Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 278

Rollenbasiertes SOA-Governance Dashboard

Dominik Bilgery

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf. Jan Königsberger
Beginn am:	18. November 2015
Beendet am:	13. Mai 2016
CR-Nummer:	H.3.4, H.5.2, J.1

Kurzfassung

Mit dem zunehmenden Einzug von serviceorientierten Architekturen (SOA) in Unternehmen wächst auch der Bedarf an leistungsfähigen Werkzeugen zur Unterstützung der SOA-Governance. Diese sind notwendig, um zu den technischen Umsetzungen auch strategische, organisatorische und personelle Änderungen einzuführen. Ein immer wichtig werdender Bestandteil solcher Werkzeuge ist das Dashboard. Es soll für verschiedene Zielgruppen, wie zum Beispiel Manager, Entwickler oder Administratoren, die wichtigsten Informationen zusammenfassen und übersichtlich darstellen. In dieser Arbeit wird ein solches Dashboard entworfen und entwickelt. Dafür wird zunächst eine umfangreiche Literaturanalyse anhand von akademischen Quellen, sowie Veröffentlichungen von Softwareherstellern, Beratungsfirmen und unabhängigen Autoren durchgeführt, in der sowohl Stakeholder als auch Kennzahlen der SOA-Governance ermittelt werden. Darauf aufbauend wird zudem ein allgemeines Rollenmodell entwickelt. Zusammenfassend werden dann die identifizierten Kennzahlen den einzelnen Rollen zugeordnet, um eine zielgruppengerechte Darstellung dieser Kennzahlen ermöglichen zu können. Auf dieser Basis wird dann ein detailliertes Konzept für ein SOA-Governance Dashboard entworfen. Dazu werden mögliche Darstellungsmöglichkeiten für Kennzahlen untersucht und ein Rechte-Management für diese entwickelt, welches gewährleisten soll, dass die Kennzahlen ausschließlich den Stakeholdern angezeigt werden, deren Rolle dies erlaubt. Um abschließend deutlich zu machen, wie eine Umsetzung des Konzepts unter Berücksichtigung der Anforderungen an eine zielgruppengerechte Darstellung der identifizierten Informationen aussehen kann und welchen Nutzen ein solches SOA-Governance Dashboard hat, wird das Konzept prototypisch implementiert und evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

Der Begriff der serviceorientierten Architektur (SOA) wurde erstmals im Jahr 1996 von dem amerikanischen Marktforschungsunternehmen Gartner eingeführt [SN96]. Der Grundgedanke dahinter war, die Geschäftsprozesse eines Unternehmens in den Vordergrund zu stellen. Es soll damit die historisch gewachsene homogene Anwendungslandschaft eines Unternehmens so optimiert werden, dass bestehende Anwendungen in verschiedene Dienste, sogenannte Services, aufgespalten werden. Diese können dann anhand von Geschäftsprozessen wieder zusammengesetzt werden. Durch diese Entkopplung soll eine flexible Anpassung der Geschäftsprozesse an die sich ständig wandelnden Marktbedingungen, des Unternehmens vereinfacht werden.

Im Laufe der Jahre wurde SOA als das Wundermittel für Integrationsprobleme angepriesen. Unternehmen versprachen sich durch die Einführung eine Reihe von Vorteilen. Prozesse und Anwendungen sollen dann flexibel und schnell entwickelt, konfiguriert und verändert werden können. Dadurch können neue Produkte oder Dienstleistungen schneller und vor allem mit geringeren Kosten realisiert werden. Diese Flexibilität und Agilität spielt für Unternehmen gerade heute, in Zeiten der Globalisierung, eine erhebliche Rolle, um wettbewerbsfähig zu bleiben [BKR11].

Diese Erwartungen konnten allerdings oft nicht erfüllt werden. Grund dafür war vor allem ein fehlerhaftes Verständnis gegenüber der serviceorientierten Architektur. Oft wurde die Einführung als ein weiteres IT-Projekt gesehen, für das es eine Art Patentrezept gibt. SOA ist aber vielmehr ein Architekturparadigma für verteilte Systeme, das nur dann die gewünschten Vorteile bringt, wenn das ganze Unternehmen sich auf die Grundprinzipien einlässt. Für eine erfolgreiche Einführung muss eine Individuallösung erarbeitet werden, die auf das gesamte Unternehmen und dessen Ziele abgestimmt ist. Diese ist meist komplex, kostenintensiv und kann sich über mehrere Jahre hinziehen, bis sie ihre ganzen Stärken entfalten kann.

