eindeutigen Namen erhalten haben. Auch müssen die Beziehungen zwischen den Domänenobjekten festgelegt sein, da sie Hinweise für die Ableitung von möglichen verhaltensspezifischen Aspekten zwischen Domänenobjekten liefern. Hier sei als Beispiel die *Aggregate Root* genannt, welche die Eignerschaft von *Aggregates* übernimmt und für deren Erzeugung verantwortlich ist.

Die Modellierung anwendungsbezogener Anforderungen von etwaigen Service-Konsumenten, die unabhängig von der abzubildenden Geschäftsdomäne zu betrachten sind und sich aus dem zugrundeliegenden Spezifikationsdokument ergeben, werden nicht betrachtet (vgl. Prämissen in Abschnitt 1.6). Denn bei der Betrachtung von Anwendungsanforderungen besteht die Gefahr, die Wiederverwendbarkeit des Microservice derart einzuschränken, dass nur spezifische Anwendungen oder Anwendungsgruppen diesen letztlich nutzen können. Stattdessen empfiehlt es sich, für anwendungsbezogene

Anforderungen einen separaten Microservice nach dem Backend For Frontend (BFF)-Muster (vergleichbar mit dem Strukturmuster Fassade [GH+94]) zu entwerfen, sofern die Anforderungen die Verortung auf Seiten der Microservices erfordern [Ne15]. Ist dies nicht der Fall, sind diese Anforderungen durch die jeweiligen Service-Konsumenten umzusetzen. Dann kann sich bspw. die Geschäftsdomäne unabhängig von einer konkreten Anwendung weiterentwickeln. Dieser Fall wird allerdings nun nicht weiter ausgeführt, sondern nur in Abbildung 4.4 veranschaulicht. Die in Kapitel 5 erarbeiteten Konzepte lassen sich allerdings auch für derartige Ansätze verwenden. Hierzu müsste lediglich die initiale Überführung des Ressourcenmodells R_1 entsprechend adaptiert werden.

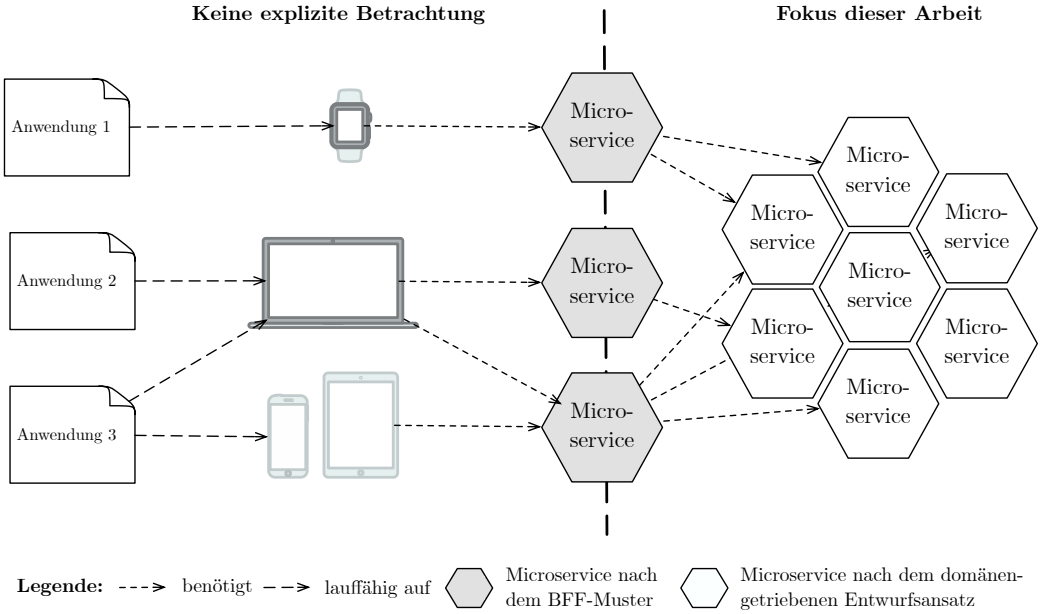


Abbildung 4.4: Veranschaulichte Microservice-Architektur bestehend aus Microservices mit dem domänengetriebenen Ansatz und nach dem BFF-Muster

Als Modellierungssprache wurde UML gewählt, obwohl deren Verwendung zu hoher Komplexität infolge des Meta-Modells der UML [OMG-UML2a, OMG-UML2b] führt, die für die Modellierung einer Domäne eigentlich nicht erforderlich ist. Grund für diese Entscheidung war, dass UML eine weit verbreitete und universelle Modellierungssprache mit entsprechender Werkzeugunterstützung im Bereich der Software-Entwicklung ist, sodass diese Wahl auch die praktische Anwendbarkeit befördert [Ge11]. Zudem lassen sich mit OCL Bedingungen festlegen, damit das Modell samt seiner Ausprägung gewissen Regeln unterliegt [SV+06, Se06]. Die Verwendung einer DSL würde zwar die Komplexität reduzieren, gleichzeitig aber auch die Werkzeugunterstützung einschränken und initialen Lernaufwand von den Modellierenden verlangen, die sich erst mit der Sprache vertraut machen müssen. Da die Modellierung der Domäne nicht im Fokus dieser Arbeit steht, wurde dieser Ansatz nicht weiter berücksichtigt. Zwar wurde UML als Modellierungssprache gewählt, doch die grundle-

genden Konzepte lassen sich auch in eine andere Sprache übertragen, sodass diese Entscheidung den domänengetriebenen Entwurfsansatz nicht gravierend einschränkt.

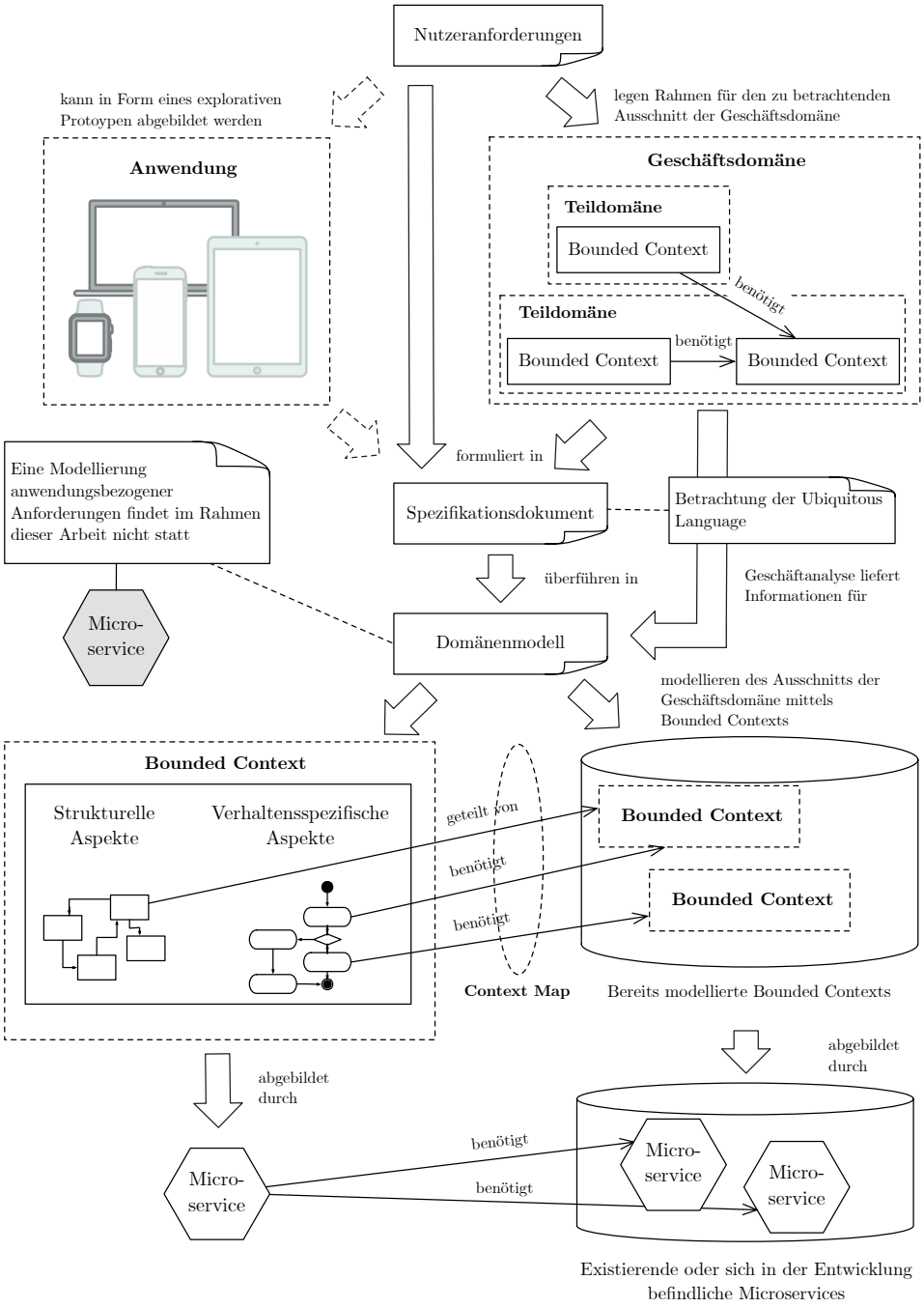


Abbildung 4.5: Artefakte für den ressourcenorientierten Entwurf eines Microservice und deren Zusammenspiel

4.2.1 Modellierung von strukturellen Aspekten

Die strukturellen Aspekte des gewählten Geschäftsausschnitts fokussieren die Domänenobjekte und deren Beziehungen zueinander, was in der Literatur auch als Informationsmodell bezeichnet wird und ein wichtiger Bestandteil des Domänenmodells ist [SL+08, Fa10, SG+17]. Für die konkrete Modellierung eignen sich sogenannte UML-Klassendiagramme, die zur offiziellen Spezifikation der UML gehören [RH08, OMG-UML2a, OMG-UML2b]. Dies betonen etwa auch Siikarla et al.: „[An] information model represents all relevant nouns and their relationships in the domain, so using class diagrams is an obvious choice“ [SL+08, S. 6]. Verwandte Ansätze beim Entwurf von ressourcenorientierten Services nutzen für die Modellierung derartiger Aspekte ebenfalls ein UML-Klassendiagramm [LS+09, PR11, HK+14, Pr15] oder erweitern die Semantik einzelner Modellelemente durch sogenannte Stereotypen auf Ebene M2 der Vier-Schichten-Architektur von UML [KW+03, Ro16].

Um nun den DDD-Ansatz mit UML zu verknüpfen, sind die Konzepte und Begrifflichkeiten des DDD-Ansatzes in UML zu übersetzen. Dies ist vergleichbar mit dem *Context Mapping* nach Evans [Ev03], bei dem UML und DDD jeweils durch einen Bounded Context repräsentiert werden. Für die Modellierung der strukturellen Aspekte wurden folgende Konzepte bei DDD identifiziert: 1) *Entity*, 2) *ValueObject*, 3) *Aggregate* und 4) *AggregateRoot*. Als *Entity* gilt meist eine reale Einheit in der Geschäftswelt – etwa eine Bestellung oder ein Bestellposten [Ge11]. Jede Entität ist eindeutig identifizierbar, sodass sie von anderen Entitäten unterschieden werden kann. Zudem kann eine Entität veränderbar (engl. mutable) oder nicht veränderbar (engl. immutable) sein. Hingegen handelt es sich bei einem ValueObject um ein unveränderliches Werteobjekt, das keine eindeutige Identität hat und meist einer Entität zugewiesen wird. Als Beispiel lässt sich hier die Wohnadresse einer Person nennen: Die Person wird durch eine *Entity* und die Wohnadresse durch ein *ValueObject* repräsentiert. Beide sind über eine Assoziation miteinander verknüpft. Ein *Aggregate* kann ein *ValueObject* oder eine *Entity* sein, das sich in einer Aggregations- oder Kompositionsbeziehung mit einem anderen Domänenobjekt befindet. Entsprechend handelt es sich bei einem *AggregateRoot*, um diejenige Entität, welcher *Aggregates* zugeordnet sind und die dahingehend eine Eignerschaft aufweist [Ev03, Ve13].

Die Übersetzung von *Aggregates* und *AggregateRoot* in UML kann, wie bereits erwähnt, durch eine Aggregations- oder Kompositionsbeziehung erfolgen. Für die Unterscheidung zwischen *Entity* und *ValueObject* ist dies allerdings nicht ganz so eindeutig. Grundsätzlich lassen sich entsprechende Attributwertpaare sowohl *Entities* als auch *ValueObjects* zuweisen. Übertragen auf das UML-Klassendiagramm entspricht dies am ehesten der Metaklasse *Class*. Um zwischen *Entities* und *ValueObjects* unterscheiden zu können, wird der leichtgewichtige Erweiterungsmechanismus der UML mithilfe von Stereotypen verwendet und damit um neue Modellelemente erweitert. Dabei werden die Stereotypen *Entity* und *ValueObject* mit dem zu erweiternden Modellelements des UML-Meta-Modells auf Ebene M2 in einem sogenannten UML-Profil (siehe Abbildung 4.6) zusammengefasst, das fortan als DDD-Profil bezeichnet wird.

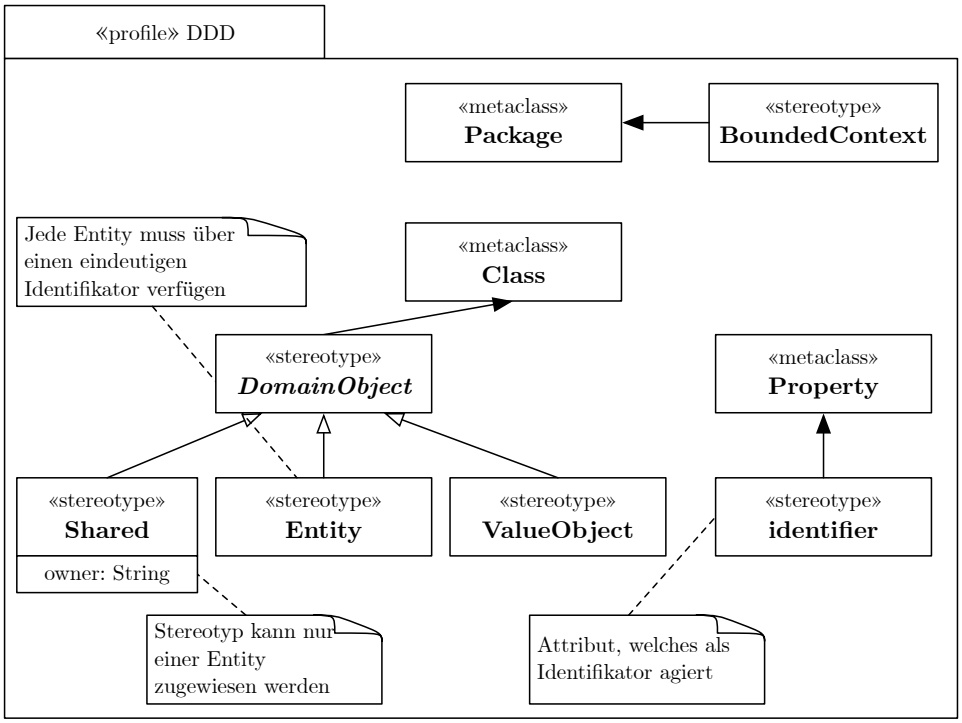


Abbildung 4.6: UML-Profil für DDD

Eine Besonderheit ist der Stereotyp `Entity`, da bei seinem Einsatz ein entsprechendes Merkmal (engl. property) gemäß des UML-Meta-Modells als Identifikator erforderlich ist. Das soll sicherstellen, dass jede *Entity* über eine eindeutige Identität verfügt. Vernon [Ve13] liefert hierzu entsprechende Lösungsansätze, wie sich eine eindeutige Identität bilden lässt. Auch Schreier zeigt in [Sc11] ein Muster für die Bildung geeigneter Identifikatoren im Kontext von REST-basierten Anwendungen. Das *EntityIdentifierPattern* (abgeleitet aus dem *ResourceIdentifierPattern* [Sc11]) zeigt, dass der Identifikator entweder eine einfache und willkürlich gesetzte Zeichenkette (*SimpleIdentifier*) ist oder aus zusammengesetzten Attributen der jeweiligen *Entity* (*ComplexIdentifier*) besteht. Die Wahl des Identifikators ist für den Entwurf nicht relevant, sondern eher als Vorgabe für das Entwicklungsteam zu sehen: „The identifiers are not important at the beginning of the design process but at the end for implementation or code generation“ [Sc11, S. 3]. Die Manifestierung des Identifkators in einem UML-Klassendiagramm stützt sich auf die Arbeit von Laitkorpi et al. [LS$^+$09]. Es wurde das entsprechende UML-Profil mit dem Stereotyp *Identifier* ergänzt, der im Gegensatz zu den anderen beiden Stereotypen die Metaklasse *Property* spezialisiert. Die Vorgabe für den Identifikator im Sinn des *EntityIdentifierPattern* kann über Kommentare zur Klasse angegeben werden. Falls keine derartige Vorgabe existiert, obliegt die Entscheidung dem jeweiligen Entwicklungsteam.

Bis zu diesem Zeitpunkt wurden lediglich die strukturellen Aspekte unabhängig von anderen Bounded Contexts betrachtet. Allerdings können auch geteilte Domänenobjekte existieren, die in mehr als

einem Bounded Context verwendet werden. Bei geteilten Domänenobjekten kann es sich allerdings nur um eine *Entity* mit eindeutiger Identität handeln, da nur so eine Referenzierung über die Grenzen eines Bounded Context möglich ist. Etwaige damit verknüpfte *ValueObjects* können dann ebenfalls als geteilt angesehen werden. Nachdem eine geteilte *Entity* identifiziert wurde, muss zunächst die grundlegende Fragestellung geklärt werden, welcher Bounded Context die Eignerschaft dieses Domänenobjekts für sich beansprucht. Sobald das geklärt wurde, kann diese Entscheidung entsprechend modelliert werden. So wird dem geteilten Domänenobjekt der Stereotyp `Shared` zugewiesen. Zusätzlich lässt sich über die Eigenschaftsdefinition `owner` angeben, welcher Bounded Context die Eignerschaft hat. Hierzu wird der Eigenschaftswert mit dem Namen des Bounded Context versehen. Falls kein Eigenschaftswert gesetzt ist, wird angenommen, dass der jeweilige Bounded Context die Eignerschaft besitzt. Ein UML-Kommentar kann zudem darauf hinweisen, falls das geteilte Domänenobjekt in einem anderen Bounded Context anders bezeichnet ist. Diese Informationen können schließlich aus den Bounded Contexts extrahiert und nach den Konzepten von DDD in einer *Context Map* festgehalten werden. Darüber hinaus ist bei geteilten Domänenobjekten zu prüfen, ob sich diese jeweils in einen anderen Bounded Context überführen lassen. Denn das geteilte Domänenobjekt muss etwa alle von den jeweiligen Bounded Contexts benötigten Informationen enthalten.

Mit OCL lässt sich schließlich eine Invariante definieren, welche die Validität des so entstandenen UML-Klassendiagramms prüft. Zudem kann noch definiert werden, welche Attribute eines Domänenobjekts optional und welche hingegen unbedingt bei dessen Instanziierung zu setzen sind. Hierfür kann ein Attribut um `[1]` (Default) bzw. `[0..1]` ergänzt werden. Weiter können zusätzliche Invarianten mittels OCL angegeben werden, damit die Ausprägung eines Domänenobjekts stets einen validen Zustand ergibt. Falls derartige Angaben fehlen, wird angenommen, dass jedes Attribut ein Pflichtattribut ist und also ein Wert gesetzt werden muss und gesonderte OCL-Bedingungen nicht existieren. Ersteres entspricht auch der UML-Spezifikation 2.5 [OMG-UML2c]. Zusätzlich lassen sich weitere zu beachtende Aspekte per UML-Kommentar ergänzen, die der Software-Architekt berücksichtigen muss.

4.2.2 Modellierung von verhaltensspezifischen Aspekten

Nachdem der vorige Abschnitt die strukturellen Aspekte fokussiert hat, geht es nun um die Modellierung der verhaltensspezifischen Aspekte. Dafür lassen sich UML-Sequenz- und UML-Zustandsdiagramme verwenden (vgl. Anhang A.4). Während im ersten Fall das Verhalten eine Folge von Interaktionen als Nachrichtenaustausch über die Zeit (entspricht der vertikalen Achse im Diagramm) ist, wird dies in einem Zustandsdiagramm als Zustandsübergang abgebildet. Hier wird die Zeit nicht durch eine Achse, sondern als das Durchlaufen eines endlichen Automaten repräsentiert. Darüber hinaus können sogenannte Invarianten definiert werden, die unabhängig von der Zeit und über die Dauer des entsprechenden aktiven Zustands gelten müssen (das wurde bereits bei der Modellierung der strukturellen Aspekte in Abschnitt 4.2.1 erwähnt, bei der die Invarianten die syntaktische Korrektheit

prüfen). Durch *Wächter* (engl. guards) lassen sich Bedingungen festlegen, die erfüllt sein müssen, um den Zustandsübergang auszulösen. In UML werden Invarianten üblicherweise nicht explizit modelliert, sondern fließen implizit in die Benennung von Zuständen ein: „A State models a situation in the execution of a StateMachine Behavior during which some invariant condition holds. In most cases this condition is not explicitly defined, but is implied, usually through the name associated with the State“ [OMG-UML2c, S. 306]. Im Hinblick auf eine systematische und nachvollziehbare Überführung auf die Implementierungsebene ist die implizite Modellierung jedoch wenig hilfreich, weshalb sich von dieser Aussage distanziert wird.

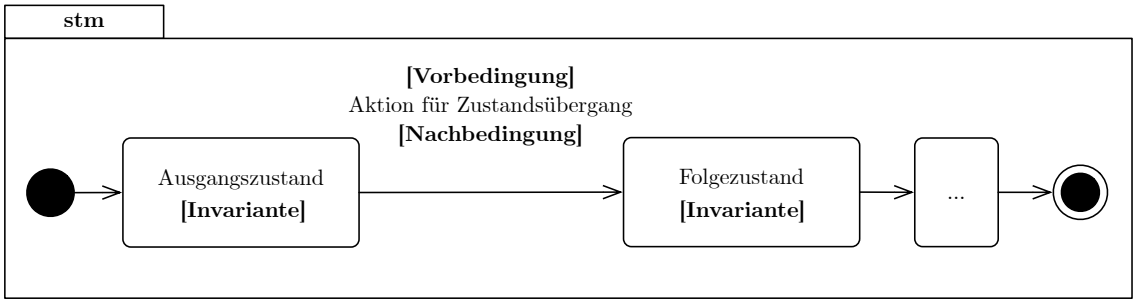


Abbildung 4.7: Verknüpfung von Design by Contract und UML-Zustandsdiagrammen

Die Möglichkeit durch Angabe von Invarianten und Wächter, die möglichen Zustände und Zustandsübergänge mit Bedingungen einzuschränken, lässt sich auf den *Design by Contract*-Ansatz von Betrand Meyer projizieren [Me92, PR11]. Dieser sieht eine formale und überprüfbare Schnittstellen-Spezifikation von Software-Komponenten mittels Vorbedingungen, Nachbedingungen und Invarianten vor, sodass er sich als logische Erweiterung von abstrakten Datentypen deuten lässt. Allerdings können mit üblichen UML-Zustandsdiagrammen nur Vorbedingungen mittels Wächter und Invarianten modelliert werden. Für Nachbedingungen existiert keine derartige Entsprechung bei UML-Zustandsdiagrammen. Deshalb können diese lediglich über Kommentare ergänzt werden. Allerdings existiert eine weitere Ausprägung eines UML-Zustandsdiagramms, das UML-Protokollzustandsdiagramm. Es beschreibt den Lebenszyklus einzelner Objekte und spezifiziert die Aufrufreihenfolge einzelner Operationen, wodurch letztlich ein Protokoll für mögliche Zustandsübergänge geschaffen wird [Se06]. Dieses erlaubt neben den zuvor genannten Bestandteilen auch die Modellierung von Nachbedingungen. Abbildung 4.7 zeigt die Verknüpfung von UML-Protokollzustandsdiagrammen und dem *Design by Contract*-Ansatz. Sie veranschaulicht, dass zu jedem Zustandsübergang eine Vorbedingung und eine entsprechende Nachbedingung gehören und dass die einzelnen Zustände entsprechende Invarianten haben. Für eine detaillierte und formale Spezifikation dieser Invarianten und Wächter kann die OCL aus Abschnitt 4.2.1 verwendet werden.

Die Modellierung von Verhaltensaspekten mithilfe von UML-Zustandsdiagrammen ist recht verbreitet, wie die Literatur insbesondere im Kontext von ressourcenorientierten Services [LS+09, RP+13] zeigt.

Diese Arbeit wählt deshalb UML-Zustandsdiagramme bzw. das UML-Protokollzustandsdiagramm. Letzteres beinhaltet wiederum Referenzen zu den Domänenobjekten, die im UML-Klassendiagramm durch Anwendung des UML-Profils DDD vorliegen. Bei dem abzubildenden Verhalten werden zwei Arten unterschieden [Ev03]:

(1) Abzubildendes Verhalten, das sich genau einem Domänenobjekt (*Entity* oder *ValueObject*) zuordnen lässt und

(2) abzubildendes Verhalten, das mehrere Domänenobjekte (*Entity* oder *ValueObject*) umfasst und sich nicht eindeutig einem dieser Domänenobjekte zuordnen lässt, um so bspw. einen Geschäftsprozess in seiner Gesamtheit abbilden zu können.

Diese unterschiedlichen verhaltensspezifischen Aspekte werden im Hinblick auf ihre Modellierung in den nächsten beiden Unterabschnitten detailliert beleuchtet.

Verhalten von Domänenobjekten

Das Verhalten von Domänenobjekten manifestiert sich in ihren Operationen, die ausschließlich den Zustand des jeweiligen Domänenobjekts verändern. Diese Art der Operationen lassen sich dabei auf das CRUD-Schema anwenden und unterliegen einem gewissen Lebenszyklus, den Evans in [Ev03, vgl. S. 123] schildert und der in leicht veränderter Form in Abbildung 4.8 dargestellt ist. Eine Leseoperation wird hier nicht explizit modelliert, da diese Methode seiteneffektfrei und idempotent sein sollte und demnach nicht den Zustand ändert. Deshalb muss sie nicht gesondert betrachtet werden. Beim Erstellen und Bearbeiten von Domänenobjekten ist dies jedoch anders, da diese Operationen in der Regel bestimmten Invarianten unterliegen, die gewährleisten, dass das Domänenobjekt die Integritätsbedingungen erfüllt. Eine solche könnte bspw. lauten, dass ein Attribut eines Domänenobjekts unter keinen Umständen einen höheren Wert als 1000 annehmen darf. Diese Integritätsbedingungen lassen sich auch aus einem UML-Klassendiagramm ableiten, sofern die entsprechenden Attribute über eine derartige Bedingung verfügen [Ka14].

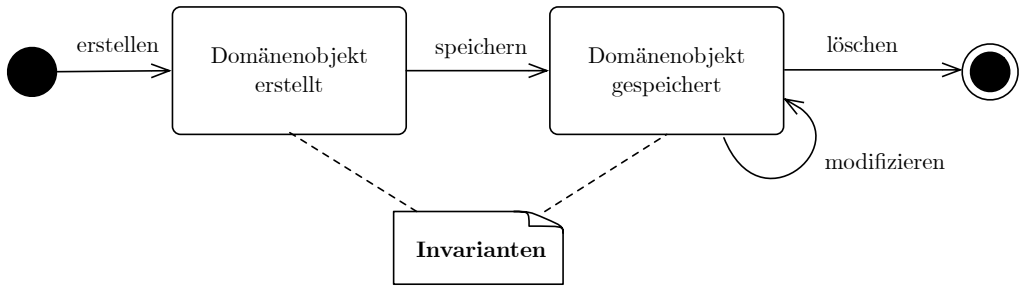


Abbildung 4.8: Lebenszyklus von Domänenobjekten nach CRUD

Kann das abzubildende Verhalten des Ausschnitts der Geschäftsdomäne im Rahmen des Bounded Context mit dem zuvor erwähnten Lebenszyklus abgebildet werden, so ist in dieser Arbeit eine separate Modellierung des Verhaltens in Form eines Protokollzustandsdiagramms nicht notwendig. Die entsprechenden Invarianten werden dem UML-Klassendiagramm entnommen, das die strukturellen Aspekte modelliert. Eine separate Modellierung ist etwa erforderlich, wenn weitere Invarianten und Vor- oder Nachbedingungen definiert und letztlich weitere Zustände betrachtet werden müssen.

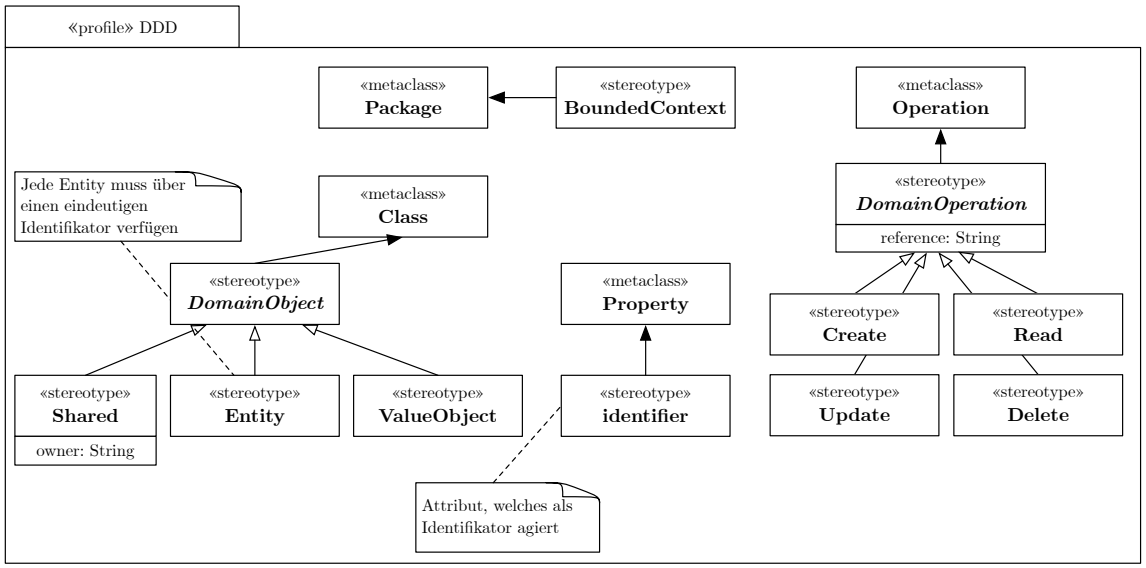


Abbildung 4.9: Erweitertes DDD-Profil zur Abbildung von CRUD-Operationen

Um die Domänenobjekte aus dem UML-Klassendiagramm mit dem entsprechenden Verhalten verknüpfen zu können, wurde das UML-Profil DDD um weitere Stereotypen ergänzt (siehe Abbildung 4.9). Durch die Stereotypen *Create, Read, Update, Delete* lassen sich Operationen annotieren, die mit dem zuvor eingeführten Lebenszyklus abgebildet werden können. Zu beachten ist hier, dass ein *ValueObject* keine Operation zum Bearbeiten wegen dessen Unveränderlichkeit besitzt.

Der Modellierungsaufwand lässt sich reduzieren, da bei der Durchführung von Projekten beobachtet wurde, dass eine Modellierung des Verhaltens nicht immer zwingend erforderlich ist. So muss eine Leseoperation nicht explizit modelliert und als entsprechende Operation angegeben werden, da sich das grundlegende Verhalten implizit aus den Attributen des Domänenobjekts ableiten lässt. In diesem Fall wird angenommen, jeder Zustand, der ein Domänenobjekt halten kann, lasse sich auch nach außen zur weiteren Verwendung geben. Gleiches gilt für Operationen für das Modifizieren und Erstellen, da auch sie sich implizit aus den Domänenobjekten ableiten lassen. So kann anhand von Multiplizitäten festgestellt werden, ob ein Attribut als optional zu betrachten ist oder nicht. Demnach ist eine Operation zum Erstellen und Modifizieren immer genau dann erfolgreich, wenn alle Pflichtattribute korrekt im Hinblick auf den zugewiesenen Datentyp befüllt sind. Eine Operation zum Löschen erfordert

ebenfalls keine explizite Modellierung, sofern keine gesonderten Bedingungen oder Invarianten zu beachten sind. Falls eine der CRUD-Operationen explizit nicht unterstützt werden soll, ist dies in einem entsprechenden Kommentar zu vermerken. Im Gegensatz dazu ist eine Modellierung genau dann sinnvoll, wenn das Verhalten nicht allein durch die Attribute bestimmt werden kann. So kann es bspw. sein, dass für eine Erstelloperation des Domänenobjekts ein anderer Datentyp verwendet wird, der in erster Linie unabhängig von den Attributen des Domänenobjekts ist. Dann muss eine Operation mit entsprechendem Eingabeparameter dem Domänenobjekt hinzugefügt werden, die mit der entsprechend durchzuführenden CRUD-Operation annotiert ist. Der Eingabeparameter ist dann die Vorbedingung. So kann später im Entwurfsprozess ermittelt werden, welche Eingabe eine Erzeugung des Domänenobjekts ermöglichen soll. Gleiches gilt für die übrigen CRUD-Operationen.

Falls weitreichende Invarianten und Bedingungen für eine Operation definiert werden müssen, reicht der reine Operationskopf, der bestimmte Vorbedingungen darstellt, nicht aus. Dann ist das abzubildende Verhalten explizit zu modellieren. Hierzu kann mittels der Eigenschaftsdefinition `reference` des entsprechenden Stereotyps ein Identifikator angegeben werden, der eindeutig einen Zustandsautomaten in Form eines Protokollzustandsdiagramms identifiziert. Darin können dann etwaige Vor- und Nachbedingungen und Invarianten definiert werden, die bei der Durchführung der jeweiligen Operation zu beachten sind. So lässt sich bspw. eine Erstelloperation definieren, die nur Attribute akzeptiert, deren Werte bestimmten Regeln unterliegen.

Verhalten von Domänenservices

Domänenservices unterscheiden sich von Domänenobjekten u. a. dahingehend, dass sie keinen eigenen Zustand halten können (vgl. Abschnitt 2.2.3). Bei der Modellierung werden jene in die strukturelle Darstellung des Bounded Context eingefügt und der entsprechende Stereotyp `DomainService` hinzugefügt. In diesem Zuge wurde das bestehende UML-Profil DDD in Abbildung 4.9 geringfügig erweitert. Das erweiterte UML-Profil findet sich in Abbildung 4.11. Obgleich die Abbildung von Domänenservices innerhalb der strukturellen Darstellung eigentlich der Trennung von Struktur und Verhalten widerspricht, ermöglicht dieses Verfahren andererseits eine ganzheitliche Sicht auf den abzubildenden Bounded Context. Gleichzeitig bildet der Domänenservice im späteren Verlauf des Entwicklungsprozesses ein entsprechendes Pendant innerhalb der Implementierung. Da sich die Domänenservices nicht auf eine einzelne CRUD-Operation reduzieren lassen, wurde der Stereotyp `Execute` eingeführt, der auch Bestandteil des erweiterten UML-Profils ist (siehe Abbildung 4.11). Für die Beschreibung des Verhaltens werden vergleichbar mit Abschnitt 4.2.2 entsprechende Vor- und Nachbedingungen definiert. Allerdings existieren im Gegensatz dazu keine Invarianten, da ein Domänenservice keinen eigenen Zustand hält.

Der innere Ablauf der offerierten Operation eines Domänenservice besteht aus Operationen, die wiederum mit Operationen der Domänenobjekte oder solchen von weiteren Domänenservices zusammenhängen. Diese Beziehungen zeigt auch das Meta-Modell in Abbildung 2.2. Der innere Ablauf, aus

weiteren Operationen bestehend, unterliegt einer bestimmten Reihenfolge und unter Umständen auch Kontrollflussbedingungen. Vergleicht man diese Anforderungen mit den vorhandenen Diagrammtypen der UML in Anhang A.4, so zeigt sich, dass sich hierfür das UML-Aktivitätsdiagramm besonders eignet [OMG-UML2c]. Eine Aktivität repräsentiert demnach die Operation eines Domänenservice, die sich wiederum als eine geordnete Folge von Aktionen darstellen lässt, wobei sich jede Aktion entweder auf die Operation eines Domänenservice oder die eines Domänenobjekts bezieht [Se06].

Namensschema für Aktionen

Operationstyp	::=	"Create" | "Read" | "Update" | "Delete" | "Execute"
Domänenobjekt	::=	Bezeichnung des Domänenobjekts aus dem UML-Klassendiagramm
Domainservice	::=	Bezeichnung des Domänenservice aus dem UML-Klassendiagramm
Aktionsname	::=	("«" external "»")? "«" Operationstyp "»" (Domänenobjekt | Domainservice)

Abbildung 4.10: Namensschema für die Benennung von Aktionen in einem UML-Aktivitätsdiagramm

Um die Aktionen einheitlich zu benennen, wird das Namensschema (vgl. Abbildung 4.10) eingeführt. Es stützt sich auf die Backus-Naur-Form (BNF), eine Metasprache zur Bildung kontextfreier Grammatiken. Dies erhöht die Konsistenz und ermöglicht zudem eine systematische und nachvollziehbare Ableitung, da die Mächtigkeit der natürlichen Sprache bei der Benennung eingeschränkt wird. Der Operationstyp ergibt sich aus den Stereotypen aus Abbildung 4.11, welche zur Abbildung von CRUD-Operationen dienen. Die möglichen Kombinationen ergeben sich aus dem kartesischen Produkt von Operationstyp und Bezeichnung der Domänenobjekte. Neben den grundlegenden Eigenschaften und Möglichkeiten durch UML-Aktivitätsdiagramme ist ein besonderer Vorteil die Modellierung der Vor- und Nachbedingungen, die direkt im Diagrammkopf angegeben werden kann. Für die genaue Modellierung dieser Bedingungen wird hier auf die UML-Spezifikation verwiesen [OMG-UML2c]. Sofern keine weitreichenden Bedingungen definiert werden müssen, lässt sich auf die Verwendung von OCL verzichten. Dies stärkt die Einfachheit und die Akzeptanz des Modellierungsverfahrens.

Abschnitt 4.1 hat die Möglichkeit diskutiert, dass als Folge des *Middle-out*-Ansatzes ein Bounded Context eine Abhängigkeit zu einem anderen aufweist. Dies ist insbesondere bei komplexeren Geschäftsprozessen der Fall, die nicht an die Grenzen eines Bounded Context gebunden sind. Um nun Operationen eines weiteren und benötigten Bounded Context verknüpfen bzw. ansprechen zu können (vgl. Abbildung 4.5), kann der Stereotyp `external` dem Aktionsnamen hinzugefügt werden. Durch einen entsprechenden Kommentar lässt sich schließlich der Bounded Context ergänzen, der diese Operation offeriert. Deshalb sind zunächst unbedingt die Bounded Contexts zu betrachten, die möglichst autonom agieren und ihre Dienstleistung selbstständig erbringen können. Zum Schluss sei

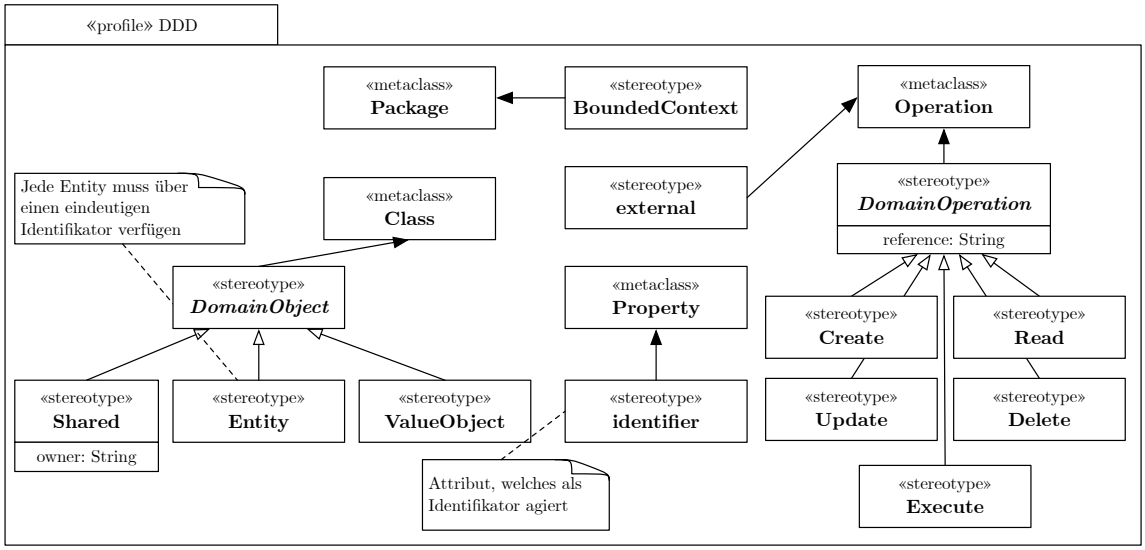


Abbildung 4.11: Vollständiges DDD-Profil

hier bemerkt, dass in der nun folgenden Überführung des Domänenmodells bzw. der darin enthaltenen Bounded Contexts in entsprechende Ressourcenmodelle (siehe Kapitel 5) dieser Fall nicht weiter betrachtet wird. Dieser Umstand wird erst im Zuge der Überführung auf die Implementierungsebene behandelt.

4.3 Betrachtetes Szenario zur Demonstration des domänengetriebenen Entwurfsansatzes

Bei dem betrachteten Szenario zur Demonstration des Entwurfsansatzes handelt es sich um ein Projekt im Rahmen des KIT-Qualipakt, das Studierende mit Sehbeeinträchtigungen unterstützen soll, damit sie sich besser auf dem Campus zurechtfinden. Das Projekt wurde in Zusammenarbeit mit dem Studienzentrum für Sehgeschädigte (SZS) durchgeführt, wobei die technische Umsetzung der Verantwortung des Instituts C&M unter Leitung von Herrn Prof. Abeck unterlag. Der KIT-Qualipakt wurde über mehrere Semester durchgeführt, in denen verschiedene Teilprojekte entworfen und entwickelt wurden, wie bspw. der AccessibilityInfoService (AIS) als Bestandteil des SmartCampus, der Informationen zur Barrierefreiheit auf dem Campus verfügbar macht [KIT-CM-SC]. Das nun beschriebene Projekt und gleichzeitig der Kontext des Szenarios ist eine logische Ergänzung des zuvor erwähnten AIS.

Projekt: Orientation with Beacons (OwB)

Das Projekt Orientation with Beacons (OwB) soll eine Ortung auf dem Campus des KIT vor allem in Gebäuden ermöglichen. Mit diesem Wissen können gezielt Informationen zur Barrierefreiheit

abgerufen und dem Studierenden bereitgestellt werden. Eine Positionsbestimmung in Gebäuden ist allerdings nicht unproblematisch, da sie sehr präzise sein muss. Deshalb ist bspw. auch das weitläufig genutzte Global Positioning System (GPS) hierfür ungeeignet. Stattdessen werden nun sogenannte Beacons verwendet, die in regelmäßigen zeitlichen Abständen Signale basierend auf Bluetooth 4.0 sowie einem festgelegten Protokoll (Apple iBeacon[3] oder Google Eddystone[4]) aussenden. Mit jedem Signal wird ein eindeutiger Identifikator übermittelt, mit dessen Hilfe ein Beacon eindeutig identifiziert werden kann. Sobald der Beacon mit einem Gebäude oder Raum verknüpft wird, lässt sich ermitteln, wo sich ein Studierender genau befindet. Sobald sich drei Beacons in der näheren Umgebung des Studierenden befinden, kann schließlich eine Triangulation durchgeführt werden, um die Position im Raum zu bestimmen. Eine Verknüpfung zum erwähnten AIS für den automatischen Abruf von Informationen zur Barrierefreiheit ist geplant, stand zum Zeitpunkt der Durchführung allerdings nicht im Fokus. Dennoch soll diese Arbeit demonstrieren, wie eine Verknüpfung aussehen könnte und wie sie sich im Sinne des domänengetriebenen Entwurfsansatzes realisieren ließe.

Auf Grundlage des beschriebenen Szenarios wurde in Zusammenarbeit mit Studierenden und unter Anwendung der genannten Konzepte zur Modellierung einer Domäne ein Domänenmodell erstellt. In Bezug auf die Demonstration wurde es so gewählt, dass möglichst viele Konzepte darin enthalten sind. Das Domänenmodell ist in Abbildung 4.12 dargestellt, während die Tabelle 4.2 eine leichtgewichtige Beschreibung der Begrifflichkeiten der Domäne liefert. Die vollständige Begriffssammlung würde an dieser Stelle keinen Mehrwert für die Demonstration des Entwurfsansatzes liefern, weshalb an dieser Stelle explizit darauf verzichtet wird.

Begrifflichkeit	Bedeutung
Accessibility-Information	Spezifizieren Informationen über einen Point-of-Interest, die für Personen mit Behinderung oder chronischen Krankheiten relevant sind. Dazu gehören u. a. Zugänge oder auch Fahrstühle für die Erreichung des Points-of-Interest.
Beacon	Ein Gerät mit einer Signal-Reichweite von 20 bis 30 Metern (abhängig von der Sendeleistung), das in regelmäßigen Zeitabständen einen eindeutigen Identifikator übermittelt, der von Bluetooth 4.0-kompatiblen Endgeräten empfangen werden kann.
Region	Eine Region bietet die Möglichkeit, den Point-of-Interest weiter zu unterteilen. Ein Point-of-Interest ist selbst eine eigene Region.
Building	Ein zum Campus gehörendes Bauwerk mit Etagen, Räumen und Gängen.
Elevator	Anlage, um Personen oder Lasten, in einer beweglichen Kabine in vertikaler Richtung zwischen zwei oder mehreren Etagen zu transportieren.
Floor	Zugangsebene in einem Gebäude zu horizontal miteinander verbundenen Räumen.

[3] https://developer.apple.com/ibeacon/ (Letzter Zugriff: 03.10.2017)
[4] https://github.com/google/eddystone (Letzter Zugriff: 03.10.2017)

Begrifflichkeit	Bedeutung
Position	Geographische Position eines Dings. Diese ist definiert als die letzte bekannte solche.
PointOfInterest	Ort, der als interessant für eine Person gilt und mit Koordinaten versehen ist.
Room	Ort in einem Gebäude, der durch Wände begrenzt und durch mindestens eine Tür betretbar ist. Zudem ist er meist eindeutig bezeichnet.
Toilet	Raum, der vor allem sanitäre Anlagen wie Toiletten und Waschbecken enthält.

Tabelle 4.2: Verwendete Begrifflichkeiten zur Abbildung der Domäne im Rahmen des Projekts OwB

4.4 Zusammenfassung

In diesem Kapitel wurde zunächst der domänengetriebene Entwurfsansatz in einen etablierten Entwicklungsprozess eingeordnet sowie die verschiedenen Rollen in letzterem erläutert. Dabei wurde dieser Entwicklungsprozess mit den grundlegenden Konzepten von DDD nach Evans [Ev03] verknüpft und somit ein direkter Bezug zu diesem Ansatz hergestellt. Danach wurden die Aktivitäten des Entwurfsprozesses für eine ganzheitliche Übersicht auf Prozessebene fokussiert.

Schließlich wurden Konzepte zur Modellierung der Domänenobjekte im Rahmen der betrachteten Bounded Contexts erläutert, aus denen sich die für den Entwurfsprozess in Kapitel 5 benötigten Modellierungsartefakte gewinnen lassen. Während das UML-Klassendiagramm die strukturellen Aspekte unter Anwendung des UML-Profils DDD beschreibt (siehe Abbildung 4.11), fokussieren die UML-Protokollzustandsdiagramme die verhaltensspezifischen Aspekte der Domänenobjekte. Mithilfe von UML-Aktivitätsdiagrammen wird wiederum jenes Verhalten adressiert, das sich keinem Domänenobjekt eindeutig zuweisen lässt, um so bspw. einen Geschäftsprozess ganzheitlich in einem Domänenservice abzubilden. Die einzelnen Aktionen als Teil einer Aktivität verweisen wiederum auf Operationen von Domänenobjekten. Die Domänenobjekte können dabei im abzubildenden Bounded Context liegen oder aber in anderen Bounded Contexts verortet sein (vgl. Abbildung 4.5).

Wird das Domänenmodell auf die Hexagonal-Architektur nach dem DDD-Ansatz bezogen [Ve13], so zeigt sich deren Einfluss auf die letztliche Architektur (siehe Abbildung 4.13) [Ve13]. In der *Domäne* manifestieren sich sowohl die strukturellen als auch die verhaltensspezifischen Aspekte des Ausschnitts der Geschäftsdomäne bzw. des Bounded Context, weshalb das UML-Klassendiagramm und ebenso etwaige UML-Zustandsdiagramme diesen Bestandteil der Architektur prägen. Hier wird allerdings nur das Verhalten betrachtet, das genau einem Domänenobjekt zugeordnet ist. Mit UML-Aktivitätsdiagrammen wird dagegen das übergeordnete Verhalten adressiert, das die Grundlage für die *Domänenservices* darstellt. Die *Anwendungsservices* werden nicht explizit betrachtet, da diese nur als Bindeglied zwischen Ports/Adapters und Domäne/Domänenservices aufgefasst werden. An dieser Stelle sei auf den Abschnitt 4.2 verwiesen, wo dieser Umstand und die letztliche Begründung für dieses

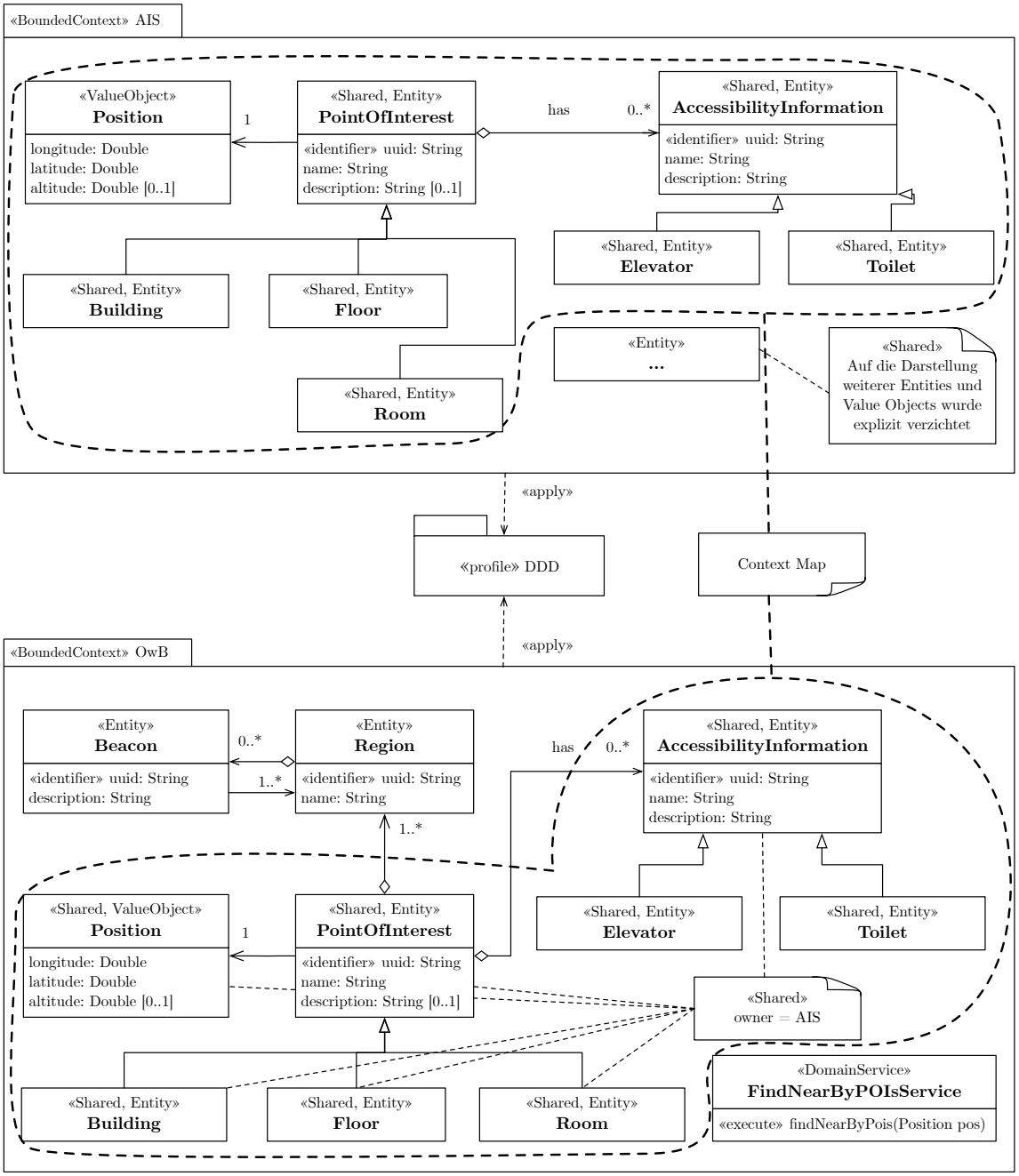


Abbildung 4.12: Domänenmodell des OwB-Projekts

Vorgehen erläutert wurde. Die Modellierung der Ports/Adapters in Form einer ressourcenorientierten Web-API ergibt sich aus den in Kapitel 5 beschriebenen Aktivitäten, die das Domänenmodell und insbesondere dessen Bounded Contexts als zugrundeliegende Artefakte diskutieren.

Die Anwendung dieser Konzepte zur Domänenmodellierung spiegelt sich wiederum in dem Szenario

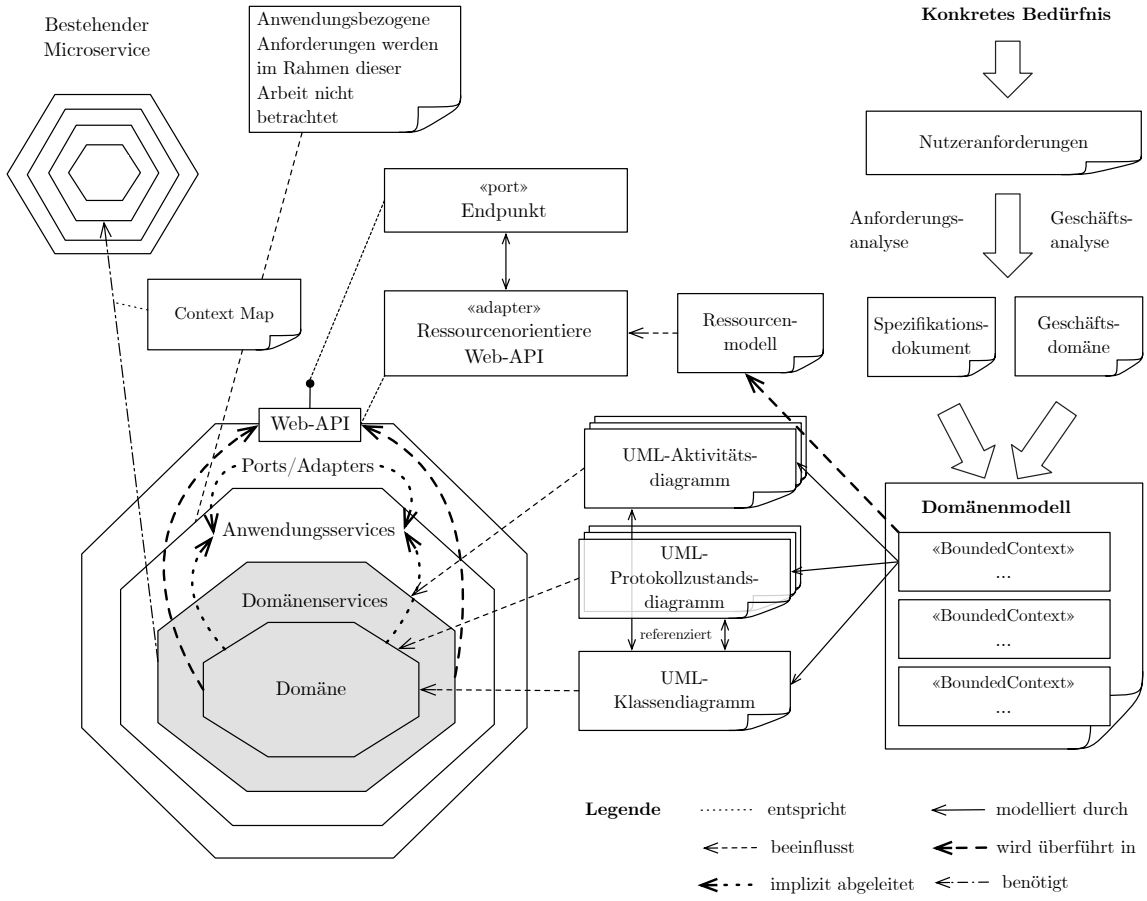


Abbildung 4.13: Einfluss der Artefakte auf die Hexagonal-Architektur nach DDD

in Abschnitt 4.3 wider, das anschließend den Entwurfsprozess demonstrieren soll. Gleichzeitig wurde damit gezeigt, dass sich mit diesen Konzepten der relevante Ausschnitt einer Geschäftsdomäne aus struktureller und auch aus verhaltensspezifischer Sicht für die Realisierung der geschäftsbezogenen Anforderungen abbilden lässt. Die anwendungsbezogenen Anforderungen müssen unterdessen durch den Service-Konsumenten selbst oder einen dedizierten Microservice erbracht werden.

5 Ressourcenorientierter Entwurf von Microservices

Nachdem mit dem vorherigen Kapitel eine Übersicht über den Entwurfsprozess sowie eine Beschreibung des Ausgangspunkts und der damit einhergehenden Modellierungsartefakte gegeben wurde, widmet sich dieses Kapitel der Beschreibung der einzelnen Entwurfsaktivitäten und damit der systematischen und nachvollziehbaren Überführung des Domänenmodells in einen ressourcenorientierten Entwurf von Microservices. Dieser kann als abgeschlossen gelten, wenn für alle Bounded Contexts des Domänenmodells die einzelnen Entwurfsaktivitäten durchgeführt wurden (vgl. Abbildung 4.3 aus Abschnitt 4.1.3 und Abbildung 5.1). Die Demonstration der Überführung wird anhand des Szenarios in Abschnitt 4.3 veranschaulicht.

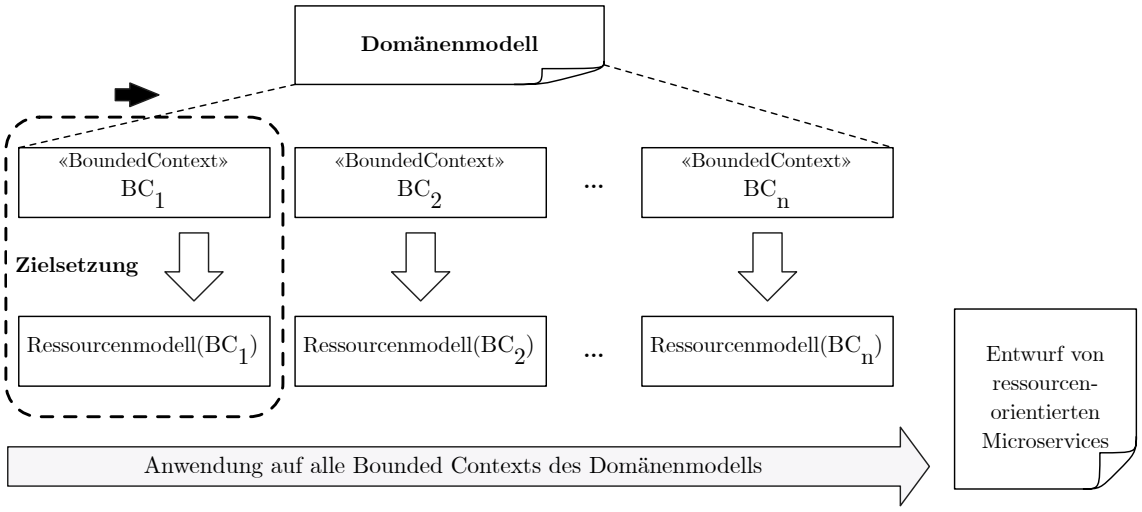


Abbildung 5.1: Übersicht über die Zielsetzung von Kapitel 5

Mit diesem Kapitel soll die zentrale Fragestellung dieser Arbeit geklärt werden, wie aus den vorliegenden Modellierungsartefakten eine systematische und nachvollziehbare Ableitung erfolgen kann, während gleichzeitig die geforderten Qualitätsaspekte der Wiederverwendbarkeit berücksichtigt werden (vgl. Abschnitt 2.3.2). Darüber hinaus muss ein Software-Architekt bei den Entwurfsentscheidungen unterstützt werden, die stets auch die Qualität beeinflussen. Aus diesem Grund wird im Kontext dieses Kapitels auch die Frage aufgegriffen, wie ein Software-Architekt bei der Überführung des Domänenmodells bzw. der Bounded Contexts in jeweils ein Ressourcenmodell in geeigneter Form unterstützt werden kann. Nur durch die Aufbereitung von entsprechendem Wissen kann dieser schnell fundierte und nachvollziehbare Entscheidungen im Hinblick auf die letztliche Qualität treffen, ohne

dass er sich tiefer mit dem spezifischen Thema auseinandersetzen muss. Das Kapitel orientiert sich dabei an den Aktivitäten des iterativen und inkrementellen Entwurfsprozesses in Abschnitt 4.1.3 und liefert für diese jeweils eine ausführliche Beschreibung, um eine nachvollziehbare Überführung zu gewährleisten.

Das Kapitel ist wie folgt strukturiert: Abschnitt 5.1 behandelt die Identifikation von Ressourcen im Rahmen eines Bounded Context entsprechend Abschnitt 4.2, um ein initiales *Ressourcenmodell* R_1 abzuleiten. Auf dieses gestützt, wird dann in Abschnitt 5.2 die Adressierung unter Betrachtung der geforderten Qualitätsaspekte der Wiederverwendbarkeit abgeleitet, über welche die zuvor identifizierten Ressourcen erreichbar sein sollen. Das gewonnene Artefakt wird als *Ressourcenmodell* R_2 bezeichnet. Abschnitt 5.3 fokussiert schließlich die verhaltensspezifischen Aspekte und deren Abbildung auf eine einheitliche und ressourcenorientierte Web-API, mit der dann Service-Konsumenten interagieren können. Für die Abbildung werden mehrere Muster betrachtet, welche u. a. die geforderten Qualitätsaspekte der Wiederverwendbarkeit in unterschiedlicher Weise beeinflussen und so eine Entscheidungsgrundlage liefern. Auf Basis des so gewonnenen *Ressourcenmodells* R_3 ergänzt Abschnitt 5.4, wie sich dafür entsprechende Repräsentationen ableiten lassen. Letztere sind die eigentlichen Nachrichteninhalte zwischen Service-Konsumenten und einem Microservice. Das Kapitel endet mit der Versionierung des *Ressourcenmodells* R_3 in Abschnitt 5.5, der die Veränderung des Ressourcenmodells und deren Auswirkung auf die Web-API fokussiert.

5.1 Identifikation von Ressourcen

Die Identifikation von Ressourcen erfolgt auf Basis der zugrundeliegenden Modellierungsartefakte wie es in Kapitel 4 dargelegt wurde. Dabei werden die identifizierten Modellierungselemente unter Anwendung des ressourcenorientierten Stils in ein Ressourcenmodell überführt. Für die Modellierung des Ressourcenmodells wird wie bei der Modellierung der strukturellen Aspekte eines Bounded Context ein UML-Klassendiagramm verwendet. In dieser Hinsicht gesellt sich die vorliegende Arbeit zu den schon in diesem Bereich verfügbaren [LS+09, RP+13, Sc14, HK+14, HL+15, Ro16].

Dieser Abschnitt untersucht folgende Fragen, um eine nachvollziehbare Identifikation von Ressourcen sowie deren Überführung in ein Ressourcenmodell sicherzustellen.

(1) Welche verschiedenen Ressourcenarten (engl. resource types) sind bei der Identifikation zu berücksichtigen?

(2) Welche zugrundeliegenden Modellierungsartefakte aus Abschnitt 4.2 ermöglichen die Identifikation der zu berücksichtigenden Ressourcenarten aus (1)?

(3) Wie können die Ressourcenarten aus (1) und die Modellierungsartefakte aus (2) eine systematische und nachvollziehbare Ableitung ermöglichen?

5.1.1 Kategorisierung von Ressourcen mittels Ressourcenarten

Für die Beantwortung der Frage (1) wurde die bestehende Literatur konsultiert, um verfügbare Arbeiten in diesem Forschungsgebiet auszuwerten. Diesbezüglich identifizierten Siikarla et al. [SL+08], Selonen [Se11], Rathod et al. [RP+13] und Tilkov et al. [TE+15] verschiedene Ressourcenarten, die in Tabelle 5.1 aufgeführt sind. Dabei wurde mit ● gekennzeichnet, ob die jeweilige Ressourcenart aus semantischer Sicht in der entsprechenden Arbeit verwendet wurde. Entsprechend repräsentiert ○ das Gegenstück, dass die Arbeit nicht diese Ressourcenart berücksichtigt. Hingegen weist ◐ darauf hin, dass diese Ressourcenart zwar nicht explizit erwähnt wird, sich ihre Verwendung dennoch aus dem Kontext der Arbeit ableiten lässt. Abschließend wird mit ◐ markiert, dass diese Ressourcenart zwar erwähnt ist, ohne jedoch weiter erörtert zu werden, oder dass ihre Rolle in der Forschungsarbeit unklar bleibt.

Aus der Tabelle 5.1 wird ersichtlich, dass zwischen den untersuchten Arbeiten Übereinstimmungen existieren. So lassen sich die Ressourcenarten R_{Prim}, R_{Sub} und R_{List} als Vereinigungsmenge aller Arbeiten auffassen. Gleichzeitig unterscheiden sich die Arbeiten in ihrer Granularität. So liefern Tilkov et al. [TE+15] eine sehr feingranulare Sicht, während bspw. Selonen et al. [Se11] auf Grundlage des Ressourcenmodells von Laitkorpi et al. [LS+09] lediglich eine Untermenge davon in ihrem Ansatz nutzen. Rathod et al. [RP+13] betrachten zwar eine vergleichsweise große Teilmenge von Tilkov et al., nehmen darauf aber keinerlei weiteren Bezug bei der Modellierung von RESTful Web Services.

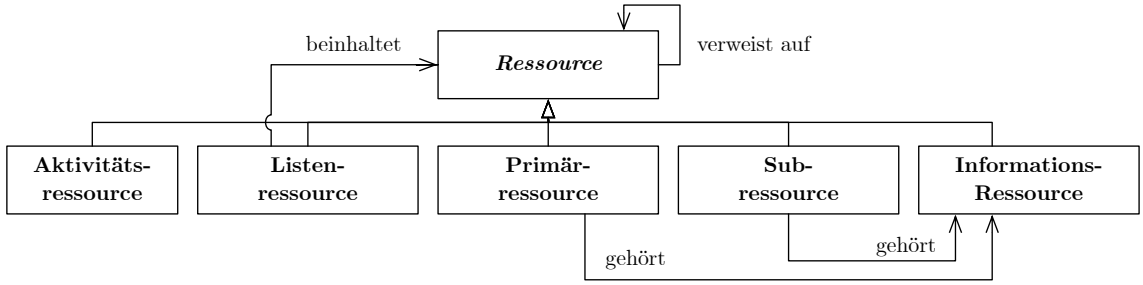


Abbildung 5.2: Meta-Modell der verschiedenen Ressourcenarten

Für eine möglichst hohe Abdeckung im Rahmen dieser Arbeit sind für die folgende Betrachtung die Ressourcenarten von Tilkov et al. die maßgebliche Orientierung. Das Meta-Modell in Abbildung 5.2 setzt diese Ressourcenarten in einen logischen Bezug zueinander und baut auf dem hierarchischen Meta-Modell von Schreier [Sc11] auf. Dabei wurde auf etwaige Multiplizitäten oder Attribute aus Gründen der Übersichtlichkeit verzichtet. Auf R_{Proj} und R_{Con} wurde an dieser Stelle verzichtet, da eine Projektionsressource lediglich eine weitere Repräsentation einer bestehenden Ressource und eine Konzeptressource lediglich eine Listenressource für Informationsressourcen darstellt. Zudem fehlt die Betrachtung der R_{Agg}, da sie anwendungsbezogene Anforderungen voraussetzt und sich diese eher in einem Microservice nach dem BFF manifestieren sollte (vgl. Abbildung 4.4 aus Abschnitt 4.2). Wei-

Ressourcenarten	Abk.	Beschreibung	[SL+08]	[Se11]	[Sc11]	[RP+13]	[TE+15]
Primärressource	R_{Prim}	Ressource, die den wesentlichen Domänenobjekten einer Domäne entspricht und direkt adressierbar ist	◐	◐	●	◐	●
Subressource	R_{Sub}	Ressource, die ein logischer Bestandteil einer anderen Ressource ist und direkt adressierbar ist	◐	●	●	◐	●
Listenressource	R_{List}	Ressource, die mehrere Ressourcen als Listenelement haben kann	●	●	●	●	●
Filterressource	R_{Filt}	Listenressource, bei der die Listenelemente bestimmte Eigenschaften aufweisen	○	●	●	(●)	●
Projektionsressource	R_{Proj}	Ressource, die nur eine Untermenge von Attributen einer anderen Ressource enthält	○	○	●	(●)	●
Aggregationsressource	R_{Agg}	Ressource, die Attribute von mehreren Ressourcen vereint	○	○	●	(●)	●
Aktivitätsressource	R_{Act}	Ressource, die ein Prozessschritt eines fachlichen Geschäftsprozesses darstellt	○	○	●	(●)	●
Paginierungsressource	R_{Pag}	Listenressource, deren Listenelemente auf mehrere virtuelle Seiten aufgeteilt wird	○	○	●	(●)	●
Informationsressource	R_{Inf}	Ressource, die sich nicht selbst dereferenzieren kann, aber einer Identität zugeordnet ist	○	○	○	○	●
Konzeptressource	R_{Con}	Ressource, die genutzt werden kann, um zusätzliche Informationsressourcen anzulegen	○	○	○	○	●

Tabelle 5.1: Kategorisierung von Ressourcen nach [SL+08, Se11, Sc11, RP+13, TE+15]

tere Besonderheiten im Vergleich zu Schreier und Tilkov et al. sind die R_{Pag} und R_{Filt}, die hier nicht als eigenständige Ressourcenarten, sondern als Fähigkeiten einer R_{List} (siehe Abbildung 5.3) interpretiert werden, die allerdings auch in Kombination auftreten können. Deshalb wurde das strukturelle Entwurfsmuster *Dekorierer* gewählt, mit dessen Hilfe sich eine R_{List} um Fähigkeiten erweitern lässt,

ohne dass die Bildung von Unterklassen erforderlich wäre [GH+94]. In den folgenden Betrachtungen gilt R_{List} als Listenressource ohne zusätzliche Fähigkeiten. Für die Bezeichnung der Fähigkeit einer R_{List} zur Paginierung wurden $R_{List_{Pag}}$ und entsprechend für die Filterung $R_{List_{Filt}}$ gewählt. Die Kombination beider Fähigkeiten ergibt $R_{List_{Pag,Filt}}$. Das Meta-Modell der verschiedenen Ressourcenarten wird im weiteren Verlauf der Arbeit insbesondere für die Ableitung der Adressierung in Abschnitt 5.2 benötigt.

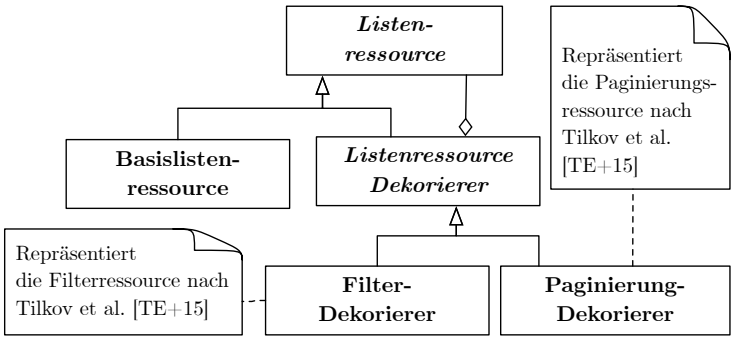


Abbildung 5.3: Fähigkeiten einer Listenressource

5.1.2 Identifikation der relevanten Modellierungsartefakte

Nachdem die endgültige Menge an zu betrachtenden Ressourcenarten identifiziert und damit die Fragestellung (1) beantwortet wurde, stellt sich nun die Frage (2), aus welchen zugrundeliegenden Modellierungsartefakten (nachfolgend als *Artefakte* bezeichnet) aus Abschnitt 4.2 diese ermittelt werden können. Dieses Vorgehen ist erforderlich, da für die Nachvollziehbarkeit erkennbar sein muss, welche Artefakte die Identifikation von Ressourcen begünstigen bzw. überhaupt erst ermöglichen. Nur so kann der Software-Architekt die entscheidenden Artefakte zur Erledigung heranziehen.

Die Ressourcenarten $R_{Prim}, R_{Sub}, R_{List}, R_{Inf}$ und R_{Act} lassen sich aus den strukturellen Aspekten des Domänenmodells ableiten und damit aus dem UML-Klassendiagramm. Obgleich die Fähigkeiten einer R_{List} keine eigenen Ressourcenarten darstellen, sind sie der Vollständigkeit wegen hier genannt. Die Fähigkeiten einer R_{List} lassen sich nicht aus den zugrundeliegenden Modellierungsartefakten ableiten, sondern erfordern ein weiteres Artefakt, das Rückschlüsse auf nutzungsspezifische Anforderungen ermöglicht (siehe die folgende Definition). Weiter korrelieren die Fähigkeiten positiv mit der Mächtigkeit einer Web-API.

> **Definition (Nutzungsspezifische Anforderungen):** *Nutzungsspezifische Anforderungen sind Anforderungen, die sich aus der Abstraktion von Nutzeranforderungen ergeben und unabhängig von einer konkreten Anwendung oder etwaigen Anwendungsgruppen sind.*

So kann bspw. eine Filterung von Informationen von verschiedenen Service-Konsumenten wiederverwendet werden, sobald die Filterung über entsprechende Parameter durch die jeweiligen Service-Konsumenten gesteuert werden kann. Als Quelle für etwaige nutzungsspezifische Anforderungen kann bspw. ein UI-Prototyp oder eine anderweitig geartete Spezifikation im Kontext der Beschreibung von anwendungsbezogenen Anforderungen dienen (vgl. Abschnitt 4.1.1). Den Zusammenhang zwischen den betrachteten Ressourcenarten und zugrundeliegenden Artefakten zeigt Abbildung 5.4.