Nachdem sehr viele SOA-Einführungen scheiterten, befassten sich verschiedenste Unternehmen und Forscher mit den Problemen [VLA09]. Gartner veröffentlichte eine Liste mit den Faktoren, an denen die meisten Unternehmen bei der Einführung scheiterten [Goa07]. Die Burton Group, die heute auch Teil von Gartner ist, untersuchte eine Reihe solcher SOA-Einführungen in verschiedenen Unternehmen [Mee08]. Der am häufigsten genannte Grund in diesen Untersuchungen war eine mangelnde Berücksichtigung der strategischen, organisatorischen und personellen Änderungen. Meist beschränkte sich die Einführung auf die technische Ebene. Dies führte dazu, dass eine Vielzahl an Services entstand, die kaum oder nur schwer wiederverwendet werden konnten. Das erhöhte mitunter die Komplexität der gesamten Architektur und brachte mehr Kosten mit sich, als dass es einsparen konnte.

1.1 Motivation

Um diesem Problem zuvorzukommen, setzen viele Unternehmen heute auf SOA-Governance, da sich gezeigt hatte, dass Unternehmen, die einer strikten Governance folgen, die Vorteile, die eine serviceorientierte Architektur mit sich bringt, ausschöpfen konnten. Diese erweitert die klassische IT-Governance um serviceorientierte Aspekte. Während die IT-Governance sich mit der Organisation, Steuerung und Kontrolle der IT befasst, bezeichnet die SOA-Governance die Definition, Durchsetzung und Steuerung von organisatorischen Regeln, Richtlinien und Standards für eine serviceorientierte Architektur. Diese soll der Ausrichtung der Services und Geschäftsprozesse an die Unternehmensstrategie durch geeignete Steuerungs- und Kontrollmaßnahmen dienen [KEHZ08].

Da auch eine SOA-Governance sehr komplex ist und schnell große Datenmengen anfallen, die nur schwer manuell verwaltet werden können, wurden hierfür über die Jahre von verschiedensten Herstellern, wie zum Beispiel Oracle, HP oder IBM, Werkzeuge entwickelt, die diese unterstützen sollen. Es hat sich allerdings gezeigt, dass diese erhebliche Unterschiede aufweisen. Oft liegt der Fokus auf der Verwaltung von Services und deren Eigenschaften im Laufe des Lebenszyklus mit dem Ziel, die Wiederverwendbarkeit von Services zu gewährleisten. Da das allerdings nur ein Teil der SOA-Governance ist, reicht dies oft nicht aus, um die Vorteile, die eine serviceorientierte Architektur mit sich bringen kann, zu erreichen. Andere Aspekte, wie zum Beispiel das Consumer- und Portfolio-Management oder das Service-Monitoring, gehören ebenfalls in ein solches SOA-Governance Werkzeug. Königsberger et al. beschreiben die Anforderungen an ein solches SOA-Governance Repository im Detail [KSM14], welches auch in dieser Arbeit als Grundlage verwendet wird.

Da das Ziel ist, langfristig alle Mitarbeiter eines Unternehmens in die serviceorientierte Architektur mit einzubeziehen und zu unterstützen, muss ein solches Werkzeug für alle Mitarbeiter geeignet sein. Um die Kommunikation sowohl innerhalb des Unternehmens als auch mit externen Personen zu verbessern, müssen geeignete Rollen definiert werden, um so die Zuständigkeiten transparenter zu handhaben. Da diese Werkzeuge oft sehr komplex sind, ist es notwendig, die Mitarbeiter zu motivieren, den vollen Funktionsumfang zu nutzen. Sehr oft muss man mühselig in solchen Werkzeugen die Informationen zusammensuchen, die man benötigt, was sehr frustrierend sein kann. Abhilfe schaffen hier sogenannte Dashboards. Ein Dashboard ist eine grafische Darstellung der wichtigsten Informationen. In den meisten Fällen handelt es sich hier um die Startseite eines Systems. Hier können dann entsprechende Informationen, für die zuvor definierten Rollen, angezeigt werden, um dem Nutzer möglichst schnell eine Übersicht der serviceorientierten Architektur zu bieten. Besonders hilfreich für den täglichen Gebrauch sind dafür Informationen zur präzisen Überwachung der SOA, wie zum Beispiel Vertragsverletzungen, Fehler, Ausfallzeiten, Verfügbarkeiten oder andere Metriken.

Derzeit gibt es auf dem Markt nur eine geringe Anzahl an Werkzeugen, die ein solches rollenbasiertes Dashboard in ihrem Funktionsumfang enthalten. Dabei sollte unumstritten sein, dass dieses zur Unterstützung der SOA-Governance beiträgt und damit auch die unternehmerischen Ziele, die mit der Einführung der serviceorientierten Architektur erreicht werden sollen, fördert [Lau08].