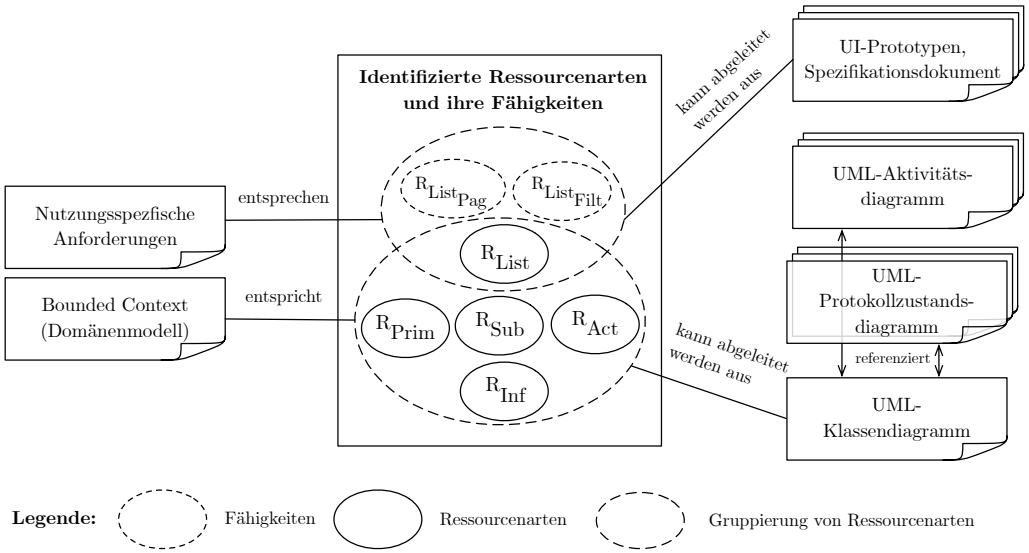


Abbildung 5.4: Erweiterte Kategorisierung von Ressourcen auf Grundlage der Artefakte für den Entwurf aus Abschnitt 4.2

5.1.3 Ableitung der Ressourcenarten aus Modellierungsartefakten

Nachdem die zu betrachtenden Ressourcenarten (R_{Prim}, R_{Sub}, R_{List}, R_{Inf} und R_{Act}) und die relevanten Artefakte aus dem vorherigen Abschnitt identifiziert sind (entspricht der Beantwortung von Fragestellung (2)), müssen passende Überführungsregeln bereitgestellt werden, um geeignete Kandidaten für Ressourcen aus dem Domänenmodell bzw. dem betrachteten Bounded Context abzuleiten und in ein Ressourcenmodell zu überführen. Obgleich es sich um hinreichend gute Kandidaten handelt, da die Geschäftsanalyse auf die Nutzeranforderungen ausgerichtet wurde, muss der Software-Architekt entscheiden, ob das Domänenobjekt oder der Domänenservice offeriert wird. Dies ist bspw. dann der Fall, wenn bei der Geschäftsanalyse ein Domänenobjekt identifiziert wurde, das im Hinblick auf die Nutzeranforderungen nicht benötigt wird. Das gilt ebenso für die verhaltensspezifischen Aspekte, die auch in diesem Sinn beurteilt werden müssen. Die Fähigkeiten einer Listenressource sowie weitere Fähigkeiten anderer Ressourcenarten werden in Abschnitt 5.3 näher betrachtet. Für die Behandlung von geteilten Domänenobjekten (Stereotyp `Shared`) wird auf Abschnitt A.5 verwiesen.

Für die Überführungsregeln wurden die bestehenden Arbeiten [LS+09, RP+13, TE+15] untersucht.

Es zeigte sich, dass bspw. weder Rathod et al. [RP+13] noch Tilkov et al. [TE+15] beschreiben, wie Ressourcenarten systematisch aus einem zugrundeliegenden Domänenmodell nach DDD identifiziert und in ein Ressourcenmodell überführt werden können. Laitkorpi et al. [LS+09] liefern an dieser Stelle lediglich eine informelle Aussage, die keine systematische Überführung ermöglicht und kein Bezug zu den identifizierten Ressourcenarten herstellt: „Find the resources and their interconnecting links by harvesting all the relevant nouns and their relationships in the problem domain“ [LS+09, S. 173]. Hingegen liefern Rathod et al. einen Ansatzpunkt in Form von Heuristiken, die beschreiben, wie Klassen in UML und deren Beziehungen in einem UML-Klassendiagramm auf das Konzept von REST abgebildet werden können. Heuristiken sind Hilfstechniken, um praktikable Lösungen trotz beschränkten Wissens abzuleiten: „they are intended to capture how real minds make decisions under constraints of limited time and knowledge“ [GT+99, S. 5]. Allerdings wird auch bei diesem Ansatz das Verständnis erschwert, da UML nicht für die ressourcenorientierte Modellierung ausgelegt ist und so beispielsweise Termini wie *Primärressource* oder *Subressourcen* keine Modellelemente sind. Ein Bezug zu den identifizierten Ressourcenarten wird daher auch hier nicht hergestellt. Deswegen wird zunächst ähnlich zu der Modellierung der strukturellen Aspekte eines Bounded Context (siehe Abschnitt 4.2.1) ein UML-Profil eingeführt, das UML um die Fähigkeit zur Modellierung von Ressourcen unter Berücksichtigung der verschiedenen Ressourcenarten erweitert.

Einführung eines UML-Profils für Ressourcenorientierung

Der Entwurf des UML-Profils stützt sich auf das Meta-Modell in Abbildung 5.2 basierend auf [Sc11], das ein Vokabular für den Entwurf von RESTful Web Services darstellt. Das UML-Profil wurde *ResourceOrientation* genannt, um klar von REST und dem zentralen Hypermedia-Aspekt abgrenzbar zu sein. Einen vergleichbaren Ansatz mit UML-Profilen verfolgen Laitkorpi et al. [LS+09], wobei die Stereotypen in diesem UML-Profil keinen direkten Bezug zu REST ermöglichen. Auch werden in diesem Ansatz die verhaltensspezifischen Aspekte mithilfe von Stereotypen in das UML-Klassendiagramm modelliert, was das Verständnis zusätzlich erschwert. Ein weiteres UML-Profil für REST wird in [SO+14] und [Ro16] geliefert. Doch auch diese Autoren stellen keinen direkten Bezug zu den Ressourcenarten her. So geht semantisches Wissen verloren. Auch liefern diese Ansätze keinen direkten Bezug zu dem DDD-Ansatz und den damit einhergehenden Begrifflichkeiten nach Evans [Ev03], obgleich der Ansatz von Sanchez et al. [SO+14] auf eine Art Domänenmodell beruht.

Das UML-Profil *ResourceOrientation* (siehe Abbildung 5.5) bildet die Basis, um Regeln zur Überführung der Domänenobjekte aus einem Bounded Context in ein Ressourcenmodell zu definieren. Darin enthalten sind die Stereotypen für die verschiedenen Ressourcenarten gemäß Abschnitt 5.1 sowie der Stereotyp `identifier`, der die Ausprägung einer Ressource eindeutig identifiziert. Zusätzlich findet sich im UML-Profil der Stereotyp `link`, da Ressourcen über eine Verknüpfung in Form von Hyperlinks verbunden sind. Zur Abbildung der Operationen etwaiger Domänenobjekte wird die *DomainOperation* aus dem UML-Profil für DDD genutzt und in *ResourceOperation* umbenannt, da

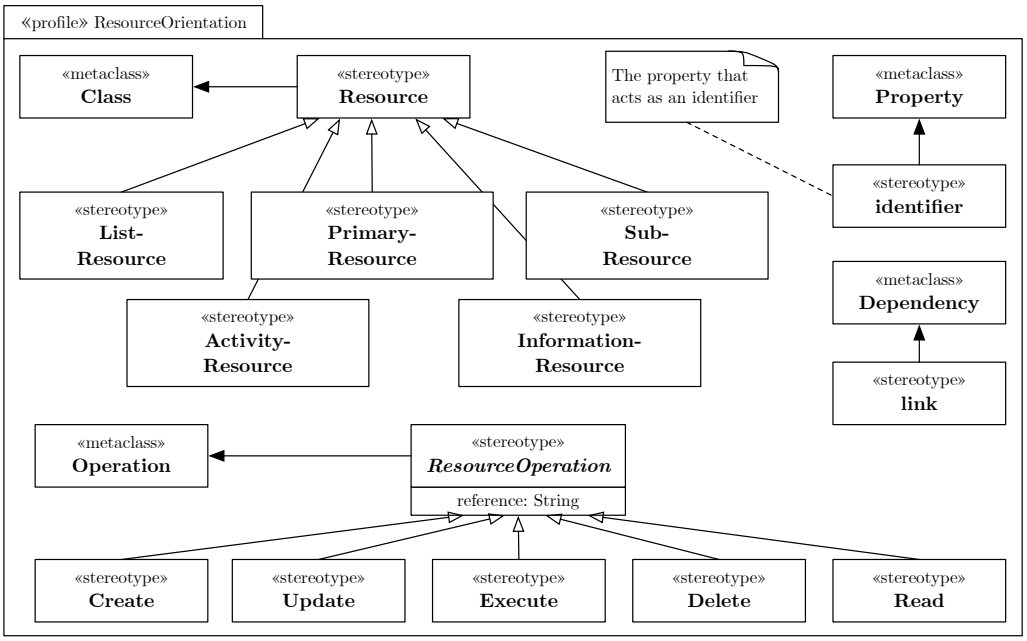


Abbildung 5.5: UML-Profil zur Abbildung von Ressourcen innerhalb eines UML-Klassendiagramms

die Bezeichnungen der Operationen keinen Bezug zu einem Anwendungsschichtprotokoll herstellen, wie es in bestehenden Ansätzen [RP+13, HL+15, Ro16] sonst der Fall ist. Die Wahl des Anwendungsschichtprotokolls ist allerdings eine technologische Entscheidung, die erst bei der Überführung des Entwurfs betrachtet werden muss (siehe Abschnitt 6.1).

Besonderheiten bei der Überführung der strukturellen Aspekte

Die Modellierung der strukturellen Aspekte erfolgt mithilfe des objektorientierten Paradigmas gemäß Evans [Ev03] (vgl. Abschnitt 4.1.1), weshalb sich darin bspw. auch Vererbungshierarchien abbilden lassen. Allerdings können mit einem ressourcenorientierten Stil Vererbungshierarchien nicht in ähnlicher Weise abgebildet werden, da sich die Ressourcen letztlich in der URL widerspiegeln. Deswegen sind im Ressourcenmodell kleinere Umformungen erforderlich. Die bisher untersuchten Ansätze haben Auflösungen von Vererbungen (auch als *Generalisierungsbeziehungen* bezeichnet) nicht betrachtet. Allerdings kann sich in diesem Fall der Konzepte aus dem Bereich des Object Relational mapping (ORM) bedient werden, die den *impedance mismatch* in adäquater Weise behandeln [IB+09]. Auf diesen Konzepten basieren mehrere nun vorgestellte Ansätze, die helfen, Vererbungen bei den Modellierungen der strukturellen Aspekte eines Bounded Context aufzulösen. Eine Überführung muss in beide Richtungen möglich sein – also ein Domänenobjekt in eine Ressource und ebenso eine Ressource in ein Domänenobjekt bzw. die Repräsentation einer Ressource in ein instanziiertes Domänenobjekt.

(1) **Single Resource Inheritance (SRI):** Bei *SRI* vereint die Ressource alle Attributwertpaare, die zu einer Generalisierungsbeziehung gehören. Das ergibt eine Repräsentation einer Ressource, bei der nicht immer alle Attribute mit einem Wert besetzt sein müssen. Das konkrete Domänenobjekt wird dann lediglich über die mit einem Wert besetzten Attribute einer Repräsentation bestimmt. Dieser Ansatz ähnelt dem *Single Table Inheritance*-Muster aus [Fo02a].

(2) **Single Resource Inheritance with unique Type (SRIT):** *SRIT* verfolgt einen vergleichbaren Ansatz wie *SRI*. Der Unterschied betrifft nur die Form, wie ein konkretes Domänenobjekt ermittelt wird. Hierzu wird die Ressource um ein zusätzliches Attribut `_type` ergänzt, das mit einem beschränkten Set an Werten belegt werden kann. Diese ergeben sich aus den einzelnen Vererbungshierarchien, sodass sich mit ihnen das konkrete Domänenobjekt ermitteln lässt und die Ermittlung mithilfe der besetzten Attribute nicht mehr notwendig ist (vgl. *SRI*).

(3) **Multiple Resource Inheritance (MRI):** Bei *MRI* wird jede Vererbungshierarchie einer Ressource als eigenständige Ressource abgebildet. Verknüpfungen in Form von Hyperlinks zwischen den verschiedenen Ressourcen bzw. deren Repräsentationen erlauben es, die Vererbungshierarchie der abgebildeten Domänenobjekte zu rekonstruieren. Dieser Ansatz ähnelt dem *Class Table Inheritance*-Muster aus [Fo02a].

(4) **Abstract Multiple Resource Inheritance (AMRI):** *AMRI* ist mit *MRI* vergleichbar und unterscheidet sich lediglich darin, dass die oberste Vererbungshierarchie als *abstrakt* definiert wird. Dadurch wird diese Vererbungshierarchie durch keine eigenständige Ressource abgebildet, sondern vererbt lediglich die Attribute. Wird dieser Ansatz mit den Mustern in [Fo02a] verglichen, so ist er der *Concrete Table Inheritance* gleichzusetzen.

Den Ansatz zur Auflösung von Vererbungen muss der Software-Architekt wählen. Um ihn bei der Wahl zu unterstützen, stellt die Tabelle 5.2 die verschiedenen Ansätze einander gegenüber. Dabei gilt ● als *zutreffend* und ○ als *nicht zutreffend*.

	SRI	SRIT	MRI	AMRI
Resultiert in einer hohen Anzahl von Ressourcen	○	○	●	●
Resultiert in einer geringen Anzahl von Ressourcen	●	●	○	○
Mit Wert besetzte Attribute entscheiden über konkrete Ausprägung	●	○	○	○
Eindeutiges Attribut entscheidet über konkrete Ausprägung	○	●	○	○
Ressourcenbezeichnung entscheidet über konkrete Ausprägung	○	○	●	●
Erweiterung um neue Ausprägungen einer Ressource zur Laufzeit	●	●	○	○

Tabelle 5.2: Gegenüberstellung der Ansätze zur Auflösung von Generalisierungsbeziehungen in einem Ressourcenmodell

Heuristiken und Überführungsregeln für das zugrundeliegende Modellierungsartefakt

Nachdem im Vorfeld eine Besonderheit bei der Überführung der modellierten strukturellen Aspekte eines Bounded Context in ein Ressourcenmodell aufgezeigt wurde, werden nun die entsprechenden Heuristiken und Überführungsregeln (nachfolgend als *Regeln* bezeichnet) aufgeführt. Hierzu wird jeder Ressourcenart im Sinn des UML-Profils eine Regel und die dazugehörige Heuristik zugeordnet, welche wiederum aus einem Set an Merkmalen besteht. Die Regeln wurden in Anlehnung an die Arbeit von Rathod et al. [RP+13] entworfen und sind in der Tabelle 5.3 dargestellt. Bei Anwendung der Heuristiken ist die Reihenfolge einzuhalten, die durch die Tabellenzeilen festgelegt wird. So sollten zunächst die Primärressourcen und anschließend die Subressourcen identifiziert werden, da auch die Ressourcen hierarchisch aufgebaut sind [LK+06, Sc11]. Die Listenressourcen lassen sich in diesem Zusammenhang als Ergänzung der Primär- oder Subressourcen auffassen.

Die Operationen der einzelnen Domänenobjekte sind derart zu überführen, dass sie der jeweiligen *ResourceOperation* des UML-Profils *ResourceOrientation* entsprechen. Weil die Operationen identisch bezeichnet sind, ist eine zusätzliche Regel nicht erforderlich. Einzig besonders ist hier, dass die Eigenschaftsdefinition `reference` auf die Operation des jeweiligen Domänenobjekts verweist, da das Verhalten diesem zugeordnet wird und nicht auf Ebene der Ressourcen abzubilden ist. Weiter findet auch das implizite Verhalten auf dieser Ebene seine Anwendung, um den notwendigen Modellierungsaufwand zu reduzieren (vgl. Abschnitt 4.2.2). Dies bedeutet, dass jede Ressource auf Grundlage eines Domänenobjekts entsprechend CRUD-Operationen nach außen veräußert, sofern das Domänenobjekt diese anbietet. Auf Basis dieses impliziten Wissens und unter Anwendung der Heuristiken sowie zugehöriger Regeln werden die Domänenobjekte ähnlich wie bei Haupt et al. [LS+09, HL+15] in ein Ressourcenmodell überführt, das ausschließlich aus Ressourcen besteht, womit letztlich die strukturellen Aspekte eines ressourcenorientierten Microservice abgebildet werden.

Die Abbildung 5.6 veranschaulicht die Identifikation und Überführung von Ressourcen in ein Ressourcenmodell. Es lässt erkennen, wie unter Anwendung der Heuristiken und der damit verknüpften Regeln ein Domänenobjekt oder ein Domänenservice in eine Ressource überführt werden kann.

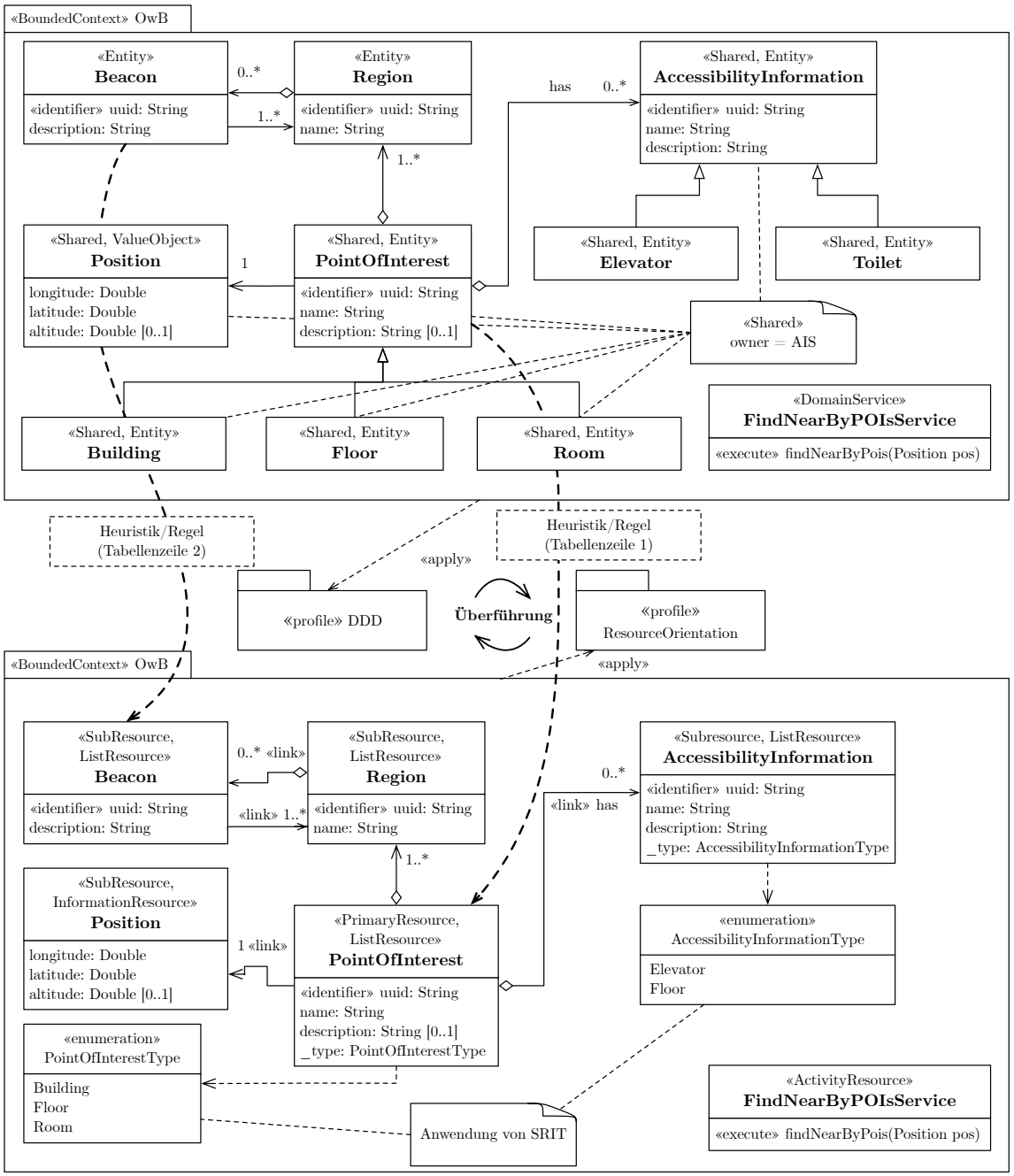


Abbildung 5.6: Beispielhafte Überführung eines modellierten Bounded Context in ein Ressourcenmodell

Art	Heuristik	Überführungsregel
R_{Prim}	Domänenobjekt in Form einer *Entity* ohne eingehende Assoziationen	1) Kopieren der *Entity* als neues Modellelement in das Ressourcenmodell bei gleichzeitigem Entfernen von allen bisherigen Stereotypen mit Ausnahme des *identifier* 2) Zuweisen des Stereotyps *PrimaryResource* an das neue Modellelement 3) Anwendung des Ansatzes zur Auflösung etwaiger Generalisierungsbeziehungen mit einem Ansatz in Tabelle 5.2
R_{Sub}	Domänenobjekt in Form einer *Entity* mit einer oder mehreren eingehenden Assoziationen	1) Kopieren der *Entity* als neues Modellelement in das Ressourcenmodell bei gleichzeitigem Entfernen von allen bisherigen Stereotypen mit Ausnahme des *identifier* 2) Zuweisen des Stereotyps *SubResource* an das neue Modellelement 3) Überführen der Assoziation aus dem Domänenmodell inklusive eines etwaigen Assoziationsnamens sowie unter Hinzunahme des Stereotyps *link* (Teil-Ganzes-Beziehungen werden als einfache gerichtete Assoziation übernommen) 4) Anwendung des Ansatzes zur Auflösung etwaiger Generalisierungsbeziehungen mit einem Ansatz in Tabelle 5.2
R_{Inf}	Domänenobjekt, das als *ValueObject* vorliegt und eine eingehende Beziehung aufweist	1) Kopieren der *Entity* als neues Modellelement in das Ressourcenmodell bei gleichzeitigem Entfernen von allen bisherigen Stereotypen 2) Zuweisen des Stereotyps *InformationResource* 3) Überführen der Assoziation aus dem Domänenmodell unter Hinzunahme des Stereotyps *link* inklusive eines etwaigen Assoziationsnamens
R_{List}	Domänenobjekt, für das es mehrere Instanzen geben kann oder das sich in einer 1..n-Beziehung befindet und den Elternteil repräsentiert	1) Identifikation der *Entity* oder des *ValueObject* im Ressourcenmodell, die nach den vorherigen zwei Regeln durch eine *PrimaryResource*, *SubResource* oder *InformationResource* repräsentiert werden 2) Zuweisen des Stereotyps *ListResource*
R_{Act}	Domänenobjekt, das als *DomainService* vorliegt, um so bspw. einen Geschäftsprozess von außen anzusteuern	1) Kopieren des *DomainService* in das Ressourcenmodell bei gleichzeitigem Entfernen von allen bisherigen Stereotypen 2) Zuweisen des Stereotyps *ActivityResource*

Tabelle 5.3: Überführungsregeln von Domänenobjekten in ein Ressourcenmodell

5.2 Ableitung der Adressierung

Auf die Identifizierung der Ressourcen eines modellierten Bounded Context und deren Überführung in ein Ressourcenmodell erfolgt nun die Ableitung der Adressierung, die letztlich den Zugriff auf die Ressourcen über das Web ermöglicht und damit der zugrundeliegende Bounded Context von Service-Konsumenten genutzt werden kann. Die Adressierung erfolgt über eine URI gemäß [RFC3986-2005], deren Syntax eine hierarchische Ordnung verfolgt: „components listed in order of decreasing significance from left to right“ [RFC3986-2005, S. 9]. Dieser Abschnitt untersucht die folgenden Fragen für eine systematische und nachvollziehbare Ableitung der Adressierung:

(1) Welche existierenden bewährten Methoden und Muster sind im Zuge der zu betrachtenden Qualitätsteilmerkmale einer Web-API zu berücksichtigen?

(2) Wie kann die Adressierung der im vorherigen Abschnitt identifizierten Ressourcen im Hinblick auf die ermittelten bewährten Methoden und Muster aus (1) systematisch und nachvollziehbar abgeleitet und abgebildet werden?

5.2.1 Bestehende Ansätze zur Adressierung von Ressourcen

Zur Beantwortung der Frage (1) wurden existierende Ansätze in Bezug auf die Adressierung von Ressourcen untersucht. Hierzu liefert [GG+16c] eine aggregierte Ansammlung bewährter Methoden im Kontext ressourcenorientierter Web-APIs, die auch Adressierung behandeln. Ergänzend liefern Palma et al. [PGH+15] einen Ansatz zum Auffinden von Fehlern bzw. Anti-Patterns innerhalb einer URI auf Grundlage sogenannter linguistischer Muster. Den Einfluss dieser Muster auf die Benutzbarkeit und die letztliche Wiederverwendbarkeit betonen auch Palma et al.: „well-designed and named RESTful APIs may attract client developers more than poorly designed or named ones [...] because client developers must understand the providers' APIs while designing and developing their Web-based systems that use these APIs. Therefore, in the design and development of RESTful APIs, their understandability and reusability are two major quality factors.“ [PGH+15, S. 171f.]. Zudem soll es möglich sein, aus der URI den grundsätzlichen strukturellen Aufbau des modellierten Bounded Context abzuleiten, da Entwickler von Service-Konsumenten in der Regel keinen Zugriff auf die zugrundeliegenden Modellierungsartefakte haben.

Die Vereinigung der linguistischen Muster und der relevanten bewährten Methoden in [GG+16c] wird in Tabelle 5.4 dargestellt. Das Befolgen dieser Empfehlungen kann die Benutzbarkeit und damit auch die Auffindbarkeit der Web-API erhöhen (vgl. Abschnitt 2.3.3). Auf den konkreten Bezug zu den einzelnen Qualitätsaspekten in Abschnitt 2.3.3 wurde hier verzichtet, da sich der Zusammenhang aus der Beschreibung ergibt.

Bezeichnung	Abgeleitet aus	Beschreibung
Domänenspezifische URI-Komponenten (*DSUCP*)	Contextualised Resource Names (M)/Domain-specific Nodes (B)	URI-Komponenten sollten stets den gleichen semantischen Kontext haben. Der semantische Kontext ergibt sich durch den logischen Zusammenhang von Ressourcen. So haben bspw. *customer* und *order* im Gegensatz zu *customer* und *warehouse stock* den gleichen semantischen Kontext [PGH+15]. Da der Kontext durch die Domänen gegeben ist, ist dies mit der bewährten Methode *Domain-specific nodes* vereinbar. Demnach sollte auch die Semantik aus dem Ressourcennamen ohne zusätzliches Wissen abgeleitet werden können [GG+16c].
Hierarchisch beschränkte URI-Komponenten (*HBUCP*)	Hierarchical Nodes (M)/Bounding URI (B)	URI-Komponenten sollten hierarchisch angeordnet sein [RFC3986-2005, PGH+15]. Gleichzeitig sollte die URI im Hinblick auf die Anzahl der URI-Komponenten nicht zu groß werden, obgleich kein eindeutiger Richtwert für die Anzahl existiert [GG+16c].
Konsistente URI-Komponenten (*CUCP*)	Tidy URIs (B, M), Verbless URIs (B, M), Using nouns (B)	Eine URI sollte einfach zu lesen sein, weshalb sie bestimmte Konventionen im Hinblick auf die URI-Komponenten befolgen sollte. Zu den Konventionen zählen die (1) Verwendung der Kleinschreibung, (2) die Vermeidung von Dateiendungen, (3) die Verwendung von Bindestrichen anstelle von Unterstrichen sowie (4) die Vermeidung eines nachgestellten Schrägstrichs (engl. trailing slash). Außerdem sollte eine URI ausschließlich aus Nomen bestehen und keine Verben beinhalten, da diese durch das verwendete Anwendungsschichtprotokoll abgebildet werden [PGH+15, RB+16b, GG+16c].
Semantik korrekter Numerus (*SCNUCP*)	Singularised/Pluralised Nodes (B, M)	Die URI-Komponenten sollten einheitlich in der Plural- oder Singularform entsprechend der Informationen beschrieben sein. So sollten URI-Komponenten die eine Liste an Informationen repräsentieren, im Plural formuliert sein, wohingegen eine bestimmte Information über den Singular der URI-Komponente gekennzeichnet wird [PGH+15, RB+16b]. Bekanntes Beispiel einer Singularform sind die aktuellen Profilinformationen eines angemeldeten Nutzers über *.../me*[1].

Tabelle 5.4: Linguistische Muster und bewährte Methoden aus [PGH+15, GG+16c, RB+16b]

Da sich die Muster und bewährten Methoden teilweise überlappen oder einander ergänzen, wurden diese in der Tabelle entsprechend zusammengefasst und eine einheitliche Bezeichnung dafür definiert: *DSUCP*, *HBUCP*, *CUCP* und *SCNUCP*. Das Resultat sind wiederum Muster, welche die bestehenden linguistischen Muster und die identifizierten bewährten Methoden vereinen. Durch *(B)* und *(M)* wurde jeweils gekennzeichnet, aus welchen Bestandteilen sich die resultierenden Muster zusammensetzen.

[1] https://developer.spotify.com/web-api/get-current-users-profile/ (Letzter Zugriff: 17.09.2017)

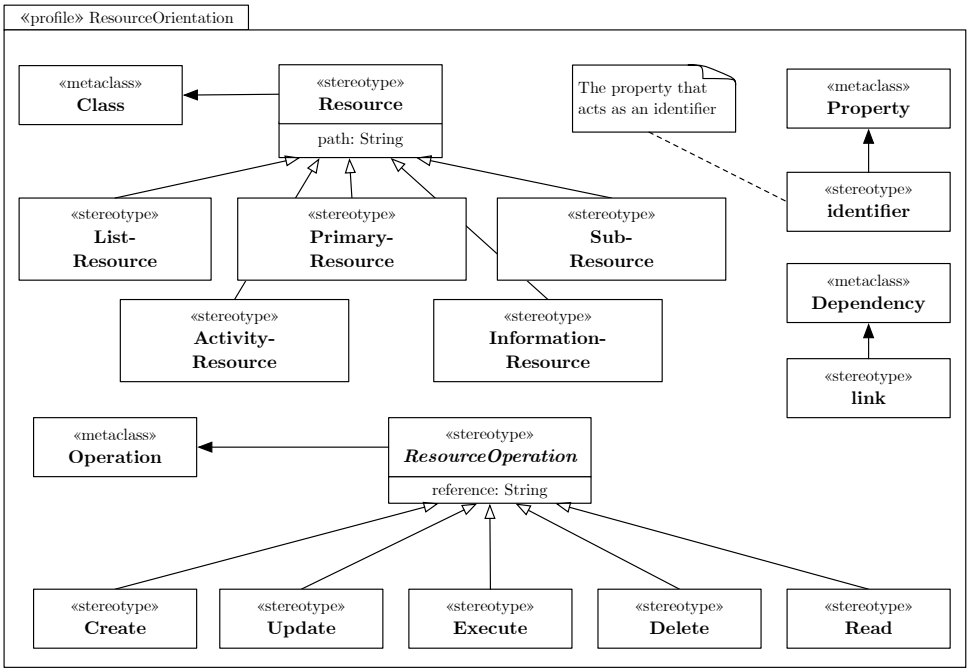


Abbildung 5.7: Erweitertes UML-Profil zur Abbildung von Ressourcen innerhalb eines UML-Klassendiagramms

Dabei steht *(M)* für ein Muster und *(B)* für eine bewährte Methode. Auf eine Formalisierung in Form eines Musters ähnlich wie in [PI+16] wurde hier verzichtet, da deren Mehrwert über die bisherige Beschreibung in Tabelle 5.4 hinaus gering wäre. Die Vorzüge betreffen vor allem die Benutzbarkeit u. a. durch die Wahrung einer einheitlichen Schreibweise sowie den Bezug zur Geschäftsdomäne bzw. zum abgebildeten Bounded Context durch Verwendung der Ubiquitous Language.

Die Musterbeschreibungen in Tabelle 5.4 zeigen, dass der modellierte Bounded Context und die damit einhergehende Terminologie in Form der Ubiquitous Language einen idealen Kandidaten für die Ableitung der Adressierung ergeben. Da der Bounded Context im vorigen Abschnitt in ein Ressourcenmodell überführt wurde, kann dieses gleichermaßen als Grundlage für die Ableitung der Adressierung dienen. So werden bspw. die Ressourcen durch entsprechende Assoziationen in einen logischen und fachlichen Bezug zueinander gesetzt. Gleichzeitig ist durch die Benennung der Ressource mit den Begrifflichkeiten der Ubiquitous Language das Muster *DSUCP* bereits implizit auf Modellebene gegeben.

5.2.2 Festlegen von Heuristiken und Ableitungsregeln

Um sicherzustellen, dass die URIs für die einzelnen Ressourcen den Mustern in Tabelle 5.4 gerecht werden, werden ähnlich wie in Abschnitt 5.1 Heuristiken und Ableitungsregeln definiert, wodurch zumindest teilweise die Fragestellung (2) beantwortet wird. Dabei hat sich herausgestellt, dass

wenige Heuristiken sowie zugehörige Ableitungsregeln ausreichend sind (siehe Tabelle 5.6). Für die Zuweisung der URI-Komponenten zu den einzelnen Ressourcen wird der Stereotyp `Resource` des zuvor eingeführten UML-Profils (siehe Abbildung 5.5) um die Eigenschaftsdefinition `path` in Anlehnung an den Ansatz von Rossi [Ro16] erweitert. Das erweiterte UML-Profil wirkt somit ergänzend und ist weiterhin kompatibel mit dem vorherigen UML-Profil. Entsprechende Änderungen am Ressourcenmodell sind daher nicht erforderlich.

Konvention	Richtig (✓)	Falsch (×)
Durchgehende Verwendung der Kleinschreibung	*order*	*Order*
Verwendung eines Bindestrichs für zusammenhängende Wörter	*order-items*	*orderItems*
Vermeidung von Dateiendungen	*order*	*order.json*
Vermeidung von Unterstrichen	*order-items*	*order_items*
Vermeidung eines nachgestellten Schrägstrichs	*order-items*	*order-items/*

Tabelle 5.5: Konventionen zur Adressierung der Ressourcen gemäß *CUCP*

Die Konsistenz der einzelnen URI-Komponenten wird durch die festgelegten Konventionen in Tabelle 5.5 gewährleistet, die bei der Festlegung der Eigenschaftsdefinition `path` stets zu beachten sind. Sie bilden damit Invarianten die vor und nach einer Änderungen an dem Ressourcenmodell erfüllt sein müssen. Bei gegebenen Unternehmensrichtlinien können die genannten Konventionen auch den Bedürfnissen des Unternehmens angepasst werden. Entscheidend ist, dass sie einmal definiert und über die Grenzen eines Bounded Context hinweg konsistent verwendet werden.

Anwendbar auf	Heuristik	Ableitungsregel
R_{Prim}, R_{Sub}, R_{Inf}, R_{Act}	Die Ressource verfügt nicht über den Stereotyp `ListResource`	Zuweisen der Singularform der Ressourcenbezeichnung als Wert für die Eigenschaftsdefinition `path`.
R_{List}	Die Ressource besitzt den Stereotyp `ListResource`	Zuweisen der Pluralform der Ressourcenbezeichnung als Wert für die Eigenschaftsdefinition `path`.
R_{Prim}, R_{Sub}	Den Ressourcen ist der Stereotyp `ListResource` zugewiesen	Der Wert für die Eigenschaftsdefinition `path` bleibt leer, da die Ressource über deren *identifier* adressiert wird.
R_{Inf}	Die Ressource befindet sich in einer 1:n-Beziehung	Zuweisen der Pluralform der Ressourcenbezeichnung als Wert für die Eigenschaftsdefinition `path`.

Anwendbar auf	Heuristik	Ableitungsregel
R_{Act}	Den Ressourcen ist der Stereotyp `ActivityResource` zugewiesen	Zuweisen einer substantivierten Ressourcenbezeichnung als Wert für die Eigenschaftsdefinition `path`, der die damit adressierte Funktionalität widerspiegelt.

Tabelle 5.6: Heuristik und Ableitungsregeln zur Zuweisung von URI-Komponenten zu Ressourcen gemäß *DSUCP*, *HBUCP*, *CUCP* und *SCNUCP*

5.2.3 Ableitung der URI-Hierarchie

Die URI-Hierarchie gemäß [RFC3986-2005] lässt sich durch das Verfolgen der gerichteten Assoziationen ableiten. Ausgehend von einer Primärressource wird zunächst geprüft, ob zu ihr eine Listenressource gehört, da Listenressourcen Container für andere Ressourcen sind: „Collection resources are a specific kind of resource which acts as a container for other resources“ [HL$^+$15, S. 3]. Falls dies zutrifft, stellt der Wert von `path` der entsprechenden Listenressource die Bezeichnung der ersten URI-Komponente dar. Um auf die einzelnen Kindelemente der Listenressource zuzugreifen, dient das Attribut, das mit dem Stereotyp `identifier` gekennzeichnet ist. Da der Wert dieses Attributs erst zur Laufzeit durch eine entsprechende Instanziierung vorliegt, wird an dieser Stelle ein Platzhalter mit `{uuid}` als weitere URI-Komponente deklariert.

Um einen Platzhalter von einem statischen Wert, der auf Modellebene definiert werden kann, zu unterscheiden, wird der Platzhalter nicht besetzt. Falls keine Listenressource zugeordnet ist, wird lediglich der Wert von `path` der entsprechenden Ressource als neue URI-Komponente aufgefasst. Danach werden rekursiv die ausgehenden Assoziationen betrachtet, die jeweils mit dem Stereotyp `link` annotiert sind. Vergleichbar mit einer Primärressource wird auch hier jeweils geprüft, ob die Ressource einer Listenressource zugeordnet ist. Ein Sonderfall ist die Informationsressource, die kein Stereotyp `identifier` hat. Falls diese Informationsressource über einen gesetzten Wert für *path* verfügt, wird sie als eigenständige URI-Komponente aufgefasst. Weiter sind Listenressourcen, die sich als Kindelement in einer Aggregationsbeziehung befinden, ebenfalls geeignete Kandidaten für die oberste Ebene der URI-Hierarchie. Dann ist zu prüfen, ob die Kindelemente in einer m:n-Relation existieren können. Falls das der Fall ist, muss diese Listenressource auf oberster Ebene der URI-Hierarchie ergänzt werden, da sonst keine Möglichkeit besteht, vorhandene Instanziierungen der jeweiligen Ressource wiederzuverwenden. Das geschilderte Vorgehen muss so lange wiederholt werden, bis jede Primärressource des Ressourcenmodells einmal Ausgangspunkt der Untersuchung war. Eine Aktivitätsressource ist besonders, da sie in der Regel keiner Ressource direkt zugeordnet werden kann und grundsätzlich unabhängig betrachtet werden muss. In diesem Fall ist sie auf der obersten Ebene der URI-Hierarchie zu finden. Falls jedoch eine Zuordnung zu einer Ressource existiert, kann sie deren jeweilige URI-Hierarchie logisch ergänzen. Auch hier ist zu erwähnen,

dass der Software-Architekt die URI-Hierarchie anpassen muss, falls die Ableitung nicht zu den gewünschten Ergebnissen führt.

Wird das beschriebene Vorgehen zur Adressierung von Ressourcen an dem Szenario in Abschnitt 4.3 angewandt, so ergibt sich die in Abbildung 5.8 abgeleitete URI-Hierarchie.

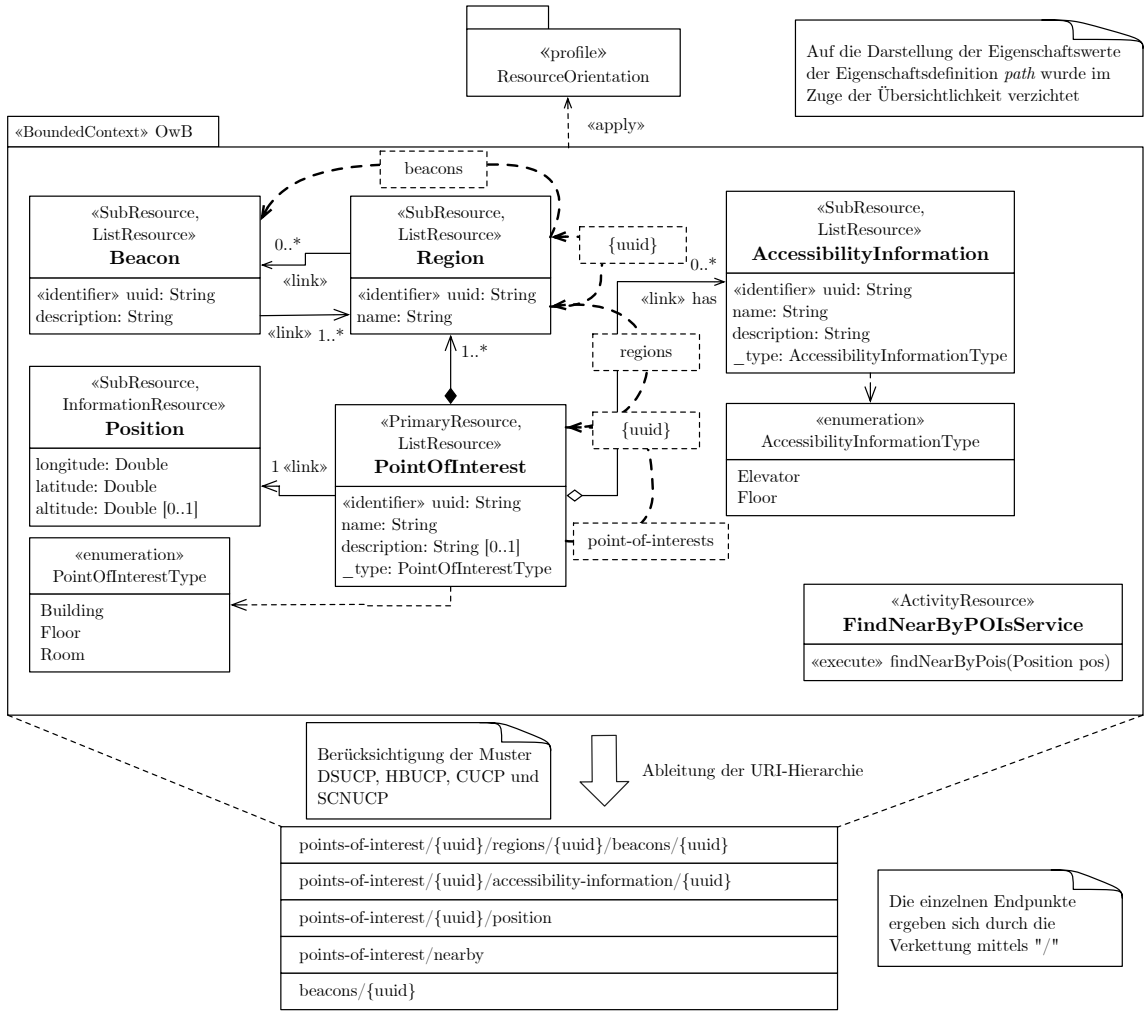


Abbildung 5.8: Ableitung der URI-Hierarchie aus dem zugrundeliegenden Ressourcenmodell R_1

5.3 Interaktion mit Ressourcen

Nachdem den identifizierten Ressourcen entsprechende Pfade zugeordnet und ein Vorgehen zur Ableitung der URI-Hierarchie aufgeführt wurde, untersucht dieser Abschnitt die Interaktionen zwischen Service-Konsumenten und Microservice bzw. dessen Ressourcen. Da eine Interaktion eine Aktivität über die Zeit darstellt, sind diese den verhaltensspezifischen Aspekten zuzuordnen. Jede Interaktion adressiert wiederum eine geschäftliche Dienstleistung, wodurch ein direkter Bezug zum Verhalten

einer Domäne bzw. eines Bounded Context hergestellt werden kann. Eine Interaktion ist demnach immer mit einer konkreten Absicht (engl. intent) verknüpft, wie bspw. Auflistung aller Beacons, die einer Region und letztlich einem Point of Interest (POI) zugewiesen wurden. Auf technischer Ebene wird eine Interaktion durch eine Verkettung von Anfragen/Antworten zwischen Microservice und letztlichem Service-Konsumenten realisiert. Dies ist vergleichbar mit einer Konversation nach Haupt et al. [HL+15] und Pautasso et al. [PI+16]. Dieser Abschnitt erörtert die Frage, wie sich das Verhalten einer Domäne mit einer Interaktion zwischen Service-Konsumenten und Microservice adressieren lässt. Dazu wurden folgende Teilfragen abgeleitet:

(1) Was ist beim Entwurf einer Interaktion zu beachten bzw. welche möglichen Einflüsse müssen betrachtet werden, und wie beeinflussen diese die Wiederverwendbarkeit?

(2) Wie kann der Software-Architekt im Hinblick auf (1) bei der Modellierung der Interaktionen unterstützt werden, und wie manifestiert sich seine Entscheidung im Ressourcenmodell?

5.3.1 Interne und externe Einflussfaktoren

Der Einfluss des Entwurfs einer Interaktion mit Ressourcen lässt sich durch *Einflussfaktoren* beschreiben, die entweder positiv oder negativ auf einen Teilnehmer der Interaktion wirken. Zu den Einflussfaktoren zählen bspw. die effiziente Ausnutzung von (Hardware-)Ressourcen bei der Ausführung von Operationen, die Mächtigkeit im Sinne der Anzahl der realisierbaren Anwendungsfälle oder die Komplexität bei der Integration/Implementierung. Diese Einflussfaktoren lassen sich Qualitätsmerkmalen bzw. -teilmerkmalen zuordnen. Als Teilnehmer einer Interaktion (nachfolgend als *Interaktionsteilnehmer* bezeichnet) gelten neben den Service-Konsumenten auch der Microservice mit seiner veräußerten Web-API. Diese ist dabei der Vertrag zwischen Interaktionsteilnehmern (wird auch als *Customer/Supplier*-Modell bezeichnet). Zusammengefasst ergibt sich im Kontext dieser Arbeit folgende Definition:

> **Definition (Interaktionsteilnehmer)** *Interaktionsteilnehmer sind aktiv oder passiv an einer Interaktion beteiligte Akteure. Die Interaktion ist dabei durch eine Web-API vertraglich festgelegt.*

Diese Arbeit unterscheidet zwischen internen und externen Einflussfaktoren (siehe Abbildung 5.9). Dabei adressieren die internen Einflussfaktoren die Implementierung des Microservice, während die externen die Integration bzw. Nutzung des Microservice fokussieren. Beide Formen können nicht unabhängig voneinander betrachtet werden, weshalb immer eine Abwägung (engl. tradeoff) zwischen den Vorzügen und Nachteilen erforderlich ist. So verlangt bspw. eine Filterung von Informationen auf Service-Konsumentenseite einen gewissen Implementierungsaufwand. Zudem steigt mit zunehmender Anzahl an Informationen auch die Übertragungsmenge, wodurch die Umsetzung einer Filterung auf Service-Konsumentenseite auch durch höhere Latenz zunehmend unpraktikabel erscheint. Eine

Filterung auf Microserviceseite verlangt auf der anderen Seite mehr Aufwand bei der Implementierung, was allerdings durch die Unterstützung in heutigen Frameworks erleichtert wird.

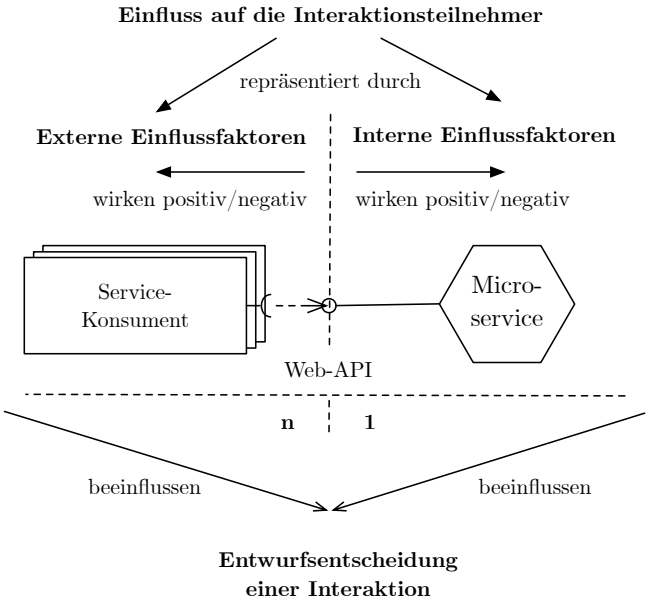


Abbildung 5.9: Interne und externe Einflussfaktoren bei einer Entwurfsentscheidung im Kontext einer Interaktion

Weiter gilt es zu beachten, dass grundsätzlich mehr als ein Service-Konsument einen Microservice nutzen kann, wodurch sich die Gewichtung der Einflussfaktoren verändern kann und daher in der Regel von einer n : 1-Beziehung auszugehen ist, wobei $n \in \mathbb{N} \setminus 0$. Deshalb sollten bei etwaigen Entwurfsentscheidungen vorrangig die externen Einflussfaktoren betrachtet werden. Doch muss der Software-Architekt auch den Grad der jeweiligen Auswirkung im Hinblick auf das abzubildende Szenario beurteilen und bewerten. So hängt bspw. eine Gewichtung von der Erfahrung des Entwicklungsteams und auch der verwendeten Technologie ab. Abschließend ist zu erwähnen, dass die Wiederverwendbarkeit eines Microservice durch die Betrachtung der externen Einflussfaktoren sich steigern oder mindern lässt und sich deshalb dahingehend eine positive Relation ableiten lässt. Dies ist wiederum darauf zurückzuführen, dass die Service-Konsumenten die Zielgruppe des Microservice darstellen. Eine hohe Nutzung des Microservice durch unterschiedliche Service-Konsumenten impliziert wiederum einen hohen Grad an Wiederverwendbarkeit.

5.3.2 Mustersprache für ressourcenorientierte Interaktionen

Nachdem mit dem vorigen Abschnitt die Fragestellung (1) betrachtet wurde, widmet sich dieser der Unterstützung des Software-Architekten bei dem Entwurf einer Interaktion und damit der Fragestellung (2). Hierfür hat sich der Einsatz von Mustern bewährt, wie es auch Buschmann et al. [BH$^+$07]

darlegen. Einen Ansatz im Kontext der Interaktion mit Ressourcen liefern Pautasso et al. [PI+16]. Die dort präsentierte Mustersprache ist allerdings auf die Umsetzung mittels HTTP ausgerichtet und betrachtet den Service-Konsumenten und damit die externen Einflussfaktoren nur teilweise (vgl. *Resource Collection Traversal*). Weiter ist unklar, wie die Muster die Qualitätsteilmerkmale der Wiederverwendbarkeit verändern und insgesamt zu den Einflussfaktoren stehen.

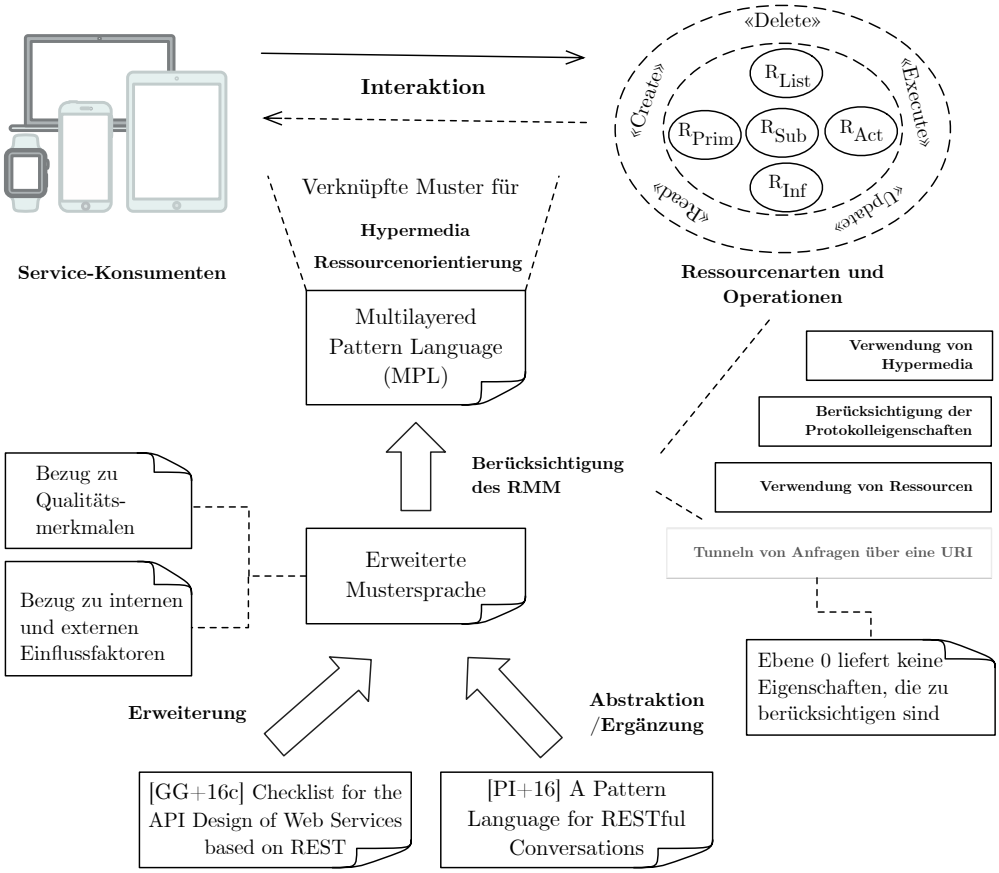


Abbildung 5.10: Vorgehen bei dem Entwurf der mehrschichtigen Mustersprache für Interaktionen

Für eine ganzheitliche Sicht auf die Interaktionen wird nun, auf [PI+16] gestützt, eine Mustersprache vorgestellt, die alle zu betrachtenden Ressourcenarten aus Abbildung 5.2 adressiert, einen Bezug zu Qualitätsmerkmalen herstellt und von einem konkreten Anwendungsschichtprotokoll abstrahiert. Ergänzende Muster wurden durch bewährte Methoden in [GG+16c] sowie durch aktuelle Untersuchungen in der Praxis ermittelt, was die Benutzbarkeit und Mächtigkeit steigert und letztlich in einer höheren Wiederverwendbarkeit resultiert. Hingegen wurde die Gruppierung zur Sicherheit aus [PI+16] ausgeklammert, da dies nicht Fokus dieser Arbeit ist. Der Aufbau der Mustersprache orientiert sich, ähnlich wie in [PI+16], an der Mustersprache in [MD97]. Eine Übersicht über die Mustersprache, in Anlehnung an das Muster *Problem/Solution Summary* in [MD97], gibt Abbildung 5.11. Die Mustersprache wurde so entworfen, dass sie das RMM auf Ebene 2 und damit die

Ressourcenorientierung sowie die Verwendung von Hypermedia (Ebene 3) berücksichtigt. Die Muster lassen sich auf unterschiedlichen Ebenen platzieren (vgl. Abschnitt 2.3.4). So ist es möglich, zunächst die Ressourcenorientierung zu fokussieren, ohne dass Überlegungen zur Unterstützung von Hypermedia erforderlich sind. Diese Entscheidung geht auch auf Rückmeldungen aus der Industrie zurück, worin die Vorzüge von Hypermedia zumindest infrage gestellt werden. Dies zeigt sich auch bei vielen als *RESTful* bezeichneten Services, die sich bei genauer Betrachtung eigentlich sich nicht als solche bezeichnen lassen, weil der Hypermedia-Aspekt fehlt. Das erwähnt auch Vinoski beiläufig: „As important as the hypermedia constraint is, developers don't seem to adequately address it" [Vi08, S. 94]. Deshalb wird diese Mustersprache nun als *Multilayered Pattern Language (MPL)* bezeichnet, um ressourcenorientierte Microservices umzusetzen und bei Bedarf um den Hypermedia-Aspekt zu ergänzen. Einen ähnlichen Ansatz mit mehreren Ebenen einer Mustersprache liefert [MM+12], wobei dort die Muster auf mehreren Ebenen existieren können, ohne einander zu ergänzen. Abbildung 5.10 illustriert das grundsätzliche Vorgehen.

Um den Bezug zum Vorigen zu vereinfachen, ist dieser Abschnitt in die verschiedenen Ressourcenarten Abschnitt 5.1.1 sowie die Absichten (engl. intents) einer Interaktion auf Service-Konsumentenseite unterteilt. Das erleichtert die Verortung der relevanten Muster in der MPL auf Basis des Ressourcenmodells, was wiederum systematisches Vorgehen ermöglicht. Etwaige Muster, die sich nicht einer bestimmten Ressourcenart zuweisen lassen, werden separat am Ende erörtert. Zur Beschreibung der Muster bzw. der MPL wurden für eine genauere sowie kompaktere Beschreibung einige Variablen eingeführt, die in Tabelle 5.7 aufgeführt sind.

Variable	Bedeutung
r	Eine Ressource $r \in \{R_{Prim} \cup R_{Sub} \cup R_{List} \cup R_{Inf} \cup R_{Act}\}$
S_K	Ein oder mehrere Service-Konsumenten
r_{inf}	Eine Informationsressource $r_{inf} \in R_{Inf}$
r_{prim}	Eine Primärressource $r_{prim} \in R_{Prim}$
r_{sub}	Eine Subressource $r_{sub} \in R_{Sub}$
r_{list}	Eine Listenressource $r_{list} \in R_{List}$
$Repr(r)$	Repräsentation der Ressource r
$RC(r_{list})$	Menge aller Ressourcen einer Listenressource r_{list}
$Repr(RC(r_{list}))$	Gesamtheit der Repräsentationen der Ressourcen einer r_{list}

Tabelle 5.7: Eingeführte Variablen und deren Bedeutung für die Musterbeschreibung

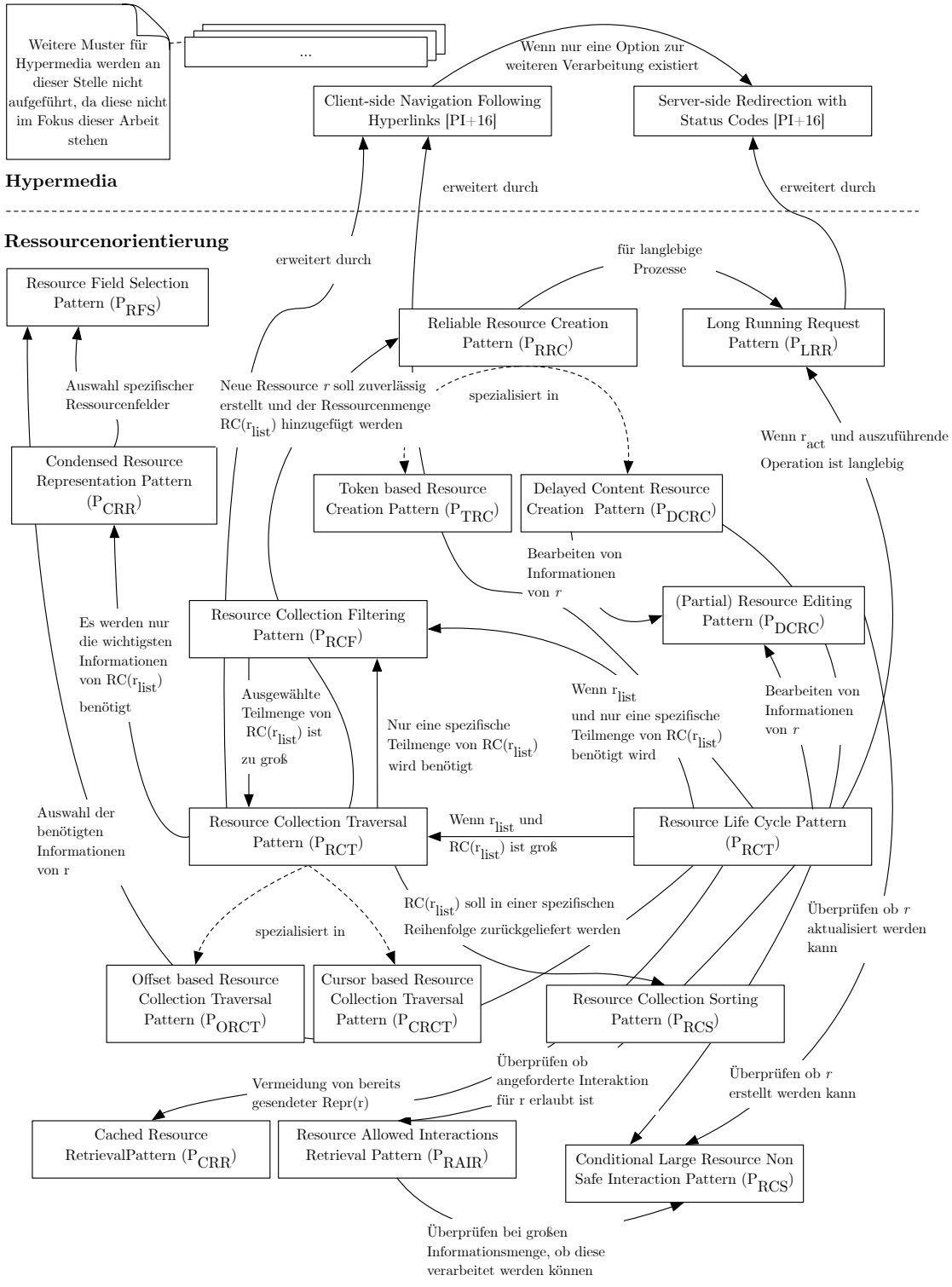


Abbildung 5.11: MPL für die Interaktion mit Ressourcen auf Grundlage von [PI+16] und RMM

Interaktionen mit einer Listenressource

Listenressourcen R_{List} gruppieren mehrere Ressourcen als Sammlung (engl. collection), um u. a. die Auffindbarkeit von Ressourcen zu erleichtern [PI+16]. Die verschiedenen Interaktionen mit einer Listenressource r_{list} zeigt Abbildung 5.12. Sie werden jeweils mit I_1 bzw. I_2 gekennzeichnet. Mit I_1 wird die Bereitstellung der zugeordneten Ressourcen $RC(r_{list})$ einer Listenressource r_{list} adressiert. Um eine Ressource aus der Menge *RC* zu adressieren, enthält die Repräsentation der einzelnen Ressource einen Identifikator oder eine URI, die helfen, gegebenenfalls weiterführende Informationen bei einer weiteren Anfrage abzurufen (siehe I_3 in Tabelle 5.3.2). Neben der Bereitstellung mehrerer Ressourcen bietet eine Listenressource auch die Möglichkeit, neue Ressourcen anzulegen und diese der Sammlung hinzuzufügen (I_2) [TE+15, HL+15, PI+16]. Vergleicht man dies mit dem Verhalten eines *Aggregate Root*, so obliegt auch diesem der Erzeugungsprozess der jeweiligen *Aggregates* [Ev03], weshalb dies als ein Beleg für die korrekte Überführung angesehen werden kann. Für derartige Interaktionen sind einige Muster etabliert, die Tabelle 5.8 mit dem Verweis auf die dazugehörige Musterbeschreibung aufführt. Die Tabellenspalte "Abstrakt" zeigt an, ob es sich um ein abstraktes Muster handelt oder nicht. Abstrakte Muster ermöglichen eine Gruppierung zusammengehöriger Muster, die den gleichen Kontext und die gleiche Problemstellung teilen. Nun werden die einzelnen Interaktionen noch einmal im Detail betrachtet und auf die erwähnten Muster bezogen.

Für diese beschriebenen Interaktionen haben sich einige Muster etabliert, die in Tabelle 5.8 mit einem Verweis auf die entsprechende Musterbeschreibung aufgeführt sind. Mit der Tabellenspalte “Abstrakt“ soll gekennzeichnet werden, ob es sich um ein abstraktes Muster handelt oder nicht. Abstrakte Muster ermöglichen eine Gruppierung zusammengehöriger Muster, die sich den gleichen Kontext und die gleiche Problemstellung teilen. Nachfolgend werden nun die einzelnen Interaktionen noch einmal im Detail betrachtet und Bezug zu den im Vorfeld erwähnten Mustern genommen.

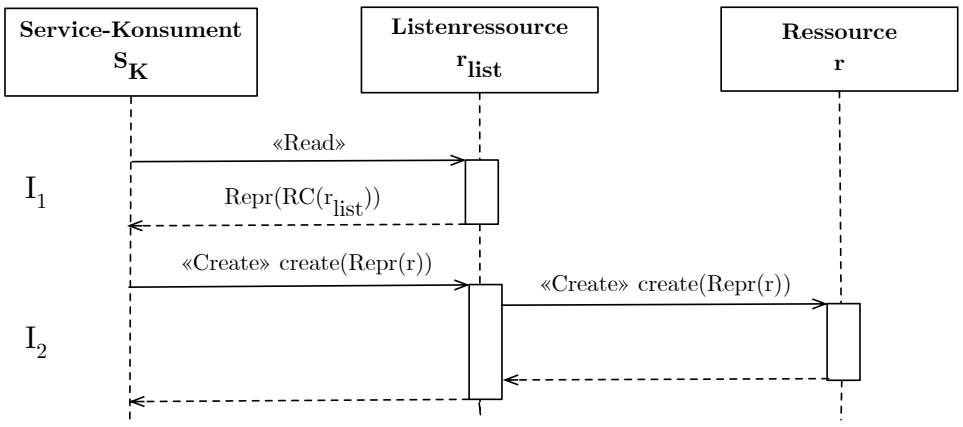


Abbildung 5.12: Interaktionen mit einer Listenressource

Muster	Abk.	Abstrakt	Absicht	Interakt.	Verweis
Resource Collection Traversal Pattern	P_{RCT}	Ja	Traversierung von Ressourcen $R \in R_{List_x}$	I_1	Anhang A.7 auf Seite 215
Offset-based Resource Collection Traversal Pattern	P_{ORCT}	Nein	Traversierung von Ressourcen $RC(r_{list})$	I_1	Anhang A.7 auf Seite 216
Cursor-based Resource Collection Traversal Pattern	P_{CRCT}	Nein	Traversierung von Ressourcen $RC(r_{list})$	I_1	Anhang A.7 auf Seite 217
Condensed Resource Representation Pattern	P_{CRR}	Nein	Reduktion der Übertragungsmenge der $RC(r_{list})$	I_1	Anhang A.7 auf Seite 218
Resource Collection Filtering Pattern	P_{RCF}	Nein	Filterung von Ressourcen der $RC(r_{list})$	I_1	Anhang A.7 auf Seite 219
Resource Collection Sorting Pattern	P_{RCS}	Nein	Sortierung von Ressourcen der $RC(r_{list})$	I_1	Anhang A.7 auf Seite 220
Reliable Resource Creation Pattern	P_{RRC}	Ja	Zuverlässiges Anlegen von neuen Ressourcen r wobei $r \notin \{R_{List} \cup R_{Act}\}$	I_2	Anhang A.7 auf Seite 223
Token-based Resource Creation Pattern	P_{TRC}	Nein	Zuverlässiges Anlegen von neuen Ressourcen r wobei $r \notin \{R_{List} \cup R_{Act}\}$	I_2	Anhang A.7 auf Seite 224
Delayed Content Resource Creation Pattern	P_{DCRC}	Nein	Zuverlässiges Anlegen von neuen Ressourcen r wobei $r \notin \{R_{List} \cup R_{Act}\}$	I_2	Anhang A.7 auf Seite 224

Tabelle 5.8: Muster für die Interaktion mit einer Listenressource

(I_1) **Bereitstellung von Ressourcen**

Die Bereitstellung von Ressourcen einer Listenressource $RC(r_{list})$ erfolgt nach einer Anfrage eines Service-Konsumenten S_K. Die Listenressource r_{list} liefert dann die Ressourcen bzw. deren Repräsentationen zurück, für welche die Listenressource eine Eignerschaft im Sinne einer Aggregationsbeziehung aufweist. Diese Menge an Ressourcen für eine Listenressource wird durch die Menge $RC(r_{list})$ beschrieben, während die Gesamtheit der zurückliefernden Repräsentationen durch $Repr(RC(r_{list}))$ abgebildet wird. Da die Menge $RC(r_{list})$ beliebig groß sein kann, empfiehlt es sich, eine entsprechende Paginierung einzusetzen, die durch die Muster P_{RCT}, P_{ORCT} und P_{CRCT} adressiert wird. Dabei ist P_{RCT} das abstrakte Muster, das aus dem *Resource Collection Traversal*-Muster von Pautasso et al. abgeleitet wurde. Trotz einer geeigneten Paginierung können die Repräsentationen $Repr(RC(r_{list}))$ vom Informationsgehalt eine große Übertragungsmenge einnehmen, obwohl nur eine Teilmenge benötigt wird. Hier liefert das Muster P_{CRR} einen entsprechenden Lösungsansatz. Schließlich können verschiedene nutzungsspezifische Anforderungen (vgl. Abschnitt 5.1.2) existieren, die durch P_{RCF}, P_{RCS},

P_{RCIS} als auch P_{RCT} in geeigneter Form adressiert und auch bei heutigen Web-APIs häufig angewendet werden.

(I_2) **Anlegen von neuen Ressourcen**
Das Anlegen von neuen Ressourcen erfolgt über eine Repräsentation der anzulegenden Ressource $Repr(r)$ wobei $r \notin \{r_{List} \cup R_{Act}\}$, die vom Service-Konsumenten S_K an eine Listenressource des Service r_{list} übermittelt wird. Die Antwort teilt dem Service-Konsumenten S_K mit, ob die Ressource r ordnungsgemäß angelegt und der Listenressource r_{list} hinzugefügt wurde. Allerdings kann nicht garantiert werden, dass der Service-Konsument S_K diese Antwort auch tatsächlich erhält. Als Beispiel ziehen Pautasso et al. hierfür die Eigenschaften von Netzwerken heran: „As all networks are not reliable, a client cannot know the reason for a missing response“ [PI+16, S. 5]. In diesem Fall wird der Service-Konsument S_K womöglich nochmals versuchen, diese Ressource anzulegen und so ungewollt Duplikate der Ressource erstellen.

Deshalb liefern Pautasso et al. zwei verschiedene Muster, um dieses Problem zu bearbeiten: *POST Once Exactly* und *POST-PUT Creation*. Obwohl die Muster auf dem Anwendungsschichtprotokoll HTTP basieren, sind sie auch auf andere Anwendungsschichtprotokolle übertragbar. Denn die Muster beziehen sich auf die Eigenschaft der Idempotenz bei der Erzeugung einer Ressource. So stellt *POST Once Exactly* durch ein Token sicher, dass eine Ressource unter Verwendung einer nicht idempotenten Methode genau einmal erzeugt wird. Hingegen verfährt *POST-PUT Creation* bei der Erzeugung in zwei Schritten. Erst wird eine nicht idempotente Methode zur Erzeugung verwendet und anschließend die angelegte Ressource mit Attributwerten unter Nutzung einer idempotenten Methode befüllt. Bei diesem Muster ist allerdings eine Ergänzung erforderlich, damit sichergestellt wird, dass Ressourcen ohne gesetzte Attributwerte nach einer bestimmten Zeit automatisch gelöscht werden. Diese Ergänzung, die Abstraktion von HTTP sowie die Verknüpfung mit den Qualitätsteilmerkmalen der Wiederverwendbarkeit finden sich in den jeweiligen Musterbeschreibungen der Muster P_{RRC}, P_{TRC} und P_{DCRC} wieder.

Interaktionen mit Primär-, Sub- und Informationsressourcen

Die Interaktionen von Primär-, Sub, und Informationsressourcen R_{Prim_n}, R_{Sub_n} und R_{Inf_n} lassen sich zusammenfassen, weshalb diese nun als $R = R_{Prim} \cup R_{Sub} \cup R_{Inf}$ bezeichnet werden, während mit r eine konkrete Ressource dieser Menge adressiert wird. Die entsprechenden Interaktionen zeigt Abbildung 5.13. Grundsätzlich existieren also drei verschiedene Interaktionen I_1, I_2 und I_3, die in diesem Abschnitt als I_n zusammengefasst sind und nun gemeinsam vorgestellt werden. Die dabei verorteten Muster sind Tabelle 5.9 zu entnehmen, die auf dieselbe Weise, wie bereits bei den Interaktionen mit einer Listenressource vorgestellt, zu interpretieren sind.

[2] Auf die Aufschlüsslung der Einflüsse auf die Interaktionsteilnehmer sowie Bezug zu etwaigen Qualitätsmerkmalen wurde verzichtet, da dieses Muster zur Interaktion keine gesonderten Vorzüge und Lasten aufweist.

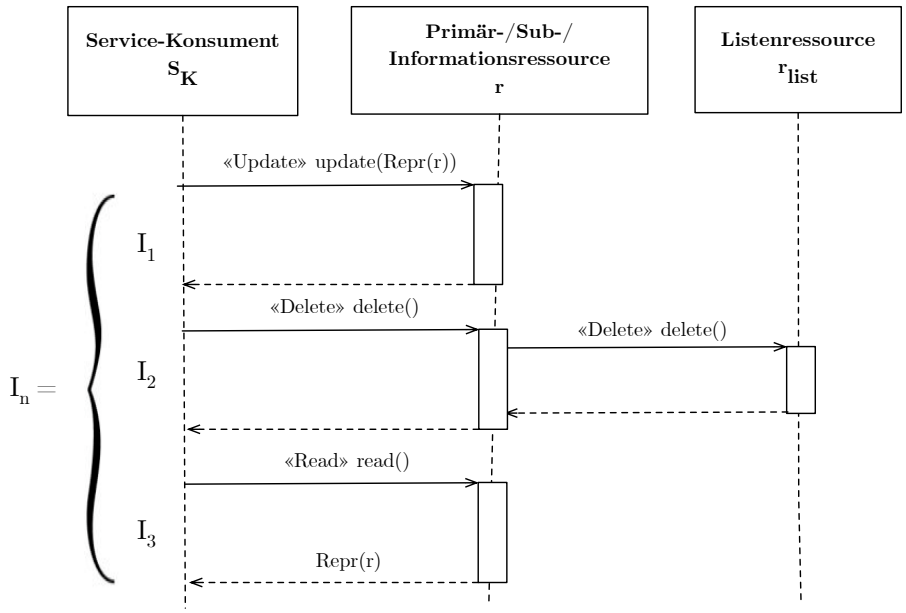


Abbildung 5.13: Interaktionen mit Primär-, Sub-, und Informationsressourcen

(I_n) **Abrufen, Bearbeiten und Löschen einer Ressource**
Mit I_1 wird das Bearbeiten einer Ressource r_n adressiert, das mit einer Repräsentation der Ressource $Repr(r)$ erfolgt. Der Status der Bearbeitung wird dem Service-Konsumenten S_K über eine entsprechende Nachricht ohne Nachrichteninhalt übermittelt. Es findet demnach keine Übermittlung der Repräsentation der geänderten Ressource $Repr(r)$ statt. Das liegt an der Anwendung des Musters *Command-Query Responsibility Segregation (CQRS)*, wodurch Schreib- und Leseoperationen unabhängig voneinander betrachtet werden sollten. Für die Bearbeitung einer Ressource r muss demnach festgelegt werden, welche Repräsentationen syntaktisch gültig sind, um den Zustand einer Ressource zu verändern. Hierfür dient das Muster P_{PRE} auf Basis von Pautasso et al. [PI+16]. Es beschreibt eine Lösung, wie Service-Konsumenten S_K die modifizierbaren Attribute einer Ressource r ableiten können. Für I_2 und I_3 liefern Pautasso et al. keine gesonderten Muster, sondern sie fassen diese unter dem *Collection Item Life Cycle Pattern* zusammen, ein Vorgehen, das diese Arbeit übernimmt, was sich auch

Muster	Abk.	Abstrakt	Absicht	Interakt.	Verweis
Partial Resource Editing Pattern	P_{PRE}	Nein	Bearbeiten einer Ressource r	I_1	Anhang A.7 auf Seite 221
Resource Life Cycle Pattern	P_{RLC}	Nein	Lebenszyklus einer Ressource r	I_1, I_2, I_3	[PI+16, S. 19f][2]

Tabelle 5.9: Muster für die Interaktion mit Primär-, Sub- und Informationsressourcen

im Muster P_{RLC} widerspiegelt. Die grundsätzlichen Interaktionen und deren Implementierung lassen sich auch der gängigen Literatur zu REST wie bspw. [RA+13, TE+15] entnehmen.