1.2 Zielsetzung

Ziel dieser Arbeit soll die Ausarbeitung eines solchen rollenbasierten SOA-Governance Dashboards sein. Grundlegend müssen die Anforderungen für eine zielgruppengerechte Darstellung der Informationen ermittelt werden. Dafür müssen zum einen die Zielgruppen eines solchen Dashboards und zum anderen notwendige Informationen für die Steuerung und Überwachung einer serviceorientierten Architektur identifiziert werden. Diese Anforderungen sollen auf Basis einer umfassenden Literaturanalyse anhand von akademischen Quellen sowie Veröffentlichungen von Softwareherstellern, Beratungsfirmen und von unabhängigen Autoren erhoben werden. Darauf aufbauend soll ermittelt werden, welche Informationen für welche Zielgruppen interessant sind und ob diese eine Zugriffsberechtigung darauf haben sollen. Die Resultate sollen dann geeignet zusammengefasst werden und als Basis für eine zielgruppengerechte Darstellung der Informationen dienen.

Auf dieser Basis soll dann ein detailliertes Konzept für ein solches SOA-Governance Dashboard entworfen werden. Der Fokus soll dabei weniger auf der technischen Umsetzung als vielmehr auf der Darstellung liegen. Das Dashboard soll die wichtigsten Informationen möglichst in grafischer Form übersichtlich darstellen. Dafür sollen verschiedene Darstellungsmöglichkeiten für entsprechende Informationen evaluiert werden. Durch die Konfiguration des Dashboards soll dem Nutzer die Möglichkeit gegeben werden, eine personalisierte Ansicht der vorhanden Informationen entsprechend seiner Präferenzen anzuzeigen. Die Struktur des inhaltlichen Aufbaus spielt dabei eine genauso große Rolle wie die benutzerfreundliche Bedienung. Dafür sollen Interaktionen untersucht werden, die Usability und User Experience erheblich steigern. Durch den immer häufigeren Einsatz von mobilen Endgeräten soll auch eine geeignete Darstellung auf kleineren Displays möglich sein. Dennoch müssen einige technische Aspekte berücksichtigt werden. So muss das Dashboard beispielsweise gewährleisten, Informationen ausschließlich der dafür vorgesehenen Zielgruppe anzuzeigen. Dementsprechend muss ein geeignetes Rechte-Management entworfen werden.

Um deutlich zu machen, wie eine Umsetzung des Konzepts unter Berücksichtigung der Anforderungen an eine zielgruppengerechte Darstellung der identifizierten Informationen aussehen könnte und welchen Nutzen ein solches SOA-Governance Dashboard hat, soll eine prototypische Umsetzung erfolgen. Dabei soll dieses Dashboard in einem vorhandenen Prototyp eines SOA-Governance Repositories implementiert werden. Bei diesem Prototyp handelt es sich um eine JavaEE-Webanwendung, die ursprünglich im Rahmen eines Studienprojekts auf Basis des SOA-Governance Meta-Models [KSM14] an der Universität Stuttgart entwickelt und im Anschluss erweitert wurde. Entsprechend soll diese Implementierung auf Basis aktueller Webtechnologien wie HTML5, CSS3, JavaScript sowie Java realisiert werden. Dabei soll stets darauf geachtet werden, dass das Dashboard beliebig erweitert werden kann. Durch einen sauberen und gut lesbaren Code, der ausreichend dokumentiert ist, soll eine einfache Wartbarkeit gewährleistet werden.

1.3 Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In diesem Kapitel werden die Grundlagen erläutert, die für diese Arbeit relevant sind. Dabei werden unter anderem Begriffe wie SOA und SOA-Governance eingeführt und erläutert. Zudem werden die Anforderungen und eine mögliche Umsetzung von Dashboards beschrieben.

Kapitel 3 – Anforderungen: In diesem Kapitel werden die grundlegenden Anforderungen an ein SOA-Governance Dashboard untersucht. Dabei werden anhand existierender Literatur mögliche Zielgruppen für ein solches Dashboard ermittelt. Zudem werden dabei Informationen einer serviceorientierten Architektur identifiziert, die in einem solchen Dashboard angezeigt werden sollten. Abschließend werden die Informationen, die dargestellt werden sollen, diesen Zielgruppen zugeordnet.

Kapitel 4 – Konzept: In diesem Kapitel wird ein Konzept für ein SOA-Governance Dashboard entworfen. Dafür wird zunächst der Aufbau des Dashboards beschrieben. Anschließend werden verschiedene Darstellungsmöglichkeiten für anzuzeigende Informationen ermittelt und untersucht. Abschließend wird dann noch geklärt, wie gewährleistet werden kann, dass die anzuzeigenden Informationen für die verschiedenen Zielgruppen angezeigt werden.