Interaktionen mit Aktivitätsressource

Eine Aktivitätsressource ermöglicht die Ausführung eines Geschäftsprozesses, der in dieser Arbeit durch einen Domänenservice repräsentiert wird. Grundsätzlich ergibt sich dadurch eine je unterschiedlich geartete Interaktion. Die konkrete Interaktion hängt davon ab, ob es um einen kurz- oder langlebigen Prozess geht, den die Anfrage des Service-Konsumenten S_K auslöst. Diese Interaktionen illustriert Abbildung 5.14, Tabelle 5.10 nennt die verorteten Muster.

Muster	Abk.	Abstrakt	Absicht	Interakt.	Verweis
Long Running Request Pattern	P_{LRR}	Nein	Ausführen von langlebigen Prozessen	I_2	Anhang A.7 auf Seite 222
Reliable Resource Creation Pattern	P_{RRC}	Ja	Zuverlässiges Anlegen von neuen Ressourcen r	I_2	Anhang A.7 auf Seite 223
Token-based Resource Creation Pattern	P_{TRC}	Nein	Zuverlässiges Anlegen von neuen Ressourcen r	I_2	Anhang A.7 auf Seite 224
Delayed Content Resource Creation Pattern	P_{DCRC}	Nein	Zuverlässiges Anlegen von neuen Ressourcen r	I_2	Anhang A.7 auf Seite 224

Tabelle 5.10: Muster für die Interaktion mit einer Aktivitätsressource

(I_1) **Kurzlebiger Prozess:** Bei einem kurzlebigen Prozess ist die Interaktion mit I_1 beendet. Das Initiieren des Prozesses ist vergleichbar mit dem grundsätzlichen Anlegen einer Ressource (siehe hierzu Interaktion I_2 im Kontext einer Listenressource), weshalb dies hier nicht nochmals im Detail aufgeführt wird. Dem Service-Konsumenten S_K wird durch Antwort Erfolg oder Misserfolg direkt mitgeteilt. Da diese Antwort auch verloren gehen kann, lassen sich die Muster P_{RRC}, P_{TRC} und P_{DCRC} auch für dieses Problem nutzen.

(I_2) **Langlebiger Prozess:** Ein langlebiger Prozess wird durch $I_{21} = I_1$ initiiert. I_{22} verschafft Informationen zum Status des Prozesses, während sich mit I_{23} der Prozess beenden lässt. Die Umsetzung dieser Interaktion basiert auf dem Muster P_{LRR}, das sich wiederum auf das Muster *Long Running Request* nach [PI+16] stützt.

Weitere Muster für die Interaktion mit einer Ressource

Nachdem eben die einzelnen Ressourcenarten aus Abschnitt 5.1.1 behandelt wurden, widmet sich dieser Abschnitt weiteren Mustern, die nicht einer Ressourcenart zugeordnet werden können. Stattdessen lassen sich diese an der Absicht der jeweiligen Interaktionen verorten, die auch durch die Stereotypen `Create`, `Read`, `Update`, `Delete` und `Execute` geprägt werden. Die Muster und deren Zuordnung zu

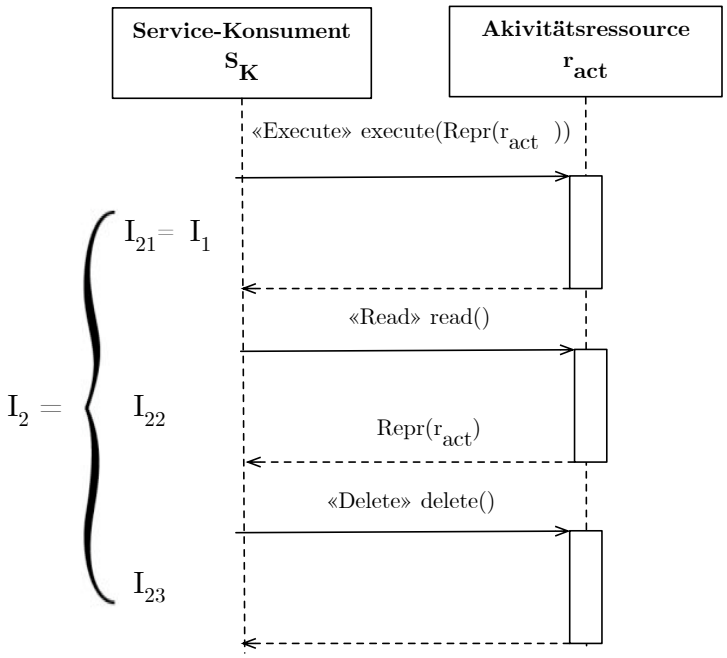


Abbildung 5.14: Interaktion mit einer Aktivitätsressource

den Absichten einer Interaktion nennt Tabelle 5.11. Auf eine weiterführende Beschreibung wird hier verzichtet und dazu auf die jeweilige Musterbeschreibung verwiesen.

Muster	Abk.	Abstrakt	Absicht	Stereotyp	Verweis
Conditional Large Resource Non Safe Interaction Pattern	P_{CLRNSI}	Nein	Bedingte Bearbeitung oder Erstellung einer Ressource *r*	`Create`, `Update`	Anhang A.7 auf Seite 225[3]
Resource Field Selection Pattern	P_{RFS}	Nein	Selektierung von Informationen einer Ressource *r*	`Read`	Anhang A.7 auf Seite 226
Conditional Resource Retrieval Pattern	P_{CoRR}	Nein	Bedingtes Abrufen einer Ressource *r*	`Read`	Anhang A.7 auf 227

Tabelle 5.11: Muster für die Interaktion mit einer Ressource unabhängig von ihrer Ressourcenart

5.3.3 Anwendung der Mustersprache auf das Ressourcenmodell

Nachdem die MPL für Interaktionen mit Ressourcen auf Grundlage von [PI+16] und [GG+16c] vorgestellt wurde, erörtert dieser Abschnitt die systematische Anwendung der Mustersprache auf das zugrundeliegende Ressourcenmodell R_2 aus Abschnitt 5.2. Es ist das Ziel, Software-Architekten bei Entwurfsentscheidungen durch ein aufbereitetes Set an Mustern im Kontext von Interaktionen

[3] Dieses Muster kann im Gegensatz zu Pautasso et al. auch zur Erstellung einer Ressource angewendet werden.

zu unterstützen. Allerdings ist weiterhin das Problem ungelöst, die relevanten Muster aus der Mustersprache zu identifizieren, das durch die zunehmende Anzahl von Mustern zusätzlich erschwert wird. Deshalb wurde die MPL auf zwei Dimensionen abgebildet, um eine kompaktere Darstellung zu erhalten: 1) Ressourcenarten und 2) deren Operationen. Die Wahl der Dimensionen richtete sich dabei nach dem Ressourcenmodell R_2, aus dem sich die Ressourcenarten und bereitgestellte Operationen einzelner Ressourcen durch Stereotypen ableiten lassen (vgl. Abschnitt 5.1.3). Die Dimensionen lassen sich allerdings auch für andere Zwecke verwenden, um bspw. auf Grundlage einer Interaktionsabsicht relevante Muster zu selektieren. Die Abbildung der MPL auf die genannten Dimensionen gibt Tabelle 5.12 wieder. Grau markierte Flächen stellen Interaktionen dar, für das die jeweilige Ressourcenart keine Operation bereitstellt. So sollte bspw. keine Bearbeitung (entspricht dem Stereotyp `Update`) einer Listenressource möglich sein, da sie lediglich eine Sammlung bestehender Ressourcen ist. Dementsprechend finden sich darin auch keine Muster wieder, die von der MPL aufgegriffen werden.

	«Create»	«Read»	«Update»	«Delete»	«Execute»
R_{Prim}	–	P_{RFS}, P_{CRR}, P_{RLC}, P_{CaRR}	P_{PRE}, P_{RLC}, P_{CLRNSI}	P_{RLC}	–
R_{Sub}	–	P_{RFS}, P_{CRR}, P_{RLC}, P_{CaRR}	P_{PRE}, P_{RLC}, P_{CLRNSI}	P_{RLC}	–
R_{Inf}	–	P_{RFS}, P_{CRR}, P_{RLC}, P_{CaRR}	P_{PRE}, P_{RLC}, P_{CLRNSI}	P_{RLC}	–
R_{List}	P_{CLRNSI}, P_{RRC}, P_{TRC}, P_{DCRC}	P_{RFS}, P_{CRR}, P_{RCT}, P_{CaRR}, P_{ORCT}, P_{CRCT}, P_{CRR}, P_{RCF}, P_{RCS}	–	–	–
R_{Act}	–	–	–	–	P_{LRR}, P_{RRC}, P_{TRC}, P_{DCRC}

Tabelle 5.12: Gruppierung der Muster in Ressourcenarten und Operationen von Ressourcenarten

Die Anwendung der Mustersprache für ressourcenorientierte Interaktionen am Ressourcenmodell R_2 kann durch dieselbe Benennung der einzelnen Interaktionsschritte mit den Annotationen `«Create»`, `«Read»`, `«Update»`, `«Delete»` und `«Execute»` ohne eine etwaige Zuweisungsvorschrift erfolgen. Das Vorgehen ist also vergleichsweise einfach und lässt sich in die nachfolgenden zwei Schritte unterteilen, die sich auch aus der erweiterten Kategorisierung der Ressourcen ergeben (siehe Abbildung 5.4 auf Seite 94). Bei nutzungsspezifischen Anforderungen (vgl. Abschnitt 5.1.2) ist dies allerdings nicht ohne eine weitere Interpretation gegebener Nutzeranforderungen möglich.

1. **Betrachtung des Ressourcenmodells R_2:**
   In diesem Schritt wird ausschließlich das Ressourcenmodell R_2 samt der darin enthaltenen Ressourcen betrachtet. Jede Ressource ist mindestens einer Ressourcenart zugewiesen, sodass

nun die Tabelle 5.12 zur Identifikation relevanter Muster herangezogen werden kann. Gleichzeitig können die bereitzustellenden Operationen «Create», «Read», «Update», «Delete» und «Execute» (entsprechen den Annotationen durch Verwendung der Stereotypen aus dem UML-Profil *ResourceOrientation*) einer Ressource *r* die Muster weiter eingrenzen. Hier muss der Software-Architekt wieder über die Anwendung der Muster entscheiden, die damit einhergehenden Vorzüge und Nachteile miteinander vergleichen und zudem erkennen, ob das geschilderte Problem für das abzubildende Anwendungsszenario zutrifft.

2. **Betrachtung der nutzungsspezifischen Anforderungen:**
   Die Betrachtung der nutzungsspezifischen Anforderungen fokussiert die Service-Konsumenten als Interaktionsteilnehmer. Bei der Identifikation von Ressourcenarten wurde bereits festgestellt, dass sich die benötigten Fähigkeiten einer Listenressource aus dem Bedürfnis der Service-Konsumenten ableiten und den nutzungsspezifischen Anforderungen zuordnen lassen. Im Kontext der MPL wurden ebenfalls Muster identifiziert, die Lösungen für nutzungsspezifische Anforderungen anbieten. Dazu zählen: a) P_{RCF} (Filterung $R_{List_{Filt}}$), b) P_{RCS} (Sortierung $R_{List_{Sort}}$), c) $P_{RCT}, P_{ORCT}, P_{CRCT}$ (Paginierung $R_{List_{Pag}}$), d) P_{RFS} (Selektion R_{Sel}), e) P_{CI} (Bedingte Bearbeitung R_{CLRNSI}), f) P_{CoRR} (Bedingtes Lesen R_{Con}) und g) P_{PRE} (Partielle Bearbeitung R_{PartUp}) . Diese Fähigkeiten werden nicht zwingend benötigt, um die Operationen einer Ressource in adäquater Weise zu adressieren, sondern wirken lediglich ergänzend. Gleichwohl kann eine nicht berücksichtigte Fähigkeit dazu führen, dass die Wiederverwendbarkeit aufgrund des damit adressierten Problems verschlechtert wird. Die Beziehung der Fähigkeiten zu den einzelnen Ressourcenarten zeigt Abbildung 5.15 übersichtlich. Diese ergänzt zudem Abbildung 5.3 auf Seite 93, bei der nur die Paginierung und die Filterung als Fähigkeiten ermittelt wurden.

Nachdem entsprechende Muster ausgewählt wurden, werden diese dem Ressourcenmodell R_2 durch UML-Kommentare an den entsprechenden Operationen angefügt. Falls es sich um implizite Operationen handelt, verweist der UML-Kommentar auf das Modellelement, da sich anhand des Musters ein Bezug zur impliziten Operation herstellten lässt. Abbildung 9.7 in Anhang A.10 illustriert die angewandte Mustersprache anhand einer Nutzeranforderung im Kontext des in Abschnitt 4.3 betrachteten Szenarios.

5.4 Festlegung von Repräsentationen

Der vorige Abschnitt hat die Interaktionen der verschiedenen Ressourcenarten betrachtet. Nun wird die Festlegung der Repräsentationen der einzelnen Ressourcen fokussiert. Der Abschnitt diskutiert folgende Fragen, die sich sowohl bei Erforschung des ressourcenorientierten Entwurfs als auch im Dialog mit Akteuren aus der Industrie ergeben haben.

(1) Wie ist ein geeigneter Medientyp für die Repräsentation einer Ressource aus Ressourcenmodell R_2 (siehe Abbildung 5.15) zu wählen?

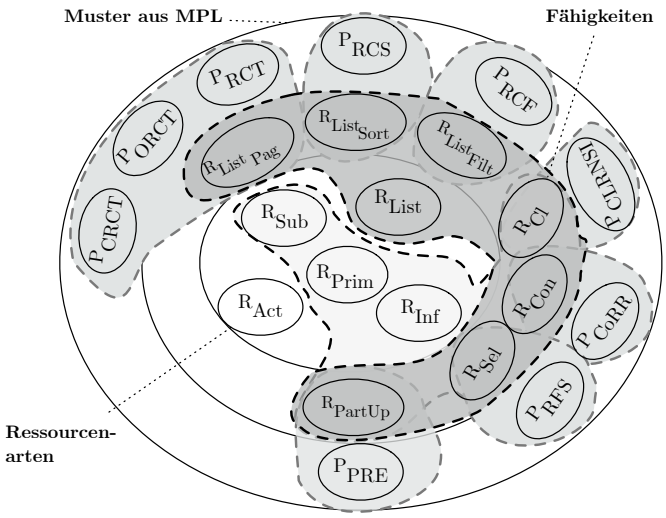


Abbildung 5.15: Ressourcenarten und deren ergänzende Fähigkeiten

(2) Nachdem mit (1) ein passender Medientyp für die Repräsentation gefunden wurde, ist zu klären: Wie kann ein passendes Schema für eine Repräsentation einer Ressource aus R_2 abgeleitet werden?

(3) Sind noch weitere Repräsentationen abzuleiten, die sich nicht aus dem Ressourcenmodell R_2 ergeben und somit keinen konkreten Anwendungskontext aufweisen?

Abbildung 5.16 gibt eine Übersicht des ganzheitlichen Prozesses zur Festlegung von Repräsentationen. Die einzelnen Unterabschnitte liefern das notwendige Hintergrundwissen und helfen, diese Fragen zu beantworten.

5.4.1 Auswahl eines geeigneten Medientyps

Bei ressourcenorientierten Web-APIs stellen Repräsentationen den zentralen Nachrichteninhalt einer Interaktion dar. Das Format des Nachrichteninhalts wird mit *Multipurpose Internet Mail Extensions (MIME)-Typen* definiert (nachfolgend als *Medientypen* bezeichnet), damit der Empfänger den Nachrichteninhalt korrekt interpretieren kann. Ein Medientyp hat grundlegend zwei Bestandteile: (1) den Primärtyp und (2) den Subtyp [RFC2045-1996]. Eine Liste verfügbarer und registrierter Medientypen ist der Internet Assigned Numbers Authority (IANA)[4] zu entnehmen. Alternativ können auch eigene Medientypen definiert werden. Hierzu ist dem entsprechenden Subtyp gemäß dem RFC 2045 [RFC2045-1996] ein *x-* oder *X-* voranzustellen. Der RFC 6838 [RFC6838-2013] erweitert diesen

[4] http://www.iana.org/assignments/media-types/media-types.xhtml (Letzter Zugriff: 28.09.2017)

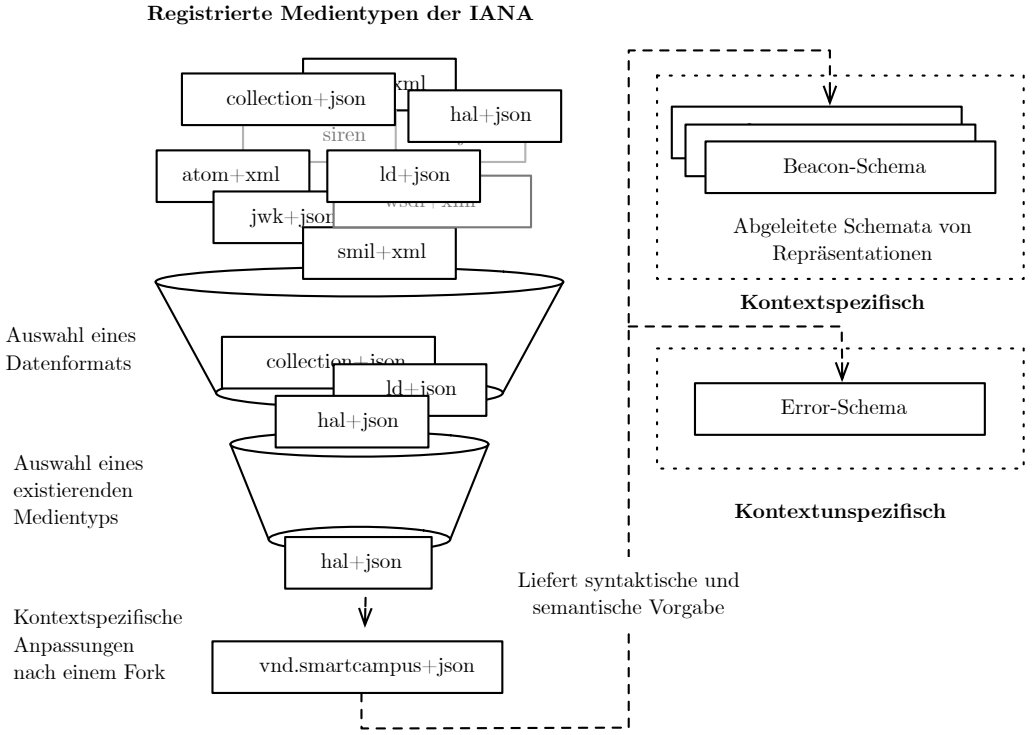


Abbildung 5.16: Übersicht des Prozesses zur Ableitung von Repräsentationen

Bestandteil der Spezifikation von Medientypen u. a. durch die Einführung sogenannter Registrierungsbäume (engl. registration trees), um die Effizienz und Flexibilität des Registrierungsprozesses von neuen Medientypen zu steigern. Registrierungsbäume sollen den Geltungsbereich (engl. scope) des jeweiligen Medientyps bestimmen. Dafür wurden vier Registrierungsbäume definiert (vgl. Tabelle 5.13). Weiter wurde ein Suffix an den Subtyp angehängt, um die Konformität zu einem Datenformat festzulegen. So zeigt bspw. das Suffix `+json` die Konformität zum Datenformat JSON an.

Bevor nun die eigentlichen Repräsentationen auf Grundlage des Ressourcenmodells abgeleitet werden können, muss ein bestehender Medientyp für die weitere Verwendung ausgewählt oder alternativ ein neuer Medientyp definiert werden. Die Definition und Registrierung eines neuen Medientyps erfolgt abhängig von dem jeweiligen Registrierungsbaum in Tabelle 5.13 und ist in [RFC6838-2013] weiter ausgeführt. Grundsätzlich muss der Software-Architekt den Medientyp wählen. Allerdings sollte nur sehr selten tatsächlich ein neuer und eigenständiger Medientyp entworfen werden, da Software-Entwickler bereits mit einigen der existierenden Medientypen vertraut sind und gegebenenfalls auch eine entsprechende Werkzeugunterstützung vorliegt. Doch schränkt die Verwendung eines existierenden Medientyps andererseits die Flexibilität ein, da substantielle Änderungen die Vorgaben des Medientyps verletzen könnten. Das nachfolgend präsentierte Vorgehen richtet sich an dieses Problem und verknüpft die genannten Vorteile existierender mit denen neu definierter Medientypen.

Registrierungs-baum	Beschreibung	Prefix	Beispiel
Standard	Geeignet für Medientypen, die unabhängig von einem konkreten Produkt sind.	Keine	application/json
Anbieter	Geeignet für Medientypen, die mit einem öffentlich verfügbaren Produkt assoziiert werden.	vnd	application/vnd.ibm.modcap
Personell	Geeignet für Medientypen, die experimentell genutzt werden oder Bestandteil eines Produktes sind, das nicht kommerziell vertrieben wird.	prs	application/prs.cww
Unregistriert	Ausschließlich für privat genutzte Medientypen geeignet.	x	application/x.prototype

Tabelle 5.13: Registrierungsbäume gemäß RFC 6838 [RFC6838-2013]

Gleichzeitig wird der Software-Architekt bei der Auswahl eines geeigneten Medientyps durch eine Vorauswahl des Datenformats unterstützt. Sollten mehrere Datenformate unterstützt werden, so kann das nachfolgende Vorgehen wiederholt werden.

1. **Auswahl eines geeigneten Datenformats:** Zuerst ist zu klären, welches Datenformat für die Repräsentation verwendet werden soll. Bekannt sind etwa XML und JSON. Aufgrund der Kompaktheit des Datenformats ist JSON zunehmend populär bei heutigen Web-APIs [TE+15]. Die Wahl des Datenformats kann die Wahl eines geeigneten Medientyps deutlich einschränken. So ließen sich z.B. durch Wahl des Datenformats JSON die möglichen Medientypen von 1257 auf 145 reduzieren[5].

2. **Auswahl eines existierenden Medientyps:** Nachdem der letzte Schritt die möglichen Datenformate eingegrenzt hat, muss nun aus dieser Menge ein geeigneter Medientyp gewählt werden. Es empfiehlt sich, einen Medientyp zu selektieren, der bereits eine Unterstützung von Hypermedia aufweist, sodass diesbezüglich keine Einschränkungen entstehen. Ob Hypermedia dann auch tatsächlich genutzt wird, muss das Entwicklungsteam entscheiden. Als Beispiele seien hier die Medientypen *HAL, Siren, Collection+JSON* oder *JSON-LD* erwähnt, die eine Unterstützung von Hypermedia aufweisen. Die Beschränkung auf Medientypen mit Hypermedia-Unterstützung kann nochmals die Auswahl eingrenzen[6].

3. **Abspaltung (engl. Fork) vom ausgewählten Medientyp:** Es wird eine Abspaltung des ausgewählten Medientyps vorgenommen und ein neuer Medientyp definiert, der den Bounded

[5] Die Ermittlung erfolgte anhand der registrierten Medientypen der IANA[4] unter der Annahme, dass alle registrierten Medientypen das Suffix für das Datenformat verwenden (Zeitpunkt der Durchführung: 29.09.2017).

[6] Genaue Zahlen können an dieser Stelle nicht geliefert werden, da jeder Medientyp im Einzelnen auf die Unterstützung von Hypermedia untersucht werden müsste.

Context in geeigneter Form beschreiben kann. Dies bedeutet, dass alle bekannten Konzepte des abgespaltenen Medientyps weiterhin gelten und sich nur unter einem neuen Medientyp wiederfinden. So sind Erweiterungen der grundsätzlichen Struktur weiterhin möglich, um die gewünschte Flexibilität zu gewährleisten. Im Kontext des SmartCampus würde sich bspw. folgende Medientyp-Bezeichnung anbieten: `application/vnd.smartcampus.hal+json`.

5.4.2 Ableitung von Schemata für Repräsentationen

Nachdem im Vorfeld der Medientyp und damit die Grundstruktur für eine Repräsentation festgelegt wurde, ist nun die konkrete Ableitung einer Repräsentation aus dem zugrundeliegenden Ressourcenmodell R_2 möglich. Der Inhalt einer Repräsentation besteht aus den Attributen und Assoziationen einer Ressource und ist mit einem Data Transfer Object (DTO) vergleichbar, das Informationen bündelt, um diese zwischen verschiedenen Prozessen oder über Systemgrenzen hinweg auszutauschen [Fo02a]. Um die letztliche Struktur einer Repräsentation zu beschreiben und auf Validität zu überprüfen, wird nun gezeigt, wie ein *Schema* abgeleitet werden kann. Als *Schema* wird in dieser Arbeit die Beschreibung der Struktur einer Repräsentation bezeichnet, die um zusätzliche Randbedingungen zur Validierung angereichert werden kann. Bekannte Beispiele für die Definition von Schemata sind bspw. JSON-Schema für JSON oder XML Schema Definition (XSD) für XML.

Das nun skizzierte Vorgehen ist auf jedes Modellelement mit Spezialisierung des Stereotyps `Resource` mit Ausnahme von `ListResource` anzuwenden. Die Ausnahme lässt sich damit begründen, dass eine Listenressource lediglich auf eine Menge an Ressourcen verweist und deshalb keine eigenständige Repräsentation besitzt. Weil JSON immer häufiger als primäres Datenformat bei ressourcenorientierten Web-APIs verwendet wird, ist JSON-Schema die Wahl für die letztliche Demonstration. Die nachfolgenden Konzepte können allerdings mit geringer Adaption auch auf andere Datenformate angewandt werden. Mittlerweile sind auch Lösungen verfügbar, um die Schemata in unterschiedliche Sprachen zu transformieren.

1. **Ableitung von Attributen:** Grundsätzlich stellt jedes Attribut einer Ressource $Attr_r$ ein Attribut einer Repräsentation dar $Attr_{Repr(r)}$. Für die Bezeichnung des Attributs $Attr_{Repr(r)}$ ist der Name des Attributs $Attr_r$ zu nutzen, wobei aus Gründen der Konsistenz auf eine einheitliche Schreibweise zu achten ist. Deshalb sollte die Bezeichnung bestimmten Vorgaben gehorchen, wie bereits in Tabelle 5.5 für die Adressierung gezeigt wurde. Für jedes Datenformat sollten spezifische Regeln festgelegt werden. Bei der Verwendung von JSON betrifft das eine einheitliche Binnenmajuskel-Schreibweise. Außerdem sind alle Attribute mit doppelten Anführungszeichen zu versehen. Diese Vorgaben sollte ein unternehmensweit eingesetzter Styleguide zusammenfassen. Ein entsprechendes Beispiel für JSON liefert Google mit [GOOGLE-JSON]. Falls dem jeweiligen Attribut $Attr_r$ eine Multiplizität `[0..1]` zugewiesen wurde, so ist das Attribut in der Repräsentation $Attr_{Repr(r)}$ als optional zu werten. Dann muss der Attributwert nicht besetzt sein.

Es können weitere Randbedingungen im jeweiligen Schema aufgenommen werden, falls sie in geeigneter Form abgebildet werden können. Die Abbildung von Randbedingungen beschränkt sich allerdings auf die erlaubten Werte, die das Attribut $Attr_{Repr(r)}$ annehmen darf.

Falls sich eine Vorbedingung nicht dieser Gruppierung zuordnen lässt, kann die Prüfung erst bei der Implementierung erfolgen. Neben der Beschreibung etwaiger Randbedingungen ist für jedes Attribut $Attr_{Repr(r)}$ eine entsprechende Bezeichnung `name` sowie optional eine Beschreibung `description` anzugeben. Diese ergeben sich aus dem zugrundeliegenden Domänenmodell bzw. aus dessen Begriffssammlung. Das verschafft den Entwicklern der Service-Konsumenten S_K mehr Einsicht in die Semantik der strukturellen Aspekte der abgebildeten Domäne. Für die Überführung der Datentypen der einzelnen Attribute aus $Attr_r$ müssen diese auf das letztliche Datenformat des Medientyps überführt werden. Da dies wiederum vom konkreten Datenformat abhängt, wird es hier nicht weiter ausgeführt.

2. **Ableitung von Assoziationen:** Jede gerichtete Assoziation eines Modellelements mit dem Stereotyp `link` wird in Repräsentationen ebenfalls durch ein Attribut abgebildet. Die Bezeichnung des Attributs ergibt sich dabei durch die Bezeichnung der gerichteten Assoziation. Falls dieser eine Bezeichnung nicht zugewiesen wurde, dient die Ressourcenbezeichnung des Ziels der gerichteten Assoziation als Bezeichnung. Wenn es sich beim Ziel der Assoziation um eine Ressource $r \in \{R_{Prim} \cup R_{Sub}\}$ handelt, dann entspricht der mögliche Wert des Attributs dem Attribut, das über den Stereotyp `identifier` verfügt. Ist es hingegen eine Informationsressource r_{inf} oder Listenressource r_{list} dann wird die $Repr(r_{inf})$ bzw. $Repr(RC(r_{list})$ direkt eingebettet bzw. ein Verweis auf das entsprechende Schema gesetzt. Alternativ lassen sich auch Verlinkungen einfügen, um direkt auf das jeweilige Ziel navigieren zu können. Ein Beispiel mittels *HAL* für eine derartige Verlinkung zeigt der Quelltextauszug 5.1.

```
/* ...*/
"_embedded": {
    "beacons": [{
      "id": "edbc0444c867abc0e",
      "_links": {
        "self": { "href": ".../beacons/edbc0444c867abc0e" }
      }
    },{
      "id": "83af6d3172a43e288c",
      "_links": {
        "self": { "href": ".../beacons/83af6d3172a43e288" }
  }}]
  },
/*...*/
```

Quelltext 5.1: Auszug einer Repräsentation mittels HAL

Zur Demonstration des Vorgehens findet sich in Abbildung 5.17 ein entsprechendes Beispiel anhand des Applikationsszenarios, was die Anwendbarkeit des Vorgehens aufzeigen soll.

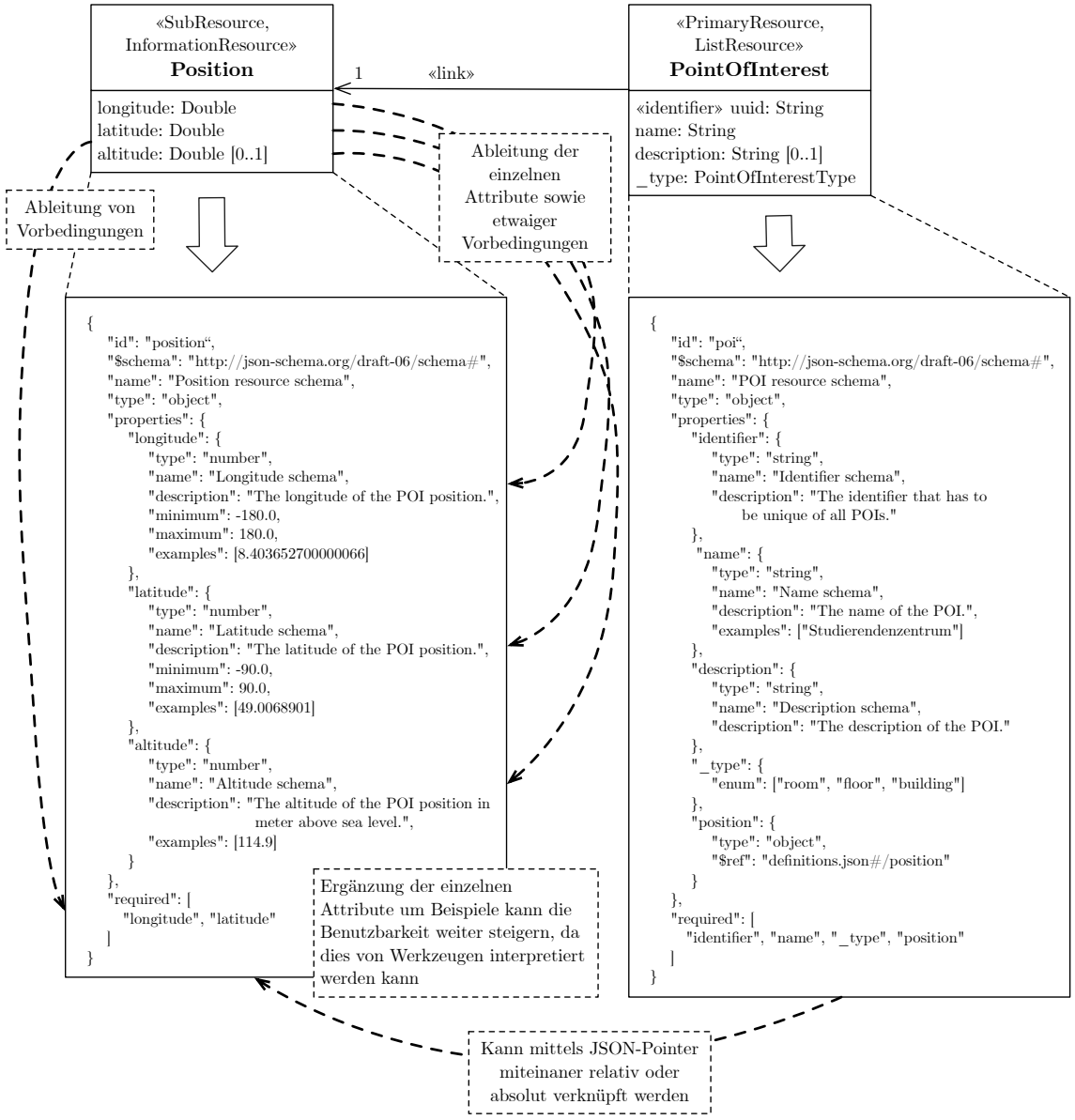


Abbildung 5.17: Auszugsweise Ableitung von Repräsentationen aus einem Ressourcenmodell R_2

5.4.3 Kontextunabhängige Repräsentation für auftretende Fehler

Für den ressourcenorientierten Entwurf von Microservices wird eine Repräsentation unabhängig vom konkreten Bounded Context immer wieder benötigt. Diese Repräsentation ist nicht von einem Ressourcenmodell ableitbar und hat deshalb auch keinen direkten Bezug zu einem Bounded Context.

Zudem wird sie erst bei etwaigen Laufzeitfehlern benötigt, die bei der Nutzung eines Microservice auftreten können und sich nie völlig ausschließen lassen. Deshalb muss die entsprechende Meldung des Fehlers aussagekräftig sein, damit Entwickler von Service-Konsumenten S_K dessen Ursache lokalisieren und im Idealfall beheben können. Sie hat demnach einen erheblichen Einfluss auf die Benutzbarkeit eines Microservice.

Der Aufbau einer derartigen Fehlermeldung ist unabhängig von einem konkreten Kontext und sollte aus Gründen der Konsistenz für alle Web-APIs ähnlich wie in Tabelle 5.5 einmal definiert werden. Der folgende Aufbau stützt sich auf den Medientyp in RFC 7807 [RFC7807-2015], der allerdings nie den Status der Registrierung erreicht hat. Dennoch wird er bspw. in einem großen Automobilkonzern angewendet und ist in leicht abgewandelter Form in [GG+16c] zu finden. Der Medientyp hat vier Bestandteile, die Tabelle 5.14 nennt. Die Tabelle hat zudem eine Spalte, die angibt, durch welches Attribut der letztliche Bestandteil in einem JSON-Schema repräsentiert wird. Auf entsprechende Beispiele zu Typ und Instanz wurde verzichtet, da diese lediglich fiktive URIs darstellen würden. Das entsprechende JSON-Schema ist Anhang A.8 zu entnehmen.

Bestandteil	Attribut	Optional	Beschreibung	Beispiel
Typ	type	Ja	URI-Verweis auf weiterführende Informationen der Fehlerkategorie	-
Titel	title	Nein	Bezeichnung der Fehlerkategorie unabhängig vom konkreten Auftreten	Verletzung von Invarianten der Domäne
Status	status	Nein	Zurückgelieferter HTTP-Statuscode, sodass dieser direkt verarbeitet werden kann	403
Detail	detail	Nein	Beschreibung des Fehlers mit Bezug zum konkreten Auftreten	Bei der Bearbeitung des Domänenobjekts *Position* muss das Attribut *longitude* in der Repräsentation gesetzt sein.
Instanz	instance	Ja	URI-Verweis auf weiterführende Informationen des spezifischen Fehlers	-

Tabelle 5.14: Repräsentation einer Fehlermeldung gemäß RFC 7807 [RFC7807-2015]

5.5 Ergänzung um eine Versionierung

Im vorigen Abschnitt wurden die verschiedenen für eine Interaktion benötigten Repräsentationen abgeleitet. Nun ist die Versionierung einer Web-API bzw. des Ressourcenmodells zu erörtern. Eine Versionierung gibt in diesem Kontext Aufschluss über die evolutionäre Stabilität einer Web-API, was auch in [LX+13] und [EZ+14] aufgezeigt wird. Die evolutionäre Stabilität lässt sich anhand der Anzahl und des Ausmaßes der notwendigen Änderungen ermitteln, die durch Änderungen an einer Web-API erforderlich werden. Allerdings lassen sich Änderungen nie völlig ausschließen. So können sich bspw. Änderungen als Folge von Veränderungen oder Erweiterungen des abgebildeten Bounded

Context ergeben. Ebenso können Fehler im laufenden Betrieb Korrekturen erfordern. Insbesondere für Service-Konsumenten können Änderungen an einer Web-API im Vergleich zu lokalen APIs erhebliche Aufwände verursachen (vgl. auch Abschnitt 5.3.1): „In traditional local applications, when some API elements (such as classes and methods) of a local API (an API without network interactions) change, the client may choose to continue using the old API if client developers do not want to upgrade. However, in the service paradigm, the service of an old API is often shut down after a certain period, and the client has to be upgraded to adapt to the new API, otherwise the client will stop working“ [LX+13, S. 300]. Eine geeignete Versionierung ist daher unumgänglich, was auch Mulloy bestätigt: „Never release an API without a version and make the version mandatory“ [Mu12, S. 13]. Deshalb sind folgende Fragen nun zu beantworten:

(1) Existieren verschiedene Arten von Änderungen an einer Web-API und falls ja, welche Arten von Änderungen gilt es zu unterscheiden?

(2) Wie können die identifizierten Änderungsarten aus (1) den Entwicklern von Service-Konsumenten kenntlich gemacht werden?

(3) Wie sieht im Hinblick auf (2) ein systematisches und nachvollziehbares Vorgehen aus um etwaige Änderungen angemessen zu behandeln?

5.5.1 Abwärts- und nicht abwärtskompatible Änderungen

Bei Änderungen an einer Web-API lassen sich abwärtskompatible und nicht abwärtskompatible Änderungen differenzieren (nachfolgend als *Änderungsarten* bezeichnet), womit die Fragestellung (1) bereits beantwortet wird [LX+13]. Der Unterschied besteht darin, dass abwärtskompatible – anders als nicht abwärtskompatible Änderungen – auf Service-Konsumentenseite keine Veränderungen verlangen. Deshalb ist darauf zu achten, vor allem abwärtskompatible Änderungen an der Web-API durchzuführen. Nicht abwärtskompatible Änderungen ergeben sich, wenn bestehende strukturelle oder verhaltensspezifische Aspekte sich ändern und sie dadurch nicht ergänzend wirken. Dies ist zum Beispiel dann der Fall, wenn eine bestehende Beziehung mit `link` gelöscht wird oder ein bestehendes Attribut einer Ressource $Attr_r$ umbenannt wird. Neue hinzugefügte Beziehungen oder neue Attribute sind hingegen keine Probleme im Hinblick auf die Kompatibilität, da diese ergänzend wirken. Hierzu liefert [LX+13] eine Auflistung möglicher Änderungsarten, die sich auf eine Studie von fünf bekannten Web-APIs stützt.

Um die entsprechende Änderungsart den Entwicklern von Service-Konsumenten kenntlich zu machen, wurde die *Semantic Versioning Specification (SemVer)* in der Version 2.0.0 [PW17] verwendet und damit die Fragestellung (2) beantwortet. Deren Eignung für die Versionierung belegt folgende Aussage: „Software using Semantic Versioning MUST declkeare a public API“ [PW17]. Auf Basis dieser Spezifikation besteht eine Versionskennung einer API aus folgenden drei Bestandteilen, die mit einem

Punkt konkateniert werden: `Major.Minor.Patch`. Die Interpretation der einzelnen Bestandteile ist Tabelle 5.15 zu entnehmen.

Bestandteil	Einheit	Default	Beschreibung
Major	$\mathbb{N}^*$	0	Repräsentiert die Hauptversion der Web-API, die inkrementiert wird, sobald eine nicht abwärtskompatible Änderung vorgenommen wird.
Minor	$\mathbb{N}^*$	0	Repräsentiert eine Zwischenversion der Web-API, die inkrementiert wird, sobald eine neue Funktionalität hinzugefügt wurde, die eine abwärtskompatible Änderung ist. Sollte die Hauptversion inkrementiert werden, wird die Nebenversion wieder auf den Default-Wert zurückgesetzt.
Patch	$\mathbb{N}^*$	0	Repräsentiert eine Verbesserung als Folge einer Fehlerkorrektur an der Web-API. Entscheidend ist dabei, dass es sich um eine abwärtskompatible Änderung handelt. Wenn die Haupt- oder Nebenversion inkrementiert wird, wird die Patch-Version wieder auf den Default-Wert zurückgesetzt.
* Die Menge der natürlichen Zahlen orientiert sich an der DIN-Norm 5473			

Tabelle 5.15: Interpretation der Versionskennung einer Web-API

Für die Manifestierung der Versionskennung im Ressourcenmodell wurde das Modellelement `package` von UML durch die Eigenschaftsdefinition `version` ergänzt. Als Eigenschaftswert ist die Versionskennung im Sinne von Tabelle 5.15 zu setzen. Die Ergänzung findet sich im erweiterten und weiterhin abwärtskompatiblen UML-Profil *ResourceOrientation* in Abbildung 5.18. Bei der initialen Erstellung erhält das Ressourcenmodell die Versionskennung `0.0.1`. Die Versionskennung kann in diesem Zusammenhang synonym für die Version eines Microservice angesehen werden.

5.5.2 Vorgehen bei Änderungen an der Web-API

Während also abwärtskompatible Änderungen keine Probleme verursachen, gehen nicht abwärtskompatible Änderungen immer mit Aufwand auf Service-Konsumentenseite einher. Da diese Änderungsart im Lauf der Zeit nicht grundsätzlich auszuschließen ist, muss ein Verfahren formuliert sein, wie mit derartigen Änderungen umzugehen ist. Das Verfahren adressiert damit die Fragestellung (3). Das nun präsentierte Vorgehen stützt sich auf Ergebnisse der Studie in [EZ+14], die den Einfluss derartiger Änderungen auf Service-Konsumenten untersucht hat. Zur Vollständigkeit wurden auch abwärtskompatible Änderungen dem Vorgehen hinzugefügt. Das Vorgehen ist in Abbildung 5.19 dargestellt. Die verwendeten Variablen im Kontext des Vorgehens erläutert Tabelle 5.16. Tabelle 5.17 vergleicht die Aktivitäten mit den Empfehlungen aus der empirischen Studie [EZ+14], um zu zeigen, wie die einzelnen Aktivitäten die Empfehlungen adressieren.

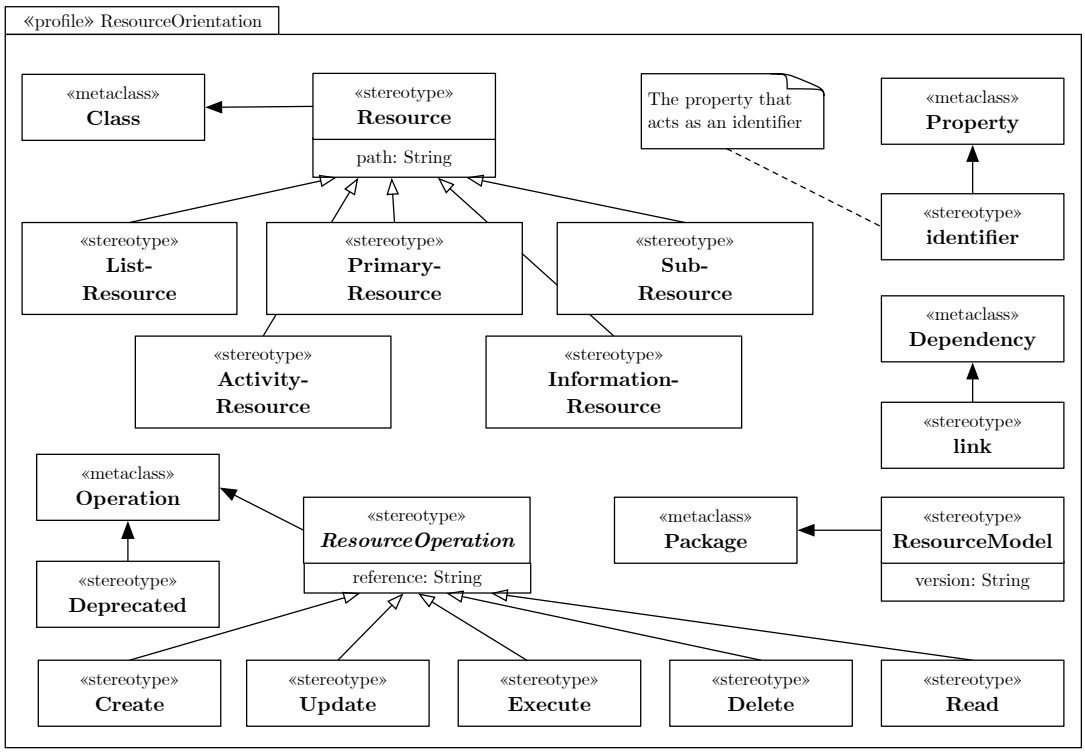


Abbildung 5.18: Erweitertes UML-Profil *ResourceOrientation* auf Basis von Abbildung 5.7

Variable	Beschreibung
$Vers_{Ma}(RM)$	Major-Version des Ressourcenmodells *RM* gemäß Tabelle 5.15
$Vers_{Mi}(RM)$	Minor-Version des Ressourcenmodells *RM* gemäß Tabelle 5.15
$Vers_{P}(RM)$	Patch-Version des Ressourcenmodells *RM* gemäß Tabelle 5.15
RM_{CUR}	Aktuelles Ressourcenmodell, das die aktuelle Web-API darstellt und von Service-Konsumenten verwendet wird
RM_{NEXT}	Neues Ressourcenmodell, das RM_{CUR} ablösen soll, wobei Folgendes gelten muss: $Vers_{Ma}(RM_{CUR}) > Vers_{Ma}(RM_{NEXT})$
$I(RM_{CUR})$	Lauffähige Instanz des Microservice mit der Web-API auf Basis von RM_{CUR}
$I(RM_{NEXT})$	Lauffähige Instanz des Microservice mit der Web-API auf Basis von RM_{NEXT}

Tabelle 5.16: Verwendete Variablen in der Beschreibung des Vorgehens

Ausgangspunkt des Vorgehens ist die durchzuführende Änderung am Ressourcenmodell, die zuerst hinsichtlich ihrer Abwärtskompatibilität geprüft werden muss. Ob es eine abwärtskompatible Änderung ist, hat bereits der vorige Abschnitt 5.5 geklärt. Abhängig von diesem Ergebnis wird dann entweder der Kontrollfluss für abwärtskompatible oder für nicht abwärtskompatible Änderungen weiterverfolgt. Dadurch lässt sich die folgende Beschreibung gliedern und übersichtlicher gestalten. Für die konkrete Umsetzung einer Versionierung dienen die Muster in Anhang A.9.

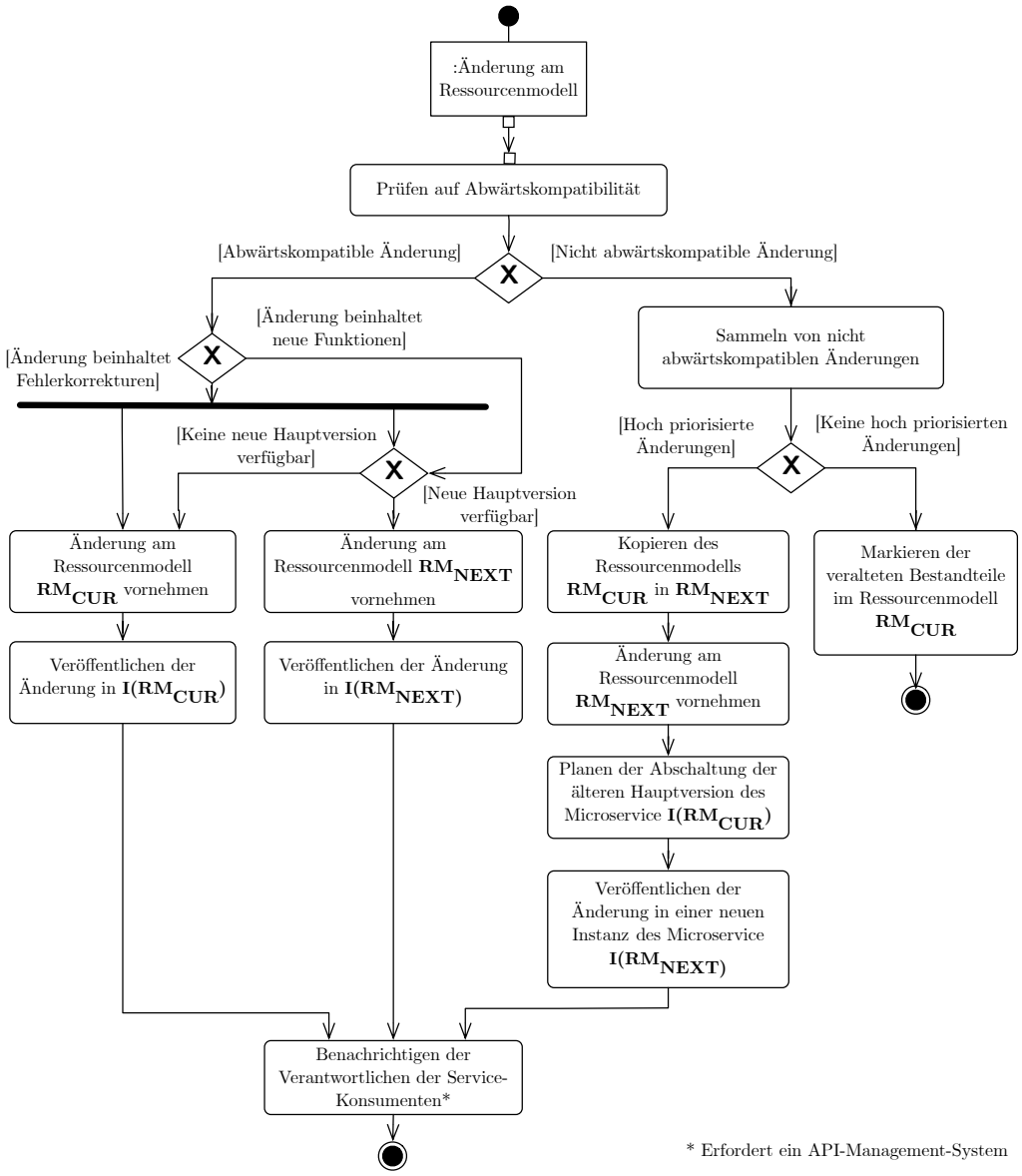


Abbildung 5.19: Vorgehen bei Änderungen an einer Web-API

Abwärtskompatible Änderung

Sobald eine erforderliche Änderung als abwärtskompatibel identifiziert ist, wird zunächst geprüft, ob es sich um eine Fehlerkorrektur (1) oder um eine neue Funktionalität (2) handelt. Im ersten Fall (1) wird die Änderung am RM_{CUR} durchgeführt und, falls vorhanden, an RM_{NEXT}, um zu gewährleisten, dass etwaige Fehler auch in der neuen Version behoben sind. Mit der Änderung wird auch die Patch-Version $Vers_P(RM)$ um 1 inkrementiert. Durch die Überführung auf die Implementierungsebene kann die Änderung in $I(RM_{CUR})$ und, sofern vorhanden, auch in $I(RM_{NEXT})$ veröffentlicht werden.

Andernfalls (2) ist die Änderung eine neue Funktionalität, welche die Web-API erweitert. Dann ist zunächst zu untersuchen, ob bereits RM_{NEXT} existiert. Falls vorhanden, wird nur dort die neue Funktionalität ergänzt. Dadurch sollen Entwickler der Service-Konsumenten dazu bewegt werden, möglichst schnell auf die neue Hauptversion zu wechseln. Nach der Durchführung der Änderung ist die $Vers_{Mi}(RM)$ um 1 zu inkrementieren. Der Kontrollfluss wird durch Benachrichtigung der Verantwortlichen für die Service-Konsumenten abgeschlossen, die für die Entwicklung von letzteren zuständig sind. Das setzt allerdings eine Form eines API-Management-Systems voraus, das u. a. Echtzeitanalysen zur Nutzung eines Microservice, Zugriffskontrolle oder auch eine Monetarisierung erlaubt. API-Management ist ein eigenes Forschungsfeld und nicht im Fokus dieser Arbeit. Deshalb muss hier der Verweis auf [De17] genügen.

Nicht abwärtskompatible Änderung

Alle nicht abwärtskompatiblen Änderungen werden gesammelt und priorisiert. Dieser Prozess berücksichtigt unterschiedliche Faktoren wie bspw. die Erhöhung des Geschäftswerts, die Neuausrichtung des abgebildeten Geschäftsausschnitts oder die geplante Abschaltung von benötigten Systemen, die von den Projektbeteiligten (engl. stakeholders) zu gewichten sind. Falls keine hohe Priorisierung vorliegt, werden lediglich offerierte Operationen im RM_{CUR} mit `Deprecated` markiert, sodass die Entwickler von Service-Konsumenten über eine baldige Änderung der Operation im Vorfeld informiert werden. Sollten hingegen die bis zu diesem Zeitpunkt gesammelten Änderungen als *hoch* priorisiert werden, dann wird zunächst das Ressourcenmodell RM_{CUR} in RM_{NEXT} kopiert.

Nun existieren zwei Ressourcenmodelle, um sicherzustellen, dass die bisherigen Service-Konsumenten weiterhin interoperabel bleiben und unbeabsichtigte Änderungen nicht vorgenommen werden. Danach werden die nicht abwärtskompatiblen Änderungen an RM_{NEXT} angewandt. Entsprechend Tabelle 5.15 wird die $Vers_{Ma}(RM)$ um 1 inkrementiert. Vor der Veröffentlichung des neuen Ressourcenmodells $I(RM_{NEXT})$ ist festzulegen, wie lange die bisherige Instanz des Microservice $I(RM_{CUR})$ mit RM_{CUR} noch verfügbar bleiben soll. Diese Entscheidung zur Dauer liegt im Ermessen der Projektbeteiligten. Doch sie sollte, so der Tenor von [EZ+14], nicht allzu großzügig gewählt werden, denn „longer periods leave developers too relaxed about the change“ [EZ+14, S. 92]. Um Entwickler von Service-Konsumenten zur Migration auf $I(RM_{NEXT})$ zu bewegen, bieten sich zudem *Blackout Tests* von Twitter an. Diese *Blackout-Tests* werden bei drohender Abschaltung eines Service über einen kurzen und zufälligen Zeitraum durchgeführt, bei der der abzulösende Microservice offline geschaltet wird: „These blackout tests last for a period of one hour and can occur at random during the days they are announced. They act as an indicator for unsuspecting users, that they should migrate“ [EZ+14, S. 88]. Die Benachrichtigung über anstehende *Blackout-Tests* wird wiederum mit der letzten Aktivität im Aktivitätsdiagramm gewährleistet, was wiederum eine Form von API-Management voraussetzt.

Empfehlung nach [EZ+14]	Aktivität	Erläuterung
Keine zu häufigen nicht abwärtskompatiblen Änderungen	Sammeln von nicht abwärtskompatiblen Änderungen	Die Aktivität stellt sicher, dass die nicht abwärtskompatiblen Änderungen gesammelt und erst durchgeführt werden, wenn eine entsprechende Priorisierung vorliegt. Die Priorisierung hängt immer vom jeweiligen Kontext ab und ist deshalb nicht verallgemeinerbar. So kann bspw. der resultierende Geschäftswert oder die Neuausrichtung der Domäne ein entscheidender Faktor sein.
Alte Versionen einer Web-API sollten nicht zu lange existieren	Planen der Abschaltung der älteren Hauptversion des Microservice $I(RM_{CUR})$	In der Aktivität wird festgelegt, wann $I(RM_{CUR})$ nicht mehr zur Verfügung steht bzw. abgeschaltet wird.
Erhebung von Nutzungsinformationen der Web-API	Benachrichtigen der Verantwortlichen für die Service-Konsumenten	In der Aktivität wird vorausgesetzt, dass ein API-Managementsystem [De17] existiert, sodass entsprechende Service-Konsumenten ermittelt und über etwaige Änderungen benachrichtigt werden können.
Einsatz von Blackout-Tests	Planen der Abschaltung der älteren Hauptversion des Microservice $I(RM_{CUR})$	In der Aktivität wird festgelegt in welchen Abständen ein Blackout-Test durchgeführt wird, um die Entwickler von Service-Konsumenten zum Upgrade auf die neue Hauptversion des Microservice zu bewegen.
Bereitstellung eines Beispiels zur Interaktion mit der Web-API	Wird durch die Überführung auf die Implementierungsebene erbracht (siehe Abschnitt 6.2)	
Statusinformation über die Stabilität einzelner Funktionalitäten	Markieren der veralteten Bestandteile im Ressourcenmodell $I(RM_{CUR})$	Wird durch die Verwendung von `Deprecated` im Ressourcenmodell gekennzeichnet. Die Kennzeichnung von *alpha* oder *beta* ist erst zum Zeitpunkt der Implementierung möglich und wird deshalb in dieser Arbeit nicht betrachtet.

Tabelle 5.17: Bewertung des Vorgehens bei Änderungen an der Web-API anhand [EZ+14]

5.6 Zusammenfassung

Dieser Abschnitt fasst die erzielten Beiträge dieses Kapitels noch einmal in übersichtlicher Form zusammen. Abbildung 5.20 veranschaulicht die Beiträge.

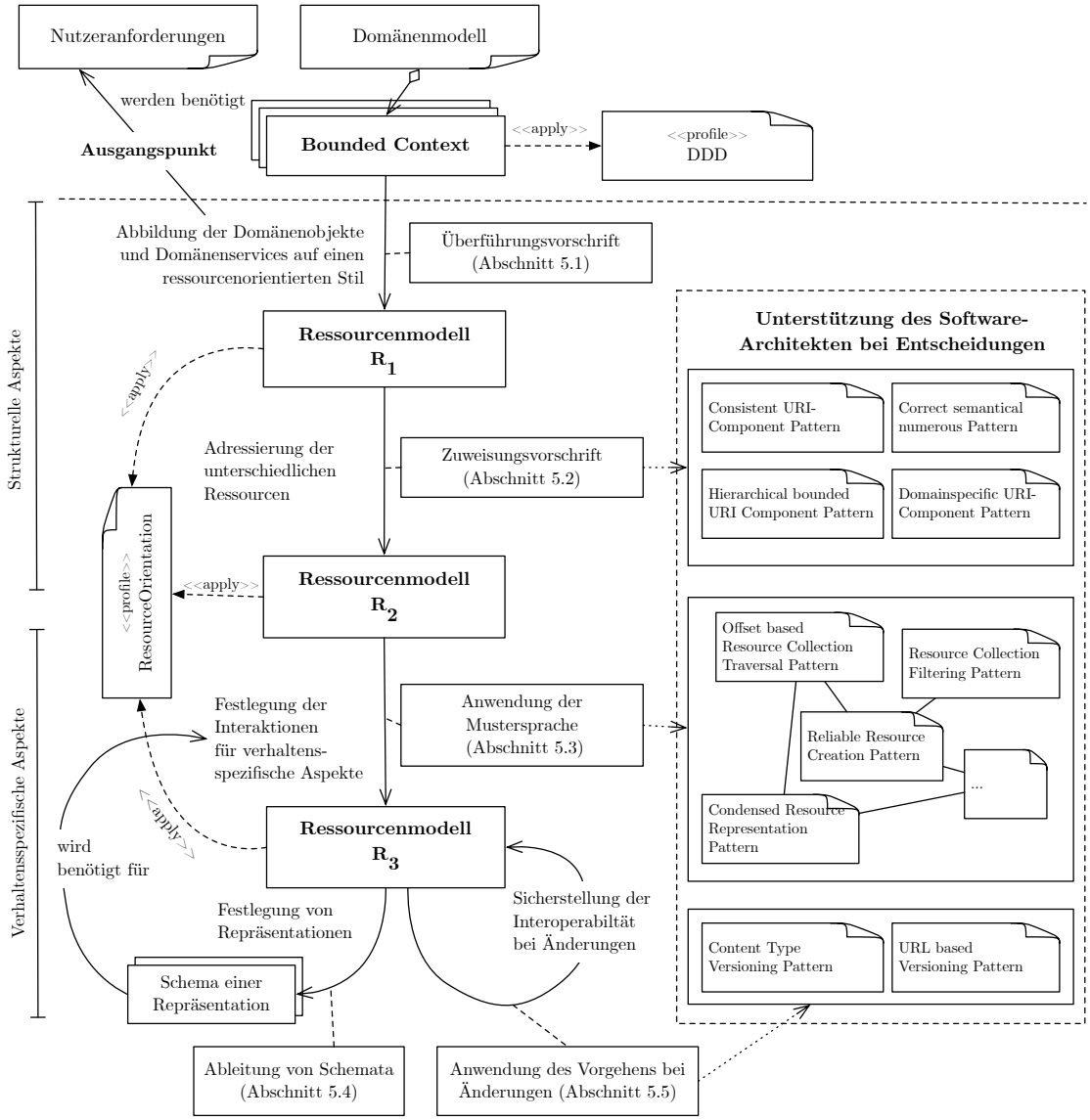


Abbildung 5.20: Übersicht über den Entwurfsprozess zur Überführung eines Bounded Context in ein Ressourcenmodell

Ausgangspunkt des präsentierten Entwurfsprozesses ist die Modellierung des Bounded Context aus Kapitel 4. Mittels einer Zuweisungsvorschrift wurde die objektorientierte Modellierung des Bounded Context unter Anwendung von Heuristiken und Regeln in ein Ressourcenmodell R_1 überführt. Dabei wurden die Domänenobjekte und Domänenservices des modellierten Bounded Context auf die

verschiedenen Ressourcenarten abgebildet, die in Tabelle 5.1 aufgeführt sind. Für die Modellierung des Ressourcenmodells wurde ein UML-Klassendiagramm verwendet, das durch ein UML-Profil für die Ressourcenorientierung erweitert wurde.

Danach wurde die Adressierung der einzelnen Ressourcen mittels entsprechender Muster abgeleitet und im Ressourcenmodell R_2 mithilfe von Eigenschaftswerten festgehalten. Zudem wurde aufgezeigt, wie sich die letztliche URL-Hierarchie und dabei die einzelnen Endpunkte des Microservice bzw. der Web-API systematisch ableiten lassen.

Nachdem bislang nur die strukturellen Aspekte betrachtet wurden, fokussierte der nächste Prozessschritt die Interaktionen mit Ressourcen und damit die verhaltensspezifischen Aspekte. Dabei wurde eine Mustersprache für Interaktionen präsentiert, die Software-Architekten bei der Bereitstellung der modellierten verhaltensspezifischen Aspekte des Bounded Context über eine ressourcenorientierte Web-API unterstützen soll. Die darin enthaltenen Muster beziehen sich auf Qualitätsteilmerkmale der Wiederverwendbarkeit und deren Einfluss auf die Teilnehmer einer Interaktion. Als letztere gelten der Microservice selbst und außerdem die Service-Konsumenten. Die Anwendung der Muster sowie deren Manifestierung im Ressourcenmodell führten zu R_3.

Eine Interaktion mit Ressourcen erfolgt über entsprechende Repräsentationen der Ressourcen, weshalb im nächsten Prozessschritt systematisch die Ableitung von Repräsentationen aus einem Ressourcenmodell aufgezeigt wurde. Durch die Ableitung wurde eine Sammlung von Schemata gewonnen, welche die Struktur der Repräsentationen auf Grundlage eines existierenden Medientyps der IANA beschreiben und gleichzeitig als Validierungsgrundlage für Nachrichteninhalte dienen.

Der letzte Beitrag untersuchte Änderungen des Ressourcenmodells R_3 im Zuge der geforderten Flexibilität heutiger Informationssysteme, die sich im Lauf der Zeit infolge verschiedener Einflussfaktoren ergeben können. Bei den zu betrachtenden möglichen Änderungen kann es sich um abwärtskompatible oder um nicht abwärtskompatible Änderungen handeln. Während abwärtskompatible Änderungen nur ein Inkrementieren der Versionsnummer der Web-API und damit des Microservice zur Folge haben, gefährden nicht abwärtskompatible Änderungen die Interoperabilität mit den Service-Konsumenten. Deswegen wurde ein Lösungsansatz vorgestellt, wie derartigen Änderungen zu begegnen ist. Der skizzierte Vorschlag basiert auf Empfehlungen einer dieses Problem untersuchenden Studie.

6 Überführung des Entwurfs auf die Implementierungsebene

Dieses Kapitel diskutiert, wie der domänengetriebene Entwurf eines ressourcenorientierten Microservice nun auf die Implementierungsebene übertragen wird. Laitkorpi et al. [LS+09] bezeichnen diesen Prozessschritt als *Service translation* und gleichzeitig als Voraussetzung für eine automatische Generierung von Quellcode. Deshalb erörtert dieses Kapitel auch grundsätzlich, wie sich die hexagonale Architektur auf Ebene der Implementierung manifestiert, wodurch letztlich die Struktur und der Aufbau eines Microservice aufgezeigt werden. Dies lässt sich grundsätzlich unabhängig von der letztlich verwendeten Technologie betrachten, da noch keine Entscheidung zur Implementierungstechnologie getroffen wurde. Das grundsätzliche Vorgehen, die Fragestellung sowie die zugrundeliegenden und benötigten Modellierungsartefakte veranschaulicht Abbildung 6.1. Die Gesamtheit der implementierten Microservices samt ihrer Vernetzung untereinander ergeben die Microservice-Architektur. Nicht diskutiert wird die Abbildung von anwendungsbezogenen Anforderungen im Kontext eines separaten Microservices, der als Fassade agiert (vgl. auch BFF-Muster aus Abschnitt 4.2).

Das Kapitel ist wie folgt strukturiert: Abschnitt 6.1 zeigt, wie das zugrundeliegende Ressourcenmodell R_3 mit einem zu wählenden Anwendungsschichtprotokoll verknüpft werden kann. Dies wird benötigt, um einen Vertrag zwischen Microservice und Service-Konsument zu definieren, den beide Interaktionsteilnehmer im Zuge der Interoperabilität bei der Implementierung einhalten müssen. Nach der Verknüpfung mit einem Anwendungsschichtprotokoll erfolgt die Ableitung der Web-API-Spezifikation in Abschnitt 6.2. Darauf aufbauend stellt Abschnitt 6.3 eine qualitätssichernde Maßnahme vor, bei der diese Web-API-Spezifikation in Bezug auf mehrere Kriterien vor der eigentlichen Implementierung bzw. Integration untersucht wird. Abschnitt 6.4 zeigt schließlich, wie die einzelnen strukturellen Modellelemente aus dem modellierten Bounded Context und dem Ressourcenmodell abgeleitet und Platzhalter für die Implementierung des Verhaltens definiert werden, die sich aus den verhaltensspezifischen Aspekten des Bounded Context ergeben. Dabei ist die explizite Trennung von Ressourcenmodell und modelliertem Bounded Context entscheidend, da sich letzterer unabhängig von der über die Web-API offerierte Funktionalität entwickeln können muss [Ev03]. Tilkov et al. bezeichnen dies als *Anti-Corruption-Layer* [TE+15]. Am Schluss steht eine ganzheitliche Übersicht der resultierenden Microservice-Architektur und die Verortung der nach diesem Ansatz entworfenen und entwickelten ressourcenorientierten Microservices im Sinne einer Makroarchitektur.

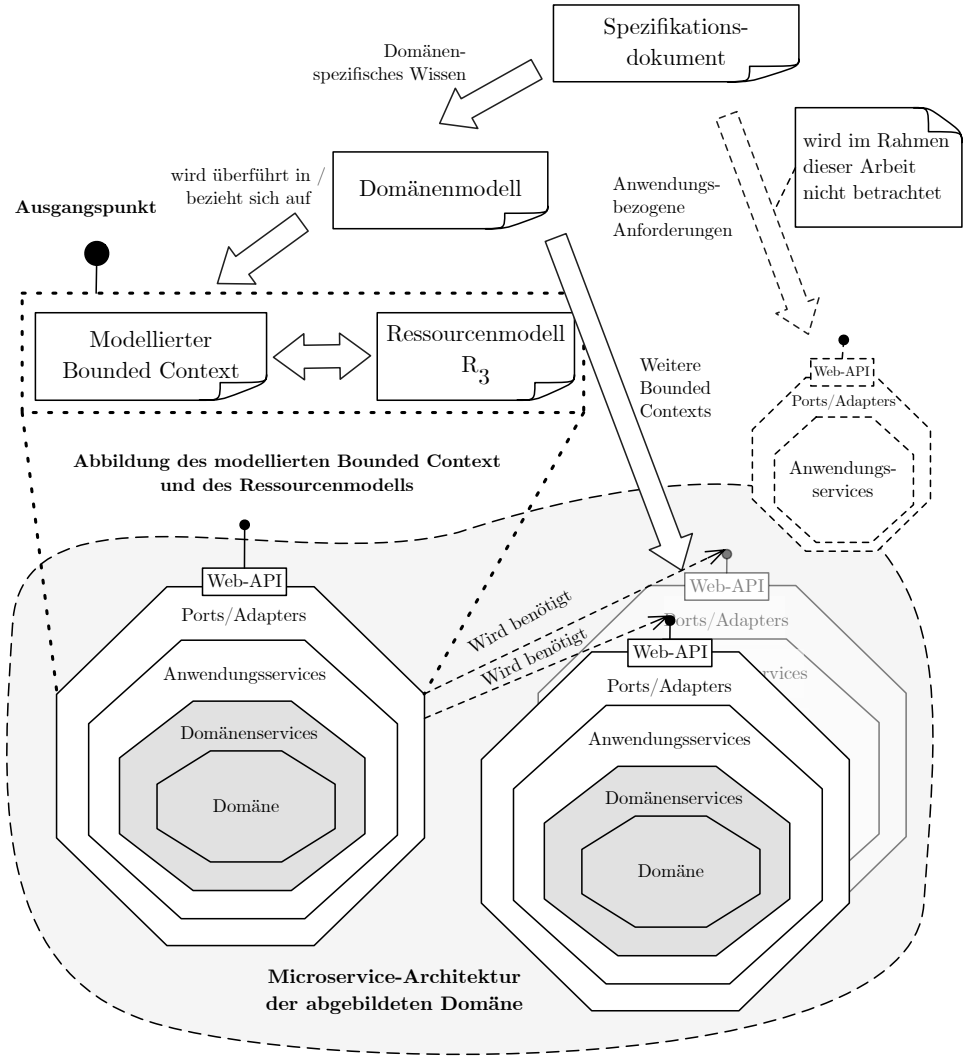


Abbildung 6.1: Übersicht über die Zielsetzung von Kapitel 6

6.1 Verknüpfung mit einem Anwendungsschichtprotokoll

Eine Interaktion mit einem ressourcenorientierten Microservice erfolgt in der Regel über ein Anwendungsschichtprotokoll, wie bspw. HTTP in der Version 1.1 [RFC7230-2014, RFC7231-2014, RFC7232-2014, RFC7233-2014, RFC7234-2014, RFC7235-2014] bzw. 2.0 [RFC7540-2015] oder Constrained Application Protocol (COAP) [RFC7252-2014] [RA+13]. Diese Anwendungsschichtprotokolle stellen Spezifikationen dar, wie u. a. eine Nachricht im Kontext einer Interaktion versendet werden kann und wie diese letztlich zu interpretieren ist. Die nachfolgende Betrachtung stützt sich wegen der höheren Interoperabilität auf HTTP als Anwendungsschichtprotokoll und nicht auf COAP. Doch die nun skizzierten Heuristiken lassen sich auch auf COAP oder vergleichbarer Protokolle anwenden, da sich HTTP und COAP hauptsächlich in den tieferen Schichten des Open Systems Inter-

connection Model (OSI)-Protokolls voneinander unterscheiden. Weiterführende Informationen zu den einzelnen Anwendungsschichtprotokollen sowie deren Unterschieden finden sich in deren offiziellen Standards. Der Nachrichteninhalt einer Interaktion wird bei ressourcenorientierten Microservices über Repräsentationen festgelegt, die ein Abbild der strukturellen Informationen auf ein Austauschformat wie bspw. JSON oder XML darstellen (vgl. Abschnitt 2.3.4). Die Wahl des Austauschformats sollte stets die zu unterstützenden Service-Konsumenten und etwaige nicht-funktionale Anforderungen berücksichtigen (vgl. Abschnitt 5.4).

Methode	Stereotyp	Semantik	Seiten-effektfrei	Idempotent
GET	`Read`	Abrufen aller Informationen einer Ressource in Form einer Repräsentation	Ja	Ja
PUT	`Update`	Bearbeiten einer bestehenden Ressource mittels einer entsprechenden Repräsentation. Falls keine Ressource mit der verwendeten URI existiert, wird eine neue Ressource angelegt	Nein	Ja
PATCH	`Update`	Partielles Bearbeiten einer Ressource. Die dafür benötigte Repräsentation beinhaltet demnach nur eine Teilmenge an Informationen der Ressource	Nein	Nein
DELETE	`Delete`	Löschen einer Ressource	Nein	Ja
POST	`Create`	Anlegen einer neuen Ressource. Gleichzeitig wird diese Methode immer angewandt, wenn keine zuvor genannten HTTP-Methoden aus semantischer Sicht zutreffend ist.	(Nein)	(Nein)
OPTIONS	-	Abrufen von Meta-Informationen einer Ressource im Hinblick auf deren offerierten Operationen	Ja	Ja
HEAD	-	Abrufen aller Meta-Informationen einer Ressource	Ja	Ja

Tabelle 6.1: Häufige HTTP-Methoden und deren Semantik

Die benötigten Interaktionen, die über die Web-API offeriert werden sollen, lassen sich aus dem zugrundeliegenden Ressourcenmodell R_3 ableiten. Dabei kann es sich um eine explizite oder um eine implizite Modellierung handeln (vgl. Abschnitt 5.1.3). Obgleich die Frage der Offerierung einer Ressource und deren Operationen bereits im letzten Prozessschritt geklärt wurde, empfiehlt es sich hier, noch einmal zu hinterfragen, ob die jeweilige Operation für eine Ressource tatsächlich über die Web-API veräußert werden sollte. Falls das nicht der Fall ist, muss die Ressource r aus dem Ressourcenmodell R_3 entfernt werden. Hinweise dafür liefern jeweils die Nutzeranforderungen oder

Rücksprachen mit Geschäftsanalysten.

Jeder Operation des Ressourcenmodells R_3 ist nach dem UML-Profil *ResourceOrientation* (vgl. Abbildung 5.18 auf Seite 129) mit einem der folgenden Stereotypen annotiert: 1) `Create`, 2) `Read`, 3) `Update`, 4) `Delete` und 5) `Execute`. Diese verschiedenen Operationen müssen nun auf die vorhandenen Methoden des entsprechenden Anwendungsschichtprotokolls überführt werden. Zur besseren Übersichtlichkeit nennt Tabelle 6.1 die häufigsten Methoden bei HTTP basierend auf [RFC7231-2014, RA+13, GG+16c]. Neben der Semantik der Methode finden sich darin grundlegende Eigenschaften der HTTP-Methoden wieder (vgl. [RFC7231-2014, S. 21f]): 1) die Idempotenz und 2) die Seiteneffektfreiheit. Eine Methode wird genau dann als idempotent bezeichnet, wenn das hintereinander ausführen dieser Methode immer das gleiche Ergebnis liefert. Eine Methode gilt als seiteneffektfrei, wenn sie keine Zustandsänderung hervorruft: „the client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource“ [RFC7231-2014, S. 22]. Die Eigenschaft "zwischenspeicherbar" (engl. cachable) wird hier nirgendwo berücksichtigt. Laut Tabelle 6.1 ist die HTTP-Methode POST im Hinblick auf die zuvor erwähnten Eigenschaften besonders. Das liegt an der Semantik, da diese HTTP-Methode auch dann eingesetzt wird, wenn keine andere HTTP-Methode zutreffend ist. Die letztliche Ausprägung der HTTP-Methode entscheidet daher der jeweilige Software-Architekt sowie der Kontext in dem sie eingesetzt wird. Die Verwendung der jeweiligen HTTP-Methode ist den Mustern in Abschnitt 5.3.2 zu entnehmen, da diese einen Bezug zu den entsprechenden HTTP-Methoden und deren Semantik über das enthaltene Beispiel ermöglichen. Gleichzeitig wurden, wie in [GG+16c] dargestellt, die HTTP-Methoden und die jeweiligen Stereotypen des UML-Profils *ResourceOrientation* zueinander in Beziehung gesetzt. Eine Besonderheit sind die HTTP-Methoden, die den Bezug zu Operationen von Domänenobjekten und Domänenservices nicht ermöglichen. Stattdessen adressieren sie Querschnittsthemen und werden deshalb mit einem "-" in der Tabelle 6.1 gekennzeichnet.

6.2 Ableitung einer Web-API-Spezifikation

Nachdem der letzte Abschnitt gezeigt hat, wie das Ressourcenmodell R_3 durch Verknüpfung mit einem Anwendungsschichtprotokoll und der damit einhergehenden technologischen Entscheidung in eine Form des PSM nach MDSD überführt werden kann, widmet sich dieser Abschnitt der Ableitung einer entsprechenden Web-API-Spezifikation. Dies ist vergleichbar mit dem Ansatz von Rossi [Ro16], der ebenfalls eine Transformation in eine Web-API-Spezifikationssprache vorsieht. Die Web-API-Spezifikation dient als Vertrag zwischen Service-Konsumenten und letztlichem Microservice, kann maschinell interpretiert werden und verfügt zumeist über eine breite Unterstützung in heutigen Entwicklungswerkzeugen (vgl. Abschnitt 2.3.5).

In dieser Arbeit wurde die Web-API-Spezifikationssprache OpenAPI (OAI) gewählt. Diese Entscheidung ist für die grundlegenden Konzepte dieser Arbeit ohne Folgen, da existierende Web-

API-Spezifikationssprachen mit vertretbarem Aufwand ineinander überführt werden können. Die Ableitung einer Web-API-Spezifikation gliedert sich in die folgenden drei Schritte, die mittels der OAI-Spezifikationssprache in der Version 3.0.0 illustriert werden. Ein beispielhafter Aufbau einer OAI-Spezifikation ist Anhang A.11 zu entnehmen.

6.2.1 Überführung von Meta-Informationen

Die Meta-Informationen beschreiben die veräußerte Web-API und damit gleichermaßen den abzubildenden Bounded Context, da die Web-API aus diesem abgeleitet wurde. Zu den benötigten Meta-Informationen zählen eine Beschreibung und eine Bezeichnung des Bounded Context, die Versionskennung der Web-API (vgl. Abschnitt 5.5) sowie Kontaktinformationen. Letztere sollen es den Nutzern der Web-API ermöglichen, mit dem Entwicklungsteam des Microservice oder der Web-API bei etwaigen Rückfragen direkt in Kontakt zu treten. Alternativ kann auch ein Support-Team diese Aufgabe übernehmen. Weitere Informationen, wie bspw. lizenzrechtliche Bedingungen zur Wiederverwendung der Web-API oder auch die allgemeinen Nutzungsbedingungen (engl. terms of use), können optional gesetzt werden. Der Quelltext 6.1 ist der Auszug einer OAI-Spezifikation für das zu betrachtende Anwendungsszenario aus Abschnitt 4.3.

```
1 openapi: "3.0.0"
2 info:
3   description: "Web API for position determination in a building."
4   version: "0.1.0"
5   title: "Web API for indoor position determination"
6   contact:
7     name: "..."
8     email: "..."
```

Quelltext 6.1: Auszug der OAI-Spezifikation mit Fokus auf den Meta-Informationen der OwB

6.2.2 Überführung von wiederverwendbaren Komponenten

Nach Überführung der Meta-Informationen folgen nun die wiederverwendbaren OAI-Komponenten, die sich nach [OAI-SPEC] (vgl. Anhang A.11) u. a. in (1) Schemata, (2) Antworten und (3) Anfrageparameter unterteilen lassen. Weitere Untergliederungen werden hier nicht behandelt, da sie nicht benötigt werden. Durch das Attribut `$ref` können OAI-Komponenten an der entsprechenden Stelle der OAI-Spezifikation referenziert werden.

(1) **Schemata:**
Die Schemata entsprechen den bisher festgelegten Schemata der Repräsentation aus Abschnitt 5.4 und können abhängig vom entsprechenden Format identisch hier eingefügt werden.

Sie lassen sich um zusätzliche Beispiele erweitern. Dadurch muss sich ein Software-Entwickler keine Gedanken über die Ausprägung etwaiger Attribute für die Interaktion machen, was das Verstehen vereinfacht und die Benutzbarkeit fördert. Werkzeuge wie bspw. *Swagger UI*[1] werten diese Informationen aus und stellen eine entsprechende Oberfläche zur Interaktion mit einer Web-API bereit.

```
schemas:
    Error:
      properties:
        detail:
          description: "Specific description of the error with
              reference to the actual occurrence"
          title: "Specific description of the error"
          type: string
        instance:
          description: "Reference to specific information on the
              error that has occurred"
          title: "Further information"
          type: string
        status:
          description: "HTTP status code of the error that
              occurred"
          maximum: 599
          minimum: 100
          title: "HTTP status code"
          type: integer
        title:
          description: "Short description of the error without
              reference to the actual occurrence"
          title: "Short description of the error"
          type: string
        type:
          description: "Reference to further information on the
              error type"
          title: "Reference to the error type"
          type: string
      required:
      - title
      - status
      type: object
```

Quelltext 6.2: Auszug der OAI-Spezifikation mit Fokus auf wiederverwendbare Schemata der OwB

[1] https://swagger.io/swagger-ui/ (Letzter Zugriff: 03.11.2017)

(2) **Antworten:**
Antworten, die sich nicht einer spezifischen Operation zuordnen lassen, werden nun als *wiederverwendbare Antwort* definiert. Entsprechende Kandidaten stellen dabei Fehlermeldungen dar, die unabhängig von einer konkreten Operation auftreten oder in irgendeiner Form gruppiert werden können. Dazu zählt bspw. eine Fehlermeldung, die eine nicht auffindbare Ressource dem Service-Konsumenten zurückmeldet. Für die Struktur einer Fehlermeldung wird schließlich auf das entsprechende Schema verwiesen (vgl. Zeile 7 in Quelltext 6.3 sowie Abschnitt 5.4.3).

```
1  responses:
2     NotFound:
3       description: "The requested resource could not be found"
4       content:
5         vnd.smartcampus.hal+json:
6           schema:
7             $ref: "#/components/schemas/Error"
```

Quelltext 6.3: Auszug der OAI-Spezifikation mit Fokus auf wiederverwendbare Antworten der OwB

(3) **Anfrageparameter:**
Neben den zu übermittelnden Antworten (2) mittels einer Repräsentation im Nachrichtenrumpf gibt es auch sogenannte Anfrageparameter, die sich ebenfalls als kontextunabhängig bezeichnen lassen und bei mehreren Ressourcen anwendbar sind. Diese Anfrageparameter können sich entweder in der URI (als Teil des Pfades oder als Query String) im HTTP-Header oder im Cookie manifestieren und beeinflussen zu einem gewissen Grad die Anfrage. Die Art der Manifestierung wird über das Attribut `in` definiert. Geeignete Kandidaten für wiederverwendbare Parameter ergeben sich bspw. aus den Mustern in Abschnitt 5.3.2 als auch dem Platzhalter, der bei der Adressierung einer Ressource in Abschnitt 5.2 definiert wurde (vgl. Zeilen 2-8 in Quelltext 6.4).

```
1  parameters:
2     uuidParam:
3       name: "uuid"
4       in: path
5       description: "UUID of the resource"
6       required: true
7       schema:
8         type: string
9     offsetParam:
10      name: "offset"
11      in: query
12      description: "Number of list resource entries to be skipped"
13      required: false
```

```
14        schema:
15          type: integer
16          format: int32
17      limitParam:
18        name: "limit"
19        in: query
20        description: "Maximum number of returned list resource
              entries"
21        required: false
22        schema:
23          type: integer
24          format: int32
```

Quelltext 6.4: Auszug der OAI-Spezifikation mit Fokus auf wiederverwendbare Parameter der OwB

6.2.3 Überführung von Interaktionen

Nachdem die vorigen Abschnitte gezeigt haben, wie Meta-Informationen und die wiederverwendbaren OAI-Komponenten einer OAI-Spezifikation überführt werden, widmet sich dieser der Festlegung der benötigten und zu offerierenden Operationen eines Microservice, mit denen Service-Konsumenten schließlich interagieren. Für die Überführung sind nun mehrere Schritte notwendig, die für jede einzelne Ressource r des Ressourcenmodells R_3 durchzuführen sind. Der Quelltext 6.5 illustriert hier die einzelnen Schritte.

```
1  paths:
2    /point-of-interests:
3      get:
4        summary: Get all points of interest
5        parameters:
6         - $ref: "#/components/parameters/limitParam"
7         - $ref: "#/components/parameters/offsetParam"
8        responses:
9          '200':
10           description: A list of points of interest
11           content:
12             application/vnd.smartcampus.hal+json:
13               schema:
14                 type: array
15                 items:
16                   $ref: "#/components/schemas/Position"
17         '400':
18           description: There was an error
19           content:
```

```
20              application/vnd.smartcampus.hal+json:
21                schema:
22                  $ref: "#/components/schemas/Error"
23          # Futher responses
24      post:
25        summary: Add a new point of interest
26        # Futher instructions
```

Quelltext 6.5: Auszug der OAI-Spezifikation mit Fokus auf den Interaktionen der OwB

1. **Überführung der Pfade:** Die Adressierung der Ressource *r* erfolgt über eine Verkettung von URI-Komponenten (nachfolgend als *Pfad* bezeichnet) sowie eine vorangestellte Basis-URI, die durch das Attribut `servers` (vgl. Anhang A.11) festgelegt werden kann. Der letztendliche Pfad einer Ressource *r* ist dem Abschnitt 5.2 zu entnehmen und manifestiert sich in der OAI-Spezifikation als eigenständiges Attribut innerhalb des Attributes `paths`.

2. **Zuordnung der HTTP-Methoden:** Auf Überführung des Pfades für die Ressource *r* folgt die der bereitzustellenden Operationen (`Read, Update, Delete, Create` und `Execute`) nach dem in Abschnitt 6.1 skizzierten Vorgehen, das sie mit dem Anwendungsschichtprotokoll HTTP verknüpft.

3. **Überführung der Anfrageparameter:** Erst wird ermittelt, ob der Pfad der Ressource r einen Platzhalter beinhaltet. Falls ja, muss ein entsprechender Anfrageparameter definiert oder referenziert werden, wobei `in:path` gelten muss. Anschließend ist zu prüfen, ob der Ressource *r* ein Muster zugewiesen ist und etwaige Anfrageparameter erforderlich sind. Dies ist bspw. bei P_{ORCT} gegeben, da dort `limit` und `offset` die Paginierung steuern und sich diese innerhalb des Query Strings manifestieren.

4. **Überführung des Anfrageinhalts:** Außer Anfrageparametern kann auch noch ein Anfrageinhalt existieren, den der Service-Konsument an die Ressource *r* übermittelt. Dies ist bspw. bei den Operationen `Create` und `Update` der Fall (vgl. die Interaktionen der verschiedenen Ressourcenarten in Abschnitt 5.3.2). Der Anfrageinhalt stellt zumeist eine Repräsentation der Ressource r dar und sollte bereits als wiederverwendbares Schema vorliegen. Deshalb lässt sich einfach mit `$ref` darauf verweisen.

5. **Festlegung der Antworten:** Abschließend sind noch die bei einer Interaktion mit der Ressource *r* möglichen Antworten sowie der semantisch korrekte HTTP-Statuscode zu definieren. Neben einer positiven Antwort müssen dabei auch alle möglichen Fehlermeldungen festgelegt werden, die sich aus den Invarianten sowie Vor- und Nachbedingungen des zugrundeliegenden Domänenobjekts ableiten lassen (vgl. Abschnitt 4.2.1 und 4.2.2). Die Fehlermeldung sollte sich dabei an dem Aufbau in Abschnitt 5.4.3 orientieren.

6.2.4 Zusammenfassung und Abbildung von weiterem Wissen

Die OAI-Spezifikation bietet die Möglichkeit, noch weiteres Wissen abzubilden, das u. a. auch betriebliche Aspekte inkludiert. Dazu zählt bspw. die Festlegung der geplanten Instanzen des Microservice bzw. der Basis-URIs, unter der die einzelnen Instanzen zu erreichen sind. Derartige Informationen werden nun nicht näher betrachtet, da sie sich nicht direkt aus dem Ressourcenmodell ergeben, zum Entwurfszeitpunkt noch nicht vorliegen oder für das weitere Verständnis dieser Arbeit nicht erforderlich sind. Hier muss der Verweis auf die OAI-Spezifikationssprache und deren möglichen Attribute genügen [OAI-SPEC].

Tabelle 6.2 zeigt, aus welchen Informationen des Ressourcenmodells oder des modellierten Bounded Context sich die einzelnen Informationsbestandteile der letztlichen OAI-Spezifikation ableiten lassen. Falls sich ein Informationsbestandteil ohne zusätzliches Wissen aus einem Modell ableiten lässt, so wird dies mit ● gekennzeichnet. Im Gegensatz dazu impliziert ○, dass der Informationsbestandteil nicht auf Grundlage eines Modells abgeleitet werden kann. Falls das zwar möglich ist, aber nicht ohne zusätzliches Wissen, wird ◐ verwendet. So kann bspw. die Beschreibung der Web-API aus dem zugrundeliegenden Bouded Context abgeleitet werden. Allerdings erfordert dies zusätzliches Wissen, da die Beschreibung nicht als eigenständiges Modellelement vorliegt. Die Antworten lassen sich grundsätzlich aus der Betrachtung der jeweiligen Interaktionen bzw. den zugrundeliegenden Absichten ableiten (vgl. Mustersprache in Abschnitt 5.3.2). Bei einer Antwort gilt es grundsätzlich zwischen erfolgreich und nicht erfolgreich zu unterscheiden. Die erfolgreichen Antworten lassen sich in der Regel aus dem Ressourcenmodell R_3 unter Anwendung der zuvor erwähnten Mustersprache ableiten. Die möglichen nicht erfolgreichen Antworten ergeben sich u. a. durch die Betrachtung der verhaltensspezifischen Aspekte mit etwaigen Vor- und Nachbedingungen sowie Invarianten. Allerdings existieren auch andere Fehler, die nicht explizit modelliert wurden, wie bspw. das Nicht-Auffinden einer bestimmten Ressource.

	Informationsbestandteile einer OAI-Spezifikation	Ressourcenmodell	Modellierter Bounded Context
Meta-Informationen	Titel	○	◐
	Beschreibung	○	◐
	Kontakt	○	○
	Versionskennung	●	○
Komponenten	Schemata	◐	○
	Antworten	◐	○
	Anfrageparameter	◐	○
Interaktionen	Pfade	●	○
	HTTP-Methoden	●	○
	Anfrageparameter	●	○

	Informationsbestandteile einer OAI-Spezifikation	Ressourcenmodell	Modellierter Bounded Context
	Anfrageinhalt	●	○
	Antworten	◐	◐

Tabelle 6.2: Zusammenfassung zur Überführung des Ressourcenmodells in eine OAI-Spezifikation

6.3 Qualitätssichernde Maßnahmen der Web-API

Der letzte Abschnitt hat eine Ableitung der Web-API-Spezifikation und der damit einhergehenden Verknüpfung mit einem Anwendungsschichtprotokoll vorgestellt. Nun ist die Qualität der Web-API-Spezifikation vor der eigentlichen Überführung auf die Implementierung sicherzustellen. Anders als bei den bisherigen Prozessschritten wird hier kein neues Entwurfsartefakt erstellt oder ein bestehendes erweitert, sondern die Web-API-Spezifikation – und damit eigentlich auch das Ressourcenmodell R_3 – in Bezug auf die definierten Qualitätsteilmerkmale der Wiederverwendbarkeit (vgl. Abschnitt 2.3.2)) bewertet. Übertragen auf die Domäne des Qualitätsmanagements für die Produktentwicklung ist dies einem *Quality Gate* gleichzusetzen, das der „Prävention, d. h. dem frühzeitigen Erkennen und Abstellen von Fehlern und Fehlermöglichkeiten mit entsprechend positivem Effekt auf die Produktreife“ [PS14, S. 415] dient. Ansätze für die Bewertung finden sich auch in [GG+16b].

Einführung von Reviews der Web-API

Um die Qualität der Web-API im Hinblick auf ihre Wiederverwendbarkeit sicherzustellen, sollte vor ihrer Veröffentlichung ein unabhängiges Team im Unternehmen die abgeleitete Web-API -Spezifikation (wie in Abschnitt 6.2 dargestellt) einem Review unterziehen. Als Review definiert die ISO/IEC/IEEE 24765 [ISO-24675]] eine „formal, documented, comprehensive, and systematic examination of a design to determine if the design meets the applicable requirements, to identify problems, and to propose solutions“ [ISO-24675, S. 102]. Zwar kann bereits für die in Kapitel 5 diskutierten Prozesse ein Review durchgeführt werden. Doch erst mit der Ableitung der Web-API-Spezifikation wird das Ressourcenmodell R_3 mit dem zu verwendenden Anwendungsschichtprotokoll verknüpft. Das ist zusammen mit dem modellierten Bounded Context die Grundlage für die Überführung auf die Implementierungsebene.

Üblicherweise führt ein Gutachterteam mit Wissen über den Entwurf von Web-APIs das Review durch. Obgleich der hier präsentierte Entwurfsansatz eine systematische und nachvollziehbare Ableitung unter Berücksichtigung von Qualitätsteilmerkmalen für die Wiederverwendbarkeit ermöglicht, können Fehler nicht ausgeschlossen werden. Aber, wenn sie sich bis zu dieser Phase identifizieren lassen, ist es möglich, sie noch vor der eigentlichen Implementierung zu korrigieren, sodass teure Fehlerkorrekturen vermieden werden. Nachträgliche Änderungen an einer Web-API können anders als bei einer lokalen

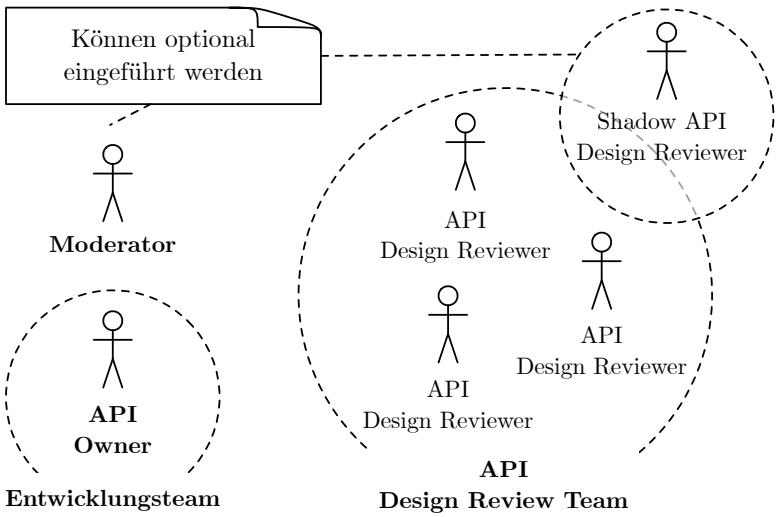


Abbildung 6.2: Beteiligte Akteure beim Review einer Web-API bzw. -Spezifikation

API kaskadierende Änderungen bei Interaktionsteilnehmern auslösen (vgl. auch Abschnitt 5.5): „things rapidly change if I put that software out on the Web as a component, and other people, whom I don't know, start building applications on top of it. If I now delete the parameter, everybody else's code will break when I upgrade" [Fo02b, S. 18]. Deshalb sollten Fehler am Entwurf vor der eigentlichen Veröffentlichung weitestgehend eliminiert sein.

Auch Unternehmen wie bspw. Google oder Zalando haben das erkannt und Review-Prozesse in den Entwurf von Services integriert. So führte Google das Verfahren *Apiness (API happiness)* [MM+16] u. a. zur Verbesserung der Benutzbarkeit einer Web-API ein, während Zalando ein sogenanntes *API Guild* [FS15] für Entwicklungsteams einführte, um Erfahrungen zur Qualität einer Web-API zu teilen und kritisch zu diskutieren, wie deren Qualität zukünftig sichergestellt werden kann. Auch andere Unternehmen, darunter ein großer Automobilhersteller, wollen derartige Programme aufsetzen und den Review einer Web-API in den Entwicklungsprozess von Microservices integrieren. Wie sehr das erforderlich ist, mag ein Beispiel von Google illustrieren: „For example, three APIs may expose the concept of a region, but do it in completely different ways (full URL vs. a short name vs. a region code). As a user, even if you have a quick way to map the representation between APIs, do the concepts really mean the same thing in the context of each API?" [MM+16, S. 851].

Im *Apiness*-Programm von Google werden u. a. die verwendeten Konventionen bei der Benennung von Ressourcen, die Strukturierung der Fehlermeldungen, die korrekte Verwendung von Standardmethoden sowie der Einsatz bestehender und etablierter Muster als auch bewährter Methoden als Untersuchungsschwerpunkt im Review untersucht [MM+16]. Zudem wurden verschiedene Rollen für unterschiedliche Tätigkeiten und Verantwortlichkeiten im Rahmen eines Reviews definiert. Für den zu etablierenden Review-Prozess wurde sich an diese angelehnt und die folgenden Rollen abgeleitet (vgl.

auch Abbildung 6.2). Die Begrifflichkeit der API lässt sich hier synonym mit Web-API verwenden. Auf die Einführung von einem *Shadow (API) Design Reviewer* und *Moderator* wird hier verzichtet, da diese Rollen die Weiterbildung von Mitarbeitern und die Qualitätssicherung des Review-Prozesses fokussieren. Sollten diese Rollen vonnöten sein, so können diese einfach in den nachfolgenden Review-Prozess integriert werden.

1. **API Owner:** Software-Architekt im Entwicklungsteam, der für den Entwurf und die Qualität einer Web-API verantwortlich ist.

2. **API Design Review Team:** Expertengruppe für den Entwurf von Web-APIs, die, wenn nötig, konsultiert werden kann und Wissen und Erfahrungen verfügbar macht.

3. **API Design Reviewer:** Mitglied im API Design Review Team, der Reviews von Web-API durchführt.

Den Review-Prozess veranschaulicht Abbildung 6.3, der den Entwurfsprozess in Abbildung 4.3 auf Seite 71 ergänzt. Den einzelnen Aktivitäten wurden entsprechende Akteure mit zuvor eingeführten Rollen zugewiesen, welche die Aktivität ausführen oder die Aktivität unterstützen (vgl. Tabelle 6.3). So ist für die Verknüpfung mit dem Anwendungsschichtprotokoll vor allem der *API Owner* verantwortlich. Sobald die Web-API-Spezifikation im OAI-Format aus dem Ressourcenmodell sowie dem verknüpften Anwendungsschichtprotokoll abgeleitet wurde, kann der *API Owner* das Review einleiten, indem er die Web-API-Spezifikation bzw. die OAI-Spezifikation dem *API Design Reviewer* übermittelt, der dann prüft, ob die in Kapitel 5 beschriebenen Muster und Konventionen eingehalten werden und so die zu berücksichtigenden Qualitätsteilmerkmale beeinflussen. Da die OAI-Spezifikation ein textbasiertes Format ist, kann der *API Design Reviewer* Kommentare und Vorschläge direkt dort vermerken, um mit dem *API Owner* zu kommunizieren. So wird Transparenz erhöht, und Entscheidungen sind für alle sichtbar und nachvollziehbar.

Aktivität	Rollenzuordnung
Verknüpfen mit einem Anwendungsschichtprotokoll	API Owner
	API Design Review Team
Reviewen der Web-API	API Owner
	API Design Reviewer
	(Moderator)
	(API Design Shadow Recruiter)

Tabelle 6.3: Rollenzuordnung zu den einzelnen Aktivitäten des Web-API-Review-Prozesses

Falls die Web-API bspw. als Folge der inkorrekten Verwendung von Standardmethoden verändert werden muss, ist das vom *API Owner* durchzuführen. Danach beginnt dieser iterative Prozess wieder

von vorn, bis das Review keinen weiteren Handlungsbedarf identifiziert. Ein Review ist aber keineswegs verpflichtend. Denn ein Projekt kann etwa einem engen Zeitplan unterliegen. So gilt bei Google: „it is critical that an API design review process is not seen as a blocker“ [MM+16, S. 855].

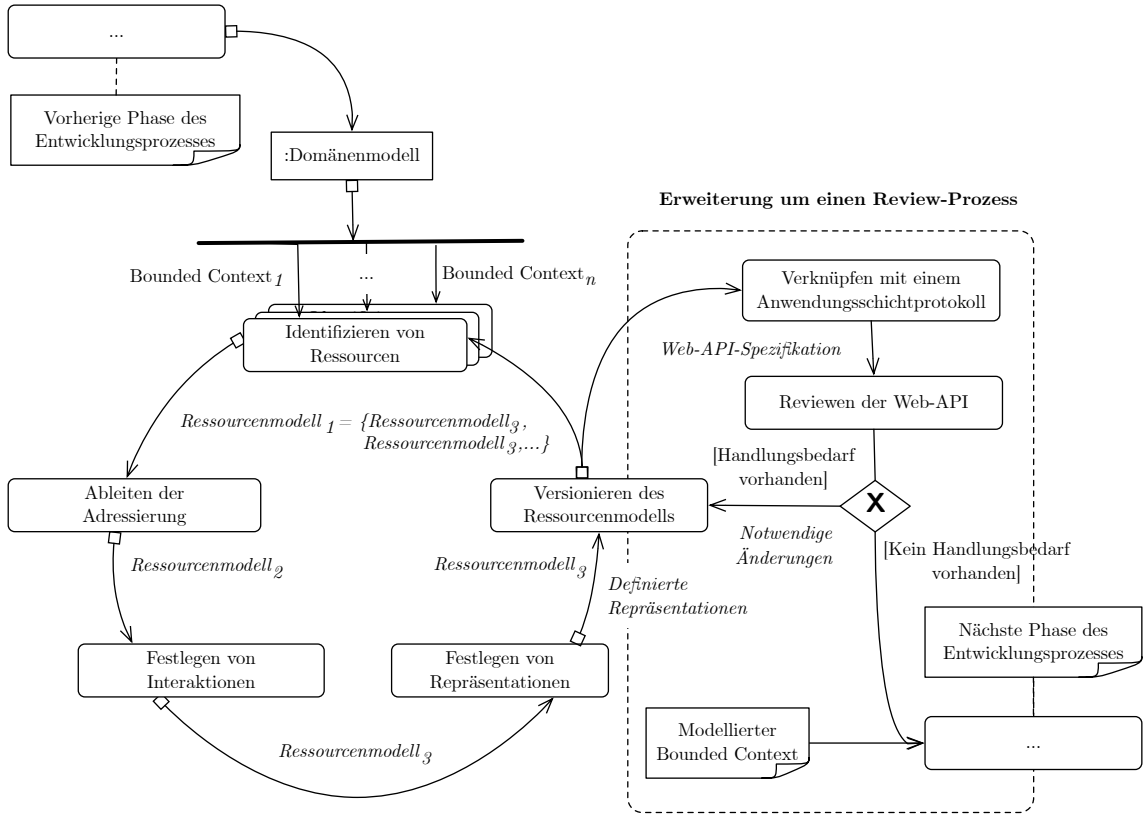


Abbildung 6.3: Review-Prozess einer Web-API als Erweiterung des Entwurfsprozesses in Abbildung 4.3

6.4 Überführung auf die Implementierungsebene

Dieser Abschnitt beschreibt die Überführung der Entwurfsartefakte – also von dem modelliertem Bounded Context und dem abgeleiteten Ressourcenmodell R_3 – auf die Implementierungsebene. Dazu wird zunächst geklärt, wie sich die hexagonale Architektur und die darin enthaltenen Schichten auf der Implementierungsebene manifestieren. Im Fokus sind nun die Ports-/Adapter- sowie die Domänenschicht, da sich etwaige anwendungsbezogenen Anforderungen nicht in dem zu implementierenden Microservice manifestieren, sondern in einem separaten Microservice zur Ausrichtung auf etwaige Service-Konsumenten verankert sind (vgl. Abbildung 6.1). Die Anwendungsservice-Schicht findet sich zwar weiterhin in der letztlichen Architektur wieder, fokussiert allerdings die Verknüpfung der Ports- und Adapterschicht mit der Domänen(service)-Schicht. Nach der grundsätzlichen Strukturierung des Microservice auf Implementierungsebene erfolgt auf Grundlage dessen die schrittweise

Überführung der einzelnen Schichten der hexagonalen Architektur.

6.4.1 Grundsätzliche Strukturierung

Die Strukturierung eines Microservice ist durch die Hexagonal-Architektur bereits vorgegeben, da sich eine Architektur auch in der Implementierung widerspiegeln und so den Software-Entwickler anleiten sollte. Die Hexagonal-Architektur sieht eine Unterteilung in die Domäne, die Domänenservices, die Anwendungsservices sowie Ports/Adapter vor, wobei die Domäne in den Mittelpunkt gerückt wird (vgl. Abschnitt 4.4). Nun werden die einzelnen Unterteilungen als Schichten bezeichnet, da die Hexagonal- Architektur als Erweiterung des architekturellen Musters *Layered Architecture* angesehen werden kann (vgl. [Ev03, Ve13]).

- adapters → **Paket für benötigte und bereitzustellende Adapter**
  - presentation → **Paket von Adaptern, die Informationen des Microservice bereitstellen**
  - infrastructure → **Paket von Adaptern, die von dem Microservice benötigt werden**
- application → **Paket für anwendungsspezifische Logik**
  - services → **Paket für Anwendungsservices**
  - utilities → **Paket für Hilfsfunktionalitäten der Anwendungsservices**
- domain → **Paket für domänenspezifische Logik**
  - entities → **Paket für Domänenobjekte**
  - services → **Paket für Domänenservices**
  - utilities → **Paket für Hilfsfunktionalitäten der Domänenobjekte/-services**
- aspects → **Paket für querschnittliche Belange, wie bspw. Logging**

Abbildung 6.4: Grundlegende Strukturierung eines Microservice nach DDD

Die einzelnen Schichten werden in der Implementierung üblicherweise durch Pakete voneinander abgegrenzt. Die Definition eines Pakets kann aus der offiziellen UML-Spezifikation abgeleitet werden: „A package is a namespace for its members, which comprise those elements“ [OMG-UML2c, S. 239]. Es bündelt demnach gleichartige Elemente und fasst diese unter einem eindeutigen Namensraum zusammen. Dies gilt auch für das Konzept der Schichten, da eine Schicht nach dem Prinzip des Separation of Concern (SoC) ein bestimmtes Anliegen (engl. concern) verfolgt und auf ihr nur gleichartige Elemente zu finden sein sollten, die zu Erfüllung des Anliegens erforderlich sind [Ri15b]. Im Hinblick auf die Implementierung manifestieren sich Pakete bei den Programmiersprachen in

unterschiedlichen Ausprägungen. So wird bei Java[2] oder auch Go[3] das gleichnamige Sprachkonstrukt *package* verwendet, während bspw. bei ECMAScript 2017[4] (Node) jede Anwendung als Paket aufgefasst wird und die Gruppierung gleichartiger Elemente durch die reine Ordnerhierarchie oder Module vollzogen wird.

Nun ergibt sich für einen Microservice nach DDD eine aus verschiedenen Paketen bestehende Basis-Struktur. Der Namensraum der einzelnen Pakete wird dabei zum einen durch einen eindeutigen Basis-Identifikator und zum anderen durch einen Paket-Identifikator festgelegt. Der Basis-Identifikator (z.B. *org.kit.tm.cm.smartcampus.navsg*) soll den eigentlichen Microservice adressieren und der Paket-Identifikator (z.B. *domain*) die einzelnen Schichten. Beide sollten zusammen eine logische Hierarchie abbilden. So kann bspw. der Basis-Identifikator die Zugehörigkeit zu einer Organisation bzw. Abteilung einer Organisation anzeigen, woraus sich wiederum verwandte Softwaresysteme ableiten lassen. Während das Entwicklungsteam und die Organisation den Basis-Identifikator selbstständig festlegen, gelten für die Paket-Identifikatoren bestimmte Vorgaben.

Dadurch soll sichergestellt werden, dass sich alle Schichten der Hexagonal-Architektur auch in der Implementierung wiederfinden und gefunden werden können. Allerdings muss das Entwicklungsteam entscheiden, ob die Pakete weiter hierarchisch unterteilt sind. Die grundlegende Struktur des Microservice nach DDD ist Abbildung 6.4 zu entnehmen. Besonderheiten sind jeweils in einem separaten Paket verortete querschnittliche Belange (*aspects*) sowie etwaige Hilfsfunktionalitäten (*utilities*).

6.4.2 Überführung der einzelnen Schichten

Der vorige Abschnitt hat die grundsätzliche Strukturierung eines Microservice nach der hexagonalen Architektur beleuchtet. Nun geht es um Überführung der Entwurfsartefakte auf die einzelnen Schichten. Dabei erfolgt die Beschreibung technologie-agnostisch, sodass sie unabhängig von der Technologie angewandt werden kann. Bei der Überführung werden die Ports-/Adapter-Schicht und die Domänenschicht fokussiert, um zu zeigen, wie sich diese möglichst lose gekoppelt entwickeln lassen. Tilkov et al. bezeichnen dies als *Anti-Corruption-Layer* [TE+15].

Zur Demonstration der Überführung wurde wegen ihrer weiten Verbreitung (so der TIOBE Index[5]) die Programmiersprache Java in der Version 8 entschieden. Mit geringem Aufwand lassen sich diese Beispiele auch auf andere Programmier- oder Skriptsprachen wie bspw. JavaScript übertragen. Auf die Verwendung eines Rahmenwerks (engl. framework), wie bspw. Spring[6], wurde explizit verzichtet, da dies zwar die Entwicklung vereinfachen würde, gleichzeitig aber auch eine weitere

[2] https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf (Letzter Zugriff: 09.12.2017)
[3] https://golang.org/ref/spec (Letzter Zugriff: 09.12.2017)
[4] https://www.ecma-international.org/ecma-262/8.0/index.html (Letzter Zugriff: 09.12.2017)
[5] https://www.tiobe.com/tiobe-index/ (Stand: September 2017)
[6] https://projects.spring.io/spring-framework/ (Letzter Zugriff: 08.12.2017)

Technologieentscheidung wäre. Abbildung 6.5 liefert eine Übersicht über die Implementierung und die Entwurfsartefakte, die als Grundlage der Überführung dienen.

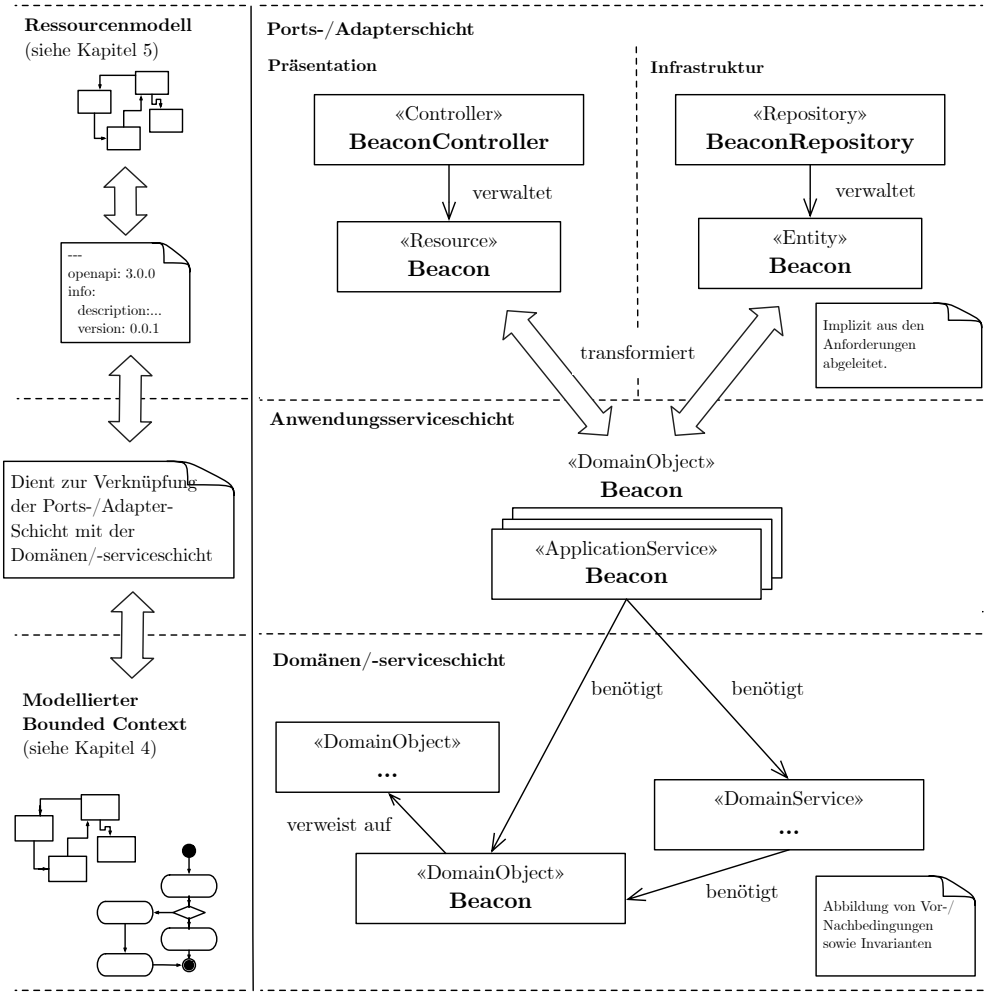


Abbildung 6.5: Übersicht über die Implementierung und die zugrundeliegenden Entwurfsartefakte

Domänen- und Domänenserviceschicht

Die Domänenschicht ist das Herzstück eines Microservice nach dem domänengetriebenen Entwurfsansatz von DDD, in der die Domänenobjekte verankert sind. Letztere haben keinen Bezug zu infrastrukturspezifischen Informationen und sind völlig lose gekoppelt, wodurch sie sich völlig unabhängig weiterentwickeln können. Für die Überführung wird jedes Domänenobjekt mit dem Stereotyp `Entity` als eigenständige Klasse mit der Bezeichnung des Domänenobjekts im Paket *domain.entities* realisiert. Analog werden die Attribute des Modellelements als Attribute der Klasse mit dem privaten Zugriffsmodifikator überführt sowie Getter-Methoden generiert, die den Zugriff auf die Attribute kapseln. So ist auch bei etwaigen Beziehungen vorzugehen. Anschließend müssen die einzelnen Operationen

überführt werden: jeweils zum Bearbeiten (*Update*), Erstellen (*Create*) sowie Löschen (*Delete*). Die Operation zum Lesen wird durch die Getter-Methoden erbracht und ist aufgrund der seiteneffektfreien und idempotenten Eigenschaft nicht gesondert zu behandeln. Als Parameter der jeweiligen Operation wird ein sogenanntes *DTO* nach dem gleichnamigen Entwurfsmuster aus [Fo02a] verwendet. Das stellt u. a. sicher, dass jeweils nur eine Operation für die Umsetzung einer CRUD-Logik benötigt wird. Zur Illustration wurde beispielhaft eine Schnittstelle für ein Domänenobjekt mittels Java erstellt (vgl. Quelltext 6.6), an der sich ein Domänenobjekt ausrichten muss.

```
1 public interface IDomainObject<T> {
2   void update(T entity) throws DomainInvariantException,
        DomainPreConditionException, DomainPostConditionException;
3   void delete() throws DomainInvariantException,
        DomainPreConditionException, DomainPostConditionException;
4   void create(T entity) throws DomainInvariantException,
        DomainPreConditionException, DomainPostConditionException;
5 }
```

Quelltext 6.6: Generische `IDomainObject`-Schnittstelle

Für jede Operation, die Seiteneffekte verursacht, wird nun eine Methode erzeugt, die vor der eigentlichen Ausführung auf etwaige Invarianten sowie Vor- und Nachbedingungen prüft[7] (vgl. Quelltext 6.7). Diese Vor- und Nachbedingungen und Invarianten sind den jeweiligen Protokollzustandsdiagrammen zu entnehmen (vgl. Abschnitt 4.2.2), die mit den entsprechenden Operationen verknüpft sind. Zudem sind auch die impliziten Randbedingungen zu betrachten (vgl. Abschnitt 4.2.1).

```
1   /*...*/
2   @Override
3   public void create(T entity)
4       throws DomainInvariantException, DomainPreConditionException,
            DomainPostConditionException {
5     this.preCreate();
6     this._create(entity);
7     this.postCreate();
8   }
9
10  protected abstract void _create(T entity);
11  /*...*/
```

Quelltext 6.7: Auszug einer generischen `DomainObjectBase`-Klasse

[7] Im Kontext von Java hat sich in der Praxis das Framework C4J (http://c4j-team.github.io/C4J/) bewährt. Für die Beschreibung wird allerdings Abstand von der aspektorientierten Programmierung genommen, um das Verständnis durch die Verwendung von bekannteren Programmierkonstrukten zu erleichtern.

Durch das Überschreiben der Methoden `preCreate` und `postCreate` lassen sich Invarianten sowie Vor- und Nachbedingungen überprüfen. Bei einer Verletzung wird eine sogenannte Ausnahme (eng. exception) geworfen, die dann auch entsprechend behandelt werden muss. Bisher wurden vorrangig die Domänenentitäten behandelt. Die Domänenobjekte mit dem Stereotyp `ValueObject` können nach dem gleichen Prinzip überführt werden. Zu beachten ist nur, dass sie über keine entsprechende Methode zum Bearbeiten verfügen (vgl. Abschnitt 4.2.1).

Die Domänenservices sind auf der Domänenserviceschicht verankert und kennzeichnen sich im Bounded Context durch die Stereotypen `DomainService`. Die Überführung auf die Implementierungsebene ist an dieser Stelle durch das Aktivitätsdiagramm in Abschnitt 4.2.2 gegeben. Die einzelnen Aktivitäten verweisen dabei auf die entsprechenden Operationen der Domänenobjekte, die letztlich auf der Implementierungsebene ausgeführt werden sollen. Ähnlich wie die Domänenschicht enthält auch diese Schicht Konstrukte, welche die Vor- und Nachbedingungen sowie die Invarianten prüfen. Da das Konzept dem der vorigen Erläuterung ähnelt, wird es nicht nochmals expliziert.

Ports-/Adapterschicht

Im Kontext dieser Arbeit werden zwei verschiedene Ports-/Adapter beleuchtet, die sich in Infrastruktur und Präsentation unterteilen lassen. Diese Aufteilung findet sich auch in der Strukturierung des Microservice wieder (vgl. Abschnitt 6.4.1). Die Adapter für die Infrastruktur repräsentieren dabei diejenigen Adapter, die von dem Microservice benötigt werden. Die Adapter für die Infrastruktur sind dann die vom Microservice benötigten. Hingegen liefern die Adapter für die Präsentation den Service-Konsumenten Informationen des Microservice. Im Kontext dieser Arbeit bzw. des Applikationsszenarios sind für die Infrastruktur ein Datenbank-Adapter und für die Präsentation die bereitzustellende Web-API vorgesehen. Das zugrundeliegende Konzept findet allerdings auch bei weiteren Adaptern seine Anwendung, weshalb die Einschränkung lediglich zur Fokussierung auf das Szenario in Abschnitt 4.3 dient.

(1) **Infrastruktur:** Um Domänenobjekte mit Datenbankeinträgen zu verknüpfen, lässt sich im Kontext von Java bspw. die JPA[8] nutzen, die vor allem die Domänenobjekte auf ein Datenbankschema abbilden soll. Für dessen Aufbau werden hierfür entsprechende Klassen angelegt und mit `@Entity` annotiert. Über weitere Annotationen kann schließlich das Datenbankschema weiter verfeinert werden. Um den Datenbankzugriff in geeigneter Form zu kapseln, lässt sich das sogenannte *Repository*-Muster verwenden: „[It m]ediates between the domain and data mapping layers using a collection-like interface for accessing domain objects“ [Fo02a, S. 280]. Das Repository liefert zudem höherwertige Funktionalitäten, die von der Anwendungsserviceschicht genutzt werden können. Dazu zählt bspw. eine Möglichkeit, etwaige Domänenobjekte

[8] Für andere Programmiersprachen existiert ein entsprechendes Pendant, weshalb die nachfolgenden Ausführungen mit geringem Aufwand auch auf andere Technologien übertragen werden können.

nach bestimmten Kriterien zu suchen. Quelltext 6.8 zeigt ein entsprechendes Beispiel.

```
1  /* ... */
2  public List<T> findAllBy(Class<T> className, Integer limit,
       Integer offset, Filters filters,
3        Sorts sorts, TimeRange timeRange) {
4      String query = this.getQueryString(className,
           filters.toHqlQuery(), sorts.toHqlQuery(),
5           timeRange.toHqlQuery());
6
7      try {
8        return this.executeQuery(entityManager ->
9            (List<T>) entityManager
10               .createQuery(query)
11               .setFirstResult(offset)
12               .setMaxResults(limit)
13               .getResultList()
14       );
15     } catch (InfrastructureException e) {
16       /* Handling different exceptions */
17     }
18     /* Further handling of the result */
19   }
20   /* ... */
```

Quelltext 6.8: Auszug einer generischen `Repository`-Klasse

Neben dem Zugriff auf die Datenbank ist entscheidend, dass die Ergebnismenge einer Datenbankabfrage in die entsprechenden Domänenobjekte überführt werden kann. Falls notwendig, muss hier eine entsprechende Transformationslogik implementiert werden.

(2) **Präsentation:** Die Web-API veräußert den Bounded Context durch eine geeignete Abstraktion und unter Anwendung eines ressourcenorientierten Stils. Als Basis für die Überführung dienen die in Abschnitt 6.2 vorgestellte OAI-Spezifikation sowie die JAX-RS-Spezifikation in der Version 2[9]. Letztere definiert u. a., wie Ressourcen über eine Web-API im Umfeld von Java EE bereitgestellt werden können. Zuerst werden hierzu für die definierten Schemata entsprechende DTO-Klassen erzeugt. Über die Annotation `@JsonProperty` kann schließlich für jedes Attribut des DTO festgelegt werden, wie sein Pendant in JSON repräsentiert wird. Dabei sind die in Abschnitt 5.4 aufgestellten Konventionen zu beachten. Die einzelnen Pfade

[9] http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-fr-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf (Letzter Zugriff: 11.10.2017).

der OAI-Spezifikation können schließlich über weitere Annotationen der JAX-RS-Spezifikation in Form von `Controller`-Klassen umgesetzt werden (vgl. Punkt 6.9).

Die `Controller`-Klassen sollen die Anfragen aufbereiten, damit Anwendungsservices (vgl. Abschnitt 6.4.2) letztere weiterverarbeiten können. Die Verknüpfung der OAI-Spezifikationsbestandteile zu JAX-RS wird hier nicht explizit aufgeführt, da diese Arbeit das grundlegende und technologieagnostische Konzept fokussiert. Jede Ressource sollte grundsätzlich über eine eigenständige `Controller`-Klasse im Sinne von SoC verfügen, wobei die Listenressource in der Regel mit ihren verknüpften Ressourcen zusammengeführt wird. Die Ansteuerung einer einzelnen Controller-Klasse basiert auf dem *FrontController*-Muster, das von einer Referenzimplementierung durch JAX-RS, wie bspw. Jersey, bereitgestellt wird [Fo02a, vgl. S. 298].

```
1  /* ... */
2  @GET
3  @Path("{uuid}")
4  @Produces({MediaType.APPLICATION_SMARTCAMPUS_HAL_JSON})
5  @ApiOperation(value = "Get Beacons with given uuid")
6  @ApiResponses(value = {
7    @ApiResponse(code = 404, message = "The beacon with the given
       uuid could not be found", response = ErrorResource.class),
8    @ApiResponse(code = 200, message = "The beacon with the given
       uuid", response = BeaconResource.class)
9  })
10 public Response getBeaconWith(
11    /* Additional API parameters for sorting, filtering,... are
        hidden */
12    @ApiParam(value = "The uuid of the beacon", required = true)
        @PathParam("uuid") String uuid) {
13  /* Further handling of the request */
14 }
15 /* ... */
```

Quelltext 6.9: Auszug einer `Controller`-Klasse

Bei der Überführung muss die OAI-Spezifikation korrekt abgebildet werden. Um sie zu validieren, lässt sich durch die Verwendung der *Swagger Core Library*[10] basierend auf den JAX-RS-Annotationen eine entsprechende OAI-Spezifikation generiert werden. Im Idealfall sollte diese resultierende OAI-Spezifikation mit der in Abschnitt 6.2 vorgestellten auf syntaktischer Ebene identisch sein.

[10] https://github.com/swagger-api/swagger-core (Letzter Zugriff: 11.10.2017).

Anwendungsserviceschicht

Auf dieser Schicht ist die Anwendungslogik verortet, die nicht von der zugrundeliegenden Domäne abhängt und die Ports-/Adapterschicht mit der Domänen- und der Domänenserviceschicht verknüpft. Die Betrachtung anwendungsbezogener Anforderungen, die sich aus dem Spezifikationsdokument ergeben (vgl. Abbildung 4.5 auf Seite 75), werden hier nicht näher betrachtet. Wenn Software-Architekten die Gefährdung der Wiederverwendbarkeit erkennen und sich dennoch dagegen entscheiden, so können sich die anwendungsbezogenen Anforderungen hier manifestieren. In dieser Arbeit findet sich auf dieser Ebene nur implizite Anwendungslogik, die sich aus der notwendigen Verknüpfung der Ports-/Adapterschicht (vgl. den vorigen Unterabschnitt) mit der Domänen- bzw. Domänenserviceschicht ableiten lässt.

```
1  /* Further attributes */
2  private Repository<Beacon> beaconRepository = new Repository<Beacon>();
3
4  /* Further methods */
5  public Beacon getBeaconWithUUID(String uuid, Parameters
       optionalParameters) throws DomainObjectNotFoundException {
6    try {
7      Optional<Beacon> beacon =
           Optional.of(this.getBeaconRepository().findById(uuid,
           optionalParameters));
8      if(beacon.isPresent()) {
9         return beacon.toDomainObject();
10     }
11     throw new DomainObjectNotFoundException();
12   } catch(DatabaseException ex) {
13       /* Handling exception */
14   }
15 }
16 /* Further methods */
```

Quelltext 6.10: Auszug der `BeaconApplicationService`-Klasse

Jegliche Interaktion mit der Domänen- bzw. Domänenserviceschicht erfolgt über die offerierte ressourcenorientierte Web-API. Sie nimmt die Anfrage entgegen, validiert die Parameter und leitet sie an den zuständigen *ApplicationService* weiter. Nun sind diese Parameter zu verarbeiten, um dem Anfragesteller bzw. dem Service-Konsumenten zu antworten. Falls auf existierende Domänenobjekte zurückzugreifen ist, erfolgt dies über die entsprechenden *Repositories* (vgl. Abschnitt 6.4.2), die Methoden zur Interaktion mit der Datenbank bereitstellen. Eine derartige Interaktion resultiert wiederum in einer entsprechenden Datenbank-Entität, die nun in ein Domänenobjekt überführt werden muss.

Das ist allerdings die Aufgabe des zuständigen Adapters (vgl. Quelltextzeile 9 in Quelltext 6.10). Der Service-Konsument erhält schließlich die Repräsentation der Ressource, aus der das Domänenobjekt hervorgegangen ist bzw. überführt wurde. Falls es nicht gefunden werden kann, wird eine Ausnahme geworfen, die in das in Punkt 6.2 definierte Schema überführt werden muss.

6.5 Microservice-Architektur nach dem domänengetriebenen Entwurfsansatz

Nachdem der letzte Abschnitt die Überführung eines ressourcenorientierten Microservice nach dem domänengetriebenen Entwurfsansatz geschildert hat, bei dem vorrangig die Mikroarchitektur eines einzelnen Microservice betrachtet wurde, fokussiert das Folgende die Makroarchitektur und damit die letztliche Microservice-Architektur (vgl.[VA+09]). Eine Makroarchitektur ist die Beschreibung eines Systems auf einer höheren Abstraktionsebene, die bspw. dessen Strukturierung betrachtet und die damit einhergehenden Schnittstellen zwischen den Systembausteinen fokussiert. Der Rahmen für die Microservice-Architektur wird durch das Domänenmodell gegeben (vgl. Abbildung 4.5).

Die Microservices nach dem präsentierten domänengetriebenen Entwurfsansatz (vgl. Kapitel 4 und 5) beinhalten keine anwendungsspezifischen Implementierungen (nachfolgend als Anwendungslogik bezeichnet), sondern veräußern lediglich geschäftliche Funktionalitäten. Dadurch besitzen letztere optimale Wiederverwendbarkeit und sind also für viele unterschiedliche Anwendungen bzw. Anwendungsgruppen nutzbar. Primäre Adressaten dieser geschäftlichen Dienstleistung sind Service-Konsumenten, die sich hier in 1) interne und 2) externe Service-Konsumenten unterteilen lassen. Als letztere gelten hier Systeme, Services und Anwendungen, die von außen auf die Microservice-Architektur zugreifen, während als interne Service-Konsumenten Microservices innerhalb der Microservice-Architektur gefasst sind. Die Verankerung der Anwendungslogik ist den externen Service-Konsumenten zuzuordnen, da die internen aus Gründen der Wiederverwendbarkeit ohne gesonderte Anwendungslogik sein sollten (vgl. Anhang A.3).

Obwohl die Anwendungslogik auf Seiten des externen Service-Konsumenten verankert werden sollte, können Anforderungen existieren, die Anwendungslogik auf Ebene der Microservices verlangen. Dies ist bspw. erforderlich, wenn rechenintensive Operationen auf die Microservice-Architektur auszulagern sind, die sich nicht als Bestandteil der abzubildenden Geschäftsdomäne ansehen lassen. Dann kann ein spezieller Microservice (nachfolgend als *BFF-Service* bezeichnet) unter Anwendung des gleichnamigen Musters BFF [Ne15, Wo15] implementiert werden, der dem Service-Konsumenten eine optimale Benutzererfahrung (engl. user experience) ermöglichen soll. Hierzu werden im BFF-Service die benötigten Microservices orchestriert und eine für den Service-Konsumenten ausgerichtete Web-API bereitgestellt. Das reduziert letztlich den Netzwerkverkehr und die notwendigen

Transformationen auf Seiten des Service-Konsumenten. Wird dieses Konzept mit bestehenden Ansätzen verglichen, so kann ein BFF-Service als anwendungsspezifisches Gateway zum Zugriff auf die Microservice-Architektur angesehen werden, das wiederum weiter unterteilt werden kann (vgl. [De17]). Grundsätzlich sind BFF-Services geeignete Kandidaten zur Implementierung von Querschnittsthemen, zu denen bspw. eine Authentifizierung oder eine Autorisierung gehört. Allerdings sind diese Querschnittsthemen meist nicht für spezifische Anwendungskontexte ausgelegt und können daher anwendungsübergreifend eingesetzt werden, weshalb dann anstelle dessen häufig ein zentrales Web-API-Gateway (kurz: API-Gateway) zu finden ist. Ergänzend liefern heutige API-Managementsysteme weiterführende Funktionalitäten zur Verwaltung von Web-APIs, wie bspw. zur Monetarisierung und Überwachung, was ebenfalls durch ein zentrales API-Gateway bereitgestellt werden kann [De17]. Ob ein BFF-Service benötigt wird, hängt stets vom jeweiligen Anwendungskontext ab. Er wird aber grundsätzlich im Zuge einer loseren Kopplung zwischen Service-Konsument und benötigten Microservices empfohlen[11]. Dann können sich Service-Konsumenten und Microservices unabhängig voneinander entwickeln, wobei stets ein Abgleich zwischen den Interaktionsteilnehmern bezüglich der benötigten und bereitgestellten geschäftlichen Dienstleistung aufgrund der inhärenten Abhängigkeit erfolgen muss. Deshalb sind Web-APIs durch deren vertragliche Eigenschaft ein zentrales Element für den Entwurf und die Entwicklung moderner Microservice-Architekturen.

Der vorige Abschnitt hat vor allem den Service-Konsumenten fokussiert. Nun stehen die Microservices der Microservice-Architektur im Mittelpunkt. Eine Microservice-Architektur ist in unterschiedliche Ebenen unterteilbar. In dieser Arbeit wurden folgende Ebenen identifiziert, die sowohl in mittelständischen als auch in Großunternehmen angewendet werden: (1) geschäftsbezogene Microservices und (2) geschäftsagnostische Microservices. Geschäftsbezogene Microservices ergeben sich direkt aus dem Domänenmodell, während die geschäftsagnostischen eher technisch orientiert sind und die höherwertigen Microservices bei der Erfüllung ihrer geschäftlichen Dienstleistung unterstützen. So kann bspw. ein Microservice zur Bildanalyse oder Speicherung großer Datenmengen von den geschäftsbezogenen Microservices verwendet werden. Derartige Funktionalitäten sind bspw. bei Google unter der Google Cloud Plattform[12] oder bei Amazon unter Amazon Web Services (AWS)[13] zusammengefasst. Werden diese Microservices mit bekannten Konzepten aus dem Bereich der SOA verknüpft, so entsprechen diese den *Utility*-Services [Er08]. Diese geschäftsagnostischen Microservices sind somit das Fundament, auf das höherwertige Microservices mit Geschäftsbezug zurückgreifen können. Diese Arbeit fokussiert allerdings diese geschäftsagnostischen Microservices nicht.

Geschäftsbezogene Microservices repräsentieren die Geschäftslogik der abzubildenden Geschäftsdo-

[11] An dieser Stelle gilt es zu anzumerken, dass einige API-Managementsysteme heutzutage auch die Orchestrierung von Microservices, Transformation von Datenstrukturen sowie zu einem gewissen Grad auch anwendungsspezifische Gateways unterstützen [De17]. Dadurch verlagern sich die Funktionalitäten eines dedizierten BFF-Service zunehmend auf ein API-Managementsystem.

[12] https://cloud.google.com/products/ (Letzter Zugriff: 10.12.2017)

[13] https://aws.amazon.com/ (Letzter Zugriff: 10.12.2017)

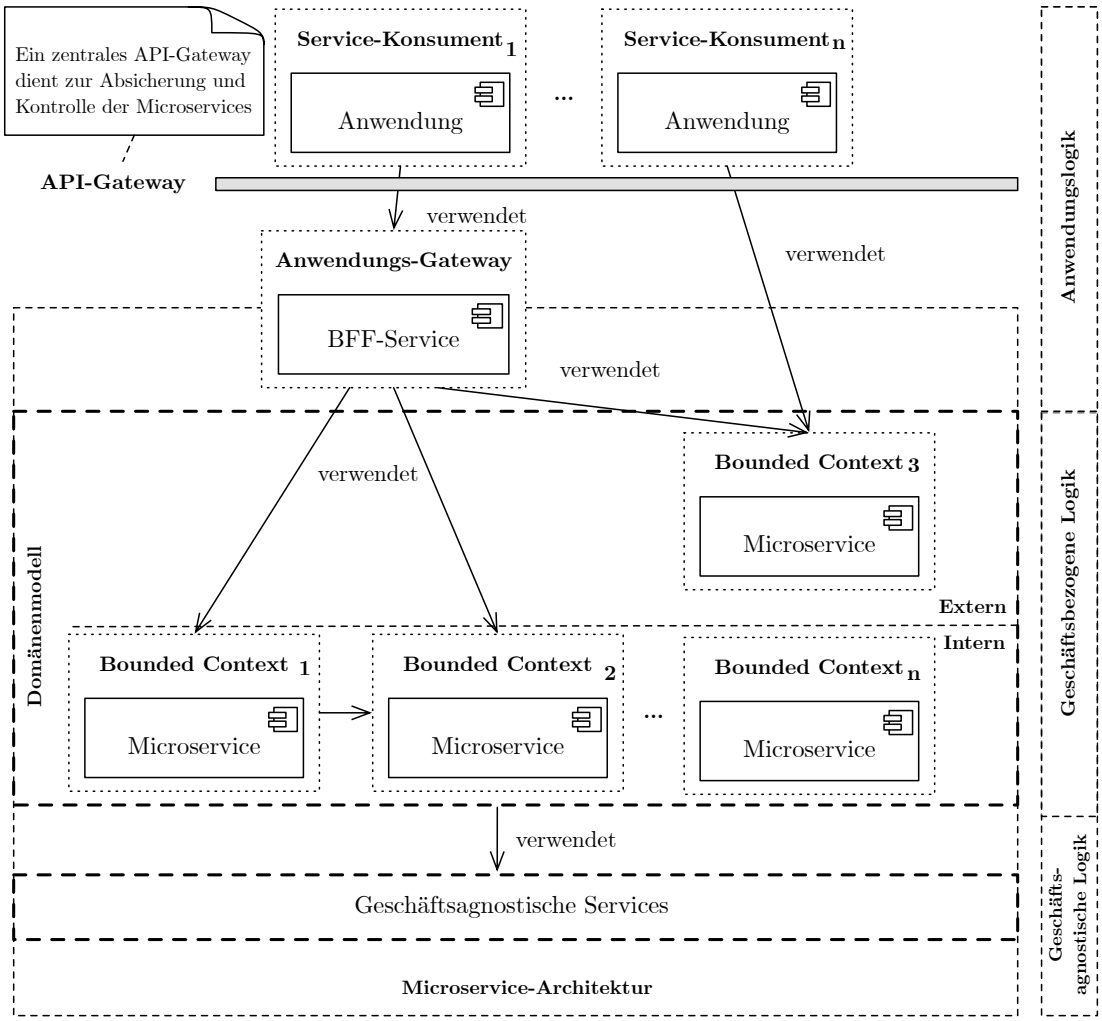


Abbildung 6.6: Übersicht über die Microservice-Architektur basierend auf einem Domänenmodell

mäne und können weiter unterteilt werden. Eine solche Möglichkeit praktiziert etwa schon der Ansatz von Evans [Ev03] als Domänenmodellierung durch eine Kerndomäne (engl. core domain) sowie mehrere unterstützende Domänen (engl. supporting domain) (vgl. Abschnitt 2.2.3). Weiter lassen sich diese Microservices auch hinsichtlich ihrer Sichtbarkeit unterscheiden. So können bspw. Microservices nur für die interne Wiederverwendung gedacht sein, während andere von externen Service-Konsumenten verwendet werden können. Der hier präsentierte domänengetriebene Entwurfsansatz ist unabhängig von der letztlichen Verortung der geschäftsbezogenen Microservices einsetzbar.

Abbildung 6.6 fasst diese Beschreibung zusammen. Abbildung 6.7 hingegen ist eine alternative Darstellung der Microservice-Architektur mit Fokus auf den einzelnen Unterteilungen, die sich auch bei Unternehmen zur Verortung der Microservices verwenden lässt.

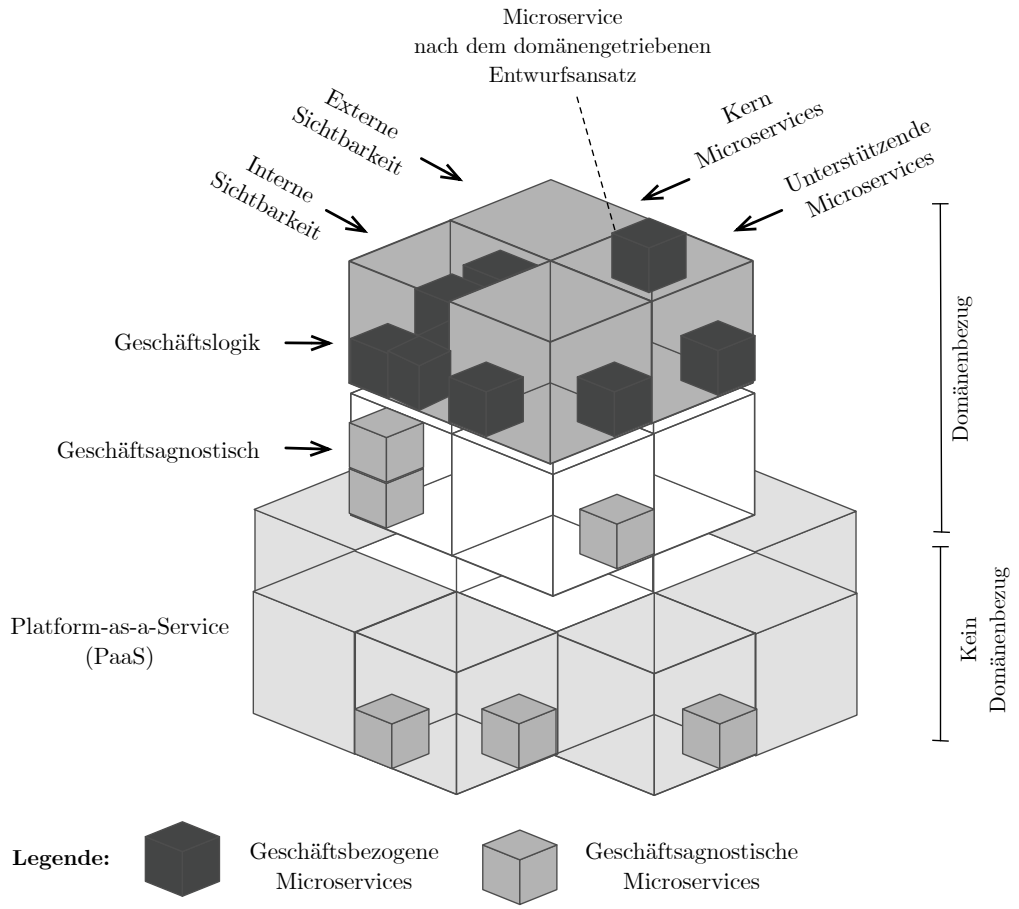


Abbildung 6.7: Übersicht über die Microservice-Architektur und deren Aufteilung sowie deren Einbettung in eine PaaS

6.6 Zusammenfassung

Ausgehend von den vorangehenden Entwurfsartefakten aus Kapitel 4 und Kapitel 5 skizzierte dieses Kapitel eine systematische Überführung auf die Implementierungsebene unter Berücksichtigung der hexagonalen Architektur nach [Ve13]. Dabei wurde zuerst das resultierende Ressourcenmodell mit einem zu wählenden Anwendungsschichtprotokoll verknüpft und dann auf dieser Basis eine Web-API-Spezifikation mittels der OAI abgeleitet. Die Web-API-Spezifikation bildet damit einen konkreten Vertrag, wie mit dem letztlichen ressourcenorientierten Microservice zu interagieren ist und ist für alle Interaktionsteilnehmer bindend. Nur so kann eine reibungslose Interaktion gewährleistet und die Entwicklung der einzelnen Interaktionsteilnehmer in geeigneter Form parallelisiert werden. Wegen der Bedeutung der Web-API für die Wiederverwendbarkeit und auch für das Gesamtsystem wurde die Qualität des gesamten Entwurfsprozesses durch Einführung eines Review weiter gesichert. Obwohl ein solches Verfahren grundsätzlich stets empfohlen wird, muss der Software-Architekt bzw.

das Entwicklungsteam des Microservice darüber entscheiden. Falls beim Review ein Handlungsbedarf festgestellt wurde, ist vor der Überführung auf die Implementierungsebene eine weitere Iteration zur Überarbeitung des Ressourcenmodells zu durchlaufen. Darauf gestützt, schilderte das Kapitel die Überführung des Entwurfs auf die einzelnen Schichten der hexagonalen Architektur, bei der zunächst die grundsätzliche Strukturierung im Quellcode aufgezeigt wurde. Danach wurde für jede Schicht erläutert, wie sie sich aus den Entwurfsartefakten ableiten lässt, was schließlich einen ressourcenorientierten Microservice ergab. Am Ende des Kapitels wurde die Makroarchitektur betrachtet, da der Entwurfsprozess bisher vor allem die Mikroarchitektur im Kontext eines geschäftsbezogenen Microservice erörterte, obgleich er wiederum Abhängigkeiten zu anderen Microservices aufweisen kann. Die resultierenden Microservices lassen sich in einer Microservice-Architektur einordnen und auf verschiedenen Ebenen gruppieren, was ein Gesamtbild der Systemarchitektur ergibt.

7 Validierung der Beiträge

Dieses Kapitel soll die Beiträge dieser Arbeit validieren. Der hier vorgestellte domänengetriebene Entwurfsansatz ermöglicht ein systematisches und nachvollziehbares Vorgehen zum Entwerfen von ressourcenorientierten Microservices und die textgestützte Überführung auf die Implementierungsebene. Die Verwendung von Mustern und die Festlegung von Konventionen während der einzelnen Phasen des Entwurfsprozesses gewährleisten, dass die so gewonnene Web-API den geforderten Qualitätsaspekten gerecht wird. Dazu zählen bspw. evolutionäre Stabilität oder die einfache Benutzbarkeit, die wiederum auch die Integration der Web-API und letztlich die Wiederverwendbarkeit eines Microservice fördert.

Die folgende Beschreibung und Klassifikation der Validierung nutzt verschiedene Validierungsarten der modellbasierten Methoden zur Vorhersage der Leistungsfähigkeit, die in [BR08] und [He08] aufgeführt wurden. Sie werden auch in der Dissertation von Durdik [Du16] in leicht abgeänderter Form zur Validierung der Dokumentation von architekturellen Entwurfsentscheidungen mithilfe von Entwurfsmustern eingesetzt. Zu den Validierungsarten nach [Du16] zählen die Machbarkeit (Typ 0), die Eignung (Typ I), die Anwendbarkeit (Typ II) und der Kosten/Nutzen (Typ III).

Die Validierung der Beiträge dieser Arbeit setzt sich aus fünf Bestandteilen zusammen, die sich jeweils einer der genannten Validierungsarten zuordnen lassen: (1) Vergleich mit/ohne Entwurfsprozess, (2) Demonstration an einem durchgängigen Beispiel, (3) kontrolliertes Experiment mit Studierenden, (4) zwei empirische Studien sowie (5) eine Demonstration im industriellen Umfeld.

Ein beispielhaftes Domänenmodell nach Evans [Ev03] sollte die Vorzüge durch den domänengetriebenen Entwurfsansatz im Vergleich zu einem fehlenden Ansatz aufzeigen (1), während dieser Ansatz zudem den Forschungsstand samt seinen Defiziten diskutiert (*Typ 0: Machbarkeit*). Die Anwendung an einem durchgängigen und zu einem gewissen Grad auch praxisnahen Beispiel (2) hingegen, soll die Eignung des domänengetriebenen Entwurfsansatzes inklusive der so hervorgebrachten Artefakte und des Prozesses zeigen (*Typ I: Eignung*). Zudem soll in einem kontrollierten Experiment (3) mit Kontrollgruppen die Unterstützung beim Entwurf eines ressourcenorientierten Microservice untersucht werden, was ebenfalls der zuvor genannten Validierungsart zuzuordnen ist. Die empirischen Studien (4) sollen schließlich die prognostizierten Vorzüge dieses Ansatzes im industriellen Umfeld belegen, während mit der Demonstration (5) vorrangig die Anwendbarkeit in der Praxis aufgezeigt werden soll (*Typ II: Anwendbarkeit*).

Dieses Kapitel ist wie folgt strukturiert: Abschnitt 7.1 erläutert die verschiedenen Validierungsarten und bezieht sie auf den domänengetriebenen Entwurfsansatz. Danach präsentiert Abschnitt 7.2 die Ziele der Validierung und deren Verknüpfung mit den Validierungsarten. Abschnitt 7.3 widmet sich der grundsätzlichen Bedrohung der Validität, um die Validierungsergebnisse kritisch zu diskutieren. Abschnitt 7.4 fokussiert die Typ 0-Validierung und schildert die Vorzüge des domänengetriebenen Entwurfsansatzes. Zudem wird der Forschungsstand mithilfe des Anforderungskatalogs (vgl. Abschnitt 3.1.2) nochmals diskutiert. Außer der Demonstration des Entwurfsansatzes an einem durchgehenden Beispiel wurde auch ein kontrolliertes Experiment durchgeführt, das in Abschnitt 7.5 behandelt wird und letztlich die Typ I-Validierung adressiert. Anschließend präsentiert Abschnitt 7.6 zwei durchgeführte Studien sowie eine Demonstration im Zuge der Anwendbarkeit des domänengetriebenen Entwurfsansatzes im Industriekontext – die Typ II-Validierung. Den Abschluss bildet eine Zusammenfassung des Kapitels.

7.1 Empirische Validierungsarten

In [Du16] präsentiert Durdik auf Grundlage von [BR08] und [He08] vier Validierungsarten, um auf diese gestützt empirische Evaluationen bzw. Validierungen im Feld der architekturellen Entscheidungsfindung durchzuführen. Diese Verfahren lassen sich auch auf andere Bereiche übertragen. Deshalb werden sie nun in Bezug auf Entwurfsprozesse vorgestellt. Abbildung 7.1 veranschaulicht die unterschiedlichen Validierungsarten und zeigt deren Verhältnis zur externen Validität. Dieses Konzept beschreibt, inwieweit die Ergebnisse eines Experiments sich verallgemeinern lassen [WR+12]. Je höher die externe Validität, desto besser lassen sich die Ergebnisse verallgemeinern.

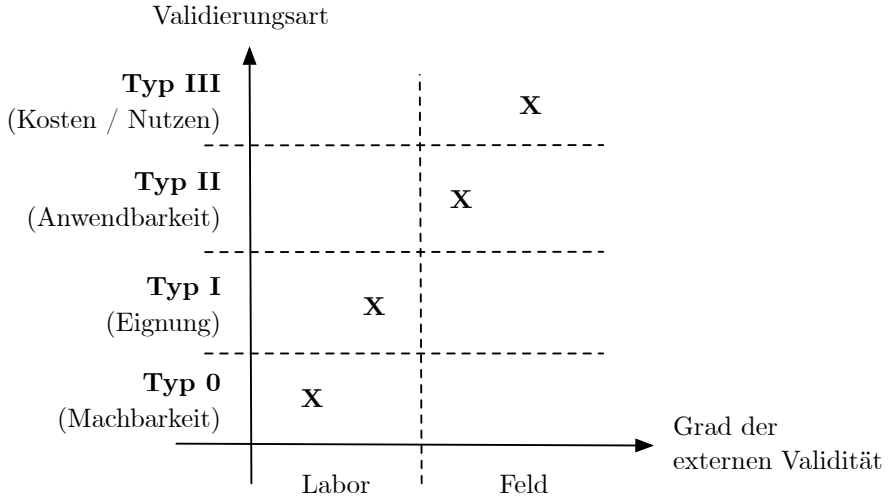


Abbildung 7.1: Übersicht über die empirischen Validierungsarten nach [Du16]

1. **Typ 0 (Machbarkeit):** Die Machbarkeit ist eine einfache Form der Validierung: Die Evaluation erfolgt, indem die aktuelle Situation mit einer hypothetischen verglichen wird, in der ein derartiger Entwurfsprozess nicht existieren würde (engl. no-approach situation) und etwa nur unklare Modellierungsartefakte vorliegen oder Überführungsschritte fehlen. Zudem lässt sich dem eigenen Ansatz der Forschungsstand gegenüberstellen. Alternativ kann die Evaluation auch auf einer Umfrage basieren [He08, Du16]. Um ein annähernd repräsentatives Ergebnis zu erhalten, sollten unterschiedliche Personengruppen aus der letztlichen Zielgruppe des Entwurfsprozesses gewählt werden. Üblicherweise erfordert diese Form der Evaluation nicht den Bezug zu einem industrienahen Kontext, da sie nur Machbarkeit bestätigen soll. Die Typ 0-Validierung verlangt im Vergleich mit anderen Validierungsarten nur geringen Aufwand [Du16].

2. **Typ I (Eignung):** Die Evaluation der Eignung des Entwurfsprozesses soll die Anwendbarkeit der vorgestellten Modellierungsartefakte, deren systematische und nachvollziehbare Ableitung, die Unterstützung bei etwaigen Entwurfsentscheidungen sowie die Überführung auf die Implementierungsebene belegen. Während die Typ 0-Validierung dies lediglich beispielhaft und auszugsweise zeigt, soll dies hier, ähnlich wie bei Durdik [Du16], an einem durchgehenden und möglichst praxisnahen Beispiel bestätigt werden. Anders als bei Durdik lassen sich für die vorliegende Evaluation auch externe Nutzer für vereinzelte Experimente hinzuziehen.

3. **Typ II (Anwendbarkeit):** Die Typ II-Validierung soll die Anwendbarkeit des Entwurfsprozesses für die letztliche Zielgruppe belegen. Die Zielgruppe sind hier Software-Architekten und -Entwickler oder auch Studierende der Informatik. Dass sich Studierende für die Validierung heranziehen lassen, zeigt [Ti00]. Im Gegensatz zur Typ I-Validierung wird hier der Entwurfsprozess in einem realen Kontext angewandt. Die Evaluation erfolgt anhand eines Vergleichs mit einem üblichen Entwurfsprozess, der bspw. keine Berücksichtigung von Qualitätsaspekten vorsieht oder keine systematische Ableitung aus Modellierungsartefakten ermöglicht. Den Vergleich führt die Zielgruppe mithilfe eines Fragenkatalogs durch, damit ein repräsentatives Ergebnis gewonnen wird. Zusätzlich kann mittels einer Demonstration im realen Kontext die Anwendbarkeit in der Praxis gezeigt werden.

4. **Typ III (Kosten/Nutzen):** Diese Form der Validierung basiert auf einer Kosten/Nutzen-Analyse, bei welcher der Entwurf eines Microservice im realen Kontext zumindest zweimal durchgeführt wird. Dabei wird der Entwurf des Microservice einmal ohne den zu validierenden Entwurfsprozess und damit u. a. auch mit dem Risiko nicht berücksichtigter Qualitätsaspekte durchgeführt und einmal unter Anwendung des Entwurfsprozesses. Der Vergleich der Kosten für beide Projekte über einen längeren Zeitraum erlaubt es, die Vorzüge des Entwurfsprozesses als Geschäftswert zu beziffern. Doch geht diese Form der Validierung mit hohen Aufwänden und Kosten einher und wird deshalb nur selten durchgeführt [He08].

Bei der Validierung des Ansatzes sollte das Ziel der Bestätigung der externen Validität verfolgt werden,

sodass die Beiträge dieser Arbeit auch in unterschiedlichen Kontexten angewandt werden können: „The external validity of an experimental evaluation refers to the generalisability of the findings to different situations. The results for a study with a low external validity do not hold if the method is applied under slightly different conditions“ [He08, S. 245]. Als einer der Einflussfaktoren kann in diesem Fall das zu betrachtende Beispiel angesehen werden. Dabei sollte sich möglichst auf ein reales Projekt bezogen werden, mit dem der Entwurfsprozess erprobt wird. Es sollten keine Bedingungen geschaffen werden, unter denen der Entwurfsprozess ausschließlich funktioniert. Je weiter man sich also vom Laborumfeld entfernt, desto stärker fördert das die externe Validität. Dies veranschaulicht auch Abbildung 7.1, in welcher die Validierungsarten durch ein "X" eingeordnet sind.

7.2 Gegenstand der Validierung

In dieser Arbeit wurden der domänengetriebene Entwurfsprozess und die damit einhergehenden Beiträge mittels der Validierungsarten vom Typ 0, Typ I und Typ II evaluiert. Typ III wurde wegen des hohen Aufwands nicht berücksichtigt, findet derzeit aber Anwendung bei einem Unternehmen der Automobilbranche, sodass zumindest eine Tendenz erkennbar wird.

1. **Typ 0 (Machbarkeit):** Die Typ 0-Validierung wurde durch den Vergleich mit und ohne Entwurfsprozess anhand eines Beispiels durchgeführt, um dessen Vorzüge zu zeigen. Zudem wurden Forschungsarbeiten zum Thema auf ihre Defizite im direkten Vergleich zum vorliegenden Entwurfsprozess analysiert. Die Resultate der Untersuchung erörtert Abschnitt 7.4.

2. **Typ I (Eignung):** Die Typ I-Validierung erfolgte anhand des realitätsnahen Beispiels im Kontext des SmartCampus am KIT (wie in Abschnitt 4.3 dargestellt) auf Grundlage von Modellierungsartefakten. In der empirischen Studie wurde die systematische und nachvollziehbare Vorgehensweise sowie die Unterstützung eines Software-Architekten bei der Ableitung eines Ressourcenmodells unter Berücksichtigung von Qualitätsaspekten demonstriert. Darüber hinaus wurde mit Kapitel 6 gezeigt, wie der letztliche Entwurf auf die Implementierungsebene übertragen und so die Implementierung der Microservices eingeleitet werden kann. Um die Unterstützung eines Software-Architekten als zentraler Bestandteil des Entwurfsprozesses zu beleuchten, wurde eine Untersuchung über mehrere Semester durchgeführt. Die Beschreibung des Experiments ist in Abschnitt 7.5 aufgeführt. Gleichzeitig wurde ein Experiment mit zwei Kontrollgruppen von Studierenden durchgeführt, bei der eine Kontrollgruppe im Besitz einer Kontrollliste war. Dieses Experiment diente als Validierung des Beitrags (vgl. [GG+16c]).

3. **Typ II (Anwendbarkeit):** Die Anwendbarkeit (Typ II-Validierung) wurde mit zwei empirischen Studien im industriellen Kontext durchgeführt, die sich auf ein kontrolliertes Experiment stützten. Darüber hinaus wurde die Anwendbarkeit an einem Industrieprojekt demonstriert. Die Ergebnisse der Studien sowie deren Durchführung sind Fokus von Abschnitt 7.6.

Vorzüge	Typ 0	Typ I	Typ II
Reduktion von Entwurfsfehlern	●	●	●
Reduktion des notwendigen Wissens bei Entwurfsentscheidungen	●	●	●
Verbesserung der Wiederverwendbarkeit des Microservice	○	●	●
Reduktion von Implementierungsfehlern	●	○	○

Tabelle 7.1: Relation zwischen den Vorzügen des Entwurfsansatzes und den verwendeten Validierungsarten

Die Tabelle 7.1 setzt die Validierungsarten in Bezug zu den Vorzügen des geschilderten domänengetriebenen Entwurfsansatzes. Dabei ist (●) als positive Relation zu werten, wonach der Vorzug des Entwurfsansatzes ein Ziel der Validierung war. Entsprechend repräsentiert (○) das Gegenstück, wonach die Umsetzung der Validierungsart nicht zur Validierung des Vorzugs beigetragen hat.

7.3 Bedrohung und Einschränkung der Validität

Dieser Abschnitt liefert die grundsätzlichen Bedrohungen einer Validität und stützt sich dabei auf die Klassifikation nach [WR+12] basierend auf [WR+00, Yi08]. Wohlin et al. [WR+12] unterscheiden vier Arten der Validität: 1) Konstruktvalidität, 2) interne Validität, 3) externe Validität und 4) Ergebnisvalidität (engl. conclusion validity), die nun definiert werden:

> **Definition (Konstruktvalidität):** *Die Konstruktvalidität erfasst, inwieweit das Ergebnis des Experiments sich auf das zugrundeliegende Konzept übertragen lässt, um dieses zu generalisieren [WR+12].*

> **Definition (Interne Validität):** *Die interne Validität erfasst, inwieweit kausale Abhängigkeiten zwischen dem Experiment und dessen Ergebnissen betrachtet wurden. Die kausalen Abhängigkeiten können die Unabhängigkeit einzelner Aspekte des Experiments beeinflussen, ohne dass der Forscher sich dessen bewusst ist [WR+12].*

> **Definition (Externe Validität):** *Die externe Validität erfasst, inwieweit etwaige Randbedingungen des Experiments dessen Verallgemeinerbarkeit für ein Industrieszenario erschweren oder verhindern [WR+12].*

> **Definition (Ergebnisvalidität):** *Die Ergebnisvalidität erfasst, inwieweit eine Schlussfolgerung aus dem durchgeführten Experiment und dessen Resultaten sich beeinflussen lässt [WR+12]. So kann bspw. fehlende Teststärke (engl. statistical power) zu falschen Schlussfolgerungen führen.*

Diese Arten der Validität werden nun im Rahmen der durchgeführten Experimente zur Validierung herangezogen, um die Gültigkeit der Ergebnisse zu diskutieren.

7.4 Evaluation der Machbarkeit

Für die Evaluation der Machbarkeit wurde der Anforderungskatalog aus Kapitel 3 herangezogen, mit dem die Arbeiten bewertet sowie der Handlungsbedarf und die Motivation für diese Arbeit ermittelt wurden. Das Ergebnis dieser Bewertung ist Tabelle 3.2 zu entnehmen. Wird die vorliegende Arbeit nun auf Basis dieses Anforderungskatalogs bewertet und mit ähnlichen Ansätzen verglichen, bringt das den Vergleich für die Typ 0-Validierung im Sinne von Abschnitt 7.1. Als Grundlage dienen die Anforderungen: (A1) Nachvollziehbarkeit, (A2) Betrachtung von strukturellen und verhaltensspezifischen Aspekten, (A3) keine Einschränkung hinsichtlich der Verwendung von Hypermedia, (A4) Unterstützung bei Entwurfsentscheidungen, (A5) Technologie- und Plattformunabhängigkeit und (A6) Abbildung des Entwurfs auf die Implementierungsebene. Das Ergebnis der Bewertung dieser Arbeit zeigt Tabelle 7.2, während die einzelnen Anforderungen (A1-A6) nachfolgend im Detail erörtert werden. Abschließend werden diese Anforderungen mit den Vorzügen der Arbeit verknüpft, um auch dahingehend eine logische Zugehörigkeit zu erreichen.

(A1) **Nachvollziehbarkeit:** Die Nachvollziehbarkeit (A1) stellt sicher, dass zwischen den verschiedenen Modellierungsartefakten stets eine systematische Ableitung gegeben ist. Dies wird gewährleistet, indem die einzelnen Überführungsschritte sich auf Überführungsvorschriften stützen. Die Überführungsvorschrift $t(x)$ hat dabei die Hauptaufgabe die Modellierungselemente x_{source} von Modell M_{source} in ein Modell M_{target} zu überführen, sodass gilt $t(x_{source}) \in M_{target}$. Hierzu wurden in dieser Arbeit entsprechende Heuristiken und Überführungsregeln aufgestellt, welche die Konzepte von DDD auf einen ressourcenorientierten Architekturstil überführen und sich auf mehrere Prozessschritte verteilen (vgl. R_1, R_2, R_3 in Kapitel 5). Um auch die Implementierung in geeigneter Form zu adressieren, wurde auch hier eine Überführung auf die letztliche Implementierungsebene aufgezeigt. Dieser Schritt ist vergleichbar mit *Service translation* aus dem Ansatz [LS+09, Se11], der allerdings nicht im Detail ausgeführt wurde. Die Überführungsvorschriften lassen sich zudem als Grundlage für weiterführende Arbeiten für eine automatisierte Überführung des Entwurfs auf die Implementierungsebene (vgl. Abschnitt 8.2) nutzen. Ohne diese Systematik könnte nicht sichergestellt werden, dass unabhängige Software-Architekten beim gleichen zugrundeliegenden Domänenmodell bzw. dem einzelnen Bounded Context zu einem vergleichbaren Ressourcenmodell kommen. Denn die Menge der erforderlichen Entwurfsentscheidungen wäre zu groß und die Folgen von einzelnen Entscheidungen für den Entwurf bzw. für dessen Qualität wären zu vielfältig. Bezogen auf MDSD, wo der Nachvollziehbarkeit eine hohe Wichtigkeit beigemessen wird, ginge eine automatische M2M-Transformation bzw. die letztliche Modell-zu-Text-Transformation (M2T) mit erheblichem Aufwand einher, da das Kreuzprodukt aller Entwurfsentscheidungen in geeigneter Form

	A1: Nachvollziehbarkeit	A2: Betrachtung von strukturellen und verhaltensspezifischen Aspekten	A3: Optionale Verwendung des Hypermedia-Aspekts	A4: Unterstützung bei Entwurfsentscheidungen	A5: Technologie- und Plattformunabhängigkeit	A6: Abbildung des Entwurfs auf die Implementierungsebene
[Ro16] UML-based Model-Driven REST API Development	○	◖	◖	○	●	○
[PI+16] A Pattern Language for RESTful Conversations	○	○	●	◖	●	○
[EdI+16] EMF-REST: Generation of RESTful APIs from Model	●	◖	●	○	◖	●
[HL+15] A conversation based approach for modeling REST APIs	◖	◖	●	◖	●	○
[RP+13] Structural and behavioral modeling of RESTful web service interface using UML	◖	●	◖	○	●	○
[Sc11] Modeling RESTful Applications	○	○	●	○	●	○
[LS+09] Towards a Model-Driven Process for Designing RESTful Web Services	◖	●	●	○	●	○
Diese Arbeit	●	●	●	●	●	●

Tabelle 7.2: Ergebnis der Bewertung dieser Arbeit im Vergleich zu bestehenden Arbeiten

abgebildet werden müsste. Zudem wäre die Folge der getroffenen Entscheidungen im Hinblick auf die Qualität durch den Software-Architekten zu ermitteln, was wiederum entsprechendes Wissen voraussetzt.

(A2) **Betrachtung von strukturellen und verhaltensspezifischen Aspekten (A2):** Ein Domänenmodell und die darin enthaltenen Bounded Contexts bestehen aus strukturellen und verhaltensspezifischen Aspekten, die angemessen modelliert werden müssen, um schließlich die Überführung auf ein ressourcenorientiertes Modell zu ermöglichen. Würden dagegen entweder ausschließlich die strukturellen oder nur die verhaltensspezifischen Aspekte betrachtet werden,

dann würden wesentliche Informationen für eine korrekte Überführung in ein Ressourcenmodell fehlen. Gleichzeitig ist eine Anfrage nur möglich, wenn eine entsprechende Ressource über die Web-API offeriert wird, was den strukturellen Aspekten zuzuordnen ist. Deshalb dienten UML-Diagramme dazu, die unterschiedlichen Aspekte abzubilden. Dafür wurde UML durch ein UML-Profil erweitert, sodass letztlich eine Überführung dieser Aspekte in ein Ressourcenmodell vereinfacht wird. Durch die Betrachtung der verhaltensspezifischen Aspekte lassen sich auch fachlich aufwändige Zustände mit Vor-, Nachbedingungen und Invarianten abbilden, was für die Ableitung der Repräsentationen und möglicher Fehler benötigt wird. So muss bei Nichteinhaltung einer Invariante eine Fehlermeldung generiert und dem Service-Konsumenten übermittelt werden. Dies liefert wiederum die Grundlage für die Ableitung entsprechender Vertragstests (vgl. Abschnitt 8.2).

(A3) **Optionale Verwendung des Hypermedia-Aspekts (A3):** Der Hypermedia-Aspekt wurde in dieser Arbeit nicht weiter beleuchtet. Zur Verdeutlichung wurde deshalb stets von ressourcenorientierten Microservices gesprochen und nicht von RESTful Microservices nach dem Architekturstil REST. Dennoch enthält der vorliegende Entwurfsansatz diesbezüglich keine Einschränkung, sodass der Hypermedia-Aspekt optional, etwa durch die Wahl eines entsprechenden Medientyps (vgl. Abschnitt 5.4) ergänzt werden kann. Viele ähnliche Ansätze setzen allerdings den Hypermedia-Aspekt als inhärent voraus, obgleich die Vorzüge nicht immer erkennbar sind, was am Beispiel von Zalando aufgezeigt wurde [ZAL-API-GUIDE]. Ein Entwurfsansatz ohne derartige Abgrenzung würde womöglich in einem Aufwand bei der semantischen Beschreibung resultieren, ohne dass der Mehrwert für das Unternehmen erkennbar wäre, was die praktische Anwendbarkeit des Ansatzes gefährden würde.

(A4) **Unterstützung bei Entwurfsentscheidungen:** Die Unterstützung des Software-Architekten bei Entwurfsentscheidungen durch Aufbereiten von relevantem Wissen reduziert seinen Aufwand beim Entwurf und kann gleichzeitig die Entwurfsqualität erhöhen. Falls diese Unterstützung fehlt, müsste er zunächst Probleme erkennen und dafür Lösungen erarbeiten. Dies verlangt allerdings, dass er mit dem Thema vertraut ist, das Problem identifizieren und sein Wissen auf diesen Fall anwenden kann. In dieser Arbeit wäre das Thema der Entwurf einer Web-API, mit dem sich ein Software-Architekt vertraut machen muss. Dies lässt sich allerdings kaum voraussetzen. Deshalb wurde das einschlägige Wissen zum Entwurf einer ressourcenorientierten Web-API komprimiert dargelegt. Um den Entscheidungsraum zu reduzieren, fokussieren die entscheidungsunterstützenden Mechanismen die Erhöhung der Wiederverwendbarkeit. Dies ist ein weiterer Vorzug dieser Arbeit.

(A5) **Technologie- und Plattformunabhängigkeit:** Durch Technologie- und Plattformunabhängigkeit sind die grundlegenden Konzepte dieser Arbeit derart abstrakt, dass sie sich für verschiedene Technologien und Plattformen anwenden lassen. Ohne diese Anforderung wäre das Konzept nur in einem bestimmten Umfeld einsetzbar, was letztlich den Wert und die Vorzüge dieser Arbeit

mindern würde. Deshalb hat sie die Konzepte von MDSD beherzigt. Das Ressourcenmodell ist ohne jedweden Bezug zu spezifischer Technologie oder Plattform. Erst mit der Verknüpfung zur letztlichen Zielarchitektur wird es um die Technologie-Komponente erweitert, sodass eine systematische Überführung auf die Implementierungsebene möglich ist. Die zugrundeliegende Web-API-Spezifikation bildete dabei den Vertrag zwischen dem ressourcenorientierten Microservice und etwaigen Service-Konsumenten.

(A6) **Abbildung des Entwurfs auf die Implementierungsebene:** Nach dem Entwurf der ressourcenorientierten Microservices in Form eines Ressourcenmodells wird dieses mitsamt den Modellierungsartefakten des Geschäftsausschnitts auf die Implementierungsebene überführt und abgebildet. Hierzu wird jedes Ressourcenmodell und der damit verknüpfte Bounded Context mit der Zielarchitektur verknüpft. Ohne diese Schritte der Verknüpfung und Abbildung ließe sich nicht gewährleisten, dass die Implementierung mit dem zugrundeliegenden Entwurf konform ist. Zudem entsteht erst durch die Implementierung und Umsetzung des ressourcenorientierten Microservice der tatsächliche Mehrwert für das Unternehmen. Deshalb hat diese Arbeit das Ressourcenmodell zunächst mit einem Anwendungsschichtprotokoll verknüpft und danach eine Web-API-Spezifikation abgeleitet, nach der sich die Implementierung zu richten hat. Schließlich wurde gezeigt, wie sich die Überführung der Entwurfsartefakte mithilfe von Java EE, das als Beispiel diente, auf die Implementierungsebene realisieren lässt.

Zuordnungen der Anforderungen zu den Vorzügen

Die hier untersuchten Anforderungen beeinflussen die Vorzüge des domänengetriebenen Entwurfsansatzes von ressourcenorientierten Microservices, wie in Kapitel 1 und 3 dargelegt wurde. Zusammengefasst ergibt sich der Einfluss, welcher Tabelle 7.3 zu entnehmen ist. Darin steht + für eine positive Relation zwischen Anforderung und aus ihr resultierenden Vorzügen, während +/- als indifferente Relation zu interpretieren ist. Immerhin kann auch eine Anforderung ohne positive Relation den Lösungsraum signifikant einschränken und auf diesem Wege die praktische Anwendbarkeit fördern. So wird bspw. mit der optionalen Verwendung des Hypermedia-Aspekts die praktische Anwendbarkeit erhöht, da dieser Aspekt in der Praxis kaum berücksichtigt wird.

7.5 Experimente im Rahmen der Typ I-Validierung

Die Typ I-Validierung demonstriert die Eignung des Entwurfsprozesses an einem realitätsnahen und durchgängigen Beispiel im Kontext des SmartCampus, das bereits durch das Szenario in Abschnitt 4.3 sowie dessen Anwendung in Kapitel 4, 5 und 6 als erfüllt anzusehen ist. Ein ähnliches Vorgehen verfolgte auch Durdik [Du16]. Daneben wurde diese Validierungsart auch durch unabhängige Experimente mit Studierenden über drei Semester in den Jahren 2016 und 2017 hinweg durchgeführt. Die Studierenden bearbeiteten Seminar- oder Praktikumsarbeiten in sogenannten *ServiceGroups*, die für

	A1: Nachvollziehbarkeit	A2: Betrachtung von strukturellen und verhaltensspezifischen Aspekten	A3: Optionale Verwendung des Hypermedia-Aspekts	A4: Unterstützung bei Entwurfsentscheidungen	A5: Technologie- und Plattformunabhängigkeit	A6: Abbildung des Entwurfs auf die Implementierungsebene
Reduktion von Entwurfsfehlern	+	+	+/-	+/-	+/-	+
Reduktion des notwendigen Wissens bei Entwurfsentscheidungen	+/-	+	+/-	+	+	+/-
Verbesserung der Wiederverwendbarkeit des Microservice	+/-	+/-	+/-	+	+	+/-
Reduktion von Implementierungsfehlern	+	+/-	+/-	+/-	+	+

Tabelle 7.3: Verknüpfung der Anforderungen mit den Vorzüge dieser Arbeit

eine zu modellierende Domäne einen oder mehrere ressourcenorientierte Microservices entwerfen und auch implementieren sollten. Die modellierte Domäne bzw. die darin enthaltenen Bounded Contexts adressierten unterschiedliche Anwendungsszenarien, wie bspw. die Navigation über den Campus mit Fokus auf der Barrierefreiheit, die Zugangskontrolle bei Identity and Access Management (IAM) oder das Verwalten von Abschlussarbeiten im Kontext des Campus-Managements. Die Eignung von Studierenden im Bereich der Informatik für derartige Evaluationen hat Tichy in [Ti00] dargelegt und wird daher als geeignet angesehen.

Um die Studierenden beim Entwurf zu unterstützen, wurden hierzu mehrere Leitfäden verfasst. Einer dieser Leitfäden enthielt die Erkenntnisse und Ergebnisse dieser Arbeit und sollte die Studierenden beim Entwurf von ressourcenorientierten Microservices angemessen anleiten und unterstützen. Am Ende jedes Semesters wurden die Studierenden um Rückmeldungen gebeten und die resultierende Web-API im Hinblick auf deren Wiederverwendbarkeit im Sinne von Abschnitt 2.3.3 bewertet, um

[1] Zu diesem Zeitpunkt lag den Studierenden noch kein derartiger Leitfaden vor.
[2] Für die Bewertung wurde das subjektive Empfinden der Studierenden erfasst.

	SSe 16[1]	WS 16/17	SSe 17
#Studierende	8	10	6
#Rückfragen zum allgemeinen Vorgehen	7	4	1
#Rückfragen zur Ableitung der Ressourcen	8	5	0
#Rückfragen zur Ableitung der Adressierung	8	1	1
#Rückfragen zur Interaktion mit Ressourcen	8	6	0
#Rückfragen zur Ableitung von Repräsentationen	8	7	2
#Rückfragen zur Versionierung	8	8	0
#Sonstige Rückfragen	7	3	0
#Befunde aus der Bewertung der Wiederverwendbarkeit	8	6	1
#Notwendige Überarbeitungen	6	2	2
Kommunikationsaufwand zwischen den Studierenden[2]	Sehr hoch	Angemessen	Niedrig

Tabelle 7.4: Ermittelte Ergebnisse für die Typ I-Validierung auf Grundlage von Rückmeldungen der Studierenden

den Leitfaden zu verbessern und auszubauen. Diese Überarbeitungen wurden wiederum in diese Arbeit überführt. Die Ergebnisse der Rückmeldungen ist Tabelle 7.4 zu entnehmen, die auf Grundlage etwaiger Kommunikationskanälen (z.B. E-Mail, Sprechstunde, Projektteamtreffen, internes Backlog-System) zwischen den Studierenden und dem entsprechenden Betreuer erhoben wurde. Allerdings sind die Ergebnisse mit Vorsicht zu betrachten, was der nächste Abschnitt im Hinblick auf die Bedrohung der Validität erörtert. Dennoch offenbaren die Ergebnisse einen klaren Trend. Dies deckt sich auch mit [GG+16c] wonach Studierende mit einer entsprechenden Kontrollliste (engl. checklist) für den Entwurf einer ressourcenorientierten Web-API schneller und zugleich produktiver sind. Das Ergebnis kann wiederum daraus resultieren, dass weniger Rückfragen gestellt und bewährte Methoden ausgewählt und angewandt wurden. Die Kontrollliste aus [GG+16c] ist eine Sammlung bewährter Methoden, die in diese Arbeit und in den Leitfaden integriert wurden.

7.5.1 Bedrohung der Validität und Einschränkung der Validität

Die Bedrohung der Validität wird in Bezug auf die in Abschnitt 7.3 genannten Arten der Validität untersucht.

1. **Konstruktvalidität:** Die Teilnehmer der Erhebung sind womöglich mit der Thematik des Entwurfs einer Web-API vertraut oder haben sich damit unter Umständen mithilfe von Literatur oder Vorlesungsunterlagen vertraut gemacht. Allerdings ist dies allgemein ein nicht vermeidbares Problem: „It is a natural threat and is common to surveys and experiments involving humans as subjects“ [Du16, S. 266]. Zudem könnten die Teilnehmer der Untersuchung das

Ergebnis verfälschen, falls sie Kenntnis über eine derartige Erhebung haben. Deswegen wurden die Studierenden über die Erhebung nicht informiert. Bei dem Entwurf einer Web-API könnten die Teilnehmer versucht haben, das erwartete Ergebnis zu erraten. Allerdings mussten sie ihre Entscheidungen in wöchentlichen Reviews begründen und protokollieren.

2. **Interne Validität:** Die jeweils ein Semester währende Untersuchung wurde in mehreren Semestern durchgeführt. Es waren ausschließlich Studierende an der Untersuchung beteiligt. Unter Umständen haben sich einige von ihnen am Semesterende vor allem um Prüfungsvorbereitungen gekümmert. Dies könnte die Menge der Rückfragen zur Seminar- oder Praktikumsarbeit reduziert haben. Des Weiteren kann auch die Erfahrung der Studierenden sie bei diesem Thema klar beeinflusst haben. Auch das lässt sich jedoch nicht verhindern. Zudem können auch Sprachbarrieren der Studierenden oder ihre Schüchternheit gegenüber dem Betreuer dazu geführt haben, dass weniger Rückfragen gestellt wurden. Doch hatten sie stets Gelegenheit, sich mit Studierenden auszutauschen, welche die Praktikums- und Seminararbeiten betreuten.

3. **Externe Validität:** Um die externe Validität nicht zu gefährden, wurde ein möglichst praxisnahes Szenario gewählt. Zudem wurden bei dem Entwurf wie bei der Implementierung Werkzeuge eingesetzt, die auch in der Industrie verwendet werden. Dies gilt ebenso für den agilen Entwicklungsprozess *Scrum*, der in der Industrie zunehmend praktiziert wird. Der Entwurfsprozess in dieser Arbeit richtet sich vorwiegend an Software-Architekten mit Erfahrung in der Modellierung, dem Entwurf und der Implementierung. Für die Untersuchung wurden jedoch vor allem Studierende herangezogen, was die Verallgemeinerbarkeit des Ergebnisses beeinflussen kann. Allerdings stellt dies andererseits sicher, dass auch Software-Architekten ohne Erfahrung mit dem Entwurf von ressourcenorientierten Microservices von dieser Arbeit profitieren können. Zudem unterschieden sich die zu bearbeitenden Anwendungsszenarien pro Semester, was sicherstellte, dass der Entwurfsprozess unabhängig von einer Domäne angewandt werden kann.

4. **Ergebnisvalidität:** Den erhobenen statistischen Daten fehlt die Teststärke, um verlässliche Aussagen zu erlauben. Allerdings wurden Studierende über mehrere Semester befragt, weshalb zumindest der Trend sich erkennen lässt. Schwankende Anzahlen der Studierenden über die Semester können allerdings das Ergebnis verfälscht haben. Zudem basieren die Erhebungen auf Rückfragen an den Betreuer bzw. den betreuenden Studierenden. Hier ist auch nicht auszuschließen, dass die tatsächliche Anzahl an Rückfragen größer ist. Das gilt ebenso für Diskussionen unter den Studierenden, die hier nicht berücksichtigt werden konnten, jedoch ist ein eindeutiger Trend im Lauf der Zeit zu erkennen.

7.5.2 Zusammenfassung der Ergebnisse

Die Ergebnisse weisen auf eine positive Relation zwischen den Rückfragen und der Anzahl an Befunden bei der abschließenden Bewertung der Web-API durch einen Mitarbeiter am Institut hin. Unter Berücksichtigung der Bedrohungen der Validität (wie in Abschnitt 7.5.1 dargelegt), lässt sich der Trend beobachten, dass der Leitfaden und somit die Beiträge dieser Arbeit den Entwurf von ressourcenorientierten Microservices unterstützen und weniger Entwurfsfehler entstehen. Allerdings sind auch weiterhin Rückfragen möglich, da manche Aspekte, wie bspw. die Berücksichtigung von spezifischer Anwendungslogik, in dieser Arbeit nicht betrachtet wurden (vgl. Abschnitt 1.6). Es wurden nur Studierende befragt, weshalb das Ergebnis nicht direkt auf den Industriekontext übertragbar ist. Die dort durchgeführten Studien erörtert der folgende Abschnitt 7.6.

7.6 Durchgeführte Studien im industriellen Kontext

Bei den folgenden Studien wurde der hier vorgestellte domänengetriebene Entwurfsprozess im Rahmen von Industrieprojekten angewandt, um vor allem die externe Validität zu untersuchen. Der Entwurfsprozess wurde von externen Software-Architekten und Entwicklern auf seine Anwendbarkeit und Tauglichkeit in konkreten Projekten bewertet. Diese Validierung lässt sich Typ II zuordnen (vgl. Abschnitt 7.1). Der Ansatz wurde anhand eines Bewertungsbogens beurteilt, den Anhang A.13 bringt. Da die Beschreibung des angewandten Entwurfsansatzes nicht wesentlich zur Validierung beiträgt, werden in den nächsten Unterabschnitten Artefakte nur in Auszügen präsentiert. Gegenwärtig wird der Entwurfsprozess auch in einem Projekt bei einem Automobilhersteller angewandt. Die Geheimhaltungspflicht erlaubt jedoch nur die Preisgabe von Einzelinformationen.

7.6.1 Studie 1: Entwurf eines Health-Microservice als Bestandteil der CTS-Domäne

Zur Überprüfung der Anwendbarkeit soll in Zusammenarbeit mit der milon Care GmbH[3] ein sogenannter Health-Microservice als logische Ergänzung für die *milon CARE*, einer Software-Lösung, nach dem domänengetriebenen Entwurfsansatz konzipiert werden. Die milon care GmbH entwickelt die *milon CARE* für die Steuerung von Trainingsgeräten sowie für die Planung, Dokumentation und Analyse von Trainings. Die *milon CARE* ist in der Domäne *Connected Training System (CTS)* verortet, mit der sämtliche Trainingsgeräte einer Fitness- oder Gesundheitseinrichtung vernetzt werden sollen, um Trainingseinheiten den konkreten Bedürfnissen des Trainierenden anzupassen. Zum besseren Verständnis zeigt Abbildung 7.2 eine Übersicht über die CTS-Domäne im Rahmen der *milon CARE* sowie die Service-Konsumenten. Auf Vollständigkeit wurde zugunsten der Übersichtlichkeit verzichtet.

[3] https://www.milon.com/ (Letzter Zugriff: 20.10.2017)

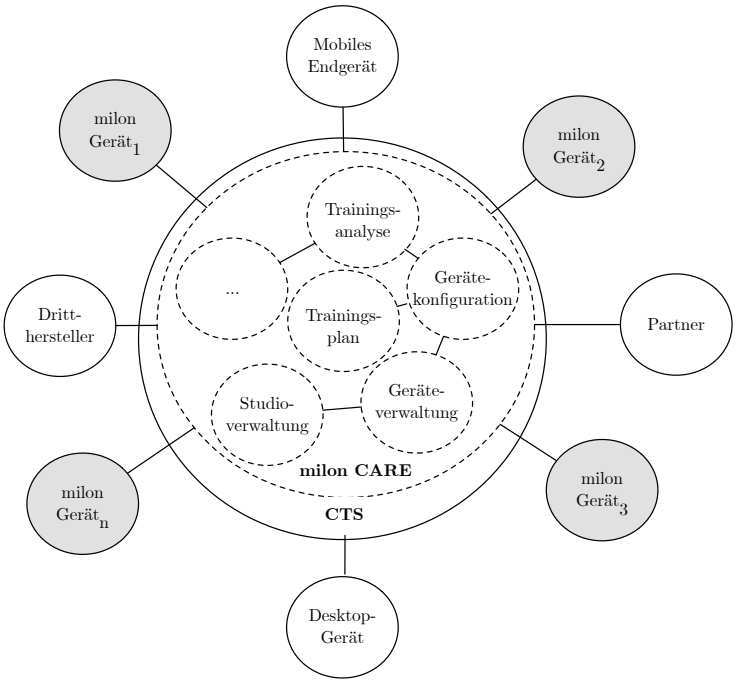


Abbildung 7.2: Übersicht über die CTS-Domäne im Rahmen der *milon CARE*

Gegebenes Szenario

Der Health-Microservice soll bestehende Plattformen von Drittherstellern, wie bspw. Polar[4] oder Garmin[5], vereinheitlichen und so eine herstellerneutrale Sicht auf die Informationen zur körperlichen Gesundheit ermöglichen. Die Web-API soll so gestaltet sein, dass sie bestimmte Qualitätsaspekte wie bspw. evolutionäre Stabilität erfüllt. Die resultierenden Artefakte sind ein modellierter Bounded Context und der Entwurf einer Web-API samt dazugehöriger Web-API-Spezifikation, um die Implementierung zu starten. Damit der Prozess der Vereinheitlichung von existierenden Plattformen überschaubar bleibt, wurden die in Tabelle 7.5 genannten Drittheristeller ausgewählt. Die Entscheidung basiert auf verfügbaren Dokumentationen der Hersteller sowie die Marktverbreitung ihrer Plattformen.

Bisheriges Vorgehen

Bisher fehlt ein vergleichbarer Entwurfsansatz bei der milon Care GmbH, um systematisch einen ressourcenorientierten Microservice auf Grundlage etwaiger Modellierungsartefakte und mit Fokus auf hoher Wiederverwendbarkeit abzuleiten. Demnach ähnelt das Verfahren hier der Evaluation der

[4] https://www.polar.com/ (Letzter Zugriff: 20.10.2017)
[5] http://www.garmin.com/ (Letzter Zugriff: 20.10.2017)

Drittherstellers	Bezeichnung der Web-API	Dokumentation der Web-API	Zeitpunkt der Untersuchung
Garmin	Health API	https://developer.garmin.com/health-api/overview/	02.10.2017
Polar	Accesslink API	https://developer.polar.com/wiki/AccessLink	02.10.2017
Life Fitness	LFconnect API	https://developers.lfconnect.com/lfopen/	02.10.2017

Tabelle 7.5: Zu betrachtende Dritthersteller im Rahmen der Validierung bei der milon Care GmbH

Machbarkeit in Abschnitt 7.4. Das bisherige Vorgehen beruhte auf der Identifikation anwendungsbezogener Anforderungen ohne konkreten Fokus auf die Wiederverwendbarkeit der Microservices. Neue Anwendungen und damit neue Service-Konsumenten, die zum Zeitpunkt des Entwurfs noch nicht ersichtlich waren, konnten die bereitgestellten Web-APIs nur mit hohem Aufwand integrieren. So mussten bspw. bestimmte Funktionalitäten auf Seiten des Service-Konsumenten ausgelagert werden, worunter auch die Benutzbarkeit litt. Zudem wurde beim initialen Entwurf eine Versionierung nicht betrachtet, weshalb Änderungen an einer Web-API nur mit großer Sorgfalt durchgeführt werden können. Es fehlt zudem weiterhin eine Dokumentation der Web-API, was den Kommunikationsaufwand zwischen den Software-Entwicklern von Service-Konsumenten und den Software-Entwicklern der *milon CARE* erhöhte.

Modellierter Bounded Context und CTS-Domäne

Abbildung 7.4 zeigt einen Auszug des modellierten Bounded Context des Health-Microservice in der CTS-Domäne. Die strukturellen Aspekte des Bounded Context wurden auf Grundlage der existierenden Plattformen von Drittherstellern (vgl. Tabelle 7.5) gemeinsam mit Domänenexperten der milon Care GmbH im Sinn der Konzepte aus Abschnitt 4.2 entworfen. Die existierenden Plattformen wurden analysiert, um auch für sie Bounded Contexts mit Schwerpunkt auf den strukturellen Aspekten zu erstellen, was als Grundlage für das Context Mapping diente. Eine Vertraulichkeitsvereinbarung (engl. non-disclosure agreement, kurz: NDA) erlaubt es nicht, diese Modellierungsartefakte näher vorzustellen. So muss Abbildung 7.3 reichen, um den grundsätzlichen Aufbau des Microservice mit Schwerpunkt auf den benötigten Adaptern zu skizzieren, der sich letztlich in der CTS-Domäne verorten lässt. So soll der Health-Microservice künftig die gesammelten Gesundheitsinformationen aus der Anamnese, aus durchgeführten Trainings (Trainingsaufnahme) und den Systemen von Drittanbietern bereitstellen und verwalten lassen, um künftige Trainings zu optimieren und für den Trainierenden effizienter zu gestalten. Die Algorithmen zur Auswertung der Gesundheitsinformationen gehören ebenfalls zu diesem Bounded Context. Die persönlichen Gesundheitsinformationen erhält ein Trainierender über einen Nutzeridentifikator. Andere Informationen über ihn, wie bspw. Name oder Alter, werden dagegen in einem separaten Microservice verwaltet.

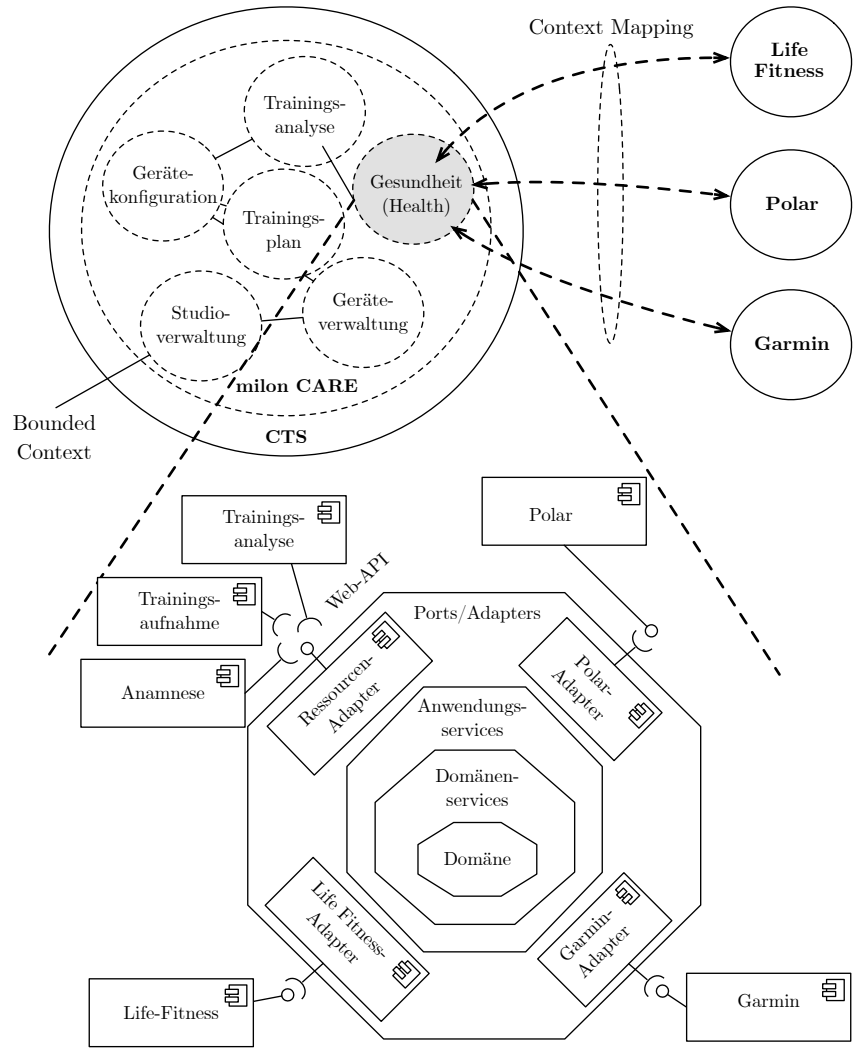


Abbildung 7.3: Grundsätzlicher Aufbau des Health-Microservice und Verortung innerhalb der CTS-Domäne

Abgeleitete Web-API-Spezifikation

Auf Basis des modellierten Bounded Context, mithilfe der in Kapitel 5 erörterten Aktivitäten und durch Überführung des Ressourcenmodells auf die Implementierungsebene, wie in Kapitel 6 dargestellt, ergibt sich die Web-API-Spezifikation, die Abbildung 7.5 mittels Swagger UI zeigt. Die Darstellung der vollständigen Web-API würde viel Platz benötigen und kaum zur Validierung beitragen. Bei der Überführung des Ressourcenmodells aus dem modellierten Bounded Context (vgl. Abbildung 7.4) wurde die mehrschichtige Mustersprache Abschnitt 5.3.2 angewandt, um eine hohe Wiederverwendbarkeit zu erreichen und Integration auf Service-Konsumentenseite Abbildung 7.3 mit minimalem Aufwand zu ermöglichen.

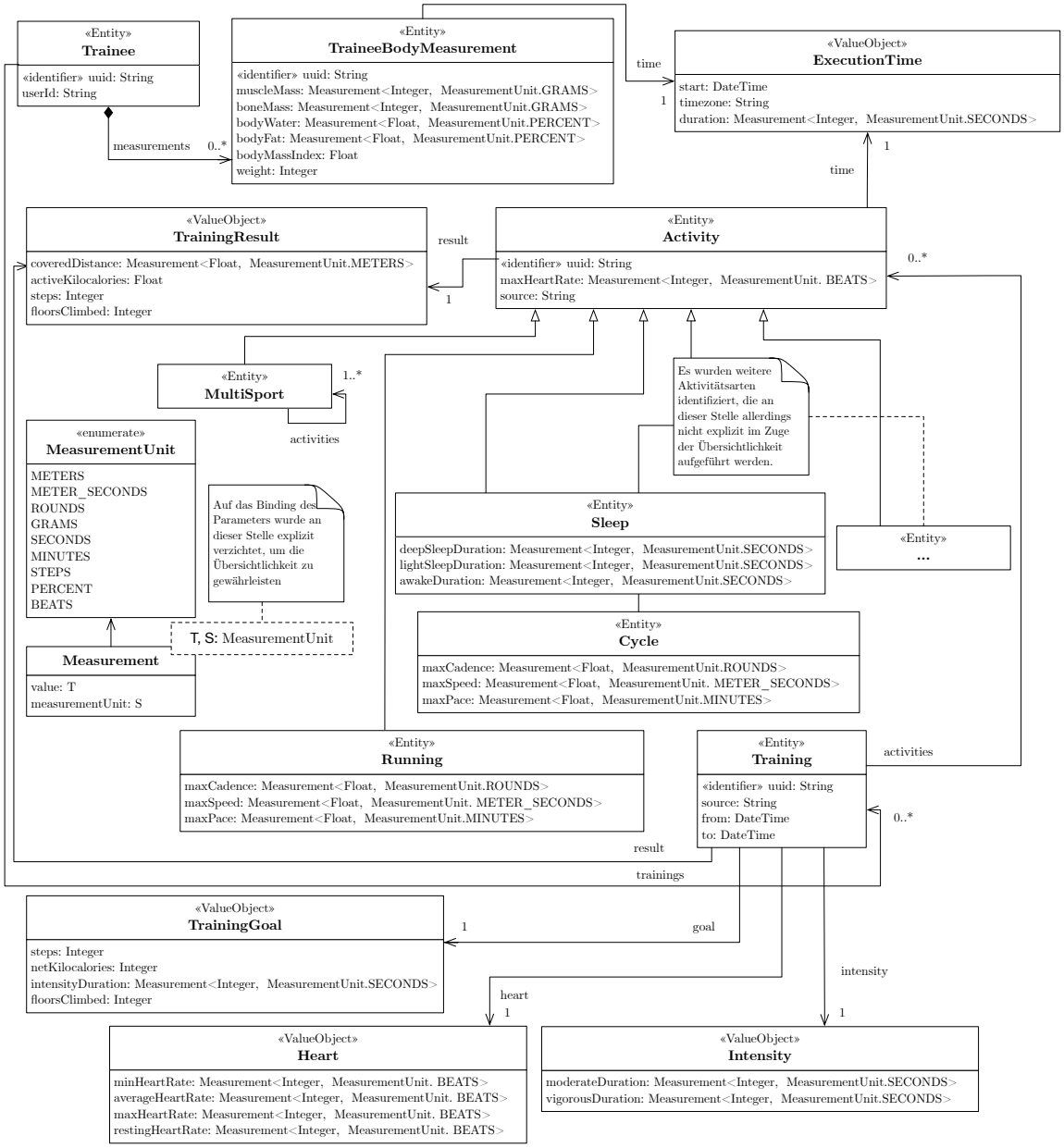


Abbildung 7.4: Auszug des abgeleiteten Domänenmodells durch Analyse der Dritthersteller und in Rücksprache mit Domänenexperten/Business-Analysten seitens der milon Care GmbH

Darüber hinaus wurden Konventionen zur Adressierung (vgl. Tabelle 5.5) und Repräsentationen entsprechend Abschnitt 5.4 festgelegt, die auch in künftigen Web-APIs im Kontext der *milon CARE* berücksichtigt werden sollen. Die Einhaltung dieser Konventionen fördert die Konsistenz und damit auch die Wiederverwendbarkeit. Durch die Anwendung der Versionierung wurde die evolutionäre Stabilität adäquat betrachtet, sodass auch diesbezüglich die Interoperabilität sichergestellt ist. Allerdings verlangt das die Anwendung des Prozesses im Sinne von Abschnitt 5.5.2.

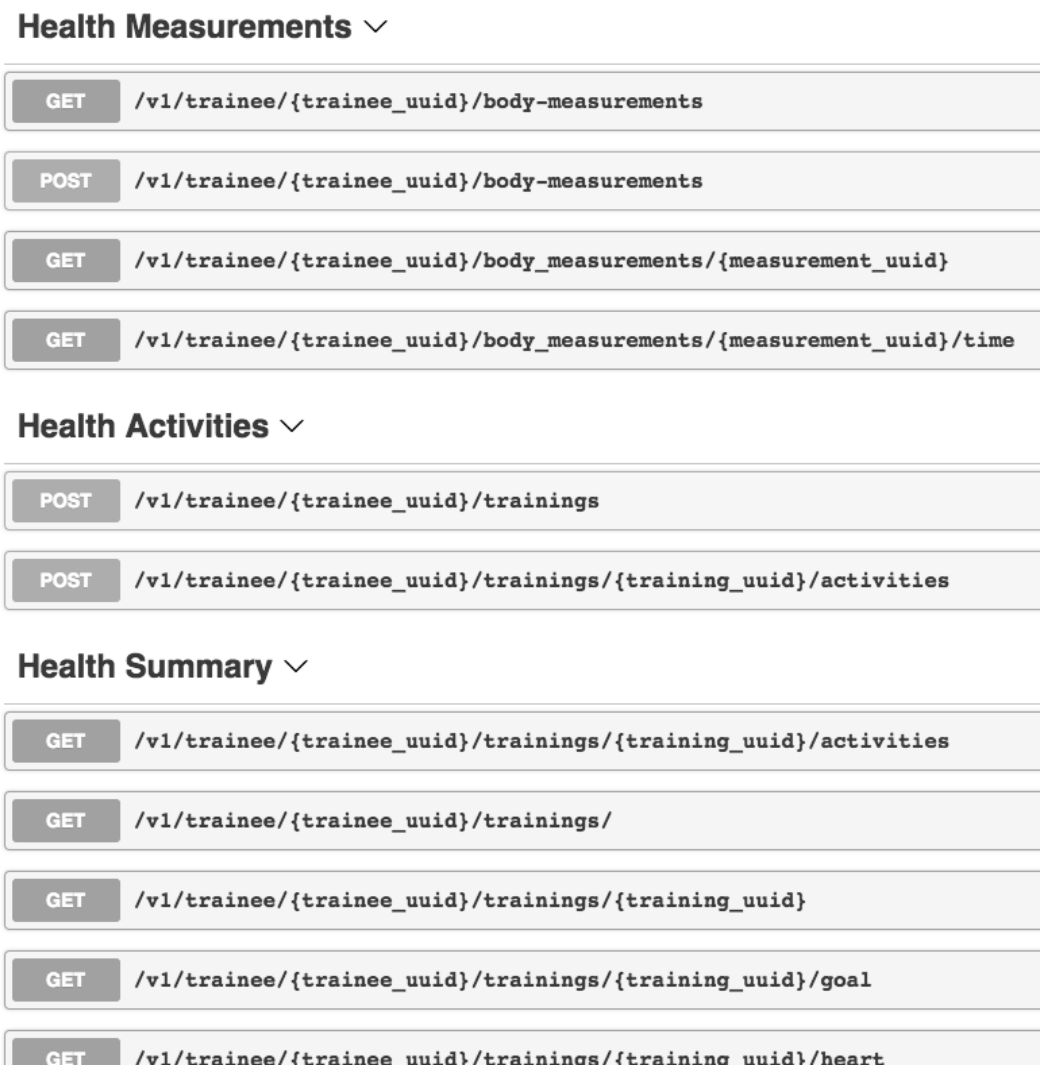


Abbildung 7.5: Auszug der Web-API-Spezifikation für den Health-Microservice, welche mittels *Swagger* UI visualisiert wurde

Bewertung der Anwendbarkeit und resultierende Vorzüge

Nach einem Workshop, der Mitarbeiter der milon Care GmbH in dem domänengetriebenen Entwurf schulen sollte, wurde die Anwendbarkeit mit einem Bewertungsbogen erfasst. Die Schulung erfolgte nach der Durchführung des geschilderten Szenarios. Tabelle 7.6 und Abbildung 7.6 repräsentieren die Einstufung der Teilnehmer des Workshops in Bezug auf Berufserfahrung und Wissen in diesem Kontext. Abbildung 7.6 gibt auf der x-Achse die Werte $1-5$ zwischen nicht *vertraut* und *sehr vertraut* an. Das Ergebnis zeigt, dass die Teilnehmer ein relativ breites Wissensspektrum in den Themengebieten *DDD*, *REST* und *Microservices* haben, dieses im Mittel aber eher im Grundlagenbereich liegt. Da diese Ergebnisse jedoch subjektiv die Sicht der einzelnen Teilnehmer wiedergeben, ist die Aussagekraft mit

Vorsicht zu betrachten, da Teilnehmer ihr Wissen in der Regel zu hoch einschätzen. Die Frage zur Anzahl am Projekt beteiligter Personen wurde bei der Bewertung nicht berücksichtigt, da bei einer Schulung auch Teilnehmer den Fragebogen ausgefüllt haben, die nur am Rande dem jeweiligen Projekt und dem Workshop beigewohnt haben. Insgesamt haben 14 Teilnehmer den Fragebogen ausgefüllt. Drei Teilnehmer taten dies nicht vollständig, ihre Bögen wurden daher nicht ausgewertet.

Rolle	#Anzahl
Software-Entwickler (Junior)	#1
Software-Entwickler (Senior)	#4
Software-Architekt (Junior)	#0
Software-Architekt (Senior)	#3
DevOp	#0
Andere	#3

Berufsjahre	#Anzahl
Weniger als 1 Jahr	#1
1 -3 Jahre	#3
3-5 Jahre	#4
5 - 8 Jahre	#2
8 - 10 Jahre	#0
Über 10 Jahre	#1

Tabelle 7.6: Übersicht über die Teilnehmer in Bezug auf ihr Wissen

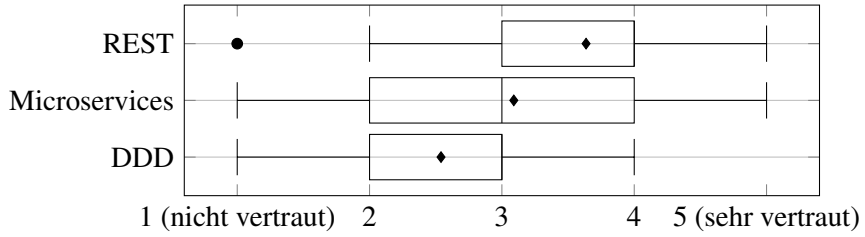


Abbildung 7.6: Subjektive Einstufung der Teilnehmer hinsichtlich ihres vorherrschenden Wissens

Es wurden wesentliche Schwerpunkte im Fragebogen erfragt: zur Modellierung des relevanten Geschäftsausschnitts der Domäne bzw. der darin enthaltenen Bounded Contexts und zur Überführung in ein Ressourcenmodell und damit der Web-API des resultierenden Microservice.

Modellierung der Bounded Contexts: Die Bewertung der Modellierung des Domänenmodells bzw. der Bounded Contexts überprüfte zuerst, ob die Teilnehmer selbst eine derartige Modellierung durchführen könnten, um die Grundlage für den Entwurfsprozess zu schaffen. Nur so lassen sich Entwurfsfehler überhaupt erst reduzieren. Es gaben $\approx 82\%$ der Teilnehmer an, dass sie mit dem gelieferten Beitrag selbstständig oder mit anfänglicher Unterstützung eine Modellierung durchführen könnten. Diese Bewertung deckt sich auch mit dem Ergebnis in Abbildung 7.7, wonach der Median bei allen Eigenschaften zwischen *sehr gut* und *gut* liegt. Lediglich $\approx 18\%$ der Teilnehmer (2 Teilnehmer) gaben an, mehr Hintergrundwissen zu benötigen. Einer von ihnen war ein Software-Entwickler (Junior) mit wenig Berufserfahrung und lediglich grundlegendem Wissen in den Themenbereichen, was ein Grund für seine Bewertung sein könnte. Der andere Teilnehmer war mit DDD wenig vertraut, was eine eigenständige Modellierung erschwert. Darüber hinaus zeigte die praktische Anwendbarkeit die größte Standardabweichung. Vermutlich lag das daran, dass die Modellierung nach Ansicht einiger

Teilnehmer aufwändig ist und zumeist das Projektbudget dafür fehle.

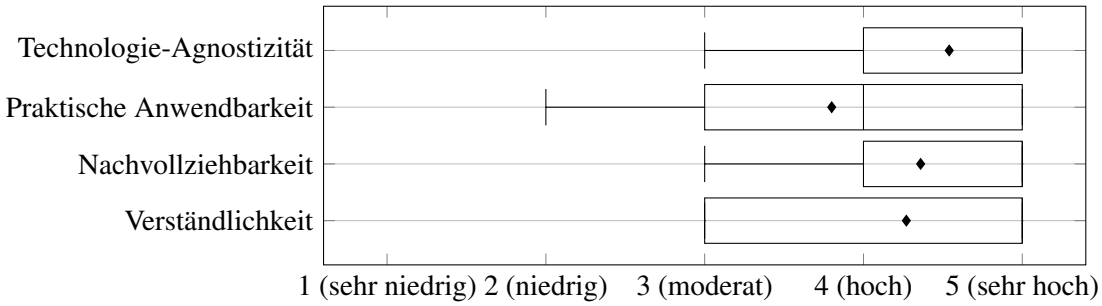


Abbildung 7.7: Ergebnis zur Bewertung der Modellierung

Im Hinblick auf die Verbesserung der Modellierung wurde die Abbildung der verhaltensspezifischen Aspekte genannt, da diese womöglich mittels UML sehr aufwändig werden könnte. Um diesen Aufwand zu reduzieren, ließen sich künftig andere Artefakte für die Beschreibung der Geschäftslogik (Verhalten des Bounded Context) nutzen. Einen passenden Ansatz hierfür liefert bspw. BDD, wo das Artefakt sich auch für die testgetriebene Software-Entwicklung verwenden lässt [RW+15]. Durch die mehrstufigen Transformationen des domänengetriebenen Entwurfsprozesses, wie in dieser Arbeit gezeigt, ist nur die erste Überführung geringfügig anzupassen, was eine derartige Erweiterung mit vertretbarem Aufwand ermöglicht. Gleichzeitig könnte dies die praktische Anwendbarkeit bei der Modellierung verbessern.

Überführung in ein Ressourcenmodell: Die Wichtigkeit einer Web-API im Unternehmen wurde von 100% der Beteiligten mit *wichtig* oder *sehr wichtig* bewertet, wodurch das Bewusstsein und die Bedeutung von gut entworfenen Web-APIs zumindest in der Mehrheit der Köpfe der Teilnehmer verankert zu sein scheint. Im Hinblick auf das bisherige Vorgehen im Unternehmen waren die Beteiligten einhellig der Meinung, dass bislang ein systematisches Vorgehen und Leitplanken zur Orientierung fehlte. Die Web-API werde bisher zumeist für eine spezifische Anwendung entworfen, was die Wiederverwendbarkeit einschränke. Dieses Ergebnis deckt sich mit dem beschriebenen Vorgehen auf Seite 176. Demnach ist ein Vergleich mit einem bisherigen Vorgehen und damit die entsprechende Fragestellung im Fragebogen für die Beteiligten obsolet. Abbildung 7.8 gibt die Bewertungen des Entwurfs von ressourcenorientierten Microservices wieder.

Wie bei der vorigen Bewertung der Modellierung wurden Unterpunkte als Bewertungsgrundlage abgeleitet, die zusammengenommen einen Median von $4,5$ und ein arithmetisches Mittel von $4,4$ ergaben. Die Bewertung der Entscheidungsunterstützung beim Entwurf deckt sich mit der Nützlichkeit der im Entwurfsprozess verankerten Muster, die alle Teilnehmer insgesamt mit *hoch* oder *sehr hoch* bewertet haben. Besonders die systematische Anwendbarkeit hielten sie für eine große Unterstützung. Die so entwickelte Web-API ist im Vergleich mit den bisherigen im Unternehmen nach Ansicht der

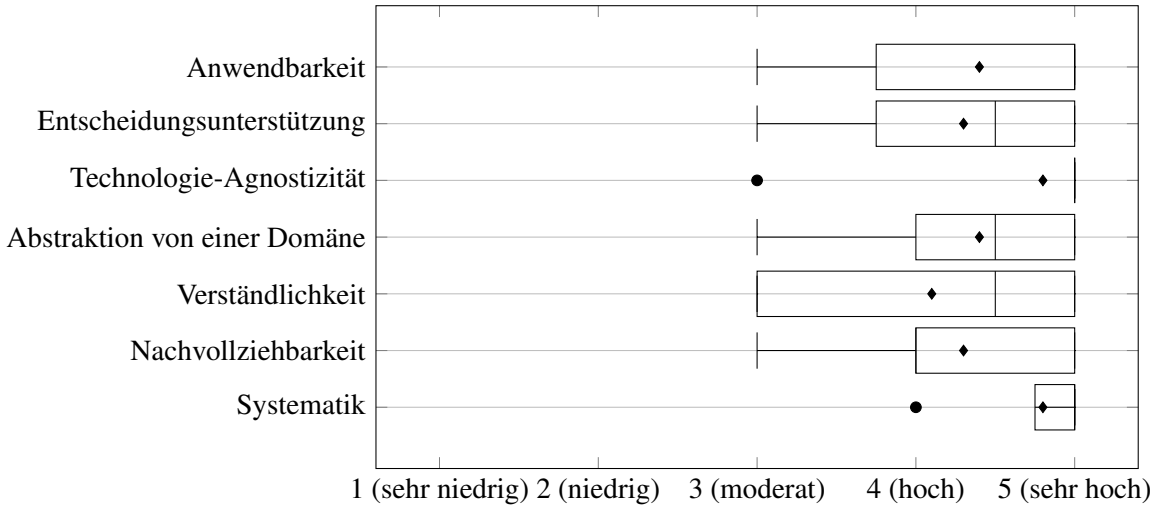


Abbildung 7.8: Ergebnis zur Bewertung der Überführung in ein Ressourcenmodell

Teilnehmer deutlich besser wiederverwendbar. Einer von ihnen sagte: „Die definierte Web-API ist aus meiner Sicht die erste [Web-]API, die nach konkreten Vorgaben entwickelt wurde und somit einer modernen [Web]-API gerecht wird“. Die Vorzüge auch im Hinblick auf die Evolution eines Microservice müssen sich, so einige Teilnehmer, im Lauf der Zeit allerdings erst noch in der Praxis bewähren, weshalb ein weiteres Projekt nach dieser Studie geplant ist. Da sich die Entwurfsentscheidungen zumeist auf Muster stützen, lässt sich die Bewährung in der Praxis stark vermuten. Die Anwendung des Entwurfsprozesses, darin war sich die Mehrheit einig, werde Review-Zyklen reduzieren und womöglich die Entwicklung von Service-Konsumenten beschleunigen (vgl. Abbildung 7.9). Ein Teilnehmer (9.1%) erkennt darin keine Reduktion von Review-Zyklen, was eventuell darauf zurückzuführen ist, dass er schon Erfahrungen bei dem Entwurf von Web-APIs besitzt.

Abschließend sei erwähnt, dass alle Teilnehmer die praktische Relevanz dieser Arbeit mit *hoch* oder *sehr hoch* bewertet haben. Also werden die erzielten Beiträge dieser Arbeit in künftigen Projekten eine Rolle spielen. Ob sie sich gänzlich etablieren lassen, muss eine Kosten/Nutzen-Analyse zeigen.

7.6.2 Studie 2: Entwurf einer Microservice-Architektur für Beweglichkeitstrainings

Die Five Konzept GmbH[6] (im Folgenden als FIVE bezeichnet) ist europaweit einer der führenden Anbieter von Trainingskonzepten für Beweglichkeitstrainings. Das Trainingskonzept richtet sich dabei sowohl an Fitnessstudios als auch an Reha- und Gesundheitszentren. Mit *FIVE touch*[7] (vgl. Anhang A.12 für entsprechende Screenshots von *FIVE touch*) wurde letztes Jahr eine Software-Anwendung vorgestellt, welche den Trainierenden bei der Ausübung seiner Übungen anleitet (vgl. Anhang A.12 für entsprechende Screenshots von *FIVE touch*). Bisher wurde die Applikation durch ein Backend-System

[6] http://www.five-konzept.de/ (Letzter Zugriff: 02.11.2017)
[7] http://www.five-konzept.de/produkte-five-touch (Letzter Zugriff: 02.11.2017)

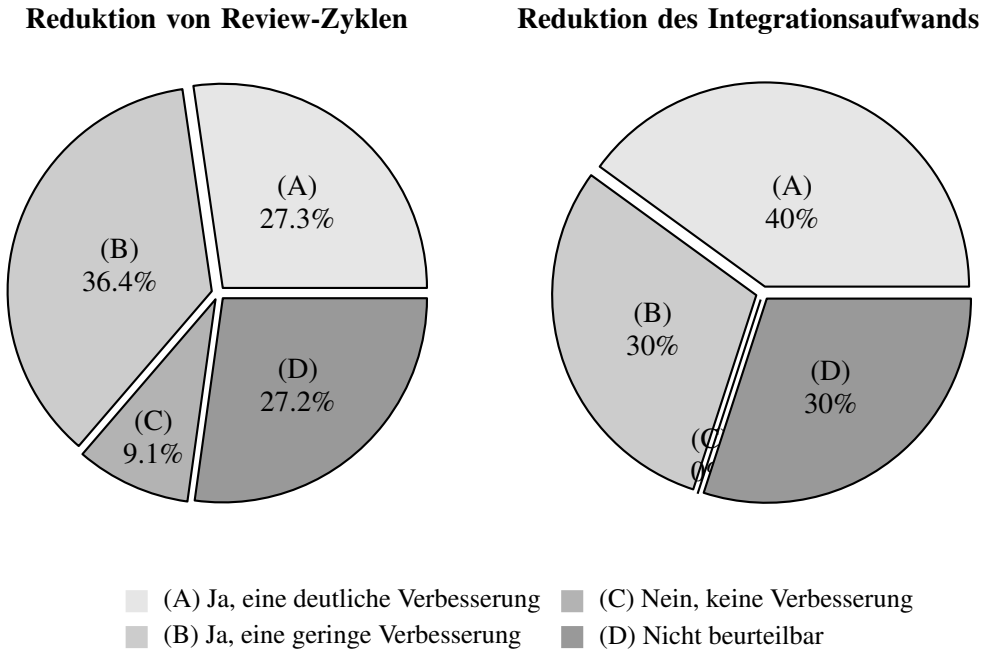


Abbildung 7.9: Ergebnisse zur Untersuchung zur Reduktion von Entwurfsfehlern und Integrationsaufwand

des Kooperationspartners milon industries GmbH gestützt. Die Trainingsgeräte greifen dabei über eine Web-API auf die *milon CARE* (vgl. Abschnitt 7.6.1) zu und ermöglichen so den Trainingsbetrieb.

Gegebenes Szenario

Im Zuge der fortschreitenden Digitalisierung verlangt die zunehmende Kooperation unter den einzelnen Unternehmen im Fitness- und Gesundheitsbereich eine offene und flexible Software-Architektur, mit der künftige Bedürfnisse schnell bedient werden können. Deshalb wurde die Konzeption und Umsetzung von *FIVE flow* beauftragt, um damit alle Stamm- und im Trainingsbetrieb erhobenen Bewegungsdaten zu verwalten und zu analysieren (der Anhang A.12 liefert mit Abbildung 9.12 und 9.13 entsprechende Screenshots des explorativen UI-Prototypen). Im Projekt *FIVE flow* soll der domänengetriebene Entwurfsansatz erprobt werden, der u. a. eine Verteilung auf mehrere Entwicklungsteams für Frontend- und Backend-Entwicklung durch eine Web-API-Spezifikation ermöglicht. Die zu entwickelnden domänengetriebenen Microservices sind unabhängig von einer konkreten Anwendung und bilden die Domäne von FIVE ab, die in der CTS-Domäne verortet ist. Neben *FIVE touch* können in der Zukunft noch weitere Anwendungen angebunden werden, die wiederum über ein anwendungsspezifisches Gateway verfügen. Daneben soll *FIVE flow* die Möglichkeit bieten, mit anderen Systemen zu kooperieren wie bspw. *milon CARE*.

Abbildung 7.10 liefert eine Übersicht über die zu entwickelnden Microservices, die durch die einzelnen

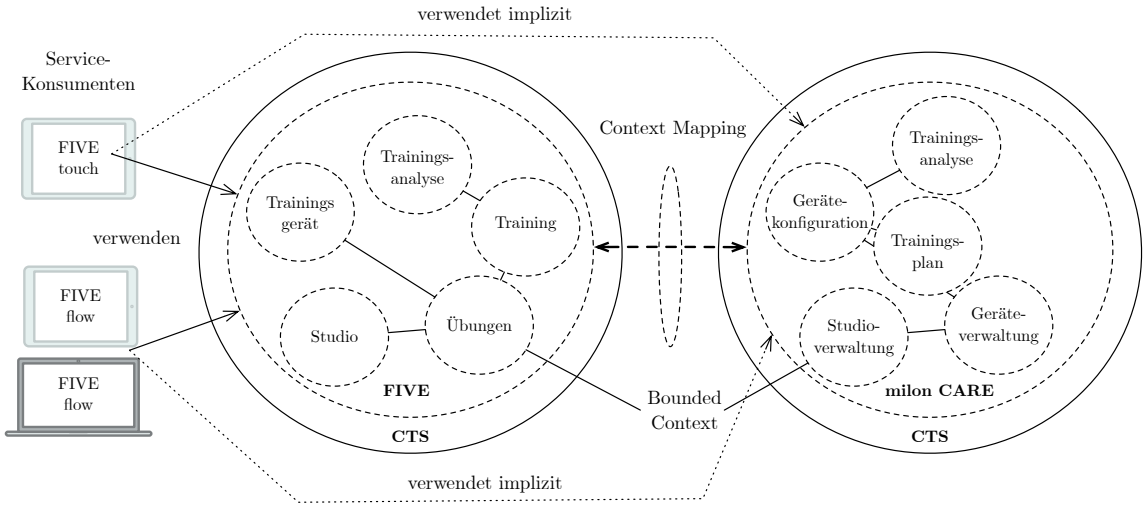


Abbildung 7.10: Ermittelte Bounded Contexts innerhalb der FIVE-Domäne

Bounded Contexts repräsentiert sind. Der Schnitt der FIVE-Domäne in einzelne Microservices erfolgte unter Anleitung des Autors dieser Arbeit.

Bisheriges und letztliches Vorgehen

Gegenwärtig fehlt für den Entwurf einer Microservice-Architektur auf Grundlage eines Domänenmodells ein einheitliches und nachvollziehbares Vorgehen, um die Integration von Service-Konsumenten zu erleichtern und außerdem die bestehende Integration langfristig im Hinblick auf die Evolution des zu entwickelnden Systems und die Domäne nicht zu gefährden. So müssen die entwickelten Microservices bspw. auch gewährleisten, dass die Interoperabilität mit der aktuellen Version von *FIVE touch* zu jederzeit sichergestellt ist. Die einzelnen Microservices im Kontext eines Minimum Viable Product (MVP) werden von mehreren Software-Entwicklern und -Architekten in einem viermonatigen Projekt unter Anwendung des domänengetriebenen Entwurfsansatzes entwickelt. Auf Ende März 2018 ist ein erster produktiver Test mit beschränktem Nutzerkreis und unter realen Bedingungen geplant.

Domänenmodell und Microservice-Architektur

Auf Darstellung der modellierten Bounded Contexts des Domänenmodells wird hier verzichtet, da dies nicht zur Validierung beiträgt. Stattdessen fokussiert dieser Abschnitt die letztliche Microservice-Architektur – das Resultat des angewandten domänengetriebenen Entwurfsprozesses (vgl. Abbildung 7.11). Wie der Service-Schnitt (vgl. Abbildung 7.10) zeigt, wurden fünf ressourcenorientierte Microservices entworfen. Vier davon sind geschäftsrelevante Microservices, die den Geschäftsausschnitt in geeigneter Form abbilden. Der geschäftsagnostische Microservice *Authentifizierer* dient

der Authentifizierung und Autorisierung der entsprechenden Service-Konsumenten. Die übrigen geschäftsagnostischen Microservices werden durch eine entsprechende PaaS bereitgestellt, sind unabhängig von einer konkreten Domäne und erfordern daher keiner separaten Entwicklung. Die erfolgreiche Authentifizierung resultiert in einem Token mit Informationen zum Nutzer und zu dessen Rechten, sogenannten Gültigkeitsbereichen (engl. scopes). Zudem existieren zwei anwendungsspezifische Gateways, die den Zugriff auf die einzelnen Microservices für die Service-Konsumenten *FIVE touch* und *FIVE flow* kapseln und eine anwendungsspezifische Web-API veräußern. So benötigt bspw. *FIVE flow* eine Möglichkeit zum Anlegen von neuen Übungen, *FIVE touch* hingegen lediglich die dem jeweiligen Training zugewiesenen Übungen. Nach Überarbeitung von *FIVE touch* können beide

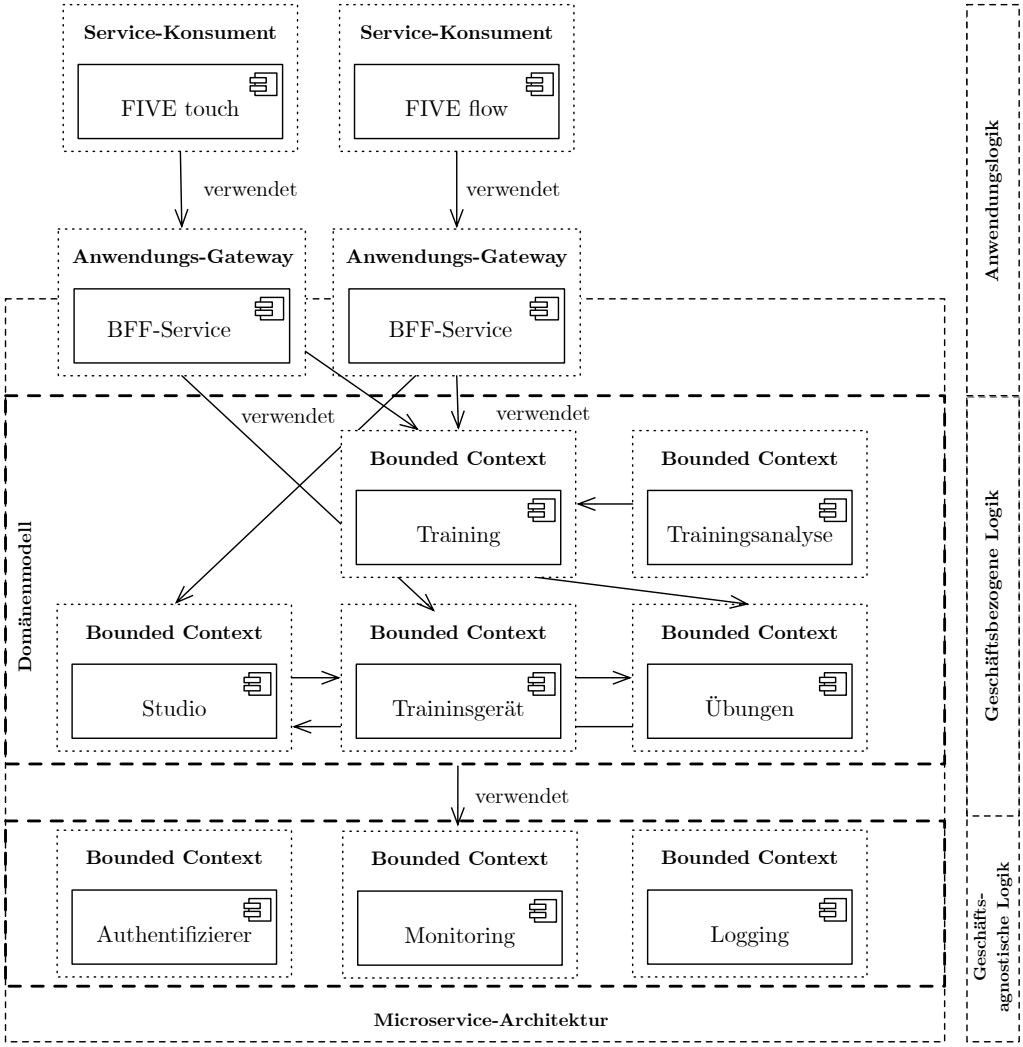


Abbildung 7.11: Microservice-Architektur für das Unternehmen FIVE insbesondere der Service-Konsumenten *FIVE flow* und *FIVE touch*

Anwendungs-Gateways in geeigneter Form zusammengeführt werden und die erlaubten Operationen des jeweiligen Service-Konsumenten über Gültigkeitsbereiche geregelt werden. Die Authentifizierung erfolgt aktuell im jeweiligen Anwendungs-Gateway über eine geteilte Bibliothek, da ein zentrales Gateway im Sinne eines API-Managementsystems gegenwärtig nicht zur Verfügung steht. Die gesamte Microservice-Architektur kann als hexagonale Architektur angesehen werden.

Bewertung der Anwendbarkeit und resultierende Vorzüge

Für die Bewertung der Anwendbarkeit des domänengetriebenen Entwurfs von ressourcenorientierten Microservices wurden sieben Software-Entwickler, drei Software-Architekten, ein technischer Produktleiter sowie der zuständige Produkt Owner per Online-Umfrage befragt, die sie anonym und in einem Zeitraum von zwei Wochen ausfüllen konnten, bevor die Umfrage inaktiv geschaltet wurde. Die Antwortquote betrug $\approx 81\%$. Ein Teilnehmer wurde bei der anschließenden Auswertung nicht berücksichtigt, da er die Umfrage nicht vollständig ausgefüllt hat. Die Befragten waren Projektteilnehmer für die Entwicklung von *FIVE flow* und gehören damit zur Zielgruppe dieser Arbeit. Tabelle 7.7 repräsentiert dabei die befragten Teilnehmer, aufgeschlüsselt nach ihren Berufsjahren und ihrer Rolle im Unternehmen. Außerdem sollten sie ihr Wissen zu den Themen *DDD*, *REST* und *Microservices* einstufen. Abbildung 7.12 zeigt das Ergebnis. Auch diese Resultate sind, wie in Abschnitt 7.6.1, mit Vorsicht zu betrachten, da sie lediglich subjektive Einschätzungen der Individuen wiedergeben. Insgesamt fühlen sich die Teilnehmer vertraut mit den Bereichen. Die nächsten Abschnitte beschreiben den Bewertungsbogen analog zum Procedere in Abschnitt 7.6.1.

Rolle	#Anzahl
Software-Entwickler (Junior)	#4
Software-Entwickler (Senior)	#3
Software-Architekt (Junior)	#0
Software-Architekt (Senior)	#3
DevOp	#0
Andere	#2

Berufsjahre	#Anzahl
Weniger als 1 Jahr	#2
1 -3 Jahre	#3
3-5 Jahre	#1
5 - 8 Jahre	#2
8 - 10 Jahre	#1
Über 10 Jahre	#3

Tabelle 7.7: Übersicht über die Teilnehmer

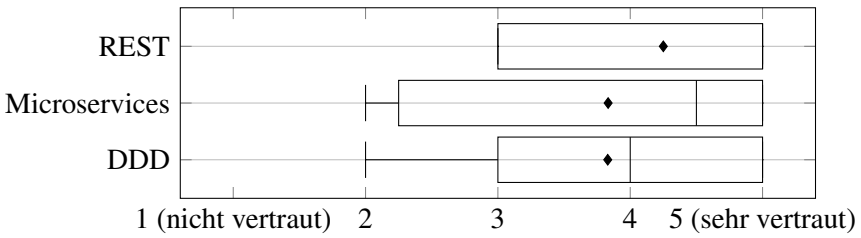


Abbildung 7.12: Subjektive Einstufung der Teilnehmer in Bezug auf ihr Wissen

Modellierung der Bounded Contexts: Für die Bewertung der Modellierung der Bounded Contexts wurde erst geprüft, inwieweit die Teilnehmer sich zutrauen, eine Modellierung selbstständig ohne Hilfe durchzuführen. Es glauben $\approx 87\%$ der Teilnehmer, unter diesen Bedingungen einen Geschäftsausschnitt modellieren zu können. Ein Beleg hierfür ist die durchgehend positive Bewertung einzelner Aspekte der Beiträge dieser Arbeit im Hinblick auf Verständlichkeit, Nachvollziehbarkeit und praktische Anwendbarkeit (vgl. Abbildung 7.13). Die Betrachtung der Technologie-Agnostizität sollte bestätigen, dass technologische Entscheidungen zum Zeitpunkt der Modellierung nicht erforderlich sind. Die Voraussetzung für den darauf aufbauenden Entwurfsprozess ist somit geschaffen, der u. a. die Inzidenz von Entwurfsfehlern verringern und ressourcenorientierte Microservices mit hoher Wiederverwendbarkeit bringen soll. Die Wiederverwendbarkeit korreliert hier positiv mit dem Integrationsaufwand auf Seiten des jeweiligen Service-Konsumenten (vgl. Abbildung 2.5.

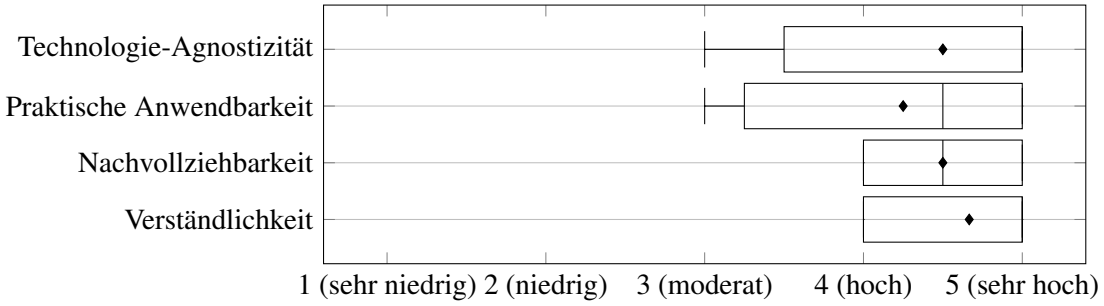


Abbildung 7.13: Ergebnis zur Bewertung der Modellierung

Es wurden keine konkreten Verbesserungsmöglichkeiten genannt. Deshalb ist anzunehmen, dass der Aufwand bei der Modellierung durch den so generierten Nutzen aufgewogen wird.

Überführung in ein Ressourcenmodell: Nachdem der vorige Abschnitt die Modellierung des Domänenmodells bzw. der Bounded Contexts fokussiert hat, widmet sich dieser deren Überführung in jeweils ein Ressourcenmodell. Eine mit Bedacht für das Unternehmen entworfene Web-API gilt $\approx 92\%$ der Teilnehmer als *sehr wichtig* und $\approx 8\%$ (1 Teilnehmer) als *wichtig*. Trotzdem fehlte zu diesem Zeitpunkt ein nachvollziehbares, systematisches Vorgehen. Stattdessen wurde eine Web-API ähnlich wie im Szenario bei der milon Care GmbH (vgl. Abschnitt 7.6.1) bei Bedarf für Anwendungsfälle entworfen und entwickelt. Zudem wurden die Web-APIs oft direkt mit zugrundeliegenden Datenbankentitäten verknüpft. Jede Änderung an den Schemata der Datenbank veränderte dann die Web-API, was wiederum eine Anpassung der Service-Konsumenten erforderlich machte und so die Interoperabilität gefährdete.

Die Bewertung des Vorgehens bei der Überführung zeigt Abbildung 7.14. Die Gesamtheit der Bewertungen wurden durch Anwendung des arithmetischen Mittels mit $\approx 4,45$ berechnet. Der Entwurfspro-

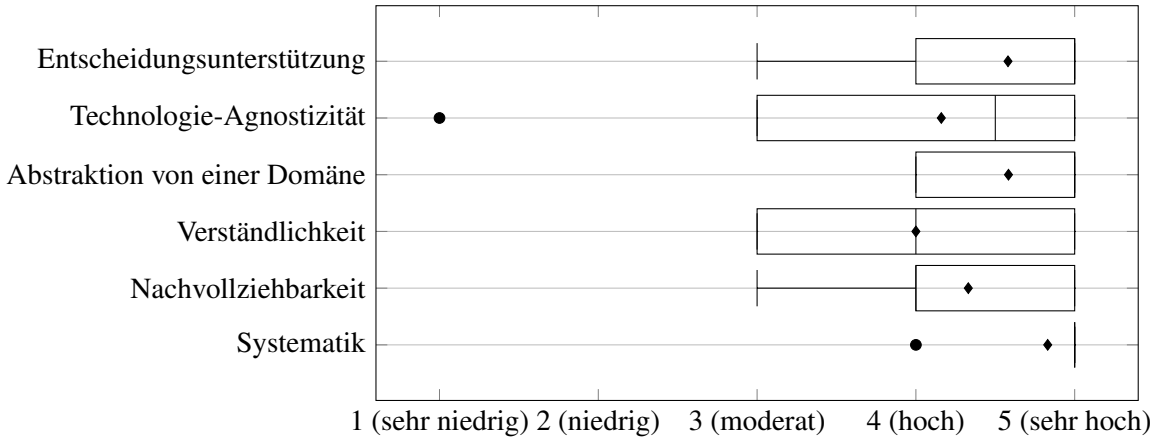


Abbildung 7.14: Ergebnis zur Bewertung der Überführung in ein Ressourcenmodell

zess wird somit als *gut* bis *sehr gut* beurteilt. Nach Ansicht der Teilnehmer haben die verankerten Muster im Entwurfsprozess Entwurfsentscheidungen nachvollziehbarer, zielgerichteter und schneller gemacht, was u. a. ein konsistenteres Gesamtbild der Systemlandschaft ergab. Diese Aussage deckt sich wiederum mit dem Ergebnis der Reduktion von Review-Zyklen. So sahen $\approx 92\%$ der Teilnehmer deren Anzahl durch Anwendung des Entwurfsprozesses verringert, was eine Reduktion von Entwurfsfehlern impliziert (vgl. Abbildung 7.15). Die Konsistenz und der Bezug zur jeweiligen Domäne reduzierten auch den Abstimmungsaufwand unter den Projektbeteiligten im Vergleich mit früheren Projekten dank geteilten Wissens (vgl. Ubiquitous Language in Abschnitt 2.2.2). Durch den Entwurf der Web-API und die Ableitung der Web-API-Spezifikation vor der Implementierung konnte der entsprechende Service-Konsument *FIVE touch* deutlich schneller entwickelt bzw. angepasst werden. Das lag auch daran, dass allenfalls noch geringfügige Änderungen an der Web-API-Spezifikation bei der Entwicklung erforderlich waren und der Abstimmungsaufwand zwischen den Entwicklungsteams minimiert wurde (vgl. Abbildung 7.15).

Bislang fehlte eine Web-API-Spezifikation im Unternehmen. Lediglich nicht formale Beschreibungen einer Web-API waren verfügbar. Künftig sollen die resultierenden Web-API-Spezifikationen für unterschiedliche Anwendungszwecke nutzbar sein. Dabei wurden vor allem deren Nutzung als interaktive und aktuelle Dokumentation ($91,7\%$), effizientere Kommunikation zwischen den Entwicklungsteams ($91,7\%$) sowie die Auffindbarkeit im Kontext von API-Management (75%) genannt. Die automatische Ableitung und Generierung von Testfällen soll später untersucht und so die vertragliche Dienstleistung eines oder mehrerer ressourcenorientierter Microservices automatisch zum Zeitpunkt der Implementierung getestet werden. Abschließend sei erwähnt, dass auch die Teilnehmer dieser Studie die praktische Relevanz der Arbeit als *sehr hoch* ($83,3\%$) oder *hoch* ($16,7\%$) bewerteten. Es werden daher vermutlich die Beiträge auch zukünftig verwendet werden.

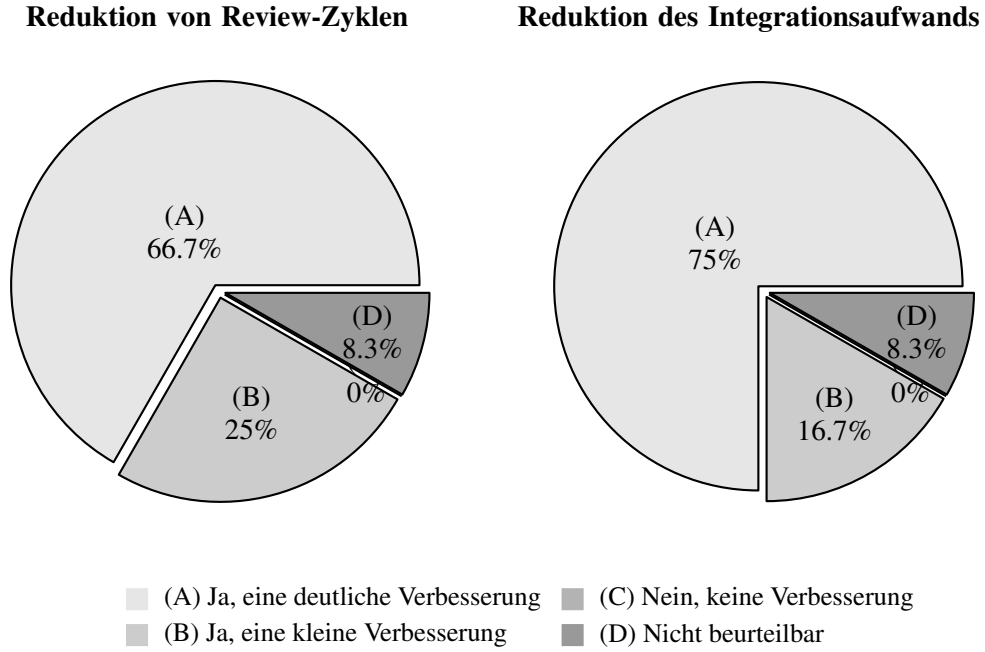


Abbildung 7.15: Ergebnisse zur Untersuchung der Reduktion von Entwurfsfehlern und Integrationsaufwand

7.6.3 Studie 3: Entwurf und Entwicklung einer Anwendung zur Erfassung von Störungen in der Produktion

Diese Studie wurde im Rahmen der digitalen Transformation eines Prozessablaufs in der Produktion eines Automobilkonzerns durchgeführt. Die Sperrklausel verbietet die Nennung von Namen oder die Beschreibung technischer Details. Diese Beschreibungen abstrahieren also von der konkreten Umsetzung. Die durchgeführte Studie adressiert die Tragfähigkeit im Rahmen eines Industrieprojekts.

Gegebenes Szenario

Zur Dokumentation von Störungen im intralogistischen Prozessablauf der Produktionswerke eines Automobilkonzerns soll eine Lösung im Zuge der digitalen Transformation auf Basis einer Microservice-Architektur geschaffen werden. Zu jeder detektierten Störung im Prozessablauf sollen neben der Ursache auch deren Auswirkungen und Maßnahmen zur Beseitigung der Störung festgehalten und zudem für die erfassten Störungen Auswertungen bereitgestellt werden, sodass sich die Quellen von beobachteten Fehler beseitigen lassen. Gleichzeitig sollen bei einer Störung weitere Geschäftsprozesse angesteuert werden, damit davon betroffene Abteilungen sofort informiert werden und somit die Störung schnellstmöglich behoben werden kann. Für die digitale Lösung soll eine Anwendung für mobile Endgeräte und für ein Desktop-System entworfen und entwickelt werden, die über eine ressourcenorientierte Web-API mit dem zuständigen Microservice kommuniziert. Während

zunächst einzelne Werke diese Anwendung im produktiven Betrieb erproben, ist eine weltweite Verteilung in allen Produktionswerken für 2018 anvisiert.

Bisheriges und letztliches Vorgehen

Zum Zeitpunkt von Entwurf und Entwicklung der Anwendung fehlte ein systematisches und nachvollziehbares Vorgehen. Deshalb wurde der domänengetriebene Entwurfsprozess im Unternehmen vorgestellt und dort schließlich vom Projektverantwortlichen genehmigt. Zur Ergänzung des domänengetriebenen Entwurfsansatzes wurde ein explorativer Prototyp erstellt (vgl. Abbildung 4.2), der den Endanwendern früh einen Eindruck über die letztliche Anwendung gibt. Zudem wurde in Kooperation mit dem Fachbereich das Domänenmodell verifiziert und eine Begriffssammlung für eine Ubiquitous Language erstellt. Die Umsetzung von Service-Konsument und Microservice erfolgte anhand der Web-API-Spezifikation mit OAI in der Version 2. Für die Umsetzung des Service-Konsumenten wurde die entworfene Web-API-Spezifikation dem verantwortlichen Entwicklungsteam übergeben, die zu Beginn einen Mock-Service[8] generierten, bevor die erforderlichen Funktionalitäten durch den Microservice schließlich verfügbar waren. Die Bereitstellung dieser Funktionalitäten erfolgte iterativ und inkrementell als agiles Entwicklungsvorgehen in fünf Sprints, wobei ein Sprint zwei Wochen dauerte.

Modellierter Bounded Context und resultierende Web-API-Spezifikation

Der modellierte Bounded Context kann wegen der Sperrklausel nicht detailliert gezeigt werden. Ein Auszug der resultierenden Web-API-Spezifikation ist Abbildung 7.16 zu entnehmen, die vom zugrundeliegenden modellierten Bounded Context abstrahiert. Da die Web-API-Spezifikation vom Microservice über einen dedizierten Endpunkt veräußert wird, lässt sie sich durch einen bestehenden geschäftsagnostischen Microservice mittels der Bibliothek *Swagger UI* visualisieren (vgl. Abbildung 7.17). Bei dem Entwurf der Web-API-Spezifikation wurden u. a. die Muster P_{RCT}, P_{LRR} und P_{RCF} als Bestandteile der Mustersprache genutzt. Die letztliche Umsetzung dieser Muster folgt den Nutzungsanforderungen, die sich aus den Anforderungen der zu realisierenden Anwendung ergeben, sowie dem Anwendungsschichtprotokoll HTTP ausgerichtet.

Service-Konsument und letztlicher Microservice

9.14 und 9.15 in Anhang A.12 verschaffen einen Eindruck vom entwickelten Service-Konsumenten. Informationen wurden, wie die Sperrklausel verlangt, unkenntlich gemacht. Ein Auszug der letztlichen Architektur des entwickelten Microservice ist Abbildung 7.17 zu entnehmen. Sie zeigt, dass der Microservice über vier Adapter verfügt. Drei von ihnen ermöglichen die Kommunikation mit

[8] Bei einem Mock-Service handelt es sich um eine Form der Attrappe, welche den eigentlichen Microservice in einem kontrollierten Umfeld mittels Beispieldaten nachahmt.

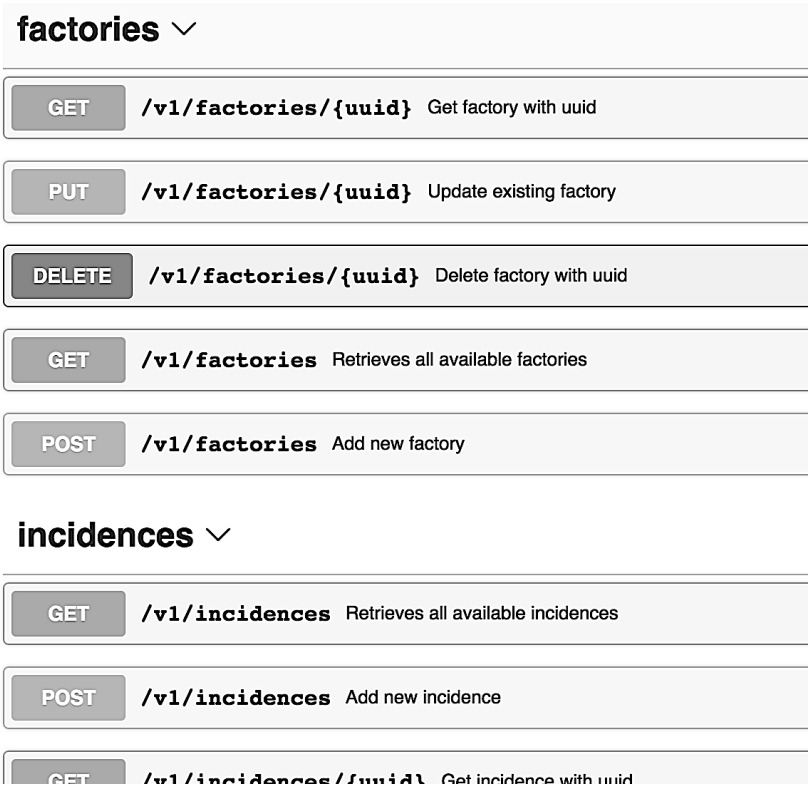


Abbildung 7.16: Zur Laufzeit visualisierte Web-API-Spezifikation mittels *Swagger UI* über dedizierten Endpunkt des Microservice

Drittsystemen, um die geforderten Nutzeranforderungen umsetzen zu können. Kern des Microservice ist die Domäne auf Grundlage des modellierten Bounded Context mit entsprechenden Invarianten sowie Vor- und Nachbedingungen. So wird bspw. vor jeder durchzuführenden Operation geprüft, ob der jeweilige Nutzer des Service-Konsumenten die notwendigen Rechte zur Ausführung hat. Die Behandlung der Rechte ist auf der Schicht der Anwendungsservices verankert. Zudem werden vor und nach jeder Änderung am Datenbestand die Integritätsbedingungen geprüft, die sich aus den Invarianten des modellierten Bounded Context ergeben. Auf ein anwendungsspezifisches Gateway wurde hier verzichtet, da dieses künftig durch ein API-Managementsystem bereitgestellt werden soll und eine Notwendigkeit für dieses Projekt gegenwärtig nicht bestand.

Bewertung der Anwendbarkeit und resultierende Vorzüge

Anders als bei den bisher präsentierten Studien unterblieb hier eine Bewertung per Fragebogen, da die Software-Entwickler vom Autor dieser Arbeit angeleitet wurden. Eine Befragung würde hier das Risiko erhöhen, dass die Ergebnisse des Bewertungsbogens das Gesamtergebnis der Studie verfälschen (vgl. Abschnitt 7.5.1). Doch konnte durch die erfolgreiche Umsetzung dieser Anwendung und Inbetriebnahme immerhin die Anwendbarkeit des Entwurfsprozesses demonstriert werden. Zudem

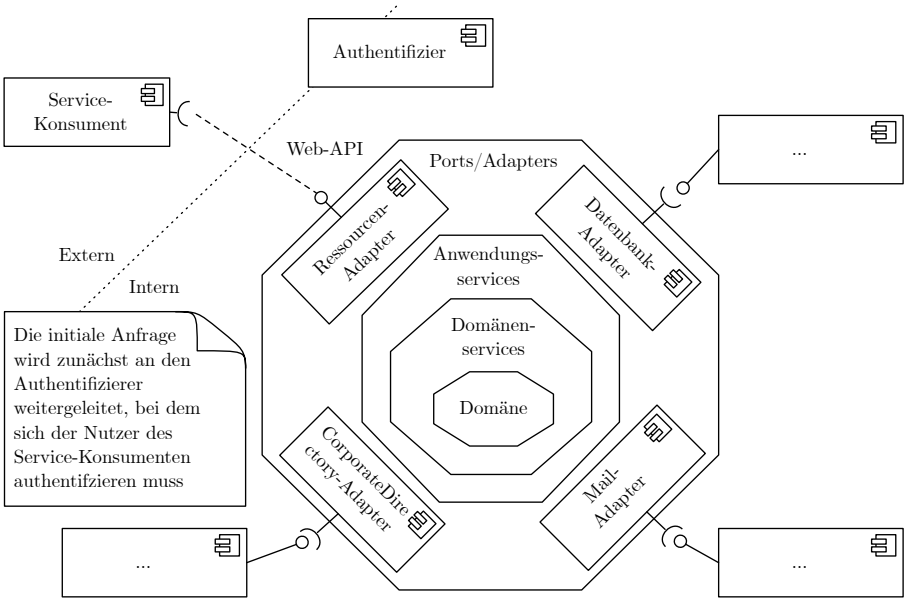


Abbildung 7.17: Übersichtliche Darstellung der Architektur des Microservice zur Erfassung von Störungen in der Produktion

wird gegenwärtig (02/2018) eine weitere auf diesen Entwurfsprozess gestützte Anwendung in der Entwicklungsabteilung des Unternehmens eingesetzt, was die Vorzüge dieser Arbeit bestätigt.

7.6.4 Bedrohung der Validität und Einschränkung der Validierung

Vergleichbar mit Abschnitt 7.5.1 wird auch hier die Validität nach den verschiedenen Arten (vgl. Abschnitt 7.3) beleuchtet und deren Bedrohungen skizziert.

1. **Konstruktvalidität:** Sämtliche Teilnehmer der Studien können abhängig von ihren Berufsjahren und ihren praktischen Erfahrungen unterschiedliches Wissen haben, das die Beantwortung der einzelnen Fragen beeinflusst. Es wurden Software-Entwickler und -Architekten mit unterschiedlichen Berufsjahren befragt, wodurch sich etwaige Ausreißer eliminieren lassen. Doch existiert das Risiko, dass Teilnehmer das gewünschte Ergebnis der Befragung erraten und ihre Antworten dahingehend ausrichten. Das Ergebnis soll dem jeweiligen Unternehmen verraten, ob der domänengetriebene Entwurfsprozess ihm nützt und bei künftigen Projekte eingesetzt werden sollte. Deshalb werden sich die Teilnehmer vermutlich nicht vom gewünschten Ergebnis der Studie beeinflussen lassen. Zudem ist die Umfrage anonym. Es werden keine Daten erhoben, die eine Identifizierung der Teilnehmer erlaubt. Diese Arbeit enthält drei voneinander unabhängige Studien. Jeder Teilnehmer hat genau an einer Studie teilgenommen, da sonst das Risiko für falsche Schlussfolgerungen und falsche Bewertungen steigen würde: „Then

you cannot conclude whether the effect is due to either of the treatments or of a combination of treatments“ [WR+12, S. 109]. Abschließend sei erwähnt, dass die Bewertungsbögen unterschiedliche Fragen und Fragetypen für eine Messung enthielten, um die Antworten der Teilnehmer zu verifizieren. Deshalb waren mehrere Fragen für eine Messung erforderlich, wodurch der Aufwand zur Beantwortung stieg. Um ihn zu beschränken, wurden die Bewertungsbögen so konzipiert, dass sich jede Version in 20 bis 25 Minuten beantworten ließ. Dieser Zeitraum ergab sich durch interne Tests mit vier unterschiedlichen Probanden.

2. **Interne Validität:** Die Studien wurden zu unterschiedlichen Zeiten und über einen längeren Zeitraum hinweg durchgeführt, weshalb hier kausale Abhängigkeiten nicht erkennbar sind. Jede einzelne Studie wurde mit den Budgets zum jeweiligen Projekt ohne Kontrollgruppe finanziert. Aus diesem Grund müssen insbesondere die Bedrohungen einer einzelnen Gruppe (engl. single group threats, siehe [WR+12, S. 106]) betrachtet werden. An den Studien haben u. a. Software-Entwickler und -Architekten mit unterschiedlichen Motivationen teilgenommen, die in ihren Unternehmen am Entwurf und der Entwicklung von Microservices beteiligt sind. Möglicherweise begegnet einer dieser Akteure wegen seines Alters einem neuen Entwurfsprozess wenig aufgeschlossen. Seine Bewertung wäre auf jeden Fall als Ausreißer bei der Auswertung erkennbar. Derartiges sollte eine Auswertung stets berücksichtigen bzw. hinterfragen. Die Bewertung des domänengetriebenen Entwurfsprozesses erfolgte nach Beendigung des jeweiligen Projekts, sodass die Teilnehmer weder negativ noch positiv beeinflusst wurden, wie bspw. als Folge der Müdigkeit bei der Durchführung der jeweiligen Studie [WR+12]. Der Bewertungsbogen enthielt Pflicht- und optionale Fragen. Wenn nicht alle Pflichtfelder ausgefüllt waren, wurde die Teilnahme aus dem Endergebnis der jeweiligen Studie als Abbruch entfernt, sodass nur vollständige Bewertungen ausgewertet wurden.

3. **Externe Validität:** Da die Studien jeweils in einem Industriekontext und in unterschiedlichen Domänen durchgeführt wurden, lässt sich externe Validität stark vermuten. Die Studien betrachteten stets einen tatsächlichen Bedarf des Unternehmens nach informationstechnischer Unterstützung mittels Microservices.

4. **Ergebnisvalidität:** In zwei Studien wurden statische Erhebungen in verschiedenen Domänen mit einer Teststärke von insgesamt 23 Teilnehmern durchgeführt, die repräsentativ ist. In der dritten Studie (im Automobilkonzern) wurden zwar keine statistische Daten erhoben, aber zumindest zeigt sich eine Tendenz durch die Fortführung des Entwurfsprozesses in einem weiteren Projekt. Obgleich die Teilnehmergruppen dank individueller Erfahrungen recht heterogen waren, was das Ergebnis der Studie beeinflussen kann, wurden diese Akteure für die Überprüfung der externen Validität gewählt. Eine homogenere Teilnehmergruppe würde zwar dieses Risiko senken, gleichzeitig aber die externe Validität reduzieren [WR+12]. Des Weiteren kann die Formulierung der Fragen das Ergebnis der Studie verfälschen, wenn sie zu Fehlinterpretationen führen. Um das Risiko für die Bedrohung der Validität weiter zu reduzieren, wurde der

Bewertungsbogen im Vorfeld zwei Probanden zum Test vorgelegt und deren Rückmeldungen eingearbeitet.

7.6.5 Zusammenfassung der Ergebnisse

Werden die Ergebnisse der Studien zusammengefasst, zeigt sich, dass der präsentierte domänengetriebene Entwurf von ressourcenorientierten Microservices mit den Vorzügen dieser Arbeit (vgl. Tabelle 7.4) positiv korreliert. Die zu validierenden Beiträge gemäß Tabelle 7.1 auf Seite 167 wurden mittels unterschiedlicher Fragestellungen überprüft. Um die Reduktion von Entwurfsfehlern zu validieren, wurde geprüft, ob dieser domänengetriebene Entwurfsprozess verständlich und nachvollziehbar ist, sich systematisch und praktisch anwenden lässt und hinlänglich von konkreten Domänen und Technologien abstrahiert. Es wurden keine schwerwiegenden Aussagen von den Teilnehmern gemacht, die auf nicht betrachtete Randfälle oder größere Probleme bei der Anwendung des domänengetriebenen Entwurfsprozesses hindeuteten. Durch die Reduktion von etwaigen Review-Zyklen der Web-API konnte schließlich gezeigt werden, dass die zugrundeliegenden Entwurfsartefakte höhere Qualität haben. Da ein Entwurf in der Regel einige Entwurfsentscheidungen im Hinblick auf verschiedene Qualitätsteilmerkmale der Wiederverwendbarkeit verlangt, muss ein Software-Architekt diese Entscheidungen mit fundiertem Wissen nachvollziehbar treffen können. Hierzu wurde geprüft, ob die verankerten Muster im domänengetriebenen Entwurfsprozess ihn angemessen unterstützen und seine Entscheidungen vereinfachen. Dies haben die Teilnehmer bei Bewertung der Nützlichkeit der Muster sowie der Entscheidungsunterstützung beim domänengetriebenen Entwurfsprozess mehrheitlich bejaht. Somit bestätigen die Umfragen auch diesen Vorzug. Die Verbesserung der Wiederverwendbarkeit ist durch die Berücksichtigung verschiedener Qualitätsaspekte im gesamten Entwurfsprozess bereits gegeben. Dennoch sollte geprüft werden, ob Service-Konsumenten dadurch schneller entwickelt werden können, was zumindest ein Hinweis für weitere Verbesserung der Wiederverwendbarkeit wäre. Die meisten Teilnehmer sind überzeugt, dass sich Service-Konsumenten u. a. durch geringere Abstimmungsaufwände und die Berücksichtigung der Qualitätsaspekte schneller und effizienter entwickeln lassen. Auch wären so neue Anwendungsfälle flexibel umsetzbar, die zu Beginn noch nicht erkennbar waren.

7.7 Zusammenfassung der Validierung

Zur Validierung wurden die Beiträge dieser Arbeit und ihre Vorzüge mit unterschiedlichen empirischen Validierungsarten bewertet: Typ 0-Validierung (Machbarkeit), die Typ I-Validierung (Eignung) und die Typ II-Validierung (Anwendbarkeit). Auf die Typ III-Validierung (Kosten/Nutzen) wurde wegen hohen Aufwands hier verzichtet und lediglich ein Ausblick gegeben. Es wurden zudem Bedrohungen und Einschränkungen der Validität aufgeführt, um damit die gewonnenen Erkenntnisse kritisch zu diskutieren.

Für die Typ 0-Validierung wurde ein Vergleich zu bestehenden und vergleichbaren Ansätzen herangezogen und damit ein Bezug zu dem Stand der Forschung hergestellt, wobei sich an dem Anforderungskatalog aus Abschnitt 3.1.2 bedient wurde. Das Ergebnis der Evaluation zeigte, dass die erzielten Beiträge die Lücken bestehender Ansätze füllen und dahingehend einen entsprechenden Lösungsansatz bereitstellen.

Danach wurde die Typ I-Validierung betrachtet, um die Eignung des Entwurfsprozesses an einem realitätsnahen Beispiel zu belegen. Die Demonstration hat diese Arbeit insbesondere durch die Kapitel 4 bis 6 schon geliefert. Deshalb wurden diese Aspekte nicht weiter ausgeführt. Zudem wurden über drei Semester Experimente mit Studierenden als Probanden durchgeführt. Auch dieses Verfahren lässt sich der Typ I-Validierung zuordnen. Diese Experimente sollten erkennen lassen, inwieweit die Beiträge die notwendigen Review-Zyklen eines Entwurfs reduzieren und etwaige Entwurfsentscheidungen unterstützen können. Obgleich die Bedrohung der Validität durch zahlreiche Faktoren gegeben ist und als relativ hoch einzustufen ist, konnte dennoch ein klarer Trend erkannt werden.

Am Schluss des Kapitels wurden drei Studien im industriellen Kontext für die Typ II-Validierung vorgestellt. Diese Studien wurden in drei getrennt operierenden Unternehmen durchgeführt, die sämtlich im Zuge einer digitalen Transformation sowie der Modernisierung der bestehenden Informationssysteme erfolgten. Die Validierung basierte stets auf der Anwendung des Entwurfsprozesses sowie in zwei Fällen mit zusätzlichen Bewertungsbögen, worin die betreffenden Teilnehmer indirekt die Beiträge dieser Arbeit anonym bewerten konnten. Die Ergebnisse zeigten, dass sich die Beiträge dieser Arbeit in der Praxis bewährt haben, der Microservice hohe Wiederverwendbarkeit besitzt und der Entwurfsprozess systematisch und nachvollziehbar aufgebaut ist. Außerdem ließen sich die Review-Zyklen des Entwurfs reduzieren und die Integration einer Web-API durch die Entwickler von Service-Konsumenten durch die Berücksichtigung von Qualitätsaspekten vereinfachen. Schließlich zeigten sich weitere Vorzüge im Vergleich mit früheren Projekten der Unternehmen. Hier wurden insbesondere die Kostenreduktion durch eine systematische und qualitätssichernde Unterstützung des Entwurfsprozesses von Microservices sowie geringere Produkteinführungszeiten (Time to Market (TTM)) durch Parallelisierung dank einer Web-API-Spezifikation und durch gesunkenen Abstimmungsaufwand genannt. Diese Erkenntnisse wurden allerdings erst nach Durchführung der Studien ersichtlich und sind noch in geeigneter Form zu validieren.

8 Fazit und Ausblick

Das Schlusskapitel dieser Arbeit fasst die Beiträge und die Vorzüge des Entwurfsansatzes zusammen. Am Ende steht ein Ausblick auf weitere Fragestellungen im Kontext des domänengetriebenen Entwurfs von ressourcenorientierten Microservices, die zeigen, wie sich diese Arbeit fortführen lässt.

8.1 Zusammenfassung

Heutige Informationssysteme unterstützen Unternehmen immer häufiger bei ihren geschäftlichen Aktivitäten, um die alltäglichen Aufgaben effektiv und effizient bewältigen zu können. Dies bedingt jedoch, dass sich die Informationssysteme immer schneller an die Bedürfnisse eines Unternehmens ausrichten müssen. Um diesen Bedürfnissen gerecht zu werden, entwickelte sich ein neuer Ansatz mittels Microservices mit denen fachliche Anforderungen im Vergleich zum ursprünglichen Ansatz einer SOA schneller umgesetzt werden können. In diesem Zuge wurde auch dem domänengetriebenen Ansatz nach Evans [Ev03] zur Strukturierung und Modellierung einer Domäne eine hohe Bedeutung zuteil. Für die Bereitstellung der immateriellen Dienstleistung eines Microservice zeigt sich eine Web-API basierend auf REST zunehmender Beliebtheit. Dies bedingt allerdings wiederum, dass die Web-API derart geartet sein muss, sodass diese bspw. eine einfache Integration mit minimalem Aufwand auf Seiten des Service-Konsumenten ermöglicht. Diese Arbeit liefert mit der Modellierung einer Domäne nach dem Ansatz von Evans, der Überführung in ein Ressourcenmodell unter Berücksichtigung von Qualitätsaspekten sowie der Abbildung auf die Implementierungsebene die entsprechenden Beiträge. Die Validierung dieser Beiträge und ihrer Vorzüge erfolgte durch Vergleich mit bestehenden Ansätzen, der Anwendung in einem praxisnahen Beispiel, durch ein kontrolliertes Experiment und durch drei Studien im industriellen Kontext.

8.1.1 Beiträge dieser Arbeit

Im Kontext des im Vorfeld beschriebenen Rahmens dieser Arbeit sowie der betrachteten Arbeiten in Kapitel 3, sind folgende Beiträge entstanden, welche die identifizierten Lücken bestehender Ansätze schließen sollen.

B1: Modellierungsartefakte für den domänengetriebenen Entwurf

Der erste Beitrag dieser Arbeit stellt die Modellierung eines Domänenmodells bzw. der darin enthaltenen Bounded Contexts basierend auf den Konzepten von Evans [Ev03] und Vernon [Ve13] dar. Zur Modellierung eines Bounded Context wurde sich für UML trotz der damit einhergehenden Komplexität als Folge dessen Meta-Modell [OMG-UML2a, OMG-UML2b] entschieden. Diese Entscheidung begründete sich durch die weite Verbreitung und die vorherrschende Werkzeugunterstützung von UML im Bereich der Software-Entwicklung. Um die Konzepte von DDD schließlich mit der UML zu verknüpfen wurde eine leichtgewichtige Erweiterung vorgenommen, welche in einem UML-Profil mit Namen *DDD* resultierte. Mittels OCL können schließlich etwaige Invarianten definiert und zum Zeitpunkt der Modellierung überprüft werden, welche sich durch die Verwendung der einzelnen Stereotypen des UML-Profils *DDD* ergeben. Bei der Modellierung wurde grundsätzlich zwischen strukturellen und verhaltensspezifischen Aspekten unterschieden, um eine vollständige Sicht und Abbildung eines Bounded Context und damit des entsprechenden Geschäftsausschnitts zu ermöglichen. Die strukturellen Aspekte wurden mithilfe eines UML-Klassendiagramms modelliert, während die verhaltensspezifischen Aspekte durch ein UML-Protokollzustandsdiagramm und ein UML-Aktivitätsdiagramm repräsentiert wurden. Diese Diagrammtypen wurden gewählt, da sie sich besonders eignen, um die jeweiligen Aspekte abzubilden. So ermöglicht bspw. das UML-Protokollzustandsdiagramm die Modellierung von Vor- und Nachbedingungen und entsprechenden Invarianten, woraus sich letztlich die möglichen Fehlerfälle als auch die validen Zustände eines Systems ableiten lassen. Querreferenzierungen erlauben, die verschiedenen Diagrammarten in Beziehung zueinander zu setzen und so das modellierte Verhalten mit einem Domänenobjekt zu verknüpfen. Um den Modellierungsaufwand schließlich zu minimieren, wurde implizites Verhalten definiert, das aus dem Lebenszyklus eines Domänenobjekts nach Evans abgeleitet wurde.

B2: Qualitätsgestützter Entwurfsprozess von ressourcenorientierten Microservices

Dieser Beitrag widmet sich dem Entwurf eines ressourcenorientierten Microservice unter Berücksichtigung von Qualitätsaspekten der Wiederverwendbarkeit basierend auf dem vorangehenden Modellierungsartefakt eines Bounded Context. Um einen systematischen und nachvollziehbaren Entwurf zu ermöglichen, wurde ein Entwurfsprozess entwickelt, der in mehrere Prozessschritte unterteilt ist und der die Verwendung von iterativen und inkrementellen Vorgehensmodellen begünstigt. Hierzu wurden zunächst Ressourcenkandidaten aus dem modellierten Bounded Context identifiziert und diese mit etwaigen Nutzeranforderungen[1] verglichen. Das ergab schließlich die Menge der zu betrachtenden Ressourcen, die mit formulierten Heuristiken und Regeln aus diesem modellierten Bounded Context in ein Ressourcenmodell überführt wurden. Wie schon bei der Modellierung eines Bounded Context wurde wieder ein geeignetes UML-Profil, hier mit dem Namen *ResourceOrientation*, entworfen und so

[1] Die Nutzeranforderungen ergeben sich durch ein konkretes Bedürfnis eines Unternehmens nach informationstechnischer Unterstützung, was zumeist in einer Software-Anwendung resultiert.

UML mit den Konzepten des Architekturstils REST verknüpft. Auf die Überführung folgte die Adressierung der Ressourcen unter Berücksichtigung von Qualitätsaspekten der Wiederverwendbarkeit einer Web-API. Die Manifestierung der Adressierung erfolgte in dem zuvor überführten Ressourcenmodell, sodass dieses Modellierungsartefakt das für die Implementierung benötigte Wissen enthält. Darauf aufbauend wurden schließlich die Interaktionen betrachtet, um auch die verhaltensspezifischen Aspekte des Bounded Context angemessen zu behandeln. Hierfür wurde eine mehrschichtige Mustersprache entworfen, die den Software-Architekten bei seinen Entwurfsentscheidungen unterstützt. Jedes darin enthaltene Muster stellt einen Bezug zu den qualitativen Auswirkungen bzw. etwaigen Qualitätsmerkmalen her. Zudem wurden die Muster technologie-agnostisch formuliert. Den Abschluss des Beitrags bildete die Betrachtung etwaiger Änderungen am Ressourcenmodell, die sich im Lauf der Zeit aus unterschiedlichen Gründen ergeben können. Hierfür wurden zur Sicherstellung der Interoperabilität mit bestehenden Service-Konsumenten ein entsprechendes Vorgehen und ebensolche Muster zur Versionierung aufgestellt.

B3: Überführung des Entwurfs auf die Implementierungsebene

Dieser Beitrag zeigt, wie ausgehend von den Entwurfsartefakten eine systematische Überführung auf die Implementierungsebene bzw. die hexagonale Architektur nach Vernon [Ve13] möglich ist. Die hexagonale Architektur lässt sich dabei als Erweiterung des Musters *Layered Architecture* deuten und stellt den abgebildeten Geschäftsausschnitt in den Mittelpunkt. Bei der Überführung wurde erst die Verknüpfung mit einem Anwendungsschichtprotokoll behandelt und schließlich auf dieser Grundlage eine Web-API-Spezifikation unter Nutzung einer Web-API-Spezifikationssprache abgeleitet. Die resultierende Web-API-Spezifikation formuliert einen Vertrag, wie mit dem ressourcenorientierten Microservice zu interagieren ist, damit sich dessen Funktionalitäten nutzen bzw. wiederverwenden lassen. Alle Interaktionsteilnehmer müssen sich an diesen Vertrag binden, damit im laufenden Betrieb eine reibungslose Interaktion gewährleistet ist. Nachträgliche Änderungen an der Web-API können allerdings in einem ungewollten Mehraufwand resultieren. Um dieses Risiko zu reduzieren, wurde der ganzheitliche Entwurfsprozess um die Dimension der Qualitätssicherung durch einen Review-Prozess ergänzt, bei dem Experten die Web-API untersuchen und Handlungsempfehlungen ausgeben, bevor die Entwurfsartefakte den Entwicklungsteams zur Implementierung übergeben werden. Danach wurde erläutert, wie die einzelnen Entwurfsartefakte auf die Implementierungsebene abgebildet werden können und sich so auf Quellcodeebene widerspiegeln. Um eine ganzheitliche Sicht und Verortung der resultierenden Microservices zu ermöglichen, wurden abschließend die Makroarchitektur und damit die resultierende Microservice-Architektur betrachtet. Hierzu wurden Ebenen zur logischen Gruppierung eingeführt und in einen Gesamtzusammenhang gesetzt.

B4: Validierung im universitären und industriellen Kontext

Der letzte Beitrag dieser Arbeit ist die Validierung des domänengetriebenen Entwurfs von ressourcenorientierten Microservices im universitären und im industriellen Kontext, worin die zuvor erläuterten Beiträge und damit auch deren Vorzüge bestätigt wurden. Die Validierung erfolgte anhand drei unterschiedlicher Validierungsarten sowie unter Betrachtung etwaiger Bedrohungen und Einschränkungen der Validität. Die erste Validierung war ein Vergleich mit ähnlichen Ansätzen in diesem Forschungsbereich, um mit Bezug zu einem hypothetisch angenommenen domänengetriebenen Entwurfsprozess die Vorzüge des vorliegenden hervorzuheben. Die zweite Validierung illustrierte die Eignung des Entwurfsprozesses durch Anwendung an einem praxisnahen Beispiel im universitären Kontext, das durch die Demonstration der Beiträge (B1 bis B3) in dieser Arbeit erbracht wurde. Gleichzeitig wurde dabei ein kontrolliertes Experiment mit Studierenden als Probanden über mehrere Semester durchgeführt, um zu untersuchen, inwieweit die Beiträge Entwurfsfehler verringern und bei Entwurfsentscheidungen unterstützen. Die letzte Validierung basiert auf der Durchführung von Studien im industriellen Kontext, in denen der domänengetriebene Entwurfsprozess erfolgreich demonstriert werden konnte. Zusätzlich konnte ein Bewertungsbogen Rückmeldungen von Teilnehmern in zwei dieser drei Studien erheben. Die Auswertung brachte eine unabhängige Sicht auf den domänengetriebenen Entwurfsprozess. Die Ergebnisse bestätigten seine Vorzüge für den Entwurf von ressourcenorientierten Microservices und damit auch die Beiträge dieser Arbeit.

8.1.2 Resultierende Vorzüge dieser Arbeit

Der nächste Abschnitt fasst die Vorzüge bei Anwendung des domänengetriebenen Entwurfs für ressourcenorientierte Microservices nochmals übersichtlich zusammen.

Reduktion von Entwurfsfehlern

Ein erster Vorzug dieser Arbeit betrifft das systematische und nachvollziehbare Vorgehen durch den domänengetriebenen Entwurfsprozess von ressourcenorientierten Microservices. Dieser wurde so gestaltet, dass eine schrittweise Überführung aus zugrundeliegenden Modellierungsartefakten möglich ist und die resultierenden Entwurfsartefakte ein Entwicklungsteam für die Implementierung adäquat anleiten können. Das gewährleistet eine Durchgängigkeit vom Entwurf bis hin zur Implementierung. Die einzelnen Schritte oder Phasen des Entwurfsprozesses wurden so konzipiert, dass sie jeweils einen Schwerpunkt bei der Anwendung eines ressourcenorientierten Architekturstils abdecken und den Software-Architekten angemessen unterstützen. So sollen Entwurfsfehler im Vorfeld vermieden werden, die im weiteren Verlauf der Implementierung Mehrkosten für Korrekturen nach sich ziehen würden. Die Validierung ergab, dass sich die Anzahl der Review-Zyklen verringern ließ, ein Hinweis dafür, dass Entwurfsfehler reduziert wurden. Die Ermittlung von Entwurfsfehlern erfolgte durch Betrachtung der einzelnen Qualitätsaspekte im Rahmen der Wiederverwendbarkeit. Der hier skizzierte

Entwurfsprozess lässt sich einfach mit bestehenden Entwurfsprozessen verknüpfen, um in der Praxis anwendbar zu sein. Das haben auch die drei unabhängigen Studien in der Industrie gezeigt. Alle Unternehmen planen die Beiträge dieser Arbeit weiter zu nutzen und auszubauen.

Reduktion des notwendigen Wissens bei Entwurfsentscheidungen

Der Entwurfsprozess wurde unter Berücksichtigung bestehender Muster und bewährter Methoden im Kontext ressourcenorientierter Microservices entwickelt. Er fokussierte die resultierende Web-API, welche die immaterielle Dienstleistung eines Microservice den Service-Konsumenten offeriert. Die Verknüpfung der Muster und bewährten Methoden mit den zu berücksichtigenden Qualitätsteilmerkmalen der Wiederverwendbarkeit zeigte den Einfluss auf die resultierende Qualität. So kann ein Software-Architekt fundierte Entwurfsentscheidungen treffen, ohne sich intensiver mit der entsprechenden Materie vertraut machen zu müssen. Hierzu wurden für die Nachvollziehbarkeit Heuristiken und Regeln aufgestellt, mit deren Hilfe er problemlos zu einem qualitätsgesicherten Entwurf gelangt. Für eine Entwurfsentscheidung sind die erforderlichen Informationen so verfügbar, dass sie sich fundiert und zielgerichtet treffen lässt. Dieser Aspekt wurde mit mehreren Validierungsansätzen untersucht, was wiederum bestätigte, dass sich die Beiträge der Arbeit praktisch anwenden lassen.

Verbesserung der Wiederverwendbarkeit ressourcenorientierter Microservices

Die Wiederverwendbarkeit ist für die serviceorientierte Ausrichtung eines Unternehmens ein zentrales Ziel, da so bestehende geschäftliche Dienstleistungen zu höherwertigen Geschäftsprozessen verschaltet und neue Anwendungsfälle bedient werden können [Jo07, Ge11]. Die Wiederverwendbarkeit korreliert dabei positiv mit der veräußerten Web-API. Deshalb muss letztere so entworfen ein, dass sie möglichst eine hohe Wiederverwendbarkeit ermöglicht. Die Wiederverwendbarkeit zeigt sich in verschiedenen Qualitätsteilmerkmalen der Web-API. Die ressourcenorientierten Microservices in dieser Arbeit wurden ohne Berücksichtigung von anwendungsbezogenen Anforderungen entworfen und entwickelt, wodurch die Ausrichtung auf eine Anwendung oder Anwendungsgruppe unterblieb und der resultierende Microservice diesbezüglich als agnostisch zu betrachten ist. Die anwendungsbezogenen Anforderungen müssen stattdessen vom Service-Konsumenten selbst oder, falls notwendig, durch einen höherwertigen Microservice erbracht werden. Die Fokussierung auf die Wiederverwendbarkeit erleichtert die Integration in Service-Konsumenten, da im Zuge dessen die Benutzbarkeit, Mächtigkeit, Interoperabilität und Auffindbarkeit angemessen behandelt werden. Die Validierung ergab eine positive Relation bei der Entwicklung von Service-Konsumenten im Hinblick auf deren Geschwindigkeit bei der Integration. Auch wurde im Entwurfsprozess das Ziel einer hohen Wiederverwendbarkeit fest verankert – ein weiterer Beleg für Verbesserungen im Vergleich mit anderen Ansätzen.

Reduktion von Implementierungsfehlern

Die resultierenden Entwurfsartefakte bei Anwendung des domänengetriebenen Entwurfsprozesses sollen das jeweilige Entwicklungsteam angemessen anleiten und sicherstellen, dass sich der Entwurf auch in der Implementierung schließlich widerspiegelt. An einem durchgehenden und praxisnahen Szenario wurde validiert, wie sich die Implementierung durch die Entwurfsartefakte anleiten lässt und wie sich die einzelnen Aspekte am Ende in der Software-Architektur manifestieren. Durch die Abbildung von Invarianten und Vor- und Nachbedingungen können Fehler vermieden werden, die ansonsten erst zur Laufzeit auftreten würden. Unterdessen liefert die Web-API-Spezifikation konkrete Vorgaben, wie der entsprechende Adapter in der hexagonalen Architektur zu implementieren ist.

Weitere Vorzüge

Die Validierung hat weitere Vorzüge dieses Entwurfsansatzes erkennen lassen, die nicht im Fokus dieser Arbeit standen. Zwar erhöhen auch sie den Nutzen dieser Arbeit, doch sind sie mit Vorsicht zu betrachten, da sie bisher nicht gezielt validiert wurden.

1. **Kürzere Produkteinführungszeiten:** Durch den Entwurf einer Web-API-Spezifikation vor der Überführung auf die Implementierungsebene lassen sich die Implementierungsprozesse des Microservice und etwaiger Service-Konsumenten voneinander trennen. In mehreren Entwicklungsteams ließ sich die Implementierung parallelisieren, sodass das Gesamtprojekt schneller abgeschlossen wurde. Gleichzeitig senkte sich der Abstimmungsaufwand zwischen diesen Entwicklungsteams. Die Möglichkeit einer Parallelisierung lässt sich damit begründen, dass eine Web-API-Spezifikation einen Vertrag für eine erfolgreiche Interaktion zwischen deren Teilnehmern (Microservice und Service-Konsument) darstellt. Gleichzeitig ließen sich Software-Entwickler entsprechend ihrer Expertise einsetzen, was durch schnellere Bewältigung des Gesamtprojekts die Wirtschaftlichkeit für das Unternehmen fördert. So könnte womöglich auch die Produkteinführungszeit verkürzt werden, was in zwei der hier durchgeführten Studien im Vergleich mit früheren Projekten beobachtet wurde.

2. **Verbesserte Auffindbarkeit bestehender Microservices:** Der Entwurf einer Web-API und das Offerieren ihrer Spezifikation über einen dedizierten Endpunkt eines Microservice ermöglicht zur Laufzeit ein Abbild der Microservice-Landschaft[2]. Dadurch lassen sich neu entwickelte Microservices leichter finden und wiederverwenden. Durch die Überführung der Web-API aus dem modellierten Bounded Context besteht zudem ein Bezug zum abgebildeten Geschäftsausschnitt, was die Auffindbarkeit mittels der Begrifflichkeiten der Domäne bzw. der

[2] Das Abbild zur Laufzeit erfordert eine dedizierte und zentrale Registrierungsinstanz bei der sich alle Microservices beim Initialisieren anmelden und so deren jeweiligen Endpunkt preisgeben.

Ubiquitous Language und die Suche nach bestehenden Funktionalitäten mit diesen Begrifflichkeiten ermöglicht. Deshalb nutzt derzeit ein Automobilhersteller einen solchen dedizierten geschäftsagnostischen Microservice[3], der offerierte Web-API-Spezifikationen aggregiert.

8.2 Ausblick

Die hier bearbeiteten Problemstellungen (vgl. Abschnitt 1.4) und die Validierung haben weitere Fragestellungen ergeben, die sich als Basis für die Weiterführung dieser Arbeit anbieten.

Automatische Generierung von Quellcode

Die Generierung von Quellcode ist der nächste logische Schritt nach der Verknüpfung des Entwurfs mit der letztlichen Zielarchitektur [LS+09]. Während die strukturellen Aspekte eines Bounded Context mittels Modell-zu-Text-Transformationen, wie bspw. Xpand oder MOF Model to Text Transformation Language (Mof2Text) automatisch erzeugt werden können, sind verhaltensspezifische Aspekte eine größere Herausforderung. Für die strukturellen Aspekte liefert [EdI+16] am Beispiel von Java EE bereits einen entsprechenden Ansatz. Die verhaltensspezifischen Aspekte eines Bounded Context werden hingegen dort nur in Form von einfachen CRUD-Operationen behandelt. Es stellt sich die Frage, wieweit sich die verhaltensspezifischen Aspekte automatisch generieren und angemessen mit den strukturellen kombinieren lassen. Daneben muss eine Möglichkeit entwickelt werden, Platzhalter zu definieren, um eine manuelle Überführung für den Software-Entwickler kenntlich zu machen.

Berücksichtigung und Abbildung von anwendungsbezogenen Anforderungen

Diese Arbeit betrachtet in Bezug auf ihre Prämissen (vgl. Abschnitt 1.6) keine spezifische Anwendungslogik, die sich aus den Nutzeranforderungen ergeben würde. Diese Anforderungen können bspw. von etwaigen BDD-Szenarien oder anders gearteten Spezifikationsartefakten abgeleitet werden. In den Studien wurde diese Ableitung unter Anleitung des Autors dieser Arbeit durchgeführt, doch nicht im Detail betrachtet. Hierzu ist zuerst zu klären, wie diese Anforderungen sich in den Spezifikationsartefakten systematisch identifizieren und angemessen auf den Entwurf überführen lassen. Es besteht dabei die grundsätzliche Herausforderung, die anwendungsbezogenen Anforderungen von dem Ausschnitt der abzubildenden Geschäftsdomäne und deren Domänenlogik zu separieren. Letztere umfasst dabei sowohl die strukturellen als auch die verhaltensspezifischen Aspekte des Ausschnitts der abzubildenden Geschäftsdomäne. Anschließend ist zu klären, wie sich die anwendungsbezogenen Anforderungen überführen und umsetzen lassen, ohne dass die Wiederverwendbarkeit beeinträchtigt ist, wie es der Fall wäre, wenn der resultierende Microservice durch die Umsetzung anwendungsbezogener Anforderungen nur von einer spezifischen Anwendung bzw. einer Anwendungsgruppe genutzt werden kann. So könnte es etwa sinnvoll sein, diese Anforderungen als eigenständigen Microservice

[3] Heutzutage wird ein Teil der benötigten Dienstleistung durch API-Managementsysteme erbracht.

in Form eines BFF-Service umzusetzen, sodass die ressourcenorientierten Microservices nach dem domänengetriebenen Entwurfsprozess weiterhin lose gekoppelt von einer konkreten Anwendung bleiben. Ein Indiz für eine derartige Vorgehensweise liefert ein führender Automobilhersteller, der auch Teile dieser Arbeit aufgreift.

Einbeziehung von Laufzeitaspekten beim Entwurf

Der Entwurf wird gegenwärtig ähnlich wie in den Arbeiten [LK+06, Ge11, Ro16] ohne Einbeziehung von etwaigen Laufzeitaspekten durchgeführt, da diese zur Entwurfszeit nicht vorliegen. Es wird mittels der Mustersprache lediglich eine Aussage getätigt, dass einige Interaktionsmuster einen positiven Einfluss auf die Bandbreitennutzung und damit letztlich auch die Laufzeit haben können (vgl. Abschnitt 5.3). Allerdings unterbleibt hier die Rückführung der gesammelten Informationen aus dem Betrieb von Microservices, da auch deren Betrieb nicht Gegenstand dieser Arbeit ist. Zudem kann eine Dienstgütevereinbarung (engl. Service Level Agreement, SLA) Zusagen zu Laufzeitaspekten erforderlich machen, was wiederum eine genaue Vorhersage verlangt. Einen möglichen Ansatz zur Vorhersage von Laufzeitaspekten könnte das Palladio Component Model (PCM) liefern [RB+16a].

Erweiterung des Prozesses um die Verwaltung von Web-APIs

Nach dem Entwurf eines ressourcenorientierten Microservice sowie dessen Überführung auf die Implementierungsebene wäre die Verwaltung der veräußerten Web-API ein möglicher nächster Schritt: „Once an API has been created, it needs to be managed using an API management platform“ [De17, S. 15]. API-Management-Plattformen ermöglichen etwa bessere Auffindbarkeit, Monetarisierung, Zugriffskontrolle und Analyse der bereitgestellten immateriellen Dienstleistung, was wiederum bspw. die Ermittlung von Key Performance Indicators (KPIs) erlaubt [De17]. Diese Arbeit hat bereits angedeutet, wie eine API-Management-Plattform die Qualität einer Web-API weiter verbessern kann. Unklar ist jedoch, wie der hier vorgestellte Ansatz erweitert werden muss, sodass auch die Verwaltung von Web-APIs in API-Management-Plattformen in geeigneter Form betrachtet wird. Ebenso wäre zu untersuchen, welche Eigenschaften einer Web-API eine Verwaltung vereinfachen oder gar erst ermöglichen.

Vertragsgestützte Überführung der Implementierung

Die Implementierung des ressourcenorientierten Microservice erfolgt anhand des Ressourcenmodells bzw. der Web-API-Spezifikation sowie des modellierten Bounded Context. Eine Herausforderung in dieser Phase des Entwicklungsprozesses ist die Sicherstellung der Konformität zum Entwurf, was eine Aufgabe des Entwicklungsteams ist. Es müssen sich daher der modellierte Geschäftsausschnitt und die Entwurfsentscheidungen in der Implementierung widerspiegeln. Das ist ein zentrales Prinzip von *DDD*, in dem das Domänenmodell die Basis für die Implementierung ist: „The vital detail about

the design is captured in the code. A well-written implementation should be transparent, revealing the model underlying it“ [Ev03, S. 37]. Deswegen müssen aus dem zugrundeliegenden Modellierungsartefakt des Bounded Context die Randbedingungen für die Testfälle unter Berücksichtigung des Ressourcenmodells zur Interaktion mit dem Microservice so abgeleitet werden, dass sich diese Testfälle nach dem Blackbox-Prinzip ergeben. Nur so kann die Grundlage dafür geschaffen werden, dass Entwurf und Implementierung trotz ihrer unterschiedlichen Abstraktionsniveaus zumindest in gewissem Umfang das Gleiche abbilden. Die Vorarbeiten wurden bereits durch Modellierung der Bounded Contexts (vgl. Abschnitt 4.2) und durch die Festlegung möglicher Antworten der Web-API basierend auf den Vor- und Nachbedingungen sowie etwaigen Invarianten erbracht.

Mit den zuvor aufgeführten Erweiterungspunkten, den im Vorfeld zusammengefassten Beiträgen sowie den damit einhergehenden Vorzügen schließt diese Arbeit ab und liefert einen Mehrwert für den Entwurf von ressourcenorientierten Microservices unter Berücksichtigung einer hohen Wiederverwendbarkeit sowie unter Anwendung eines domänengetriebenen Entwurfsansatzes.

9 Anhang

A Ergänzungen

Der folgende Abschnitt liefert Ergänzungen zu einzelnen Abschnitten dieser Arbeit, die zu den erzielten Ergebnissen beigetragen haben. Im Vergleich zu anderen Beiträgen im Rahmen dieser Arbeit haben diese Ergänzungen allerdings nur eine ungeordnete Rolle, da diese lediglich zusätzliches und aufbereitetes Wissen bereitstellen, das nicht für das Gesamtverständnis dieser Arbeit notwendig ist.

A.1 Vergleich von REST und WS*

Abbildung 9.1 verdeutlicht den Unterschied zwischen REST-basierten und WS*-Services. Abbildung und ihre Erläuterung basieren auf der Arbeit von Pautasso et al. [PW10]. Dadurch wird u. a. deutlich, dass REST-basierte Services durch die Möglichkeit der Verwendung unterschiedlicher Nachrichtenformate im Vergleich zu WS*-Services eine höhere Interoperabilität aufweisen.

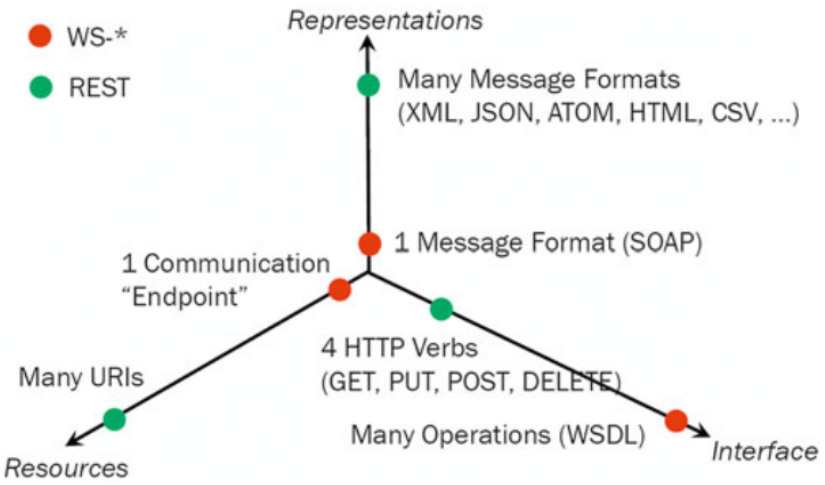


Abbildung 9.1: Entscheidungsspielraum von REST vs. WS-* [PW10, S. 35]

A.2 Nachteile von monolithische Architekturen

Bei einem Monolithen handelt es sich laut Dragoni et al. [DG+17], um eine Software-Anwendung, bei der die einzelnen Komponenten nicht unabhängig voneinander ausgeführt werden können. Für die Auslieferung der Software-Anwendung genügt lediglich ein Software-Artefakt.

Obwohl monolithische Software-Anwendungen samt ihrer zugrundeliegenden Software-Architektur in den letzten Jahren etwas in Verruf geraten ist, sind Monolithen nicht grundsätzlich schlecht. So eignen sie sich bspw. für kleine und überschaubare Entwicklungsprojekte und vereinfachen die Auslieferung sowie den Betrieb. Sobald allerdings bestimmte nicht-funktionale Anforderungen eine unternehmenskritische Rolle spielen, wie bspw. die Skalierbarkeit oder die Flexibilität, können Monolithen ein großes Hindernis im Hinblick auf die Neuausrichtung der Software-Anwendung sein [JN+16]. Um eine bessere Entscheidung im Hinblick auf das Verfolgen eines monolithischen Entwurfsansatzes im Vergleich zu einer Anwendung mit mehreren unabhängigen und verteilten Komponenten treffen zu können, sind nun einige Nachteile von Monolithen aufgeführt.

- **Erschwerte Wartbarkeit:** Etwaige Modifikationen am Quellcode der Software-Anwendung als Folge neuer Anforderungen werden durch die stetig wachsende Quellcodebasis erschwert. Zudem können Modifikationen weitläufige Auswirkungen für die gesamte Software-Anwendung haben, die im Vorfeld noch gar nicht erkennbar sind [JN+16, DG+17].

- **Erschwerte Flexibilität:** Einander widersprechende nicht-funktionale Anforderungen einzelner Komponenten können nicht isoliert behandelt werden, da sie in einem einzigen Software-Artefakt ausgeliefert werden. Dies bedingt, dass sich die Ausführungsplattform stets an den speicher- und rechenintensivsten Komponenten ausrichten muss [DG+17].

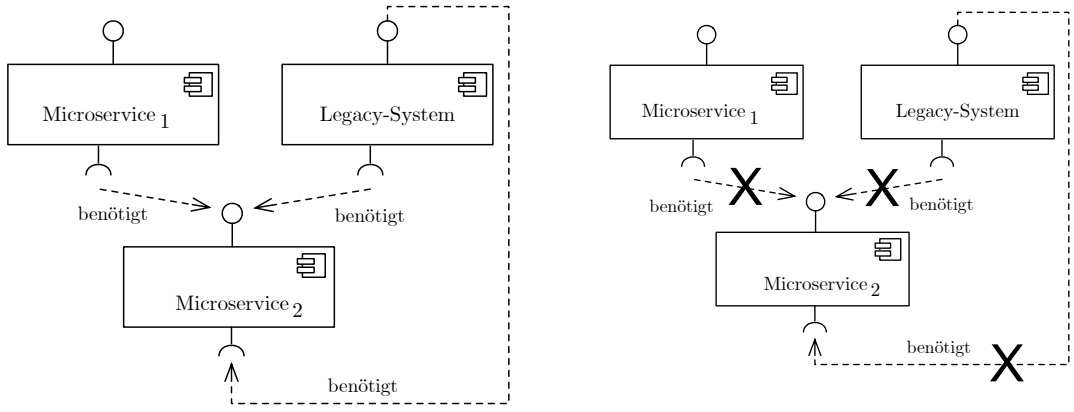
- **Erschwerte Skalierbarkeit:** Eine Skalierung kann entweder horizontal oder vertikal erfolgen. Im Kontext der horizontalen Skalierung geschieht dies über eine weitere Instanziierung der Software-Anwendung unter gleichzeitigem Verteilen etwaiger Anfragen auf die verfügbaren Instanzen. Die Instanziierung kann allerdings nur grobgranular erfolgen, was dazu führt, dass alle darin enthaltenen Komponenten neu instanziiert werden. Da eine feingranulare Skalierung einzelner Komponenten nicht möglich ist, droht schlechte Ressourcenausnutzung [JN+16, DG+17].

A.3 Qualitätsmerkmale eines Microservice

Dieser Abschnitt greift die Definition eines Microservice aus Abschnitt 2.1.1 auf und leitet die damit einhergehenden Qualitätsmerkmale eines Microservice her. Ein Qualitätsmerkmal ist laut Gebhart „die Eigenschaft einer Funktionseinheit, anhand derer ihre Qualität beschrieben und beurteilt wird, die jedoch keine Aussage über den Grad der Ausprägung enthält“ [Ge11, S. 30]. Dabei lassen sich die Qualitätsmerkmale wiederum weiter in sogenannte Qualitätsteilmerkmale verfeinern [Ba08, Ge11]. Das resultierende Qualitätsmodell wird nach Balzert [Ba08] auch als Factor-Criteria-Metrics (FCM)-Modell bezeichnet.

Hohe Autonomie

Als Autonomie gilt laut Erl [Er07, Er17] die Fähigkeit, sich selbst zu verwalten (engl. self-govern). Entscheidungen können ohne Einbeziehung externer Parteien getroffen werden. Dabei wird der Grad der Autonomie durch die Anzahl der bei einer Entscheidung involvierten Parteien bestimmt. Für hohe Autonomie muss die Anzahl der Parteien bzw. die damit einhergehenden Abhängigkeiten reduziert werden [Ge11]. Wird diese Betrachtungsweise auf Microservices übertragen, handelt es sich bei den Parteien, um die Abhängigkeiten, die zur Erbringung der immateriellen Dienstleistung erforderlich sind. Die Abhängigkeit schließt dabei andere Microservices mit ein, aber auch weitere Systeme, wie bspw. Datenbanksysteme. Abbildung 9.2 zeigt eine schematische Gegenüberstellung von verschiedenen Graden der Autonomie. Durch das Vermeiden der Abhängigkeiten (b) lässt sich demnach ein höherer Grad der Autonomie erreichen.



(a) Niedrige Autonomie

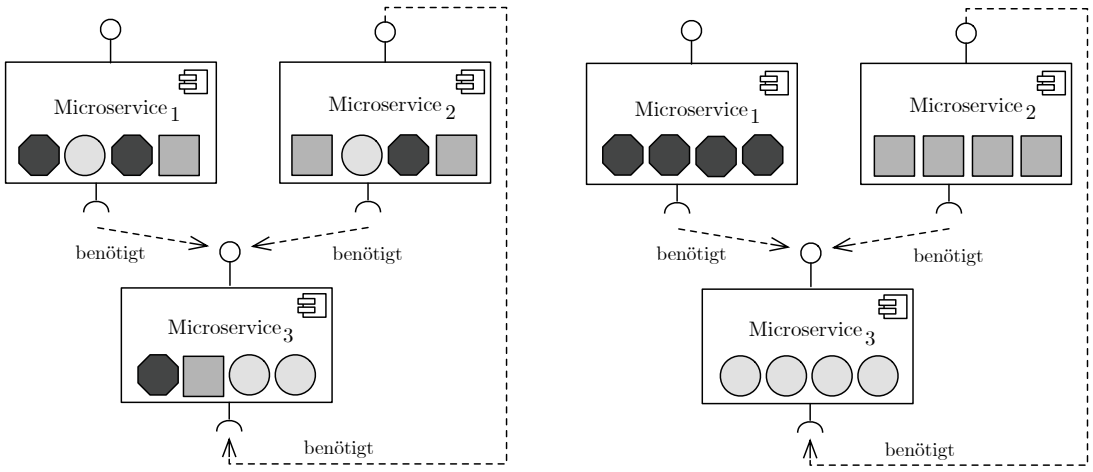
(b) Hohe Autonomie

Abbildung 9.2: Schematische Darstellung der Autonomie

Das Ziel einer hohen Autonomie ist es, dass ein Microservice seinen abgebildeten Ausschnitt der Geschäftsdomäne möglichst eigenständig erbringen kann und somit eine unabhängige Verteilung eines Microservice ohne zusätzliche Koordination bspw. zwischen verschiedenen Entwicklungsteams oder anderen organisatorischen Einheiten ermöglicht wird. Daneben wird die Ausfallsicherheit (engl. resilience/reliability) des Microservice erhöht, da die Abhängigkeiten im Betrieb eine weitere potentielle Fehlerquelle darstellen. Gleichzeitig können durch mehr Unabhängigkeit und die Abgeschlossenheit, mit der Microservices operieren, etwaige Fehler im System schneller lokalisiert werden, da „the failure doesn't cascade [through the whole system]“ [Ne15, S. 5]. Abschließend bleibt zu erwähnen, dass sich durch eine hohe Autonomie auch die Vorsehbarkeit (engl. predictability) eines Microservice im Hinblick auf sein Verhalten verbessert [Er07, Ge11, Er17]. Denn beim höchsten Grad der Autonomie ist nur der Microservice selbst für sein Verhalten verantwortlich.

Hohe Kohäsion

Als Kohäsion definiert Vogel die „Abhängigkeiten innerhalb eines Systembausteins“ [VA+09, S. 133]. Dabei ist eine hohe Kohäsion genau dann gegeben, wenn ein Systembaustein alle Informationen, „zum Verstehen und Ändern relevante[r] Eigenschaften in seiner Beschreibung vereint“ [VA+09, S. 133]. Verstärkt wird das Prinzip der hohen Kohäsion durch das *Single Responsibility Principle (SRP)*, laut welchem alle Dinge, die sich aus dem gleichen Grund ändern, zusammengehören und alle Dinge, die sich aus unterschiedlichen Gründen ändern, zu trennen sind [He10, S. 152] aus [Ne15]. In Bezug auf Microservices ist der zuvor erwähnte Systembaustein mit einem Microservice gleichzusetzen. Zum besseren Verständnis soll die Abbildung 9.3 dienen, in der eine zusammenhängende bzw. kohäsive geschäftliche Dienstleistung durch jeweils eine geometrische Figur repräsentiert wird. Eine Änderung würde bei geringer Kohäsion die Änderung an weiteren Microservices verlangen, während bei hoher Kohäsion dies nur für einen Microservice erforderlich ist. Hohe Kohäsion hat demnach auch einen positiven Einfluss auf die Autonomie eines Microservice.



(a) Niedrige Kohäsion **(b)** Hohe Kohäsion

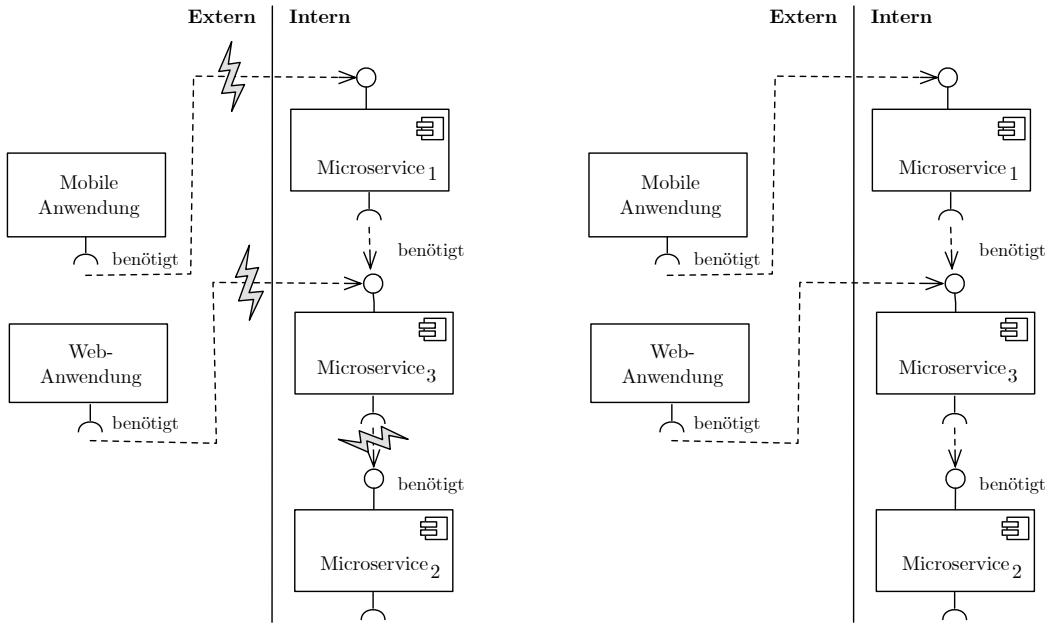
Abbildung 9.3: Schematische Darstellung der Kohäsion

Durch hohe Kohäsion soll die Wartbarkeit erleichtert werden, damit Fehler schnell lokalisiert und beseitigt werden können. Gleichzeitig wird das Prinzip der lokalen Änderbarkeit adressiert, wodurch eine Änderung an einem Microservice keine Änderungen an anderen Microservices verlangt [VA+09]. Das senkt Kommunikationsaufwand mit weiteren Entwicklungsteams oder anderen organisatorischen Einheiten bei Änderungen.

Hohe Interoperabilität

Die Interoperabilität beschreibt nach ISO/IEC 25010:2011 [ISO-25010], in welchem Umfang zwei oder mehrere Systeme, Produkte oder Komponenten etwaige Informationen zu deren weiteren Ver-

wendung untereinander austauschen können. Auch Erl [Er17] betont diesbezüglich das Teilen von Daten: „Interoperability refers to the sharing of data“ [Er17, S. 44]. Die Übertragung dieser Definition auf Microservices adressiert die nachrichtenbasierte Interaktion zwischen Microservice und Service-Konsumenten und stellt die Web-API als deren Bindeglied in den Mittelpunkt. Abgeleitet aus [WP+16] lassen sich Service-Konsumenten grundsätzlich in zwei Kategorien einteilen: 1) den internen und 2) den externen Service-Konsumenten (vgl. Abschnitt 6.5).



(a) Niedrige Interoperabilität **(b)** Hohe Interoperabilität

Abbildung 9.4: Schematische Darstellung der Interoperabilität

Hohe Interoperabilität soll den Aufwand für die notwendige Integration reduzieren, da „Software programs that are not interoperable need to be integrated“ [Er17, S. 44]. Das erhöht die Wiederverwendbarkeit der bereitgestellten Dienstleistung der Microservices. So ermöglicht die Interoperabilität bspw. eine Komposition auf Ebene der Microservices oder die Entwicklung von neuen Anwendungen mit geringem Aufwand, welche die bereitgestellte Dienstleistung nutzen, um neue Anwendungsfälle zu bedienen und damit einen Mehrwert im Sinne eines Nutzungsversprechens (engl. value proposition) zu schaffen.

Abbildung 9.4 stellt das Ziel einer hohen Interoperabilität schematisch dar. Bei geringer Interoperabilität können sich Schwierigkeiten bei der Integration in eine Android- oder Web-Anwendung ergeben können. Gleiches gilt bei einer Komposition von Microservices innerhalb der Unternehmensgrenze. Im Fall einer hohen Interoperabilität sollte grundsätzlich "jeder mit jedem" über eine Web-API kommunizieren können, ohne dass spezielle Vorgaben, wie bspw. die Verwendung einer bestimmten Technologie, erfüllt sein müssen.

A.4 UML-Diagrammtypen zur Abbildung von Verhalten

UML stellt eine Reihe von verschiedenen Diagrammtypen zur Abbildung von Verhalten bereit, die jeweils unterschiedliche Aspekte des Verhaltens in Vordergrund rücken. So fokussiert bspw. das Aktivitätsdiagramm einen zeitlichen Ablauf von sogenannten *Aktionen*, die wesentliche Ablaufschritte eines abzubildenden Verhaltens darstellen [OB12]. Abbildung 9.5 gibt einen Überblick der Diagrammtypen aus der UML 2.5 [OMG-UML2c] im Kontext der Verhaltensmodellierung.

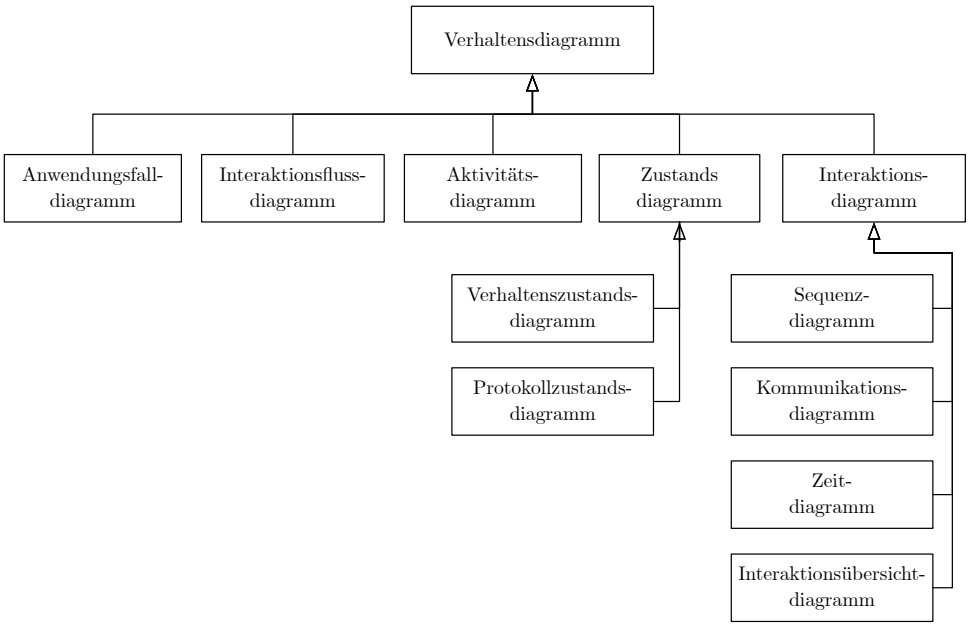


Abbildung 9.5: UML-Diagrammtypen zur Abbildung von Verhalten

A.5 Behandlung von geteilten Domänenobjekten auf Ressourcenebene

Geteilte Domänenobjekte mit dem Stereotyp `Shared` verfügen über eine Eigenschaftsdefinition `owner`, welche die Eignerschaft der Domänenobjekte angibt. Die Eignerschaft bezieht sich dann auf einen Bounded Context. Das Verhalten eines Domänenobjekts sollte nur bei einem Bounded Context implementiert werden, der die Eignerschaft hat. Dadurch können der Implementierungsaufwand reduziert und zukünftige Änderungen an zentraler Stelle durchgeführt werden[1]. Der jeweilige Microservice muss schließlich das Verhalten nach dem in dieser Arbeit vorgestellten domänengetriebenen Entwurfsprozess veräußern und so die immaterielle Dienstleistung über eine Web-API bereitstellen.

[1] Die Verschlechterung der Autonomie abhängiger Microservices wurde an dieser Stelle im Zuge der resultierenden Vorzüge des Vorgehens explizit in Kauf genommen. Falls die Autonomie bei der Betrachtung eines jeweiligen Szenarios schwerer wiegt, so kann das jeweilige Verhalten eines Domänenobjekts auch mehrfach implementiert werden. Das geteilte Domänenobjekt ist dann als nicht geteiltes Domänenobjekt zu betrachten.

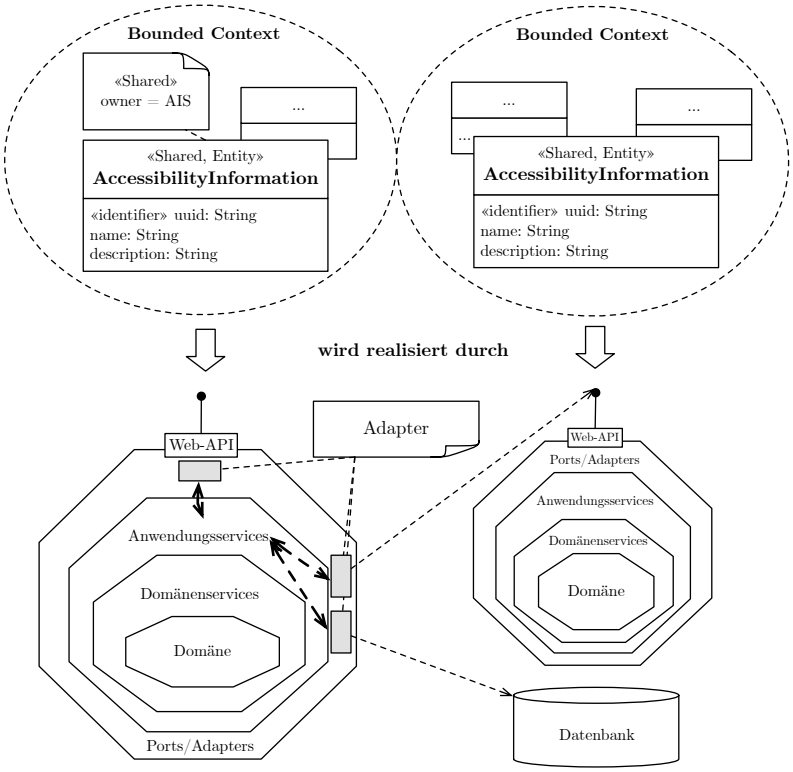


Abbildung 9.6: Schematische Darstellung zur Behandlung von geteilten Domänenobjekten

Bei Bounded Contexts ohne Eignerschaft zu einem oder mehreren geteilten Domänenobjekten sind die gleichen Regeln und Heuristiken für die Überführung in ein Ressourcenmodell anzuwenden wie bei einem nicht geteilten Domänenobjekt. Die Besonderheit liegt lediglich in der Implementierung bzw. in der Überführung auf die Zielarchitektur. Anstatt das Verhalten der jeweiligen Domänenobjekte zu implementieren, wird ein Adapter entwickelt, der entsprechende Anfragen an die Web-API des Microservice mit der Eignerschaft durchführt. Für die Kommunikation zwischen den Microservices muss die entsprechende Context Map (nicht dargestellt) berücksichtigt werden. Die Verwendung von einem oder mehreren abhängigen Microservices bleibt somit den Service-Konsumenten verborgen. Abbildung 9.6 fasst das Vorgehen zur Behandlung von geteilten Domänenobjekten in schematischer Form zusammen.

A.6 Musterbeschreibungsvorlage

Die Tabelle 9.1 repräsentiert die Vorlage für die Beschreibung der Muster im Rahmen dieser Arbeit. Ein Muster besteht aus acht Bestandteilen und lehnt sich an der existieren Mustersprache von Pautasso et al. [PI+16] an. Im Gegensatz zu der verwandten Arbeit wurde die Beschreibung der Muster um den Bestandteil *Beispiel* erweitert, welches die Umsetzung der Lösung mittels HTTP demonstriert. Eine ausführliche Beschreibung der einzelnen Bestandteile ist der Arbeit [MD97] zu entnehmen.

Zweck und Name des Musters inklusive einer geeigneten Abkürzung zur einfacheren Referenzierung	
Kontext	Beschreibt den Kontext, in dem das Muster seine Anwendung finden kann und liefert den Rahmen für das zugrundeliegende Problem
Problem	Beschreibt das Problem, für welches das Muster eine entsprechende Lösung bereitstellt
Zwänge	Beschreibt die Zwänge, aus denen das Problem hervorgeht
Lösung	Beschreibt die Lösung des Problems
Vorzüge	Beschreibt die Vorzüge der Lösung
Lasten	Beschreibt die Nachteile der Lösung
Einsatz	Listet praxisnahe Beispiele, die das Muster anwenden
Beispiel	Liefert eine beispielhafte und konkrete Umsetzung der geschilderten Lösung

Tabelle 9.1: Tabellarische Vorlage für die Beschreibung der Muster im Rahmen dieser Arbeit

A.7 Muster zur Interaktion mit Ressourcen

Es folgen die Muster, die im Rahmen der Mustersprache in Abschnitt 5.3.2 aufgeführt wurden. Dort wurden für die genauere Beschreibung der Muster bereits die Variablen eingeführt, weshalb nun darauf verwiesen wird (vgl. Tabelle 5.7 auf Seite 110). Weiter wurde im Zuge der Betrachtung der internen und externen Einflüsse einzelner Muster drei Kennzeichnungen eingeführt mit dessen Hilfe die jeweiligen Vorzüge und Lasten annotiert werden können (vgl. Tabelle 9.2). Da die Muster nicht auf Microservices beschränkt sind, ist der Begriff des Microservice in diesem Kontext mit einem Service im Allgemeinen gleichzusetzen.

Markierung	Bedeutung
$\leftarrow \vert \times$	Wirkt positiv oder negativ auf Service-Konsumenten
$\times \vert \rightarrow$	Wirkt positiv oder negativ auf zu implementierendem Microservice
$\leftarrow \vert \rightarrow$	Wirkt positiv oder negativ auf Service-Konsumenten und den zu implementierenden Microservice
$\times \vert \times$	Wirkt weder positiv noch negativ auf Service-Konsumenten oder den zu implementierenden Microservice

Tabelle 9.2: Markierung der Vorzüge und Lasten eines Musters im Hinblick auf deren Einfluss

Muster zur Interaktion mit einer Listenressource

Traversierung von Listenressourcen mit dem Resource Collection Traversal Pattern (P_{RCT}) auf Basis von [PI+16]	
Kontext	Ressourcen sind häufig in Listenressourcen R_{List} gruppiert, um u. a. deren Auffindbarkeit zu erleichtern [PI+16]. Dabei soll der S_K in der Lage sein, alle Ressourcen einer Listenressource $RC(r_{list})$ abzurufen.

Traversierung von Listenressourcen mit dem Resource Collection Traversal Pattern (P_{RCT}) auf Basis von [PI+16]	
Problem	Wie kann ein Service-Konsument S_K inkrementell die Ressourcen einer Listenressource $RC(r_{list})$ abrufen?
Zwänge	Bei einer großen Sammlung an Ressourcen kann es nach Haupt et al. [HL+15, PI+16] unpraktikabel sein die komplette Liste an Ressourcen $RC(r_{list})$ bzw. deren Repräsentationen $Repr(r_{list})$ dem Service-Konsumenten S_K zurückzuliefern. Daneben sind nicht alle Service-Konsumenten S_K daran interessiert die komplette Liste zu erhalten. Gerade im mobilen Bereich liegt der Fokus auf einer effizienten Ausnutzung der Bandbreite. Andere Service-Konsumenten S_K wiederum benötigen einen kompletten Abzug der Ressourcen $RC(r_{list})$.
Lösung	Ein Service-Konsument S_K übermittelt mit der Anfrage an eine r_{list} die Mengenanzahl der gewünschten Ressourcen. Als Antwort erhält der Service-Konsument S_K die maximale Anzahl der angeforderten Ressourcen bzw. deren Repräsentationen (Teilmenge von $RC(r_{list})$).
	Daneben wird die Antwort der Anfrage in Form einer Nachricht um zusätzliche Meta-Informationen angereichert, sodass (falls vorhanden) weitere Ressourcen der r_{list} abgerufen werden können. Letzteres ermöglicht dem Service-Konsument S_K iterativ über die Ressourcen einer Listenressource r_{list} zu traversieren. Für die Ausprägung der Meta-Informationen und die damit einhergehende Traversierung existieren, zwei unterschiedliche Lösungsvarianten, die in den Mustern P_{ORCT} (vgl. Tabelle 9.4) und P_{CRCT} (vgl. Tabelle 9.5) aufgeführt sind. Grundsätzlich können beide Muster gleichzeitig eingesetzt werden. Die entsprechende Auswahl ist den Software-Entwicklern der Service-Konsumenten S_K überlassen.
Vorzüge[2]	1) *Mächtigkeit ($\leftarrow$ | $\times$):* Reduzierte Bandbreitenausnutzung, da an die Service-Konsumenten S_K nur eine Teilmenge der Ressourcen einer Listenressource $RC(r_{list})$ als eine Ansammlung von Repräsentationen $Repr(r_{list})$ übermittelt wird [PI+16]. 2) *Mächtigkeit ($\leftarrow$ | $\times$):* Service-Konsumenten S_K können selbstständig bestimmen, wie groß die Teilmenge von $RC(r_{list})$ ist. 3) *Komplexität ($\leftarrow$ | $\times$):* Service-Konsumenten S_K müssen keine Logik implementieren, um große Datenmengen effizient verarbeiten zu können.
Lasten[3]	1) *Komplexität ($\times$ | $\rightarrow$):* Erhöhter Aufwand auf Seiten des Microservice, da die Logik zur Traversierung implementiert werden muss.
Einsatz	Ist der entsprechenden Lösungsvariante zu entnehmen.
Beispiel	

Tabelle 9.3: Resource Collection Traversal Pattern (P_{RCT}) zur Traversierung von Listenressourcen

Versatzbasierte Traversierung von Listenressourcen mit dem Offset-based Resource Collection Traversal Pattern (P_{ORCT})	
Kontext	Ist dem Muster in Tabelle 9.3 zu entnehmen.
Problem	
Zwänge	

[2] Die weiteren Vorzüge sind der ausgewählten Lösungsvariante zu entnehmen.
[3] Die weiteren Lasten sind der entsprechenden Lösungsvariante zu entnehmen.

Versatzbasierte Traversierung von Listenressourcen mit dem Offset-based Resource Collection Traversal Pattern (P_{ORCT})	
Lösung	Der Service-Konsument S_K sendet eine Anfrage an eine Listenressource r_{list} mit entsprechenden Kontrollinformationen, um eine spezifische Teilmenge der Ressourcen einer Listenressource $RC(r_{list})$ abzurufen. Die Listenressource r_{list} antwortet daraufhin mit der angeforderten Teilmenge bzw. deren Repräsentationen sowie Kontrollinformationen, mit deren Hilfe weitere Teilmengen der Listenressource r_{list} abgerufen werden können. Die Antwort kann weitere Meta-Informationen etwa zur Anzahl zugeordneter Ressourcen beinhalten. Diese Information ist besonders wichtig, wenn eine Paginierung auf Benutzeroberfläche abgebildet werden muss. Bei den Kontrollinformationen kann es sich bspw. um Hyperlinks handeln (vgl. Muster *Resource Collection Traversal Pattern* in [PI+16]) oder um Identifikatoren, um eine Teilmenge genau zu adressieren. So kann bspw. eine Teilmenge mit einem numerischen Index und einem Offset adressiert werden [GG+16c]. Dies ist vergleichbar mit einem *LIMIT* und einem *OFFSET* aus dem Bereich der Datenbankabfragesprachen.
Vorzüge	1) *Komplexität ($\times\vert\rightarrow$) :* Verbreitete Unterstützung im Kontext heutiger Frameworks im Vergleich zum Muster P_{CRCT}. 2) *Mächtigkeit ($\leftarrow\vert\times$):* Die Service-Konsumenten S_K können auf eine bestimmte Teilmenge navigieren [Ni14].
Lasten	1) *Benutzbarkeit ($\leftarrow\vert\times$):* Durch nebenläufige Änderungen können bei der Traversierung etwaige Ressourcen übersprungen oder doppelt angezeigt werden.
Einsatz	Dieses Muster findet Einsatz bei der Facebook Graph API[4], bei Zalando APIs[5] oder auch bei großen Blog-Archiven, in denen die Blog-Beiträge über mehrere Seiten verteilt sind [PI+16].
Beispiel	Der Service-Konsument S_K sendet eine Anfrage mittels HTTP-GET an eine r_{list} mit Kontrollinformationen durch das Setzen der Anfrageparameter `limit` und `offset`, woraufhin er eine Repräsentation der angeforderten Teilmengen von Ressourcen $RC(r_{list})$ erhält. Mit den zurückgelieferten Meta-Informationen `next, previous, first, last` kann schließlich eine weitere Teilmenge adressiert werden [GG+16c].

Tabelle 9.4: Offset-based Resource Collection Traversal Pattern (P_{ORCT}) zur Traversierung von Listenressourcen

Zeigerbasierte Traversierung von Listenressourcen mit dem Cursor-based Resource Collection Traversal Pattern (P_{CRCT})	
Kontext	Ist dem Muster in Tabelle 9.3 zu entnehmen.
Problem	
Zwänge	
Lösung	Der Service-Konsument S_K sendet eine Anfrage an den Microservice mit entsprechenden Kontrollinformationen, um eine bestimmte Anzahl von Ressourcen einer Listenressource r_{list} zu erhalten. Die Listenressource r_{list} antwortet daraufhin mit der angeforderten Teilmenge bzw. deren Repräsentationen sowie Kontrollinformationen mit denen die nächste Teilmenge abgerufen werden kann.

[4] https://developers.facebook.com/docs/graph-api/using-graph-api/v2.4#paging (Letzter Zugriff: 14.10.2017)
[5] https://zalando.github.io/restful-api-guidelines/pagination/Pagination.html (Letzter Zugriff: 14.10.2017)

Zeigerbasierte Traversierung von Listenressourcen mit dem Cursor-based Resource Collection Traversal Pattern (P_{CRCT})	
	Die Kontrollinformationen sind so aufgebaut, dass sie einen Zeiger in Form einer sequentiellen Kennung repräsentieren, der genau eine Ressource adressiert. Dabei ist darauf zu achten, dass die Zeiger auf Service-Konsumentenseite nicht zwischengespeichert werden, da sie im Lauf der Zeit durch Veränderungen auf Microserviceseite ungültig werden. Daneben können auch Meta-Informationen übermittelt werden, welche bspw. die Anzahl zugeordneter Ressourcen einer Listenressource $RC(r_{list})$ beinhalten.
Vorzüge	1) *Benutzbarkeit ($\leftarrow \mid \times$):* Traversierung aller Ressourcen auch bei nebenläufigen Änderungen [Ni14]. Dies ist insbesondere bei Echtzeitanwendungen entscheidend.
Lasten	1) *Komplexität ($\times \mid \rightarrow$):* Unterstützung im Kontext heutiger Frameworks noch nicht so verbreitet im Vergleich zu P_{ORCT}. 2) *Mächtigkeit ($\times \mid \rightarrow$):* Keine Möglichkeit, auf eine bestimmte Teilmenge zu navigieren: „due to the highly dynamic nature of the data, we can't provide page numbers for cursor based pagination" [Ni14, Abs. 2].
Einsatz	Dieses Muster findet seinen Einsatz bei der Facebook Graph API[6], Twitter API[7] oder auch in Zalando APIs[8].
Beispiel	Der Service-Konsument S_K sendet eine Anfrage mittels HTTP-Methode GET an eine r_{list}, woraufhin er eine Teilmenge von $RC(r_{list})$ erhält. Die Größe dieser Teilmenge kann mittels des Anfrageparameters *limit* festgelegt werden. Neben der Teilmenge an Ressourcen erhält der Service-Konsument S_K zusätzlich Kontrollinformationen in Form eines Zeigers `next_cursor` und `previous_cursor`, die er bei der nächsten Anfrage als Anfrageparameter anhängen kann. Sollte einer dieser Zeiger nicht gesetzt sein, so befindet sich die Traversierung am Anfang oder Ende von $RC(r_{list})$. Falls mehrere Muster zur Paginierung eingesetzt werden, so empfiehlt es sich bei der ersten Anfrage $cursor = -1$ zu setzen. Dadurch kann schließlich die Verwendung der zeigerbasierten Traversierung eindeutig ermittelt werden.

Tabelle 9.5: Cursor-based Resource Collection Traversal Pattern (P_{CRCT}) zur Traversierung von Listenressourcen

Gekürzte Repräsentation von Ressourcen Condensed Resource Representation Pattern (P_{CRR})	
Kontext	Bei der Traversierung von Ressourcen einer Listenressource $RC(r_{list})$ entsprechend P_{CRCT} und P_{ORCT} werden die einzelnen Repräsentationen der Ressourcen dem Service-Konsumenten S_K übermittelt. Derartige Repräsentationen können abhängig von den Attributen und Beziehungen einer Ressource sehr umfangreich sein, obgleich womöglich nur eine Teilmenge dieser Informationen benötigt wird. Dies ist bspw. dann der Fall wenn eine Übersicht der Ressourcen auf Service-Konsumentenseite angezeigt werden soll.
Problem	Bei der Traversierung von Ressourcen werden unter Umständen nicht benötigte Informationen einer Ressource übermittelt, sodass die verfügbare Bandbreite nicht effizient ausgenutzt wird.

[6] https://developers.facebook.com/docs/graph-api/using-graph-api/v2.4#paging (Letzter Zugriff: 14.10.2017)
[7] https://dev.twitter.com/overview/api/cursoring (Letzter Zugriff: 14.10.2017)
[8] https://zalando.github.io/restful-api-guidelines/pagination/Pagination.html (Letzter Zugriff: 14.10.2017)

Gekürzte Repräsentation von Ressourcen **Condensed Resource Representation Pattern (P_{CRR})**	
Zwänge	In bestimmten Szenarien ist die verfügbare Bandbreite sehr begrenzt, sodass nur die benötigten Informationen dem Service-Konsumenten S_K übermittelt werden sollten. Dies ist vergleichbar mit dem Muster Lazy Loading (vgl. [Fo02a]).
Lösung	Bei der Traversierung der Ressourcen einer Listenressource $RC(r_{list})$ wird dem Service-Konsumenten S_K für die einzelnen Ressourcen jeweils eine gekürzte Repräsentation zurückgeliefert, die mindestens einen Identifikator beinhalten muss. Dadurch können weiterführende Informationen über eine separate URI abgerufen werden. Die Wahl der jeweiligen Attribute obliegt der Entscheidung des entsprechenden Entwicklungsteams, das sich ggf. an Anwendungsfällen orientieren kann. Falls ein eindeutiger Identifikator fehlt, was bspw. bei Informationsressourcen der Fall ist, so ist die vollständige Repräsentation zu nutzen.
Vorzüge	1) *Mächtigkeit ($\leftarrow$ | $\times$):* Es werden nicht alle Informationen einer Ressource bei der Traversierung übermittelt, was zu einer Verringerung der Übertragungsmenge und damit besseren Ausnutzung der vorhandenen Bandbreite führt.
Lasten	1) *Mächtigkeit ($\leftarrow$ | $\times$)*: Es sind mehrere Anfragen notwendig, um alle Informationen einer Ressource abzurufen.
Einsatz	Dieses Muster findet zumeist Einsatz bei der Paginierung. Zum Beispiel werden bei den Suchergebnissen heutiger Suchmaschinen die Informationen nur auszugsweise angezeigt und erst beim weiteren Verfolgen der Verlinkung vollständig geladen.
Beispiel	Der Service-Konsument S_K sendet eine Anfrage mittels der HTTP-Methode GET und erhält eine Liste an Ressourcen $RC(r_{list})$ bzw. deren Repräsentationen $Repr(RC(r_{list}))$. Letztere sind so aufgebaut, dass sie partielle Informationen der Ressourcen beinhalten. Um schließlich alle Informationen abzurufen, kann wiederum der enthaltene Identifikator `uuid` verwendet werden, was allerdings eine erneute Anfrage unter Verwendung der HTTP-Methode GET an eine andere URL zur Folge hat. Die entsprechende URL ergibt sich aus der Adressierung der mit dem Identifikator verknüpften Ressource.

Tabelle 9.6: Condensed Resource Representation Pattern (P_{CRR}) zur Reduktion der Übertragungsmenge

Filterung von Listenressourcen **Resource Collection Filtering (P_{RCF})**	
Kontext	Listenressourcen r_{list} können viele Ressourcen beinhalten, obwohl Service-Konsumenten $S_K(M_S)$ unter Umständen nur an diejenigen Ressourcen $R_T \subset RC(r_{list})$ interessiert sind, für die bestimmte Kriterien gelten. Dabei kann es sich bspw. um ein Kriterium handeln, das Ressourcen nach gewissen Datumsgrenzen filtert.
Problem	Bei einer großen Sammlung an zugeordneten Ressourcen einer Listenressource r_{list} ist es für die Bandbreitennutzung unvorteilhaft, sofort alle Ressourcen abzurufen und dann auf Seiten des Service-Konsumenten S_K zu filtern. Dieser müsste zudem die Menge an Informationen effizient und in adäquater Zeit filtern können.

Filterung von Listenressourcen **Resource Collection Filtering (P_{RCF})**	
Zwänge	Service-Konsumenten S_K operieren unter Umständen in einem Umfeld mit eingeschränkter Bandbreite und ohne die Hardware-Ressourcen, die bspw. in modernen Rechenzentren zur Verfügung stehen.
Lösung	Der Service-Konsument S_K sendet eine Anfrage an eine Listenressource r_{list}, bei der er zusätzlich Filterkriterien als Anfrageparameter übergibt. Diese Filterkriterien müssen derart formal beschrieben sein, sodass möglichst viele Ausprägungen abgebildet werden können. Hierzu existieren verschiedene Lösungsansätze, wie bspw. Resource Query Language (RQL)[9], The Feed Item Query Language (FIQL)[10] oder die darauf basierende RESTful Service Query Language (RSQL)[11].
Vorzüge	1) *Mächtigkeit ($\leftarrow$ |×):* Es werden nur die angeforderten Ressourcen an den Service-Konsumenten S_K übermittelt, um die Bandbreitennutzung zu reduzieren. Die Verlagerung der Filterung in den Microservice verringert die Auslastung der Hardware-Ressourcen auf Seiten des Service-Kosumenten S_K. 2) *Mächtigkeit ($\leftarrow$ |×):* Die Service-Konsumenten S_K können selbstständig bestimmen, nach welchen Informationen gefiltert werden soll und welche von diesen sie benötigen. 3) *Komplexität ($\leftarrow$ |×)*: Die Filterung von Ressourcen wird zum Microservice verlagert, wodurch keine derartige Filterlogik beim (S_K implementiert werden muss.
Lasten	1) *Komplexität (×| $\rightarrow$)*: Die Verlagerung der Filterung zum Microservice verlangt eine Implementierung, die abhängig von der gewählten Beschreibungssprache mit einem unterschiedlichem Implementierungsaufwand einhergeht.
Einsatz	Eine Filterung von Ressourcen einer Listenressource $RC(r_{list})$ ist gegenwärtig bei vielen Web-APIs verfügbar, etwa bei der Zalando Shop API[12] oder der Facebook Insights API[13].
Beispiel	Mit dem Abrufen von Ressourcen einer Listenressource $RC(r_{list})$ mittels der HTTP-Methode GET übermittelt der Service-Konsument S_K über den Anfrageparameter `filter` entsprechende Filterkriterien. Diese sind im einfachsten Fall eine Liste mit zu filternden Attributwertpaaren, die jeweils mit einem Komma voneinander getrennt sind: `?filter=attribut1=attributwert1&attribut2=attributwert2&...` [GG+16c]. Andere Ausprägungen der Filterkriterien ist den genannten Lösungsansätzen zur Beschreibung von Filterkriterien zu entnehmen.

Tabelle 9.7: Resource Collection Filtering (P_{RCF}) zur Filterung einer Listenressource

Sortierung von Listenressourcen **Resource Collection Sorting (P_{RCS})**	
Kontext	Die Ressourcen einer Listenressource $RC(r_{list})$ können in unsortierter oder sortierter Reihenfolge dem Service-Konsumenten S_K zurückgeliefert werden. Doch kann ein Service-

[9] https://github.com/persvr/rql (Letzter Zugriff: 09.10.2017).
[10] https://tools.ietf.org/html/draft-nottingham-atompub-fiql-00 (Letzter Zugriff: 09.10.2017).
[11] https://github.com/jirutka/rsql-parser (Letzter Zugriff: 09.10.2017).
[12] https://github.com/zalando/shop-api-documentation/wiki/Filters#general-filters (Letzter Zugriff: 09.10.2017).
[13] https://developers.facebook.com/docs/marketing-api/insights/v2.10 (Letzter Zugriff: 09.10.2017).

Sortierung von Listenressourcen Resource Collection Sorting (P_{RCS})	
	Konsument S_K auch eine mehrstufige Sortierung anhand verschiedener Attribute einer Ressource benötigen.
Problem	Die Sortierung hat einen wesentlichen Einfluss auf die Traversierung der Ressourcen einer Listenressource $RC(r_{list})$. Aus diesem Grund ist es notwendig, dass ein Service-Konsument S_K die jeweiligen Sortierkriterien selbst bestimmen kann. Eine Sortierung auf Seiten des Service-Konsumenten S_K bedingt, dass zunächst alle Ressourcen abgerufen werden müssen.
Zwänge	Um eine Sortierung durchführen zu können, müsste der Service-Konsument S_K alle Ressourcen einer Listenressource $RC(r_{list})$ abrufen und anschließend eine Sortierung entsprechend der Sortierkriterien durchführen. Allerdings erfordert dies ausreichend Bandbreite und hinlängliche Hardware-Ressourcen, um die Repräsentationen der Ressourcen $Repr(RC(r_{list}))$ effizient verarbeiten zu können.
Lösung	Der Service-Konsument S_K sendet eine Anfrage an die Listenressource r_{list}, bei der zusätzlich Sortierkriterien als Anfrageparameter übergeben werden können. Die Sortierkriterien stellen eine Auflistung von Attributen der Ressourcen einer Listenressource $RC(r_{list})$ dar. Bei der Auflistung kann jedem Attribut eine entsprechende Kennzeichnung mitgegeben werden, die eine auf- oder absteigende Sortierung impliziert. Die Reihenfolge der Attribute zeigt die Präferenz der Sortierung an.
Vorzüge	1) *Mächtigkeit* ($\leftarrow \mid \times$): Ein Service-Konsument S_K kann selbst bestimmen, wie die Menge an Ressourcen einer Listenressource $RC(r_{list})$ sortiert wird. 2) *Mächtigkeit* ($\leftarrow \mid \times$): Ein Service-Konsument S_K muss nicht alle Ressourcen einer Listenressource $RC(r_{list})$ abrufen, um eine korrekte Sortierung der Ressourcen zu erhalten. 3) *Komplexität* ($\leftarrow \mid \times$): Die Logik zur Sortierung wird vom Service-Konsumenten S_K auf die Seite des Microservice verlagert, der in der Regel über mehr Hardware-Ressourcen verfügt.
Lasten	1) *Komplexität* ($\times \mid \rightarrow$): Der Microservice muss die entsprechende Logik zur Sortierung implementieren und dem Service-Konsumenten S_K über die Web-API bereitstellen.
Einsatz	Das Muster findet Einsatz bei OpenStack[14] oder mit beschränkter Mächtigkeit bei Twitter[15].
Beispiel	Mit dem Abrufen von Ressourcen einer Listenressource $RC(r_{list})$ mittels der HTTP-Methode GET übermittelt der Service-Konsument S_K über den Abfrageparameter `sort` zusätzliche Sortierkriterien. Diese orientieren sich an den Attributen der Ressourcen und werden per Komma voneinander getrennt. Über - und + wird für jedes Attribut festlegt, ob dieses auf- oder absteigend sortiert werden soll: `sort=-attribute1,+attribute2,...`.

Tabelle 9.8: Resource Collection Sorting (P_{RCS}) zur Sortierung einer Listenressource

Muster zur partiellen Bearbeitung einer Ressource Partial Resource Editing/Creation Pattern (P_{PRE}) auf Basis von [PI$^+$16]	
Kontext	Der Service-Konsument S_K möchte eine Ressource $r \notin R_{List} \cup R_{Act}$ bearbeiten, verfügt aber nicht über das Wissen, welche Informationen zur Bearbeitung der Ressource vonnöten sind.

[14] https://specs.openstack.org/openstack/api-wg/guidelines/pagination_filter_sort.html (Letzter Zugriff: 08.10.2017)
[15] https://developer.twitter.com/en/docs/ads/general/guides/sorting (Letzter Zugriff: 08.10.2017)

Muster zur partiellen Bearbeitung einer Ressource Partial Resource Editing/Creation Pattern (P_{PRE}) auf Basis von [PI$^+$16]	
Problem	Üblicherweise müssen sämtliche Informationen, die eine Ressource $r \notin R_{List} \cup R_{Act}$ mittels einer Repräsentation offeriert, unabhängig von ihrem Änderungsstatus an die Ressource r übermittelt werden. Allerdings stehen dem Service-Konsumenten S_K zum einen nicht immer alle Informationen zur Verfügung, und es sind nicht immer alle Informationen einer Ressource auch editierbar.
Zwänge	Die Bandbreite sollte möglichst effizient ausgenutzt werden, weshalb nicht geänderte Informationen nicht an die Ressource übermittelt werden sollten. Gleichzeitig muss ersichtlich sein, welche Informationen editierbar sind und welche zur Bearbeitung vonnöten sind.
Lösung	Jede Ressource muss über ein Schemata verfügen, das erforderliche und optionale Attribute sowie weitere Randbedingungen festlegt. Dieses Schemata muss durch Entwicklern von Service-Konsumenten einsehbar sein, sodass sie Entsprechendes implementieren können. Alternativ kann dies auch zur Laufzeit dynamisch erfolgen, sofern das Schemata öffentlich über eine URL erreichbar ist (vgl. *Partial Resource Editing Pattern* [PI$^+$16]). Um nur eine partielle Änderung an einer Ressource vorzunehmen, existieren mittlerweile verschiedene Repräsentationsformate, wie bspw. JSON Merge Patch [RFC7386-2014], JSON Patch [RFC6902-2013] oder XML Patch [RFC7351-2014].
Vorzüge	1) *Benutzbarkeit* ($\leftarrow$ | $\times$): Die Software-Entwickler erhalten ein wohldefiniertes Schema der Ressource über die benötigten Informationen zur Überarbeitung einer Ressource. Weiter können so Fehler in der Anfrage im Vorfeld erkannt und behoben werden. 2) *Mächtigkeit* ($\leftarrow$ | $\times$): Ein Service-Konsument S_K muss nur noch die minimal benötigten Informationen an die Ressource übermitteln.
Lasten	1) *Komplexität* ($\times$ | $\rightarrow$): Bei der Implementierung müssen verschiedene Konstellationen von Anfragen behandelt werden, da nicht mehr davon auszugehen ist, dass alle Informationen einer Ressource übermittelt werden. Weiter muss jeweils ein Schema erstellt werden.
Einsatz	Das Muster wird in [Al10] beschrieben und ist im Rahmenwerk (engl. framework) *Ruby on Rails* angewandt [PI$^+$16].
Beispiel	Der Service-Konsument S_{K_1} sendet eine Anfrage mittels der HTTP-Methode PUT an die zu bearbeitende Ressource und übermittelt alle benötigten Informationen zu deren Bearbeitung. Ein anderer Service-Konsument S_{K_2} nutzt wiederum die HTTP-Methode PATCH um eine partielle Änderung durchzuführen. Da es sich bei dieser HTTP-Methode im Gegensatz zur HTTP-Methode PUT allerdings um eine nicht idempotente Methode handelt (vgl. Tabelle 6.1), müssen ähnliche Vorkehrungen getroffen wie bei P_{RRC}.

Tabelle 9.9: Partial Resource Editing/Creation Pattern (P_{PRE})

Langlaufende Anfragen mit dem Long Running Request Pattern (P_{LRR}) auf Basis von [PI$^+$16]	
Kontext	Ist dem Muster in [PI$^+$16] zu entnehmen.
Problem	
Zwänge	

Langlaufende Anfragen mit dem Long Running Request Pattern (P_{LRR}) auf Basis von [PI$^+$16]	
Lösung	Der Service-Konsument S_K übermittelt die benötigten Informationen zum Ausführen einer langlaufenden Operation an die zuständige Ressource r_{act}, die dann mit dem entsprechenden Identifikator antwortet. Mit dessen Hilfe kann sich ein Service-Konsument S_K über den aktuellen Verarbeitungsstatus informieren oder auch die aktuelle Verarbeitung vorzeitig abbrechen. Falls die langlaufende Operation einen Rückgabewert liefern soll, so kann dieser nach erfolgreicher Verarbeitung anstelle des Verarbeitungsstatus zurückgeliefert werden. Für die Umsetzung empfiehlt es sich, ein entsprechendes Schema der Ressource zu definieren, das die Struktur der letztlichen Repräsentation formal beschreibt.
Vorzüge	1) *Mächtigkeit (*$\leftarrow \mid \times$): Ein Service-Konsument S_K kann langlaufende Operationen ausführen, ohne dass er blockiert wird und kann sich zu jederzeit über den aktuellen Verarbeitungsstatus informieren. 2) *Mächtigkeit (*$\leftarrow \mid \times$): Auslagerung von aufwändigen Operationen des Service-Konsumenten S_K zum Microservice.
Lasten	1) *Komplexität (*$\times \mid \rightarrow$): Die asynchrone Verarbeitung erfordert einen erhöhten Implementierungsaufwand für den Microservice, da u. a. ein separater Prozess gestartet werden muss.
Einsatz	Ist dem Muster in [PI$^+$16] zu entnehmen.
Beispiel	Ist dem Muster in [PI$^+$16] zu entnehmen.

Tabelle 9.10: Long Running Request Pattern (P_{LRR}) für langlaufende Anfragen

Zuverlässiges Anlegen von neuen Ressourcen mit Reliable Resource Creation Pattern (P_{RRC})	
Kontext	Neben der Interaktion mit bestehenden Ressourcen muss ein Service-Konsument S_K auch neue Ressourcen anlegen können. Zum Zeitpunkt des Anlegens der Ressource existiert allerdings noch kein Identifikator, da dieser erst durch den Microservice erzeugt und anschließend der Ressource zugewiesen wird. Des Weiteren kann es sein, dass die Antwort des Microservice den Service-Konsumenten S_K nicht erreicht, sodass dieser annehmen muss, die neue Ressource wäre nicht angelegt worden.
Problem	Alle Netzwerke können als nicht zuverlässig angesehen werden, wodurch die Garantie auf Erhalt einer Antwort nicht besteht. Gleichzeitig kann der Service-Konsument S_K hier den Grund für die fehlende Antwort nicht erkennen [PI$^+$16]. Unklar ist dann, wie zu verhindern wäre, dass mehrere Ressourcen wegen fehlender Antwort angelegt werden.
Zwänge	Bekannte Anwendungsschichtprotokolle bieten verschiedene Methoden zur Interaktion, sodass zuerst eine geeignete Methode für die Erstellung ausgewählt werden muss. Zudem verlangt eine mögliche Dopplung bei der Erstellung von Ressourcen eine aufwändige Fehlerbehandlung beim Service-Konsumenten S_K, weshalb vor dem Anlegen eine Prüfung erfolgen muss [PI$^+$16]. Dieses Muster liefert daher einen Ansatz, wie eine Dopplung von Ressourcen auf Seiten des Microservice vermieden werden kann und nicht von einem konkreten Anwendungsschichtprotokoll abhängt. Eine aufwändige Logik zur Behandlung von etwaigen Fehlern auf Seiten des Service-Konsumenten S_K ist dann nicht mehr notwendig.

Zuverlässiges Anlegen von neuen Ressourcen mit Reliable Resource Creation Pattern (P_{RRC})	
Lösung	Mehrere Lösungsvarianten bearbeiten dieses Problem. Es wird hier auf das Muster in Tabelle 9.12 sowie in Tabelle 9.13 verwiesen.
Vorzüge[16]	1) *Korrektheit* ($\times\vert\rightarrow$): Vermeidung von doppelt angelegten Ressourcen. 2) *Komplexität* ($\leftarrow\vert\times$): Auslagerung der Logik zur Prüfung einer Dopplung auf Seiten des Microservice.
Lasten[17]	1) *Komplexität* ($\times\vert\rightarrow$): Erhöhter Aufwand bei der Implementierung des Microservice, da die Logik zur Prüfung auf Dopplung implementiert werden muss.
Einsatz	Ist der entsprechenden Lösungsvariante zu entnehmen.
Beispiel	

Tabelle 9.11: Reliable Resource Creation Pattern (P_{RRC}) zum zuverlässigen Anlegen von Ressourcen

Zuverlässiges Anlegen von neuen Ressourcen mit dem Token-based Resource Creation Pattern (P_{TRC}) auf Basis von [PI+16]	
Kontext	Ist dem Muster in Tabelle 9.11 zu entnehmen.
Problem	
Zwänge	
Lösung	Der Microservice stellt einen Endpunkt bereit, an dem ein Service-Konsument S_K ein eindeutiges Token erhält. Es kann schließlich zum Anlegen einer Ressource verwendet werden. Falls es nicht bereits in einer vorherigen Anfrage verwendet wurde, wird eine neue Ressource angelegt und diese der entsprechenden r_{list} zugewiesen. Per Antwort quittiert der Microservice das erfolgreiche Anlegen der Ressource. Sollte der Service-Konsument S_K keine Antwort erhalten, wiederholt er seine Anfrage. Falls das Anlegen der Ressource erfolgreich war und lediglich die Antwort verloren ging, wird die erneute Anfrage direkt abgewiesen und dem Service-Konsumenten S_K mitgeteilt, dass das Token schon einmal verwendet wurde. Um etwaige nicht benutzte Tokens zu löschen, sollten diese nur eine beschränkte Zeit ihre Gültigkeit haben.
Vorzüge	Keine weiteren Vorzüge im Vergleich zum abstrakten Muster
Lasten	1) *Komplexität* ($\times\vert\rightarrow$): Der Microservice muss in der Lage sein, ein eindeutiges Token zu generieren sowie dessen Prüfung auf bisheriger Nutzung ermöglichen. Weiter muss ein dedizierter Endpunkt für die Token-Generierung bereitgestellt werden. 2) *Mächtigkeit* ($\leftarrow\vert\times$): Mehrere Anfragen sind notwendig, um eine Ressourcen anzulegen.
Einsatz	Dieses Muster basiert auf zwei existierenden Mustern aus [Al10] gemäß Pautasso et al. [PI+16]. Ein vergleichbarer Einsatz findet sich bei der Online-Überweisung mittels einer Transaktionsnummer, da auch dort diese als ein Token für die einmalige Nutzung generiert wird und nach einer beschränkten Zeit die Gültigkeit verliert.

[16] Die weiteren Vorzüge sind der ausgewählten Lösungsvariante zu entnehmen.
[17] Die weiteren Lasten sind der entsprechenden Lösungsvariante zu entnehmen.

Zuverlässiges Anlegen von neuen Ressourcen mit dem Token-based Resource Creation Pattern (P_{TRC}) auf Basis von [PI$^+$16]	
Beispiel	Der Service-Konsument S_K sendet eine Anfrage mittels der HTTP-Methode GET an den bereitgestellten Endpunkt für die Generierung eines Tokens. Nach seiner Zustellung kann nun mithilfe der HTTP-Methode PUT eine weitere Anfrage gestellt werden. Durch die idempotente Eigenschaft der HTTP-Methode PUT und durch die einmalige Nutzung des Tokens kann eine Dopplung der Ressource ausgeschlossen werden.

Tabelle 9.12: Token-based Resource Creation Pattern (P_{TRC}) zum zuverlässigen Anlegen von Ressourcen

Zuverlässiges Anlegen von neuen Ressourcen mit dem Delayed Content Resource Creation Pattern (P_{DCRC}) auf Basis von [PI$^+$16]	
Kontext	Ist dem Muster in Tabelle 9.11 zu entnehmen.
Problem	
Zwänge	
Lösung	Der Service-Konsument S_K übermittelt eine leere Repräsentation im Sinne der anzulegenden Ressource. Der Microservice erstellt daraufhin die Ressource und sendet den erzeugten Identifikator zurück an den Service-Konsumenten S_K. Anschießend kann dieser mittels einer idempotenten Methode die eigentlichen Informationen der Ressource nachliefern, woraufhin der Service-Konsument S_K die Erstellung entsprechend quittiert. Falls der Service-Konsument S_K keinen Identifikator erhält, wiederholt er seine Anfrage. Dies kann dazuführen, dass mehrere Ressourcen mit leeren Inhalten über einen gewissen Zeitraum existieren können. Sie müssen daher nach einem festgelegten Zeitraum wieder entfernt werden.
Vorzüge	Keine weiteren Vorzüge im Vergleich zum abstrakten Muster
Lasten	1) *Mächtigkeit* ($\leftarrow \mid \times$): Mehrere Anfragen sind notwendig um eine Ressourcen anzulegen. 2) *Korrektheit* ($\times \mid \rightarrow$): Für eine gewisse Zeit existieren leere Ressourcen, bis die zweite Anfrage erfolgt ist. Eine Validierung etwaiger Invarianten ist erst mit der zweiten Anfrage und damit mit der Übermittlung der Informationen möglich.
Einsatz	Dieses Muster findet bspw. in der DayTrader API[18] seinen Einsatz [PI$^+$16].
Beispiel	Der Service-Konsument S_K initiiert eine Anfrage mittels HTTP-POST und einer leeren Repräsentation der anzulegenden $Repr(r)$ an den Endpunkt der r_{list}. Nach dem erfolgreichen Anlegen der Ressource antwortet der Microservice mit dem Identifikator der Ressource. Dieser Identifikator wird schließlich für die zweite Anfrage genutzt, um mittels der idempotenten HTTP-Methode PUT die eigentlichen Informationen zu der schon angelegten Ressource nachzureichen.

Tabelle 9.13: Delayed Content Resource Creation Pattern (P_{DCRC}) zum zuverlässigen Anlegen von Ressourcen

[18] https://bitworking.org/news/2007/06/RESTify-DayTrader#orders-should-be-reliable (Letzter Zugriff: 18.09.2017)

Bedingte Bearbeitung/Erstellung einer Ressource mit dem Conditional Large Resource Non Safe Interaction Pattern (P_{CLRNSI}) auf Basis von [PI$^+$16]	
Kontext	Ist dem Muster in [PI$^+$16] zu entnehmen.
Problem	
Zwänge	
Lösung	Der Service-Konsument S_K übermittelt im Vorfeld an die Änderung oder Erstellung einer Ressource entsprechende Meta-Informationen. Diese Meta-Informationen beschreiben die Anfrage und die zu übermittelnden Informationen, sodass der Microservice entscheiden kann, ob er diese Informationen verarbeiten kann oder nicht. Die Entscheidung wird wiederum dem Service-Konsumenten S_K mitgeteilt.
Vorzüge	1) *Mächtigkeit* ($\leftarrow \mid \times$): Es werden nur Informationen übermittelt, die auch tatsächlich verarbeitet werden können. 2) *Komplexität* ($\times \mid \rightarrow$): Die Auswertung von Meta-Informationen wird heutzutage von vielen Frameworks unterstützt, wodurch der Implementierungsaufwand gering ist.
Lasten	1) *Komplexität* ($\times \mid \rightarrow$): Für die Prüfung der Meta-Informationen muss eine entsprechende Logik implementiert werden
Einsatz	Ist dem Muster in [PI$^+$16] zu entnehmen.
Beispiel	Ist dem Muster in [PI$^+$16] zu entnehmen.

Tabelle 9.14: Conditional Large Resource Non Safe Interaction Pattern (P_{LRR}) für bedingte Bearbeitung/Erstellung einer Ressource

Selektion von Ressourcenfeldern mit dem Resource Field Selection Pattern (P_{RFS})	
Kontext	Ein Service-Konsument S_K benötigt unter Umständen nur bestimmte Informationen einer Ressource, weshalb er die benötigten Informationen einer Ressource mitteilen muss.
Problem	Es ist unklar, wie ein Service-Konsument S_K die benötigten Informationen einer Ressource selektieren kann und wie die verfügbaren Informationen einer Ressource in geeigneter Form offeriert werden können, damit er eine Selektierung vornehmen kann.
Zwänge	Die Bandbreite ist begrenzt, weshalb nur vom Service-Konsumenten S_K benötigte Informationen ihm übermittelt werden sollten. Gerade bei großen zu übertragenden Informationen, wie bspw. kodierten Bilder, ist das wichtig. Zudem werden Verbindungsabbrüche mit zunehmender Übertragungsdauer wahrscheinlicher – insbesondere bei mobilen Endgeräten oder häufig wechselnden Lokationen.
Lösung	Jede Ressource beschreibt ihre Struktur und damit die enthaltenen Informationen mithilfe eines Schemas, das zur Laufzeit ausgewertet werden kann, wenn es über einen Endpunkt erreichbar ist oder bei der Implementierung des Service-Konsumenten S_K herangezogen werden kann. In beiden Fällen erhält letzterer somit die notwendigen Meta-Informationen. Bei der Anfrage an eine Ressource kann der Service- Konsument S_K mittels der Abfrageparameter eine kommaseparierte Liste mit gewünschten Informationen übergeben, auf Basis deren die Selektion erfolgt. Dieses Vorgehen gleicht der Selektion bei Datenbankabfragen.
Vorzüge	*Mächtigkeit* ($\leftarrow \mid \times$): Durch die Selektion von Daten vor der Übertragung wird die Bandbreite effizienter ausgenutzt.

Selektion von Ressourcenfeldern mit dem Resource Field Selection Pattern (P_{RFS})	
	Mächtigkeit ($\leftarrow \mid \times$): Der Service-Konsument S_K legt selbstständig die gewünschten Informationen fest.
Lasten	1) *Komplexität* ($\times \mid \rightarrow$): Die Selektion erfordert einen Implementierungsaufwand auf Seiten des Microservice.
Einsatz	Die Selektion von Informationen findet bspw. Einsatz bei der Facebook Graph API[19], der Google Drive API[20] und findet Erwähnung in [GG+16c].
Beispiel	Der Service-Konsument S_K sendet eine Anfrage mit der HTTP-Methode GET und dem Anfrageparameter `fields` mit einer zugewiesenen kommaseparierten Liste in der Form `attribute1, attribute2,...`. Diese Attribute müssen mit den definierten Attributen im jeweiligen Schema der Ressource übereinstimmen.

Tabelle 9.15: Resource Field Selection (P_{RFS}) zur Selektion von Ressourcenfeldern

Bedingte Anfrage an Ressourcen mit dem Conditional Resource Retrieval Pattern (P_{CoRR})	
Kontext	Der Service-Konsument S_K benötigt die aktuellsten Informationen einer Ressource und sendet daraufhin eine entsprechende Anfrage. Der Microservice hat im Zuge dessen zustandlose Eigenschaft keine Kenntnis darüber, ob der Service-Konsument S_K diese Ressource bereits im Vorfeld angefragt hat.
Problem	Der Microservice übermittelt dem Service-Konsumenten S_K womöglich die gleichen Informationen zu einer Ressource, die der Service-Konsument S_K bereits im lokalen Speicher hält.
Zwänge	Ähnlich zu P_{RFS} sollten auch hier nur die notwendigen Informationen mit Rücksicht auf die verfügbare Bandbreite übermittelt werden.
Lösung	Der Service-Konsument S_K übermittelt mit der Anfrage eine Meta-Information, mit deren Hilfe der Microservice erkennt, ob der Service-Konsument S_K diese Ressource bereits angefragt hat und ob diese sich seit der letzten Anfrage geändert hat. Beispiele derartiger Meta-Informationen sind bspw. ein Zeitstempel oder ein Hashwert.
Vorzüge	*Mächtigkeit* ($\leftarrow \mid \rightarrow$): Es werden Informationen einer Ressource nicht mehrfach übermittelt.
Lasten	*Komplexität* ($\leftarrow \mid \rightarrow$): Der Microservice muss eine entsprechende Logik bereitstellen, um die Meta-Informationen auszuwerten, die der Service-Konsument S_K vorhalten muss.
Einsatz	Das Muster findet Einsatz bei Zalando APIs[21], der Facebook Marketing API[22] und der GitHub API[23].

[19] https://developers.facebook.com/docs/graph-api/using-graph-api (Letzter Zugriff: 09.01.2018)
[20] https://developers.google.com/drive/v3/web/performance (Letzter Zugriff: 09.01.2018)
[21] https://zalando.github.io/restful-api-guidelines/pagination/Pagination.html (Letzter Zugriff: 14.10.2017)
[22] https://developers.facebook.com/docs/marketing-api/etags (Letzter Zugriff: 15.10.2017)
[23] https://developer.github.com/v3/#current-version (Letzter Zugriff: 15.10.2017)

Bedingte Anfrage an Ressourcen mit dem Conditional Resource Retrieval Pattern (P_{CoRR})	
Beispiel	Das HTTP-Headerfeld `Last-Modified` kann verwendet werden, um den letzten Änderungszeitstempel der angeforderten Ressource zu setzen. Durch dessen Setzen im HTTP-Headerfeld `If-Modified-Since` für anstehende Anfragen an die Ressource kann der Server feststellen, ob der Service-Konsument S_K bereits über die neueste Version verfügt. Wenn ja, sendet der Service einen entsprechenden HTTP-Statuscode 304 ("Not modified") ohne Nachrichteninhalt. Sollte es hingegen nicht der Fall sein, wird die entsprechende Repräsentation der Ressource übertragen. Ähnliches ist mit sogenannten *ETags* realisierbar, die statt eines Änderungszeitstempels einen Identifikator nutzen, der den Zustand einer Ressource zum Zeitpunkt x genau identifiziert.

Tabelle 9.16: Conditional Resource Retrieval Pattern (P_{CoRR})

A.8 JSON-Schema für den Aufbau einer Fehlermeldung

Das folgende JSON-Schema (vgl. Quelltext 9.1) zeigt die Repräsentation eines Fehlers im Kontext einer ressourcenorientierten Web-API. Die Repräsentation orientiert sich dabei an Abschnitt 5.4.3.

```
1  {
2    "definitions": {},
3    "schema": "http://json-schema.org/draft-06/schema#",
4    "properties": {
5      "detail": {
6        "description": "Specific description of the error with reference
             to the actual occurrence",
7        "title": "Specific description of the error",
8        "type": "string"
9      },
10     "instance": {
11       "description": "Reference to specific information on the error
             that has occurred",
12       "title": "Further information",
13       "type": "string"
14     },
15     "status": {
16       "description": "HTTP status code of the error that occurred",
17       "maximum": 100,
18       "minimum": 599,
19       "title": "HTTP status code",
20       "type": "integer"
21     },
22     "description": {
23       "title": "Short description of the error",
```

```
24        "description": "Short description of the error without reference
              to the actual occurrence",
25        "type": "string"
26      },
27      "type": {
28        "description": "Reference to further information on the error
              type",
29        "title": "Reference to the error type",
30        "type": "string"
31      }
32    },
33    "required": [
34      "detail",
35      "status",
36      "title"
37    ],
38    "type": "object"
39  }
```

Quelltext 9.1: JSON-Schema für den Aufbau einer Fehlermeldung

A.9 Muster zur Versionierung einer Web-API

Die Beschreibung der Muster zur Versionierung orientieren sich an der Musterbeschreibungsvorlage in Tabelle 9.1 sowie den eingeführten Variablen in Tabelle 5.7 orientiert, während die Tabelle 9.2 zur Interpretation der Einflüsse dient. Für die letztliche Umsetzung der Versionierung im Kontext einer Web-API dienen die nachfolgenden Muster P_{UV} und P_{CTV}. Auf ein abstraktes Muster ähnlich zu dem Muster P_{RCT} (vgl. Tabelle 9.3 auf Seite 215) wurde hier verzichtet, da es keinen Mehrwert liefert.

Versionierung mit dem URL-based Versioning Pattern (P_{UV})	
Kontext	Im Lauf der Zeit kann sich die Web-API durch unterschiedliche Faktoren verändern, was wiederum die Service-Konsumenten S_K beeinflusst. Gleichzeitig müssen diese angeben können mit welcher Version einer Web-API sie kompatibel sind.
Problem	Service-Konsumenten S_K können mit unterschiedlichen Versionen einer Web-API interoperabel sein. Aus diesem Grund muss die Möglichkeit bestehen, dass ein Service-Konsument S_K die unterstützte Version auswählen kann, wodurch letztlich die korrekte Instanz des Microservice für die Interaktion adressiert wird.

Versionierung mit dem URL-based Versioning Pattern (P_{UV})	
Zwänge	Es existiert ein Vorgehen (vgl. Abschnitt 5.5.2) um nicht abwärtskompatible Änderungen an einer Web-API in geeigneter Form zu behandeln, wodurch mehrere *Major*-Versionen von ihr gleichzeitig existieren können. Ein Service-Konsument S_K unterstützt in der Regel eine *Major*-Version einer veräußerten Web-API und ist mit dieser interoperabel. Wenn mehrere Versionen einer Web-API existieren, muss der Service-Konsument S_K die interoperable Version auswählen können. Falls irrtümlich eine neuere oder ältere Version adressiert wird, kann eine reibungslose Interaktion nicht gewährleistet werden, da einer *Major*-Versionen mit ihrer Vorgängerversion nicht abwärtskompatibel ist.
Lösung	Der Service-Konsument übermittelt mit der Anfrage die unterstützte *Major*-Version der Web-API des Microservice als Bestandteil der URL. Ein externes Vermittlungssystem oder der entsprechende Microservice selbst wertet diese Information aus und verarbeitet die Anfrage passend zur übermittelten *Major*-Version.
Vorzüge	1) *Interoperabilität (*$\leftarrow |\times$*)*: Der Service-Konsument kann selbstständig die unterstützte *Major*-Version auswählen. 2) *Benutzbarkeit (*$\leftarrow |\times$*)*: Die verwendete *Major*-Version ist in der URL für Software-Entwicklern von Service-Konsumenten sofort ersichtlich.
Lasten	1) *Komplexität (*$\times|\rightarrow$*):* In der Regel ist ein zusätzliches Vermittlungssystem notwendig, das die Auswertung und Weiterleitung an die entsprechende Instanz des Microservice durchführt.
Einsatz	Der Einsatz des Musters findet Anwendung bei der Facebook Graph API[24], Saleforce Chatter[25] oder PayPal REST APIs[26].
Beispiel	Der Service-Konsument S_K sendet eine Anfrage an die URL `.../v3/points-of-interest`, wodurch die dritte *Major*-Version der Web-API adressiert wird. Die konkrete Adressierung der Microservice-Instanz mit der Web-API-Version v3 wird durch ein Vermittlungssystem bereitgestellt.

Tabelle 9.17: URL-based Versioning Pattern (P_{UV}) zur Versionierung einer Web-API

Versionierung mit dem Content Type Versioning Pattern (P_{CTV})	
Kontext	.
Problem	Ist identisch mit dem Muster in Tabelle 9.17
Zwänge	

[24] https://developers.facebook.com/docs/graph-api/using-graph-api/ (Letzter Zugriff: 27.09.2017)
[25] https://resources.docs.salesforce.com/208/latest/en-us/sfdc/pdf/salesforce_chatter_rest_api.pdf (Letzter Zugriff: 27.09.2017)
[26] https://developer.paypal.com/docs/api/ (Letzter Zugriff: 29.09.2017)

Versionierung mit dem Content Type Versioning Pattern (P_{CTV})	
Lösung	Der Service-Konsument S_K übermittelt mit der Anfrage entsprechende Meta-Informationen, welcher Inhaltstyp (engl. content type) der Repräsentation und welche Version des Inhaltstyps unterstützt wird. Bei dem Inhaltstyp kann es sich bspw. um XML oder JSON handeln. Die Version des Inhaltstyps entspricht dagegen der *Major*-Version der unterstützen Web-API, oder der entsprechende Microservice wertet diese Information aus und verarbeitet die Anfrage entsprechend der übermittelten *Major*-Version.
Vorzüge	1) *Interoperabilität* ($\leftarrow|\times$): Der Service-Konsument S_K kann selbstständig die Version auswählen, mit welcher er interopabel ist. 2) *Mächtigkeit* ($\leftarrow|\times$): *Major*-Version kann sich auf einzelne Ressourcen einer Web-API beziehen und ist somit feingranularer im Vergleich zu (P_{UV}).
Lasten	1) *Komplexität* ($\times|\rightarrow$): In der Regel ist ein zusätzliches Vermittlungssystem notwendig, das die Auswertung und Weiterleitung an die entsprechende Instanz des Microservice durchführt. 2) *Benutzbarkeit* ($\leftarrow|\times$): Die *Major*-Version ist nicht sofort erkennbar, sondern erst bei Analyse der Header-Informationen.
Einsatz	Die Versionierung über den Inhaltstyp einer Nachricht findet bei der GitHub API[27] seine Anwendung.
Beispiel	Der Service-Konsument S_K setzt das HTTP-Headerfeld *Accept* mit `application/edu.smartcampus.v3+json`, wodurch dem Microservice mitgeteilt wird, dass der Inhaltstyp JSON und die dritte *Major*-Version von dem entsprechendem Service-Konsumenten S_K unterstützt und auch erwartet wird.

Tabelle 9.18: Content Type Versioning Pattern (P_{CTV}) zur Versionierung einer Web-API

A.10 Beispielhafte Anwendung der benötigten Interaktionsmuster für einen Service-Konsumenten

Die Abbildung 9.7 repräsentiert eine beispielhafte Anwendung von Mustern für die Listenressource *PointOfInterest*. Die Nutzeranforderung war die Grundlage für die Identifizierung relevanter Muster aus der MPL:

> **Nutzeranforderung:** Der Studierende möchte wegen seiner körperlichen Einschränkungen nur POIs eines bestimmten Typs auswählen können, die für ihn geeignet sind. Die POIs sollen nach der Entfernung zu seinem aktuellen Standort sortiert angezeigt werden.

Aus der Nutzeranforderung lässt sich ableiten, dass das Domänenobjekt und damit die letztliche *PointOfInterest*-Ressource adressiert wird. Entsprechend des Ressourcenmodells R_1 bzw. R_2 handelt es sich dabei um eine Listenressource, weshalb die MPL hinsichtlich dieser Dimension gefiltert

[27] https://developer.github.com/v3/#current-version (Letzter Zugriff: 27.09.2017)

werden kann. Zusätzlich handelt es sich bei der Nutzeranforderung um einen lesenden Zugriff, was einer Operation mit dem Stereotyp `Read` entspricht. Laut Tabelle 5.12 ergeben sich daher die potentiell anwendbaren Muster P_{RFS}, P_{CRR}, P_{RCT} , P_{CaRR}, P_{ORCT}, P_{CRCT}, P_{CRR}, P_{RCF} und P_{RCS}. Weiter lässt die Nutzeranforderung erkennen, dass dem Nutzer eine Filterung und eine Sortierung ermöglicht werden sollen, was den Mustern P_{RCF} und P_{RCS} entspricht.

Die Entscheidung auf Anwendung der Muster obliegt dem Software-Architekten, weshalb sie hier als Musterkandidaten bezeichnet werden. Falls die POIs über mehrere virtuellen Seiten auf dem Service-Konsumenten verteilt werden sollen, um bspw. *Lazy Load* zu ermöglichen und damit die Ladezeiten zu reduzieren [Fo02a], wären auch das abstrakte Muster P_{RCT} sowie deren Verfeinerungen entsprechende Musterkandidaten. Sobald sich der Software-Architekt für die umzusetzenden Muster entschieden hat, werden diese mittels UML-Kommentare ergänzt.

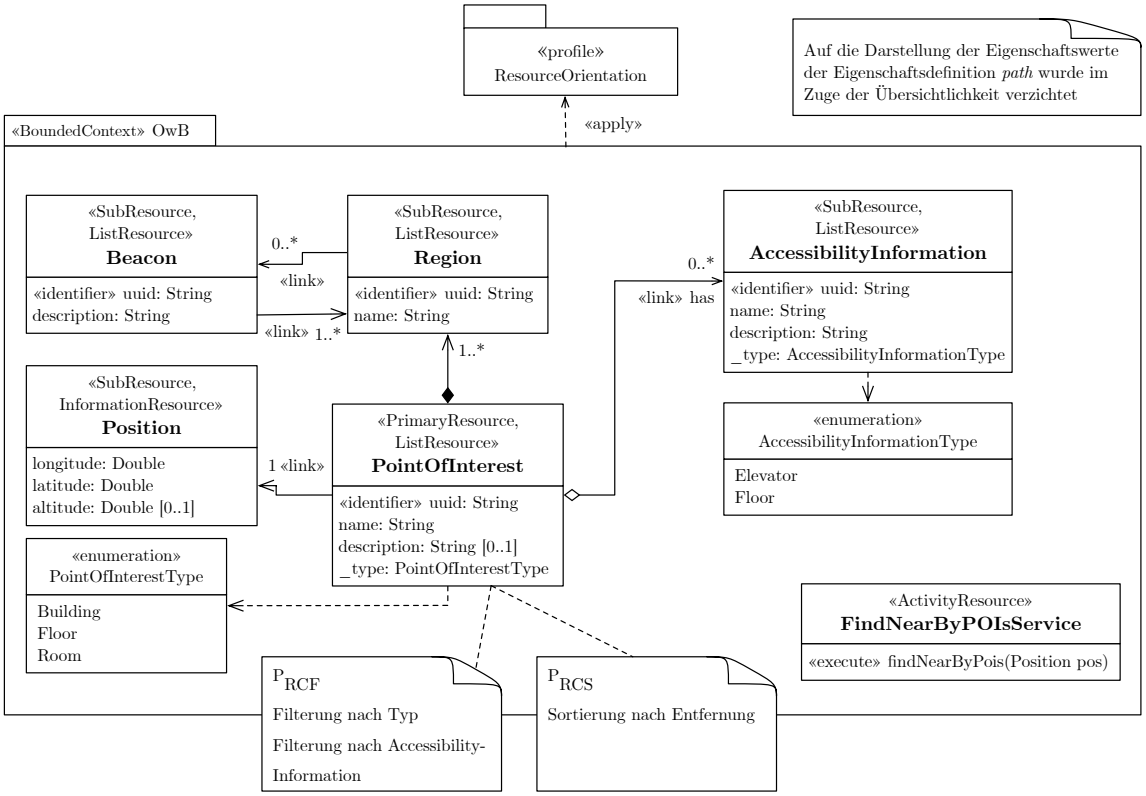


Abbildung 9.7: Ressourcenmodell R_3 nach beispielhafter Anwendung der MPL

A.11 Aufbau einer OAI-Spezifikation

Mit diesem Abschnitt werden die wichtigsten Bestandteile einer OAI-Spezifikation im Kontext dieser Arbeit entsprechend der Version 3.0.0 erläutert. Die einzelnen Bestandteile werden durch entspre-

chende Attribute der entsprechenden Sprache (YAML oder JSON) repräsentiert. Für weiterführende Informationen wird auf die offizielle Spezifikation [OAI-SPEC] verwiesen.

Zum einfacheren Verständnis findet sich nachfolgend eine beispielhafte OAI-Spezifikation mittels der YAML (vgl. Quelltext 9.2), während die Tabelle 9.19 einen entsprechenden Bezug darauf nimmt und eine kurze Erläuterung zu den einzelnen Attributen liefert. Mit der Spalte *Pflicht* wird gekennzeichnet, ob es sich um ein Pflichtattribut handelt. Daneben wird mit der Spalte *Kategorie* eine logische Gruppierung der Attribute vorgenommen.

```
1  openapi: "3.0.0"
2  info:
3    description: "Kurze Beschreibung des abgebildeten
         Geschäftsausschnitts."
4    version: "1.0.0"
5    title: "Bezeichnung der Web-API"
6    contact:
7      name: "Support-Team"
8      email: "support@sampledomain.com"
9  servers: []
10 components:
11   schemas: {}
12   responses: {}
13   parameters: {}
14   examples: {}
15   headers: {}
16 paths:
17   /<pfad>:
18     <http-methode>:
19       summary: "Kurze Beschreibung der Operation"
20       description: "Beschreibung der Operation"
21       responses:
22         <status code>:
23           description: "Beschreibung der Antwort"
```

Quelltext 9.2: Beispielhafte OAI-Spezifikation (Version 3.0.0)

Zeile	Attribut	Pflicht	Kategorie	Beschreibung
1	`openapi`	Ja	Spezifikation	Versionskennung der OAI-Spezifikationssprache nach SemVer
2	`info`	Ja	Meta-Informationen	Container für Meta-Informationen der Web-API
3	`description`	Ja	Meta-Informationen	Kurze Beschreibung der Web-API

Zeile	Attribut	Pflicht	Kategorie	Beschreibung
4	`version`	Ja	Meta-Informationen	Versionskennung der Web-API nach SemVer
5	`title`	Ja	Meta-Informationen	Bezeichnung der Web-API
6-8	`contact`	Nein	Meta-Informationen	Informationen über den Verantwortlichen des Microservice u.a. bestehend aus `name` und `email`
9	`servers`	Nein	Betrieb	Container für Informationen zu den lauffähigen Instanzen des Microservice
10	`components`	Nein	Komponenten	Container für viele wiederverwendbaren Komponenten einer OAI-Spezifikation wie bspw. Schemata für Repräsentationen, Parameter oder auch Beispiele
11	`schemas`	Nein	Komponenten	Wiederverwendbare Schemata, wie bspw. Quelltext 9.1
12	`responses`	Nein	Komponenten	Wiederverwendbare Antworten
13	`parameters`	Nein	Komponenten	Wiederverwendbare Anfrageparameter
14	`examples`	Nein	Komponenten	Wiederverwendbare Beispiele
15	`headers`	Nein	Komponenten	Wiederverwendbare HTTP-Headerinformationen
16	`paths`	Ja	Interaktionen	Container für bereitgestellten Endpunkte durch die Web-API
17	`/<pfad>`	Nein	Interaktionen	Festlegung des Pfades, auf dem eine Ressource adressiert werden kann. Der `<pfad>` ist hier als Platzhalter zu interpretieren
18	`<http-methode>`	Nein	Interaktionen	Festlegung der HTTP-Methoden, die unter dem zuvor gewählten Pfad verfügbar sind. Ähnlich zur vorherigen Beschreibung ist `<http-methode>` als Platzhalter zu interpretieren
19	`summary`	Nein	Interaktionen	Kurze Beschreibung der durchzuführenden Operation
20	`description`	Nein	Interaktionen	Beschreibung der durchzuführenden Operation
21	`responses`	Ja	Interaktionen	Container für mögliche Antworten auf eine Anfrage
22	`<status code>`	Nein	Interaktionen	HTTP-Statuscode, der mit einer Antwort auf eine Anfrage assoziiert ist
23	`description`	Ja	Interaktionen	Beschreibung der Antwort

Tabelle 9.19: Beschreibung wichtiger Bestandteile einer OAI-Spezifikation

Unter Verwendung des Werkzeugs *Swagger UI*[28] in der aktuellsten Version lässt sich eine grafische Repräsentation der Web-API (vgl. Abbildung 9.8) auf Grundlage der OAI-Spezifikation generieren, die zudem eine direkte Interaktion mit der Web-API ermöglicht. Letzteres erfordert allerdings zumindest eine lauffähige Version des mit der Web-API assoziierten Microservice sowie Informationen in Zeile 9. Um die Validität der OAI-Spezifikation zu gewährleisten, wurden die notwendigen Platzhalter (`<http-methode>` und `<status-code>`) durch einen validen Wert ersetzt.

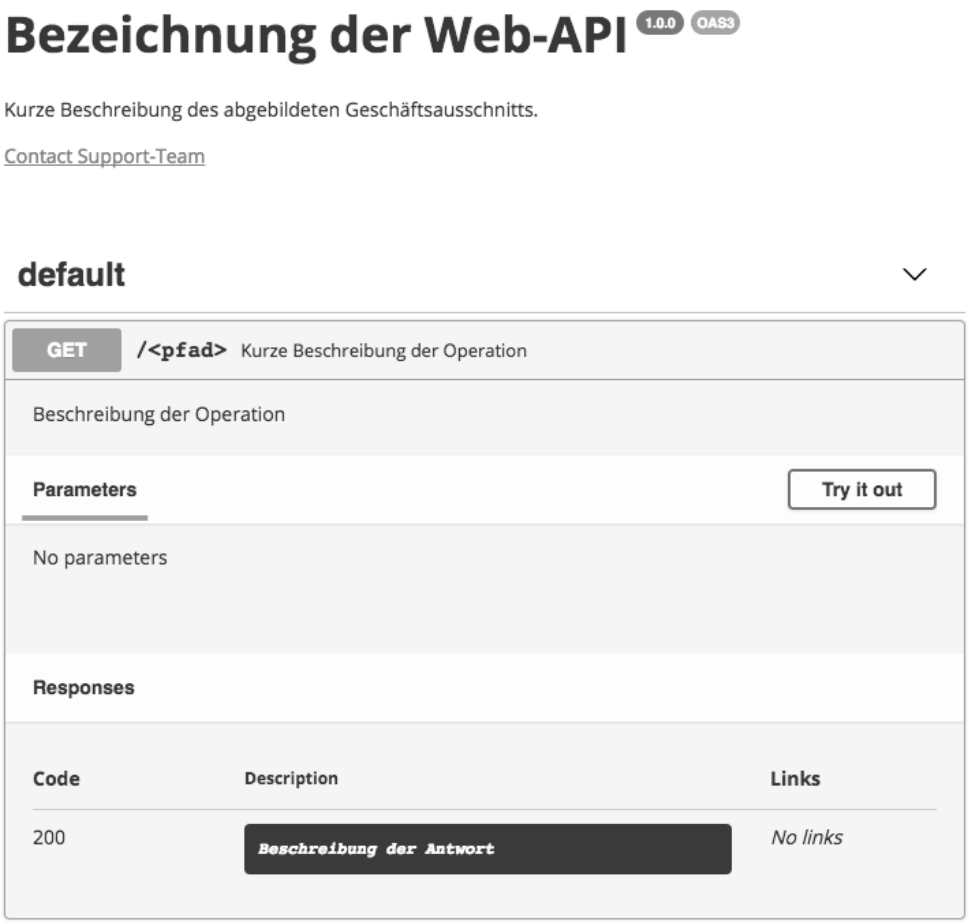


Abbildung 9.8: Visuelle Darstellung einer beispielhaften OAI-Spezifikation mithilfe Swagger UI

A.12 Screenshots von *FIVE touch*, *FIVE flow* und der Anwendung zur Erfassung von Störungen in der Produktion

Dieser Abschnitt liefert mit den Abbildungen 9.9, 9.10 und 9.11 entsprechende Einblicke von *FIVE touch*, das im Zuge der Entwicklung von *FIVE flow* (vgl. explorativer UI-Prototyp in Abbildung 9.12 und 9.13) an dessen anwendungsspezifisches Gateway angepasst wird. *FIVE touch* wird auf einer eigens entwickelten Plattform basierend auf Android mithilfe von Mobile Device Management (MDM)

[28] https://swagger.io/swagger-ui/ (Letzter Zugriff: 02.10.2017)

verteilt. Die entwickelte Plattform setzt sich dabei aus einer dedizierten Stromversorgung mittels austauschbarer Akkus sowie Near Field Communication (NFC)-Scanner zusammen.

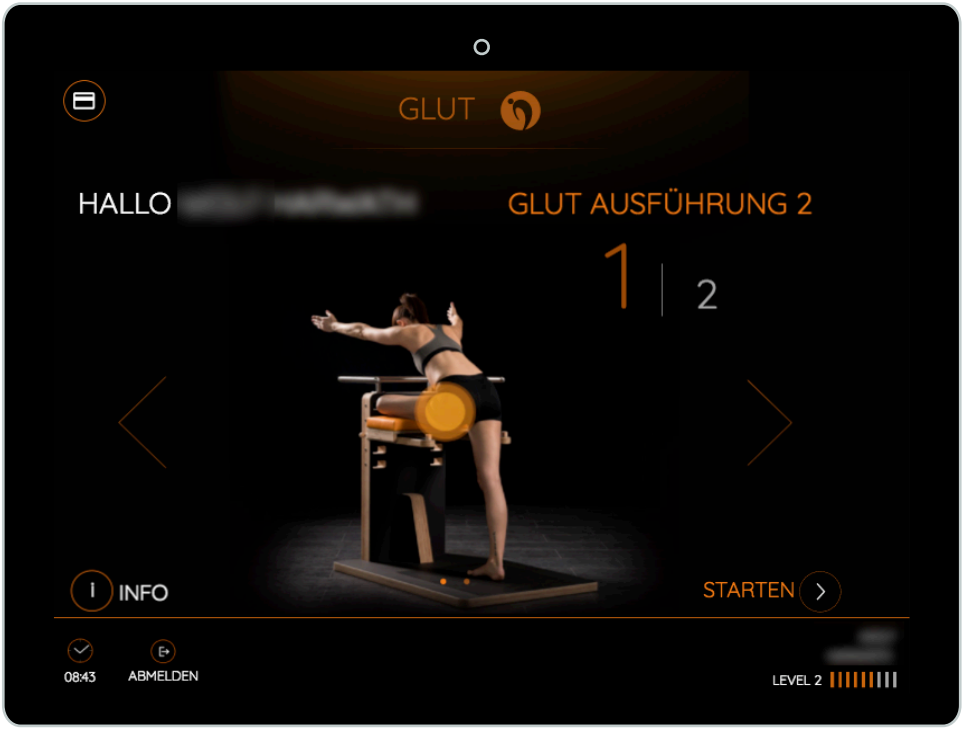


Abbildung 9.9: Screenshot der Startseite von *FIVE touch* in der Version 1.0.0

Abbildung 9.14 und 9.15 liefern zwei entsprechende Eindrücke vom entwickelten Service-Konsumenten in Form einer Anwendung, der zum einen als hybride Anwendung für Apple iOS-Endgeräte (>= iOS 9) und zum anderen als Web-Anwendung für Desktop-Systeme verteilt wurde. Beide Anwendungen basieren auf der gleichen Quellcode-Basis und unterscheiden sich lediglich in der Ausrichtung auf die jeweilige Zielplattform.

A.13 Bewertungsbogen für die empirische Studie im Industriekontext

Der Bewertungsbogen umfasst acht Seiten und wurde im Hinblick auf die anvisierten Vorzüge dieser Arbeit entworfen (vgl. Abbildung 9.16, 9.17, 9.18 und 9.19). Dabei wurde u. a. darauf geachtet, dass mehrere Fragen existieren, deren möglichen Antworten ergänzend oder substraktiv wirken, um die Bedrohung der Validität dahingehend zu reduzieren (vgl. Abschnitt 7.5.1). Die Ergebnisse des Bewertungsbogens sind den durchgeführten Studien in Abschnitt 7.6 zu entnehmen.

Abbildung 9.10: Screenshot eines aktuellen Trainings von *FIVE touch* in der Version 1.0.0

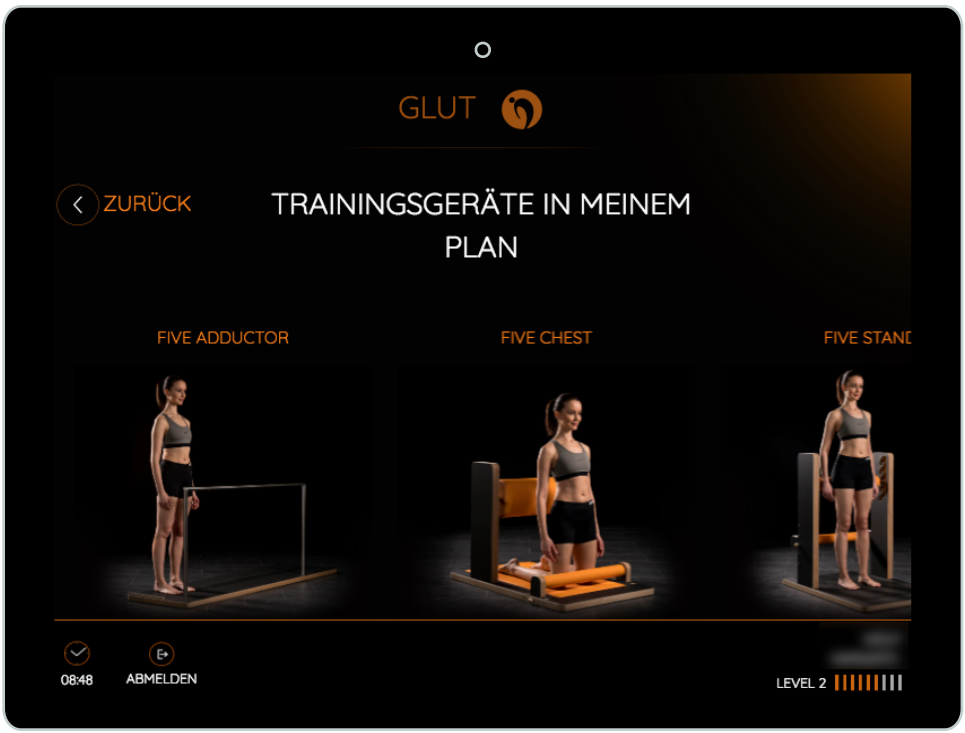


Abbildung 9.11: Screenshot des Trainingsplans von *FIVE touch* in der Version 1.0.0

Abbildung 9.12: Screenshot der Startseite des explorativen UI-Prototyps von *FIVE flow*

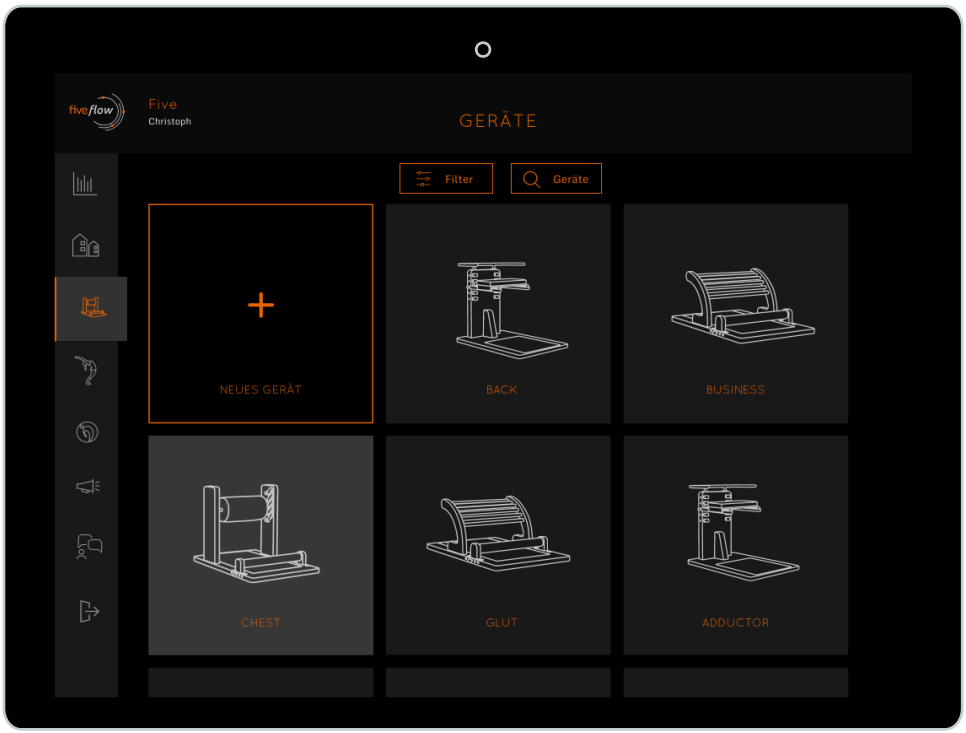


Abbildung 9.13: Screenshot der Geräteübersicht des explorativen UI-Prototyps von *FIVE flow*

Abbildung 9.14: Mandantenfähige Übersicht der erfassten Störungen in einem Produktionswerk

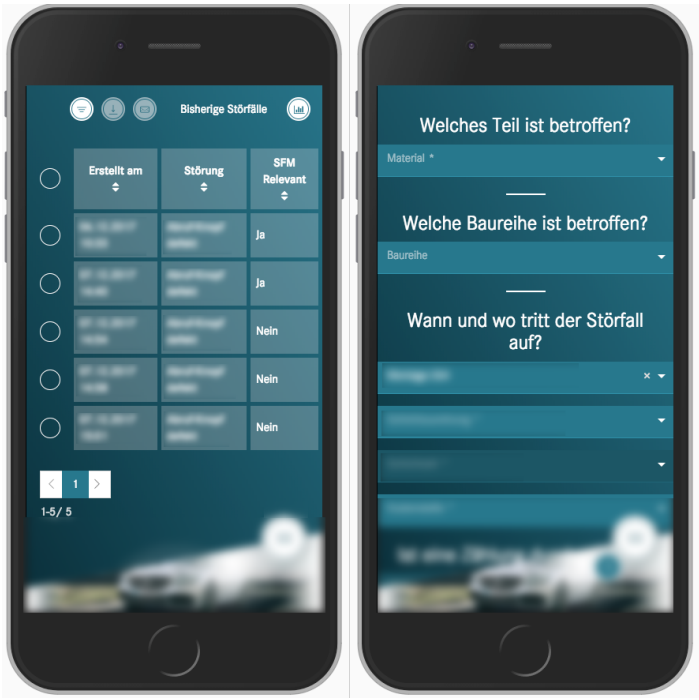


Abbildung 9.15: Übersicht der erfassten Störungen innerhalb eines Produktionswerks auf einem mobilen Endgerät sowie Auszug der Oberfläche zur Erfassung neuer Störungen

Bewertungsbogen – Domänengetriebener Entwurf von ressourcenorientierten Microservices

Einleitung

Sehr geehrte Teilnehmerin, sehr geehrter Teilnehmer,

im Rahmen meiner Promotion beschäftige ich mich in der Forschungsgruppe Cooperation & Management (C&M, Lehrstuhl von Prof. Abeck) des Karlsruher Instituts für Technologie (KIT) mit dem domänengetriebenen Entwurf von ressourcenorientierten Microservices. Das Ziel dieser wissenschaftlichen Arbeit ist die Entwicklung eines systematischen und nachvollziehbaren Vorgehens, um auf Grundlage des Ausschnitts einer Domäne ressourcenorientierte Microservices abzuleiten, der eine hohe Wiederverwendbarkeit aufweist, wodurch eine Integration durch die letztlichen Service-Konsumenten erleichtert wird.

Die zentralen Aspekte der Arbeit stellen dabei die Modellierung eines Domänenmodells nach dem Ansatz von Evans [Ev13] sowie ein Vorgehen zur Überführung des Domänenmodells in ein Ressourcenmodell, anhand dessen letztlich die Web-API-Spezifikation abgeleitet werden kann. Die Unterstützung des Software-Architekten bei der Überführung im HInblick auf die Qualität der Web-API stellt dabei ein Schwerpunkt dar.

Um die von mir erarbeiteten Konzepte zu validieren, lade ich Sie herzlich dazu ein, im Nachgang an Ihr durchgeführtes Projekt diese mit den nachfolgenden Fragen zu bewerten.

Für Ihre Teilnahme danke ich Ihnen im Voraus recht herzlich. Bei technischen oder inhaltlichen Fragen stehe ich Ihnen selbstverständlich geme zur Verfügung.

Dauer der Umfrage: 20-25 min
Anzahl der Fragen: 27

Informationen zum Datenschutz:
Die Bewertung ist verschlüsselt und wird anonym durchgeführt. Es können keine Rückschlüsse auf die Teilnehmenden gezogen werden. Es werden insbesondere keine IP-Adressen gespeichert.

Mit freundlichen Grüßen
Pascal Giessler

[Ev13] Evans, Eric: Domain-Driven Design: Tackling Complexity In the Heart of Software, 2013.

Wie sehr sind Sie mit dem Konzept des domänengetriebenen Entwurfs nach Evans [Ev13] vertraut? *

[Ev13] Evans, Eric: Domain-Driven Design: Tackling Complexity In the Heart of Software, 2013

◯ sehr vertraut ◯ vertraut ◯ bedingt vertraut ◯ wenig vertraut ◯ nicht vertraut

Wie sehr sind Sie mit dem Entwurf von Microservices vertraut? *

◯ sehr vertraut ◯ vertraut ◯ bedingt vertraut ◯ wenig vertraut ◯ nicht vertraut

Wie sehr sind Sie mit dem Architekturstil REST vertraut? *

◯ sehr vertraut ◯ vertraut ◯ bedingt vertraut ◯ wenig vertraut ◯ nicht vertraut

Wie wichtig ist Ihnen/Ihrem Unternehmen die Qualität einer Web-API? *

◯ sehr hoch ◯ hoch ◯ moderat ◯ niedrig ◯ sehr niedrig

Durchführung

Beschreiben Sie das Projekt, bei der die erarbeiteten Konzepte Ihre Anwendung gefunden haben, in drei bis vier Sätzen. *

Welche Rolle begleiteten Sie in dem Projekt? *

Abbildung 9.16: Bewertungsbogen für die empirische Studie zur Validierung des domänengetriebenen Entwurfsprozesses (Seite 1/2 von 8)

Wieviel Personen waren am Projekt beteiligt? *

- ◯ 1 Person
- ◯ 2-4 Personen
- ◯ 5-10 Personen
- ◯ Mehr als 10 Personen

Durchführung - Modellierung

Die nachfolgenden Fragen widmen sich der Modellierung eines Domänenmodells, anhand dessen ein Ressourcenmodell abgeleitet wird.

Konnte mittels der bereitgestellten Arbeit selbständig eine Modellierung durchgeführt werden? *

- ◯ ja
- ◯ ja (mit anfänglicher Unterstützung)
- ◯ nein (benötigt mehr Hintergrundwissen)
- ◯ nein

Wenn nein, welches Domänenwissen konnte nicht abgebildet werden?

Wie bewerten Sie das Vorgehen zur Modellierung und die resultierenden Modellierungsartefakte? *

	sehr hoch	hoch	moderat	niedrig	sehr niedrig
Verständlichkeit	◯	◯	◯	◯	◯
Nachvollziehbarkeit	◯	◯	◯	◯	◯
Praktische Anwendbarkeit	◯	◯	◯	◯	◯
Abstraktion	◯	◯	◯	◯	◯

Weche Verbesserungsvorschläge existieren für das Vorgehen der Modellierung und die resultierenden Artefakte?

Durchführung - Überführung des Domänenmodells

Die nachfolgenden Fragen widmen sich der Überführung des Domänenmodells in ein Ressourcenmodells, anhand dessen letztliche API-Spezifikation abgeleitet wird.

Welche Wichtigkeit wird eine ("gute") Web-API in Ihrem Unternehmen zuteil? *

◯ sehr wichtig ◯ wichtig ◯ eher nicht wichtig ◯ nicht wichtig

Erläutern Sie in zwei bis drei Sätzen, wie bislang eine Web-API entworfen wurde? *

Abbildung 9.17: Bewertungsbogen für die empirische Studie zur Validierung des domänengetriebenen Entwurfsprozesses (Seite 3/4 von 8)

Welche Qualitätsaspekte fördern die Wiederverwendbarkeit einer Web-API? *

Fehlende Qualitätsaspekte können ergänzt werden

	sehr wichtig	wichtig	eher nicht wichtig	nicht wichtig	nicht beurteilbar
Konsistenz	◯	◯	◯	◯	◯
Vermeidung von Fehlern	◯	◯	◯	◯	◯
Bezug zur abgebildeten Domäne	◯	◯	◯	◯	◯
Effiziente Nutzung von Ressourcen	◯	◯	◯	◯	◯
Evolutionäre Stabilität	◯	◯	◯	◯	◯
Keine technologischen Vorgaben	◯	◯	◯	◯	◯
Bereitstellung einer Dokumentation	◯	◯	◯	◯	◯
Nutzungsspezifische Ausrichtung	◯	◯	◯	◯	◯
	◯	◯	◯	◯	◯

Wie bewerten Sie die Gesamtheit des beschriebenens Vorgehens zur Überführung in ein Ressourcenmodell? *

	sehr niedrig	niedrig	moderat	hoch	sehr hoch	nicht beurteilbar
Systematik	◯	◯	◯	◯	◯	◯
Nachvollziehbarkeit	◯	◯	◯	◯	◯	◯
Verständlichkeit	◯	◯	◯	◯	◯	◯
Abstraktion von einer konkreten Domäne	◯	◯	◯	◯	◯	◯
Technologie-Agnostizität	◯	◯	◯	◯	◯	◯
Entscheidungsunterstützung	◯	◯	◯	◯	◯	◯
Praktische Anwendbarkeit	◯	◯	◯	◯	◯	◯

Wie hoch sehen Sie die Nützlichkeit der beschriebenen Muster für den Entwurf einer Web-API? *

◯ sehr hoch ◯ hoch ◯ moderat ◯ niedrig ◯ sehr niedrig

Inwieweit haben die Muster den Entwurf unterstützt? Begründen Sie in drei bis fünf Sätzen. *

Bewerten sie die Qualität der resultierenden Web-API im Vergleich zu ihrem bisherigen Vorgehen? Begründen Sie in zwei bis drei Sätzen. *

Konnten die Review-Zyklen des Entwurfs einer Web-API verringert werden? *

◯ ja, deutlich
◯ ja, die Review-Zyklen konnten gesenkt werden
◯ nein, diese sind gleich geblieben
◯ nicht beurteilbar, da wir kein API-Review einsetzen

Konnten etwaige Service-Konsumenten schneller entwickelt werden im Vergleich zu Ihren bisherigen Web-APIs? *

◯ ja, sehr sogar
◯ ja, zu einem gewissen Grad
◯ nein, bislang kein positiver Einfluss ersichtlich
◯ nicht beurteilbar, da noch kein Service-Konsument entwickelt wurde
◯

Abbildung 9.18: Bewertungsbogen für die empirische Studie zur Validierung des domänengetriebenen Entwurfsprozesses (Seite 5/6 von 8)

Für was wird die resultierende API-Spezifikation in Ihrem Unternehmen eingesetzt? *

- ☐ Interaktive Dokumentation
- ☐ Generierung von Testfällen
- ☐ Generierung der Implementierung
- ☐ Auffindbarkeit im Kontext von API-Management
- ☐ Kommunikation zwischen verschiedenen Entwicklungsteams
- ☐ _____

Weche Verbesserungsvorschläge existieren für die Überführung des Domänenmodells in ein Ressourcenmodell?

Statistik

Seit wie vielen Jahren sind Sie im Bereich der Softwareentwicklung tätig? *

- ◯ seit weniger als 1 Jahr
- ◯ seit 1-3 Jahren
- ◯ seit 3-5 Jahren
- ◯ seit 5-8 Jahren
- ◯ seit 8-10 Jahren
- ◯ seit mehr als 10 Jahren

Welche Rolle begleiten Sie in Ihrem Unternehmen/Ihrer Institution? *

- ◯ Software-Entwickler (Junior)
- ◯ Software-Architekt (Junior)
- ◯ Software-Entwicker (Senior)
- ◯ Software-Architekt (Senior)
- ◯ _____

Bitte beschreiben Sie Ihre Rolle in maximal zwei Sätzen.

Wie viele Web-APIs werden derzeit in Ihrem Unternehmen verwaltet *

- ◯ Mehr als 10 Web-APIs
- ◯ 5-9 Web-APIs
- ◯ 2-4 Web-APIs
- ◯ 1 Web-API
- ◯ Keine

Wie hoch sehen Sie die praktische Relevanz der erzielten Beiträge dieser Arbeit?

- ◯ sehr hoch
- ◯ hoch
- ◯ moderat
- ◯ niedrig
- ◯ sehr niedrig

Feedback

Was möchten Sie mir noch mitteilen? Hier finden Sie Platz für Anregungen, Wünsche, Kommentare oder Kritik.

Abbildung 9.19: Bewertungsbogen für die empirische Studie zur Validierung des domänengetriebenen Entwurfsprozesses (Seite 7/8 von 8)

B Abkürzungsverzeichnis

AIS AccessibilityInfoService.

API Application Programming Interface.

AWS Amazon Web Services.

BDD Behaviour Driven Development.

BFF Backend For Frontend.

BNF Backus-Naur-Form.

BPMN Business Process Model and Notation.

C&M Cooperation & Management.

CD Continuous Delivery.

CDI Context Dependency Injection.

CI Continuous Integration.

CIM Computation Independent Model.

COAP Constrained Application Protocol.

CQRS Command-Query Responsibility Segregation.

CRUD Create-Read-Update-Delete.

DDD Domain Driven Design.

DSL Domain Specific Language.

DSML Domain Specific Modeling Language.

DTO Data Transfer Object.

DX Developer Experience.

EJB Enterprise Java Bean.

EMF Eclipse Modeling Framework.

FCM Factor-Criteria-Metrics.

FIQL The Feed Item Query Language.

GPS Global Positioning System.

HATEOAS Hypermedia As The Engine Of Application State.

HTTP HyperText Markup Language.

IAM Identity and Access Management.

IANA Internet Assigned Numbers Authority.

IOSB Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung.

ISO International Organization for Standardization.

IT Informationstechnologie.

Java EE Java Enterprise Edition.

JAX-RS Java API for Representational State Transfer.

JAXB Java Architecture for XML Binding.

JPA Java Persistence API.

JSON JavaScript Object Notation.

KIT Karlsruher Institut für Technologie.

KPI Key Performance Indicator.

M2M Modell-zu-Modell-Transformation.

M2T Modell-zu-Text-Transformation.

.

MDM Mobile Device Management.

MDSD Model Driven Software Development.

MIME Multipurpose Internet Mail Extensions.

Mof2Text MOF Model to Text Transformation Language.

MPL Multilayered Pattern Language.

MVP Minimum Viable Product.

NFC Near Field Communication.

OAI OpenAPI.

OCL Object Constraint Language.

ORM Object Relational mapping.

OSI Open Systems Interconnection Model.

OwB Orientation with Beacons.

PaaS Platform as a Service.

PCM Palladio Component Model.

PIM Platform Independent Model.

POI Point of Interest.

PSM Platform Specific Model.

RAML RESTful API Modeling Language.

REST REpresentational State Transfer.

RMM Richardson-Maturity-Modell.

RQL Resource Query Language.

RSQL RESTful Service Query Language.

RUP Rational Unified Process.

SemVer Semantic Versioning Specification.

SLA Service Level Agreement.

SOA serviceorientierten Architektur.

SOAP Simple Object Access Protocol.

SoC Separation of Concern.

SRP Single Responsibility Principle.

SZS Studienzentrum für Sehgeschädigte.

TDD Test Driven Development.

UI User Interface.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

UUID Universally Unique IDentifier.

UX User Experience.

WADL Web Application Description Language.

WSDL Web Service Description Language.

XMI XML Metadata Interchange.

XML Extensible Markup Language.

XSD XML Schema Definition.

YAML Yet Another Multicolumn Layout.

C Abbildungsverzeichnis

D Tabellenverzeichnis

E Quelltextverzeichnis

F Literaturverzeichnis

[AA+10] ARNOWITZ, J. ; ARENT, M. ; BERGER, N.: *Effective Prototyping for Software Makers*. Elsevier Science, 2010 (Interactive Technologies). – ISBN 9780080468969

[AA+16] ALSHUQAYRAN, Nuha ; ALI, Nour ; EVANS, Roger: A Systematic Mapping Study in Microservice Architecture. In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, IEEE, November 2016. – ISBN 978–1–5090–4781–9, 44–51

[Al10] ALLAMARAJU, Subbu: *RESTful Web Services Cookbook*. O'Reilly, 2010. – I–XV, 1–293 S. – ISBN 978–0–596–80168–7

[AS+11] ADAMCZYK, Paul ; SMITH, Patrick H. ; JOHNSON, Ralph E. ; HAFIZ, Munawar ; WILDE, Erik (Hrsg.) ; PAUTASSO, Cesare (Hrsg.): *REST: From Research to Practice*. New York, NY : Springer New York, 2011. http://dx.doi.org/10.1007/978-1-4419-8303-9. http://dx.doi.org/10.1007/978-1-4419-8303-9. – ISBN 978–1–4419–8302–2

[Ba08] BALZERT, Helmut: *Lehrbuch der Software-Technik: Softwaremanagement*. 2. Heidelberg : Spektrum, 2008. – ISBN 978–3–8274–1161–7

[Ba16] BALLAV, Alipta: *Mental Models and User Experience*. Webseite, 2016. – URL: https://www.uxmatters.com/mt/archives/2016/05/mental-models-and-user-experience.php [Letzter Zugriff: 01.08.2017]

[BBA09] BRENNAN, K. ; BUSINESS ANALYSIS, International I.: *A Guide to the Business Analysis Body of Knowledge (BABOK Guide), Version 2.0*. International Institute of Business Analysis, 2009 (IT Pro). – ISBN 9780981129228

[BD09] BRUEGGE, Bernd ; DUTOIT, Allen H.: *Object-Oriented Software Engineering Using UML, Patterns, and Java*. 3rd. Upper Saddle River, NJ, USA : Prentice Hall Press, 2009. – ISBN 0136061257, 9780136061250

[Be07] BECK, K.: *Implementation Patterns*. Pearson Education, 2007 (Addison-Wesley Signature Series (Beck)). – ISBN 9780132702553

[BF14] BOURQUE, Pierre ; FAIRLEY, Richard E. ; BOURQUE, Pierre (Hrsg.) ; FAIRLEY, Richard E. (Hrsg.): *Guide to the Software Engineering Body of Knowledge - SWEBOK v3.0*. 2014 version. IEEE CS, 2014

[BG+16] BUTZIN, Bjorn ; GOLATOWSKI, Frank ; TIMMERMANN, Dirk: Microservices approach for the internet of things. In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA* 2016-Novem (2016). http://dx.doi.org/10.1109/ETFA.2016.7733707. – DOI 10.1109/ETFA.2016.7733707. – ISBN 9781509013142

[BH+07] BUSCHMANN, Frank ; HENNEY, Kevlin ; SCHMIDT, Douglas C.: *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*. Chichester, UK : Wiley, 2007. – ISBN 978–0–471–48648–0

[BJ+04] BERGHOLTZ, Maria ; JAYAWEERA, Prasad ; JOHANNESSON, Paul ; WOHED, Petia: Bringing speech acts into UMM. In: *1st. Int. REA Technology Workshop* (2004)

[BM14] BÜLTHOFF, Frederik ; MALESHKOVA, Maria: RESTful or RESTless - Current State of Today's Top Web APIs. In: *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, 2014, 64–74

[BR08] BÖHME, Rainer ; REUSSNER, Ralf: Validation of Predictions with Measurements. In: EUSGELD, Irene (Hrsg.) ; FREILING, Felix C. (Hrsg.) ; REUSSNER, Ralf (Hrsg.): *Dependability Metrics: Advanced Lectures*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, S. 14–18

[Br17] BROWN, Kyle G.: Apply the Strangler Application pattern to microservices applications. (2017), Februar, S. 1–6. – URL: https://www.ibm.com/developerworks/cloud/library/cl-strangler-application-pattern-microservices-apps-trs/cl-strangler-application-pattern-microservices-apps-trs-pdf.pdf [Letzter Zugriff: 15.05.2017]

[Bu13] BURGER, Erik J.: Flexible views for view-based model-driven development. In: *Proceedings of the 18th international doctoral symposium on Components and architecture* ACM, 2013, S. 25–30

[Cl04] CLARKE, Steven: *Measuring API Usability*. Webseite, Mai 2004. – URL: http://www.drdobbs.com/windows/measuring-api-usability/184405654 [Letzter Zugriff: 06.08.2017]

[Cl06] CLARKE, Steven: Describing and Measuring API Usability with the Cognitive Dimensions. In: *Cognitive Dimensions of Notations 10th Anniversary Workshop*, 2006

[Co68] CONWAY, Melvin E.: How do committees invent. In: *Datamation* 14 (1968), Nr. 4, 28–31. http://www.melconway.com/Home/pdf/committees.pdf

[Co99] COOPER, Alan: *The Inmates Are Running the Asylum*. Indianapolis, IN, USA : Macmillan Publishing Co., Inc., 1999. – ISBN 0672316498

[De17] DE, Brajesh: *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization*. 1st. Apress, 2017

[DG+17] DRAGONI, Nicola ; GIALLORENZO, Saverio ; LAFUENTE, Alberto L. ; MAZZARA, Manuel ; MONTESI, Fabrizio ; MUSTAFIN, Ruslan ; SAFINA, Larisa: Microservices: yesterday, today, and tomorrow. (2017), April. http://dx.doi.org/10.13140/RG.2.1.3257.4961. – DOI 10.13140/RG.2.1.3257.4961

[DIN-9000:2015-11] NORMUNG, Deutsches I.: *Qualitätsmanagementsysteme - Grundlagen und Begriffe*. 2015

[DIN-9241-11] NORMUNG, Deutsches I.: *Ergonomie der Mensch-System-Interaktion – Teil 11: Gebrauchstauglichkeit: Begriffe und Konzepte*. 2016

[DIN-9241-210] NORMUNG, Deutsches I.: *Ergonomie der Mensch-System-Interaktion – Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme*. 2011

[DL12] DIKMANS, Lonneke ; LUTTIKHUIZEN, Ronald van: *SOA Made Simple*. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK. : Packt Publishing Ltd, 2012. – ISBN 978–1–849684–16–3

[DM+07] DUVALL, P.M. ; MATYAS, S. ; GLOVER, A.: *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, 2007 (Addison-Wesley Signature Series). – ISBN 9780321630148

[DSF+16] DE SANTIS, S. ; FLOREZ, L. ; NGUYEN, D.V. ; ROSA, E. ; REDBOOKS, IBM: *Evolve the Monolith to Microservices with Java and Node*. IBM Redbooks, 2016. – ISBN 9780783442112

[Du16] DURDIK, Zoya: *Architectural Design Decision Documentation through Reuse of Design Patterns:*. KIT Scientific Publishing, 2016 (The Karlsruhe Series on Software Design and Quality). – ISBN 9783731502920

[EdI+16] ED-DOUIBI, Hamza ; IZQUIERDO, Javier Luis C. ; GÓMEZ, Abel ; TISI, Massimo ; CABOT, Jordi: EMF-REST: Generation of RESTful APIs from Models. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. New York, NY, USA : ACM, 2016 (SAC '16). – ISBN 978–1–4503–3739–7, 1446–1453

[Er05] ERL, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2005. – ISBN 0131858580

[Er07] ERL, Thomas: *SOA: Principles of Service Design*. Upper Saddle River, NJ : Prentice Hall, 2007. – ISBN 9780132344821

[Er08] ERL, Thomas: *SOA : Principles of Service Design*. Upper Saddle River, NJ : Prentice Hall, 2008

[Er17] ERL, Thomas: *Service-Oriented Architecture: Analysis and Design for Services and Microservices*. 2. Boston : Prentice Hall, 2017 (Prentice Hall Service Technology Series). – ISBN 978–0–13–385858–7

[Ev03] EVANS: *Domain-Driven Design: Tackling Complexity In the Heart of Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003. – ISBN 0321125215

[EZ+14] ESPINHA, Tiago ; ZAIDMAN, Andy ; GROSS, Hans-Gerhard: Web API growing pains: Stories from client developers and their code. In: *Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, 2014, 84–93

[Fa10] FAIRBANKS, George: *Just enough software architecture : a risk-driven approach*. Marshall & Brainerd, 2010. – ISBN 0984618104

[Fi00] FIELDING, Roy T.: *Architectural styles and the design of network-based software architectures*, University of California, Irvine, Dissertation, 2000. `https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf`. – 1–151 S.

[Fi08] FIELDING, Roy T.: *REST APIs must be hypertext-driven*. Webseite. `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`. Version: Oktober 2008, Abruf: 11.04.2014 09:47. – URL: `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven` [Letzter Zugriff: 02.08.2017]

[Fo02a] FOWLER, Martin: *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman, Amsterdam, 2002. – ISBN 978–0–32–112742–6

[Fo02b] FOWLER, Martin: Public versus published interfaces. In: *IEEE Software* 19 (2002), Nr. 2, S. 18–19. http://dx.doi.org/10.1109/52.991326. – DOI 10.1109/52.991326. – ISSN 07407459

[Fo10] FOWLER, Martin: *Richardson Maturity Model - steps toward the glory of REST*. Webseite, 2010. – URL: http://martinfowler.com/articles/richardsonMaturityModel.html [Letzter Zugriff: 18.07.2017]

[Fo14] FOWLER, Martin: *Microservices - a definition of this new architectural term*. Webseite, 2014. – URL: https://martinfowler.com/articles/microservices.html [Letzter Zugriff: 03.01.2018]

[FS15] FRAUENSTEIN, Thomas ; SCHMEISKY, Holger: *On APIs and the Zalando API Guild*. Online. https://jobs.zalando.com/tech/blog/on-apis-and-the-zalando-api-guild/. Version: September 2015, Abruf: 19.08.2017

[GB+12] GOLDSCHMIDT, Thomas ; BECKER, Steffen ; BURGER, Erik: Towards a Tool-Oriented Taxonomy of View-Based Modelling. In: SINZ, Elmar J. (Hrsg.) ; SCHÜRR, Andy (Hrsg.): *Modellierung* Bd. 201, GI, 2012 (LNI). – ISBN 978–3–88579–295–6, 59–74

[Ge11] GEBHART, Michael: *Qualitätsorientierter Entwurf von Anwendungsdiensten*. KIT Scientific Publishing, 2011. – ISBN 978–3–86644–704–2

[Ge14] GEBHART, Michael: Query-Based Static Analysis of Web Services in Service-Oriented Architectures. In: *International Journal on Advances in Software* (2014), S. 136–147

[GG+15] GIESSLER, Pascal ; GEBHART, Michael ; SARANCIN, Dimitrij ; ABECK, Sebastian: Best Practices for the Design of RESTful Web Services. (2015), November, S. 392–397. ISBN 978–1–61208–438–1

[GG+16a] GEBHART, Michael ; GIESSLER, Pascal ; ABECK, Sebastian: Challenges of the Digital Transformation in Software Engineering. In: *ICSEA: 11th International Conference on Software Engineering Advances*, 2016. – ISBN 978–1–61208–498–5, S. 136–141

[GG+16b] GEBHART, Michael ; GIESSLER, Pascal ; ABECK, Sebastian: Flexible and Maintainable Service-Oriented Architectures with Resource-Oriented Web Services. In: EL-SHEIKH, Eman (Hrsg.) ; ZIMMERMANN, Alfred (Hrsg.) ; JAIN, Lakhmi C. (Hrsg.): *Emerging Trends in the Evolution of Service-Oriented*

and Enterprise Architectures. Cham : Springer International Publishing, 2016. – ISBN 978–3–319–40564–3, S. 23–39

[GG+16c] GIESSLER, Pascal ; GEBHART, Michael ; STEINEGGER, Roland ; ABECK, Sebastian: Checklist for the API Design of Web Services based on REST. In: *I*nternational Journal On Advances in Internet Technology[GG+15], S. 41–51

[GH+94] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1994

[Gi16] GIAMAS, Alex: *From Monolith to Microservices, Zalando's Journey*. Online. `https://www.infoq.com/news/2016/02/Monolith-Microservices-Zalando`. Version: Februar 2016, Abruf: 05.01.2017

[Gl07] GLINZ, Martin: On Non-Functional Requirements. In: *RE*, IEEE Computer Society, 2007. – ISBN 0–7695–2935–6, 21–26

[GOOGLE-JSON] GOOGLE: *Google JSON Style Guide 0.9*. Webseite, 2017. – URL: `https://google.github.io/styleguide/jsoncstyleguide.xml` [Letzter Zugriff am: 2017-11-26]

[GT+99] GIGERENZER, G. ; TODD, P.M. ; GROUP, ABC R.: *Simple Heuristics that Make Us Smart*. Oxford University Press, 1999 (Evolution and cognition). – ISBN 9780195143812

[Ha16] HANSCHKE, Inge: *Enterprise Architecture Management - einfach und effektiv: Ein praktischer Leitfaden für die Einführung von EAM*. 2. Carl Hanser Verlag GmbH & Company KG, 2016. – ISBN 9783446449350

[He08] HEIKO, Koziolek: *Parameter dependencies for reusable performance specifications of software components*, Universität von Oldenburg, Diss., März 2008

[He10] HENNEY, K.: *97 Things Every Programmer Should Know: Collective Wisdom from the Experts*. O'Reilly Media, 2010. – ISBN 9781449388676

[HF10] HUMBLE, J. ; FARLEY, D.: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010 (Addison-Wesley Signature Series (Fowler)). – ISBN 9780321670229

[HG+17] HIPPCHEN, Benjamin ; GIESSLER, Pascal ; STEINEGGER, Roland ; ; ABECK, Sebastian: Designing Microservice-Based Applications by Using a Domain-Driven Design Approach. In: *International Journal On Advances in Software* Bd. 10, 2017, S. 432–445

[HK+14] HAUPT, Florian ; KARASTOYANOVA, Dimka ; LEYMANN, Frank ; SCHROTH, Benjamin: A Model-Driven Approach for REST Compliant Services. In: *IEEE International Conference on Web Services (ICWS)* (2014), S. 129–136. http://dx.doi.org/doi.ieeecomputersociety.org/10.1109/ICWS.2014.30. – DOI doi.ieeecomputersociety.org/10.1109/ICWS.2014.30

[HL+15] HAUPT, F. ; LEYMANN, F. ; PAUTASSO, C.: A Conversation Based Approach for Modeling REST APIs. In: *2015 12th Working IEEE/IFIP Conference on Software Architecture*, 2015, S. 165–174

[Hu16] HUNTER, Kirsten L.: *Irresistible APIs - Irresistible APIs*. Manning Publications, 2016

[HW03] HOHPE, Gregor ; WOOLF, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003. – ISBN 0321200683

[HY+15] HEFFNER, Randy ; YAMNITSKY, Michael ; MINES, Christopher ; FLEMING, Nate: *Sizing The Market For API Management Solutions*. Webseite, April 2015. – URL: https://www.forrester.com/report/Sizing+The+Market+For+API+Management+Solutions/-/E-RES121342?docid=121342&cm_mmc=Forrester-_-Blogs-_-Related%20Research-_-11696 [Letzter Zugriff: 19.04.2017]

[IB+09] IRELAND, C. ; BOWERS, D. ; NEWTON, M. ; WAUGH, K.: A Classification of Object-Relational Impedance Mismatch. In: *2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications*, 2009, S. 36–43

[IBM-RUP-98] IBM: Rational Unified Process - Best Practices for Software Development Teams. (1998). – URL: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf [Letzter Zugriff: 27.03.2017]

[IP+16] IVANCHIKJ, Ana ; PAUTASSO, Cesare ; SCHREIER, Silvia: Visual Modeling of RESTful Conversations with RESTalk. In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and*

Systems. New York, NY, USA : ACM, 2016 (MODELS '16). – ISBN 978–1–4503–4321–3, 114–114

[IS15] IYER, Bala ; SUBRAMANIAM, Mohan M.: *The Strategic Value of APIs*. Webseite, Januar 2015. – URL: `https://hbr.org/2015/01/the-strategic-value-of-apis` [Letzter Zugriff: 19.04.2017]

[ISO-24675] *Systems and software engineering - Vocabulary*. 2010

[ISO-25010] ISO/IEC: *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Geneva, CH, 2011

[ISO-42010] ISO/IEC/IEEE: Systems and software engineering – Architecture description. In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (2011), 1, S. 1–46

[Iv16] IVANCHIKJ, Ana: RESTful Conversation with RESTalk - The Use Case of Doodle. In: BOZZON, Alessandro (Hrsg.) ; CUDRE-MAROUX, Philippe (Hrsg.) ; PAUTASSO, Cesare (Hrsg.): *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*. Cham : Springer International Publishing, Mai 2016. – ISBN 978–3–319–38791–8, 583–587

[JB+11] JACOBSON, Daniel ; BRAIL, Greg ; WOODS, Dan: *APIs: A Strategy Guide*. O'Reilly Media, Inc., 2011. – ISBN 1449308929, 9781449308926

[JN+16] JARAMILLO, David ; NGUYEN, Duy V. ; SMART, Robert: Leveraging microservices architecture by using Docker technology. In: *SoutheastCon 2016*, 2016, S. 1–5

[Jo07] JOSUTTIS, Nicolai: *Soa in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007. – ISBN 0596529554

[Jo08] JOSUTTIS, Nicolai: *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*. Heidelberg : dpunkt, 2008. – ISBN 978–3–89864–476–1

[Ka14] KARLE, Thomas: *Kollaborative Softwareentwicklung auf Basis serviceorientierter Architekturen*. Karlsruher Institut für Technologie, 2014. – ISBN 9783866448728

[KIT-CM-SC] C&M: *SmartCampus*. Webseite, 2017. – URL: `https://cm.tm.kit.edu/smartcampus.php` [Letzter Zugriff: 03.10.2017]

[KL+16] KANG, Hui ; LE, Michael ; TAO, Shu: Container and Microservice Driven Design for Cloud Infrastructure DevOps. In: *2016 IEEE International Conference on Cloud Engineering, IC2E 2016, Berlin, Germany, April 4-8, 2016*, 2016, 202–211

[Kr10] KRESS, Markus: *Intelligent Business Process Optimization for the Service Industry*. KIT Scientific Publishing, 2010. – ISBN 978–3–86644–454–6

[KW+03] KLEPPE, A.G. ; WARMER, J.B. ; BAST, W.: *MDA Explained: The Model Driven Architecture : Practice and Promise*. Addison-Wesley, 2003 (The Addison-Wesley object technology series). – ISBN 9780321194428

[La14] LANTHALER, Markus: Leveraging Linked Data to Build Hypermedia-Driven Web APIs. In: PAUTASSO, Cesare (Hrsg.) ; WILDE, Erik (Hrsg.) ; ALARCÓN, Rosa (Hrsg.): *REST: Advanced Research Topics and Practical Applications*. Springer New York, 2014. – ISBN 978–1–4614–9298–6, S. 107–123

[LF04] LEWIS, James ; FOWLER, Martin: *Microservices - a definition of this new architectural term*. Online. `https://martinfowler.com/articles/microservices.html?cm_mc_uid=34013674245414737751113&cm_mc_sid_50200000=147499071`. Version: März 2004, Abruf: 04.05.2017

[LG11] LANTHALER, Markus ; GUTL, C: A semantic description language for RESTful data services to combat Semaphobia. In: *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference* 5 (2011), Juni, 47–53. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5936597`. ISBN 9781457708725

[LG12] LANTHALER, Markus ; GÜTL, Christian: On using JSON-LD to create evolvable RESTful services. In: *Third International Workshop on RESTful Design* (2012), April. `http://dl.acm.org/citation.cfm?id=2307827`

[LK+06] LAITKORPI, Markku ; KOSKINEN, Johannes ; SYSTÄ, Tarja: A UML-based approach for abstracting application interfaces to REST-like services. In: *Proceedings - Working Conference on Reverse Engineering, WCRE* (2006), S. 134–146. `http://dx.doi.org/10.1109/WCRE.2006.8`. – DOI 10.1109/WCRE.2006.8. – ISBN 0769527191

[LN+15] LE, Vinh D. ; NEFF, Melanie M. ; STEWART, Royal V. ; KELLEY, Richard ; FRITZINGER, Eric ; DASCALU, Sergiu M. ; HARRIS, Frederick C.:

Microservice-based architecture for the NRDC. In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, IEEE, Juli 2015. – ISBN 978–1–4799–6649–3, 1659–1664

[LS+09] LAITKORPI, M. ; SELONEN, P. ; SYSTA, T.: Towards a Model-Driven Process for Designing RESTful Web Services. In: *2009 IEEE International Conference on Web Services*, 2009, S. 173–180

[LX+13] LI, Jun ; XIONG, Yingfei ; LIU, Xuanzhe ; ZHANG, Lu: How does web service API evolution affect clients? In: *Proceedings - IEEE 20th International Conference on Web Services, ICWS 2013*, 2013. – ISBN 9780768550251

[MD97] MESZAROS, Gerard ; DOBLE, Jim: Pattern Languages of Program Design 3. Version: 1997. http://dl.acm.org/citation.cfm?id=273448.273487. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997. – ISBN 0–201–31011–2, Kapitel A Pattern Language for Pattern Writing, 529–574

[Me92] MEYER, Bertrand: Applying "Design by Contract". In: *Computer* 25 (1992), Oktober, Nr. 10, 40–51. http://dx.doi.org/10.1109/2.161279. – DOI 10.1109/2.161279. – ISSN 0018–9162

[Me10] MELZER, Ingo: *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. Spektrum Akademischer Verlag, 2010. – ISBN 9783827425508

[MM+12] MUNDIE, David ; MOORE, Andrew P. ; MCINTIRE, David: Building a Multidimensional Pattern Language for Insider Threats. In: *Proceedings of the 19th Conference on Pattern Languages of Programs*. USA : The Hillside Group, 2012 (PLoP '12). – ISBN 978–1–4503–2786–2, 1–11

[MM+16] MACVEAN, Andrew ; MALY, Martin ; DAUGHTRY, John: API Design Reviews at Scale. In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '16* (2016). http://dx.doi.org/10.1145/2851581.2851602. – DOI 10.1145/2851581.2851602. ISBN 9781450340823

[MMJ92] MUNOZ, Richard ; MILLER-JACOBS, Harold H.: In Search of the Ideal Prototype. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 1992 (CHI '92). – ISBN 0–89791–513–5, 577–579

[Mu12] MULLOY, Brian: Web API Design - Crafting Interfaces that Developers Love. (2012), März. http://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

[Ne15] NEWMAN, Sam: *Building Microservices*. 1st. O'Reilly Media, Inc., 2015. – ISBN 9781491950357

[Ni14] NIMESH, Rakhitha: *Paginating Real-Time Data with Cursor Based Pagination*. Webseite, 2014. – URL: https://www.sitepoint.com/paginating-real-time-data-cursor-based-pagination/ [Letzter Zugriff: 10.07.2017]

[NM+16] NADAREISHVILI, I. ; MITRA, R. ; MCLARTY, M. ; AMUNDSEN, M.: *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, 2016. – ISBN 9781491956342

[OAI-REPO] OAI: *Open API Initiative*. Webseite, 2017. – URL: https://www.openapis.org/specification/repo [Letzter Zugriff: 13.08.2017]

[OAI-SPEC] OAI: *OpenAPI Specification Version 3.0*. Webseite, 2017. – URL: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md [Letzter Zugriff: 16.08.2017]

[OAI-WEB] OAI: *Open API Initiative*. Webseite, 2017. – URL: https://openapis.org/ [Letzter Zugriff: 27.03.2017]

[OB12] OESTEREICH, B. ; BREMER, S.: *Analyse und Design mit der UML 2.5: Objektorientierte Softwareentwicklung*. Oldenbourg Wissenschaftsverlag, 2012. – ISBN 9783486716672

[OMG-UML2a] OMG: OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1 / Object Management Group. Version: August 2011. http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF. 2011. – Forschungsbericht

[OMG-UML2b] OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1 / Object Management Group. Version: August 2011. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF. 2011. – Forschungsbericht

[OMG-UML2c] OMG: OMG Unified Modeling Language (OMG UML), Version 2.5 / Object Management Group. Version: März 2015. http://www.omg.org/spec/UML/2.5/PDF. 2015. – Forschungsbericht

[Pa72] PARNAS, David L.: On the Criteria to Be Used in Decomposing Systems into Modules. In: *Communications of the ACM* 15 (1972), Dezember, Nr. 12, 1053–1058. http://dx.doi.org/10.1145/361598.361623. – DOI 10.1145/361598.361623. – ISSN 0001–0782

[Pa11] PANSA, Ingo: *Dienstorientierte Integration von Managementwerkzeugen*. 2011

[PGH+15] PALMA, Francis ; GONZALEZ-HUERTA, Javier ; MOHA, Naouel ; GUÉHÉNEUC, Yann-Gaël ; TREMBLAY, Guy: Are RESTful APIs Well-Designed? Detection of their Linguistic (Anti)Patterns. In: *International Conference on Service-Oriented Computing* (2015), November

[PI+15] PAUTASSO, Cesare ; IVANCHIKJ, Ana ; SCHREIER, Silvia: Modeling RESTful Conversations with Extended BPMN Choreography Diagrams. In: *Software Architecture: 9th European Conference, ECSA 2015, Dubrovnik/Cavtat, Croatia, September 7-11, 2015. Proceedings*. Springer International Publishing, 2015. – ISBN 978–3–319–23727–5, S. 87–94

[PI+16] PAUTASSO, Cesare ; IVANCHIKJ, Ana ; SCHREIER, Silvia: A Pattern Language for RESTful Conversations. In: *Proceedings of the 21st European Conference on Pattern Languages of Programs*. New York, NY, USA : ACM, 2016 (EuroPlop '16). – ISBN 978–1–4503–4074–8, 1–22

[PR+08] PEREPLETCHIKOV, Mikhail ; RYAN, Caspar ; FRAMPTON, Keith ; SCHMIDT, Heinz W.: Formalising Service-Oriented Design. In: *JSW* 3 (2008), Nr. 2, 1–14. http://dblp.uni-trier.de/db/journals/jsw/jsw3.html#PerepletchikovRFS08

[PR11] PORRES, Ivan ; RAUF, Irum: Modeling Behavioral RESTful Web Service Interfaces in UML. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. New York, NY, USA : ACM, 2011 (SAC '11). – ISBN 978–1–4503–0113–8, 1598–1605

[Pr15] PREHOFER, Christian: Models at REST or Modelling RESTful Interfaces for the Internet of Things. (2015). ISBN 9781509003662

[PS14] PFEIFER, Tilo ; SCHMITT, Robert: *Masing Handbuch Qualitätsmanagement*. Carl Hanser Verlag GmbH & Company KG, 2014. – ISBN 9783446439924

[PW10] PAUTASSO, Cesare ; WILDE, Erik: RESTful Web Services: Principles, Patterns, Emerging Technologies. In: *Proceedings of the 19th International Conference on World Wide Web*. New York, NY, USA : ACM, 2010 (WWW '10). – ISBN 978–1–60558–799–8, 1359–1360

[PW17] PRESTON-WERNER, Tom: *Semantic Versioning 2.0.0*. Webseite, 2017. – URL: http://semver.org/ [Letzter Zugriff am: 2017-09-25]

[PZ+08] PAUTASSO, Cesare ; ZIMMERMANN, O. ; LEYMANN, Frank: Restful web services vs. 'big'web services: making the right architectural decision. In: *Proceeding of the 17th international conference on World Wide Web* (2008), 805–814. http://dx.doi.org/10.1145/1367497.1367606. – DOI 10.1145/1367497.1367606. ISBN 978–1–60558–085–2

[RA+13] RICHARDSON, Leonard ; AMUNDSEN, Mike ; RUBY, Sam: *RESTful Web APIs*. O'Reilly Media, 2013. – ISBN 978–1–449–35806–8

[RB+16a] REUSSNER, Ralf H. ; BECKER, Steffen ; HAPPE, Jens ; HEINRICH, Robert ; KOZIOLEK, Anne ; KOZIOLEK, Heiko ; KRAMER, Max ; KROGMANN, Klaus: *Modeling and Simulating Software Architectures – The Palladio Approach*. Cambridge, MA : MIT Press, 2016. – 37–73 S. http://mitpress.mit.edu/books/modeling-and-simulating-software-architectures

[RB+16b] RODRÍGUEZ, Carlos ; BÁEZ, Marcos ; DANIEL, Florian ; CASATI, Fabio ; TRABUCCO, Juan C. ; CANALI, Luigi ; PERCANNELLA, Gianraffaele: REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. In: *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, 2016, 21–39

[RFC2045-1996] NOTTINGHAM, Mark ; WILDE, Erik: RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies / Freed, Ned and Borenstein, Nathaniel S. and Vaudreuil, Gregory M. Version: November 1996. https://tools.ietf.org/html/rfc2045. 1996. – Vorgeschlagener Standard

[RFC2396-1998] BERNERS-LEE, Tim ; FIELDING, Roy T. ; MASINTER, Larry: RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax. Version: August 1998. https://www.ietf.org/rfc/rfc2396.txt. 1998. – Standard

[RFC3986-2005] BERNERS-LEE, Tim ; FIELDING, Roy T. ; MASINTER, Larry: RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. Version: 2005. https://www.ietf.org/rfc/rfc3986.txt. 2005. – Standard

[RFC4122-2005] LEACH, Paul J. ; MEALLING, Michael ; SALZ, Rich: RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. Version: Juli 2005. https://tools.ietf.org/html/rfc4122#. 2005. – Standard

[RFC6838-2013] NOTTINGHAM, Mark ; WILDE, Erik: RFC 6838: Media Type Specifications and Registration Procedures / Freed, Ned and Klensin, John C. and Hansen, Tony. Version: Januar 2013. https://tools.ietf.org/html/rfc6838. 2013. – Vorgeschlagener Standard

[RFC6902-2013] BRYAN, Paul C. ; NOTTINGHAM, Mark: RFC 6902: JavaScript Object Notation (JSON) Patch. Version: April 2013. https://tools.ietf.org/html/rfc6902. 2013. – Forschungsbericht

[RFC7230-2014] FIELDING, Roy ; RESCHKE, Julian F.: RFC 7230: Hypertext Transfer Protocol - HTTP/1.1: Message Syntax and Routing. Version: Juni 2014. https://tools.ietf.org/html/rfc7230. 2014. – Standard

[RFC7231-2014] FIELDING, Roy ; RESCHKE, Julian F.: RFC 7231: Hypertext Transfer Protocol - HTTP/1.1: Semantics and Content. Version: Juni 2014. https://tools.ietf.org/html/rfc7231. 2014. – Standard

[RFC7232-2014] FIELDING, Roy ; RESCHKE, Julian F.: RFC 7232: Hypertext Transfer Protocol - HTTP/1.1: Conditional Requests. Version: Juni 2014. https://tools.ietf.org/html/rfc7232. 2014. – Standard

[RFC7233-2014] FIELDING, Roy ; LAFON, Yves ; RESCHKE, Julian F.: RFC 7233: Hypertext Transfer Protocol - HTTP/1.1: Range Requests. Version: Juni 2014. https://tools.ietf.org/html/rfc7233. 2014. – Standard

[RFC7234-2014] FIELDING, Roy ; NOTTINGHAM, Mark ; RESCHKE, Julian F.: RFC 7234: Hypertext Transfer Protocol - HTTP/1.1: Caching. Version: Juni 2014. https://tools.ietf.org/html/rfc7234. 2014. – Standard

[RFC7235-2014] FIELDING, Roy ; RESCHKE, Julian F.: RFC 7235: Hypertext Transfer Protocol - HTTP/1.1: Authentication. Version: Juni 2014. https://tools.ietf.org/html/rfc7235. 2014. – Standard

[RFC7252-2014] SHELBY, Zach ; HARTKE, Klaus ; BORMANN, Carsten: RFC 7252: Constrained Application Protocol. Version: Juni 2014. https://tools.ietf.org/html/rfc7252. 2014. – Standard

[RFC7351-2014] WILDE, Erik: RFC 7351: A Media Type for XML Patch Operations. Version: August 2014. https://tools.ietf.org/html/rfc7351. 2014. – Forschungsbericht

[RFC7386-2014] HOFFMAN, Paul ; SNELL, James M.: RFC 7386: JSON Merge Patch. Version: Oktober 2014. https://tools.ietf.org/html/rfc7386. 2014. – Forschungsbericht

[RFC7540-2015] BELSHE, Mike ; PEON, Roberto ; THOMSON, Martin: RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2). Version: Mai 2015. https://tools.ietf.org/html/rfc7540. 2015. – Standard

[RFC7807-2015] NOTTINGHAM, Mark ; WILDE, Erik: RFC 7807: Problem Details for HTTP APIs. Version: 2015. https://tools.ietf.org/html/rfc7807. 2015. – Vorgeschlagener Standard

[RH08] REUSSNER, Ralf (Hrsg.) ; HASSELBRING, Wilhelm (Hrsg.): *Handbuch der Software-Architektur*. 2. Heidelberg : dpunkt, 2008. – ISBN 978–3–89864–559–1

[RI+08] RAUF, Iram ; IQBAL, Muhammad Zohaib Z. ; MALIK, Zafar I.: UML based modeling of web service composition-A survey. In: *Proceedings - 6th ACIS International Conference on Software Engineering Research, Management and Applications, SERA 2008* (2008), S. 301–307. http://dx.doi.org/10.1109/SERA.2008.39. – DOI 10.1109/SERA.2008.39. ISBN 9780769533025

[Ri15a] RICHARDS, Mark: *Microservices vs. Service-Oriented Architecture*. 2nd. O'Reilly Media, Inc., 2015. – ISBN 978–1–491–94161–4

[Ri15b] RICHARDS, Mark: *Software Architecture Patterns*. O'Reilly Media, Inc., 2015. – ISBN 9781491925409

[Ro09] ROBILLARD, Martin P.: What Makes APIs Hard to Learn? Answers from Developers. In: *IEEE Softw.* 26 (2009), November, Nr. 6, 27–34. http://dx.doi.org/10.1109/MS.2009.193. – DOI 10.1109/MS.2009.193. – ISSN 0740–7459

[Ro16] ROSSI, Davide: UML-based Model-Driven REST API Development. In: MAJCHRZAK, Tim A. (Hrsg.) ; TRAVERSO, Paolo (Hrsg.) ; MONFORT, Valérie (Hrsg.) ; KREMPELS, Karl-Heinz (Hrsg.): *International Conference on Web Information Systems and Technologies (WEBIST)*, SciTePress, 2016. – ISBN 978–989–758–186–1, S. 194–201

[RP+13] RATHOD, D. M. ; PARIKH, S. M. ; BUDDHADEV, B. V.: Structural and behavioral modeling of RESTful web service interface using UML. In: *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*, 2013, S. 28–33

[RS+12] RENZEL, Dominik ; SCHLEBUSCH, Patrick ; KLAMMA, Ralf: Today's Top "RESTful" Services and Why They Are Not RESTful. Version: 2012. http://dx.doi.org/10.1007/978-3-642-35063-4_26. In: WANG, X. S. (Hrsg.) ; CRUZ, Isabel (Hrsg.) ; DELIS, Alex (Hrsg.) ; HUANG, Guangyan (Hrsg.): *Web Information Systems Engineering - WISE 2012: 13th International Conference, Paphos, Cyprus, November 28-30, 2012. Proceedings*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. – ISBN 978–3–642–35063–4, 354–367

[RW+15] ROSE, S. ; WYNNE, M. ; HELLESOY, A.: *The Cucumber for Java Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2015 (Pragmatic programmers). – ISBN 9781941222294

[Sc11] SCHREIER, Silvia: Modeling RESTful Applications. In: *Proceedings of the Second International Workshop on RESTful Design*. New York, NY, USA : ACM, 2011 (WS-REST '11). – ISBN 978–1–4503–0623–2, 15–21

[Sc12] SCHWARZ, Hannes: *Universal traceability: a comprehensive, generic, technology-independent, and semantically rich approach*, University of Koblenz-Landau, Diss., 2012. http://d-nb.info/1020501030

[Sc14] SCHREIBMANN, Vitaliy: Design and Implementation of a Model-Driven Approach for Restful APIs. (2014)

[Se06] SEEMANN, Jürgen W. Jochen und von Gudenberg G. Jochen und von Gudenberg ; YORK, Springer Berlin Heidelberg N. (Hrsg.): *Software-Entwurf mit UML 2 - Objektorientierte Modellierung mit Beispielen in Java, 2. Auflage*. Springer, 2006. http://dx.doi.org/10.1007/3-540-30950-0. http://dx.doi.org/10.1007/3-540-30950-0. – ISBN 978–3–540–30949–9

[Se11] *Kapitel* 11. In: SELONEN, Petri: *From Requirements to a RESTful Web Service: Engineering Content Oriented Web Services with REST*. New York, NY : Springer New York, 2011. – ISBN 978–1–4419–8303–9, 259–278

[SG+17] STEINEGGER, Roland ; GIESSLER, Pascal ; HIPPCHEN, Benjamin ; ABECK, Sebastian: Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications. In: *SOFTENG: The Third International Conference on Advances and Trends in Software Engineering*, 2017

[SL+08] SIIKARLA, Mika ; LAITKORPI, Markku ; SELONEN, Petri ; SYSTA, Tarja: Transformations have to be developed ReST assured. In: VALLECILLO, Antonio (Hrsg.): *Theory and Practice of Model Transformations*, 2008. – ISBN 3540699260, S. 1–15

[SM07] STYLOS, Jeffrey ; MYERS, Brad: Mapping the Space of API Design Decisions. In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, 2007. – ISSN 1943–6092, S. 50–60

[SO+14] *Kapitel* 8. In: SANCHEZ, Robson Vincius V. ; OLIVEIRA, Ricardo Ramos d. ; FORTES, Renata Pontin de M.: *RestML: Modeling RESTful Web Services*. New York, NY : Springer New York, 2014. – ISBN 978–1–4614–9299–3, 125–143

[SR+04] SOUZA, Cleidson de ; REDMILES, David ; DAVID, Li-te C. ; PATTERSON, John ; MILLEN, David: Sometimes You Need to See Through Walls — A Field Study of Application Programming Interfaces. In: *the 2004 ACM conference on Computer supported cooperative work* 6 (2004), Nr. 3, S. 63–71. `http://dx.doi.org/10.1145/1031607.1031620`. – DOI 10.1145/1031607.1031620. ISBN 1581138105

[SS12] STRAUCH, Jakob ; SCHREIER, Silvia: RESTify: From RPCs to RESTful HTTP Design. In: *Proceedings of the Third International Workshop on RESTful Design*. New York, NY, USA : ACM, 2012 (WS-REST '12). – ISBN 978–1–4503–1190–8, 11–18

[St07] STEFFENS, U.: *MDD, SOA und IT-Management (MSI 2007) Workshop, Oldenburg, April 2007*. Gito, 2007. – ISBN 9783940019066

[Su16] SURWASE, Vijay: REST API Modeling Languages – A Developer's Perspective. In: *International Journal of Science Technology & Engineering* 2 (2016), April, Nr. 10, S. 634–637

[SV+06] STAHL, Thomas ; VOELTER, Markus ; CZARNECKI, Krzysztof: *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. – ISBN 0470025700

[Sz98] SZYPERSKI, C.: *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998

[TE+15] TILKOV, Stefan ; EIGENBRODT, Martin ; SCHREIER, Silvia ; WOLF, Oliver: *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt, 2015. – ISBN 9783864901201

[Ti00] TICHY, Walter F.: Hints for Reviewing Empirical Work in Software Engineering. In: *Empirical Softw. Engg.* 5 (2000), Dezember, Nr. 4, 309–312. `http://dx.doi.org/10.1023/A:1009844119158`. – DOI 10.1023/A:1009844119158. – ISSN 1382–3256

[Ti15] TILKOV, S.: The Modern Cloud-Based Platform. In: *IEEE Software* 32 (2015), März, Nr. 2, S. 116–116. http://dx.doi.org/10.1109/MS.2015.51. – DOI 10.1109/MS.2015.51. – ISSN 0740–7459

[VA+09] VOGEL, Oliver ; ARNOLD, Ingo ; CHUGHTAI, Arif ; IHLER, Edmund ; KEHRER, Timo ; MEHLIG, Uwe ; ZDUN, Uwe: *Software-Architektur: Grundlagen – Konzepte – Praxis*. 2. Heidelberg : Spektrum, 2009. http://dx.doi.org/10.1007/978-3-8274-2267-5. http://dx.doi.org/10.1007/978-3-8274-2267-5. – ISBN 978–3–8274–1933–0

[Ve13] VERNON, Vaughn: *Implementing Domain-Driven Design*. 1st. Addison-Wesley Professional, 2013. – ISBN 0321834577, 9780321834577

[Vi08] VINOSKI, Steve: RESTful Web Services Development Checklist. In: *Internet Computing, IEEE* 12 (2008), Nr. 6, 94–96. http://dx.doi.org/10.1109/MIC.2008.130. – DOI 10.1109/MIC.2008.130

[W3C-RDF] W3C: *RDF Schema 1.1*. Webseite, Feburar 2014. – URL: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/ [Letzter Zugriff: 20.01.2018]

[W3C-SPAQRL] W3C: *SPARQL 1.1 Overview*. Webseite, 2013. – URL: https://www.w3.org/TR/rdf-sparql-query/ [Letzter Zugriff: 20.01.2018]

[W3C-WADL] W3C: *Web Application Description Language*. Webseite, 2017. – URL: https://www.w3.org/Submission/wadl/ [Letzter Zugriff: 20.03.2017]

[WA+07] WAHLI, Ueli ; ACKERMAN, Lee ; DI BARI, Alessandro ; HODGKINSON, Gregory ; KESTERTON, Anthony ; OLSON, Laura ; PORTIER, Bertrand: *Building SOA Solutions Using the Rational SDP*. IBM Redbooks, 2007

[Wa14] WATSON, Robert: Applying the Cognitive Dimensions of API Usability to Improve API Documentation Planning. In: *Proceedings of the 32Nd ACM International Conference on The Design of Communication CD-ROM*. New York, NY, USA : ACM, 2014 (SIGDOC '14). – ISBN 978–1–4503–3183–8, 1–2

[Wi17] WINN, Duncan C.: *Cloud Foundry: The Definitive Guide: Develop, Deploy, and Scale*. O'Reilly Media, 2017. – ISBN 9781491932537

[Wo15] WOLFF, Eberhard: *Microservices: Grundlagen flexibler Softwarearchitekturen*. Heidelberg : dpunkt, 2015. – ISBN 978–3–86490–313–7

[WP+10] WEBBER, Jim ; PARASTATIDIS, Savas ; ROBINSON, Ian: *REST in Practice: Hypermedia and Systems Architecture*. 1st. O'Reilly Media, Inc., 2010. – ISBN 0596805829, 9780596805821

[WP+16] WISNEWSKI, Benjamin ; PEREPA, Bhargav ; SEELEMANN, Ilene ; YILDIRIM, Kurtulus ; GUPTA, Rahul ; HAMDY, Soad ; GUCER, Vasfi: *Getting Started with IBM API Connect - Concepts and Architecture Guide*. IBM Redbooks, 2016 `http://www.redbooks.ibm.com/redpapers/pdfs/redp5349.pdf`

[WR+00] WOHLIN, Claes ; RUNESON, Per ; HÖST, Martin ; OHLSSON, Magnus C. ; REGNELL, Björn ; WESSLÉN, Anders: *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA : Kluwer Academic Publishers, 2000. – ISBN 0–7923–8682–5

[WR+12] WOHLIN, Claes ; RUNESON, Per ; HÖST, Martin ; OHLSSON, Magnus C. ; REGNELL, Björn ; WESSLÉN, Anders: *Experimentation in Software Engineering*. Springer-Verlag Berlin Heidelberg, 2012. – ISBN 978–3–642–29044–2

[Yi08] YIN, Robert K.: *Case Study Research: Design and Methods (Applied Social Research Methods)*. Fourth Edition. Sage Publications, 2008. – ISBN 1412960991

[ZAL-API-GUIDE] ZALANDO: *Zalando RESTful API and Event Scheme Guidelines*. Online. `https://zalando.github.io/restful-api-guidelines/#hypermedia`. Version: 2017, Abruf: 21.10.2017