Kapitel 5 – Prototyp: In diesem Kapitel wird das entworfene Konzept anhand der identifizierten Anforderungen prototypisch umgesetzt. Dabei erfolgt eine Beschreibung der Implementierung der grundlegenden Komponenten, die als Vorbereitung für die folgenden Implementierungsschritte dient. Abschließend werden dann die im Konzept beschriebenen Funktionalitäten umgesetzt. Hierfür erfolgt sowohl eine Beschreibung der Implementierung als auch der umgesetzten Anwendungsfälle.

Kapitel 6 – Zusammenfassung: In diesem Kapitel wird die Herangehensweise der Arbeit mit ihren Resultaten beschrieben. Zudem werden die aufgetretenen Schwierigkeiten geschildert und abschließend ein Ausblick gegeben, an welchen Stellen der Arbeit angeknüpft werden kann.

2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die für diese Arbeit relevant sind. Dafür wird zunächst eine Einführung in Unternehmensarchitekturen in der Informationstechnologie gegeben, die einen kleinen Rückblick gibt, was es für Modelle gab und inwiefern die serviceorientierte Architektur, auf die in diesem Kapitel verstärkt eingegangen wird, die Welt der Unternehmensarchitekturen auf einen neuen Stand gebracht hat. Anschließend wird die Notwendigkeit einer Governance anhand von allgemein bekannten Formen, erläutert. Der Fokus dieses Kapitels liegt auf der Beschreibung der SOA-Governance. Hierbei wird ein Meta-Model vorgestellt, das die Anforderungen an eine solche SOA-Governance beschreibt. Abschließend erfolgt eine Erläuterung von Dashboards, die deren Sinn und Zweck deutlich machen soll und welche Anforderungen an solche Dashboards gestellt werden. Zusätzlich wird hier noch bewährter Umsetzungsprozess eines solchen Dashboards erläutert.

2.1 Unternehmensarchitekturen

Unternehmensarchitektur bezeichnet grundlegend das Zusammenspiel von Elementen der Informationstechnologie und der geschäftlichen Tätigkeit in einem Unternehmen. Eine Unternehmensarchitekturinitiative sollte stets von der obersten Managementebene ausgehen und soll damit die Ausrichtung der Unternehmens-IT an den Geschäftszielen sicherstellen. Dafür machen sie fachliche Vorgaben, sie definieren Konstruktionsprinzipien und legen die einzusetzende Infrastruktur fest. Sie dienen dazu, die bestehende Anwendungslandschaft überblicken zu können und dabei, neue Systeme zu planen, zu entwickeln und sie gut integrieren zu können. [Nie05]

2.1.1 Enterprise Application Integration

Enterprise Application Integration (EAI) ist ein Ansatz zur prozessorientierten Integration von Anwendungssystemen in heterogenen IT-Anwendungsarchitekturen, der den Austausch von Informationen zwischen Anwendungen im Unternehmen und über Unternehmensgrenzen hinweg ohne wesentliche Veränderung der existierenden Systeme ermöglicht [Kai02]. EAI kann somit als Vorgänger der serviceorientierten Architektur verstanden werden.

2.1.2 Serviceorientierte Architektur

Eine serviceorientierte Architektur verfolgt das Prinzip der Modularisierung. Das bedeutet, dass Geschäftsprozesse und damit bestehende Anwendungen in viele kleine Teile zerlegt werden. Diese Teilprozesse werden dann von sogenannten Services bereitgestellt und können so wieder zu

einem größeren Prozess zusammengeführt werden [Bel08]. Diese Zusammenführung nennt man Orchestrierung. Ein Service ist eine in sich abgeschlossen gekapselte Software-Komponente einer bestimmten Funktionalität, die über eine klar definierte Schnittstelle zur Verfügung gestellt wird. Über die Schnittstellen verschiedener Services können dann Anwendungen erstellt werden, die auf die jeweiligen Geschäftsprozesse zugeschnitten sind. So wäre beispielsweise ein Geschäftsprozess eines Onlinehandels folgender: Bestellung – Bearbeitung – Versand – Rechnungsstellung – Zahlungseingang. Jeder dieser Teilprozesse kann dann als Service dargestellt werden. Dabei ist es nicht zwingend notwendig, dass alle Services im eigenen Unternehmen bereitgestellt werden. So kann auch der Fall auftreten, dass die Services für die Rechnungsstellung und den Zahlungseingang von einem externen Dienstleister übernommen werden. Da auch diese Aufgabe nicht ganz trivial ist, vor allem dann, wenn einzelne Services gewartet werden müssen oder sogar Services ausgetauscht werden sollen, sieht die serviceorientierte Architektur das Prinzip der losen Kopplung vor.

Das Prinzip der losen Kopplung verhält sich ähnlich zum Prinzip der Objektorientierung [BHA04], wobei versucht wird, die Abhängigkeiten zwischen Komponenten so zu reduzieren, dass Änderungen dieser möglich sind ohne die Beziehung jener zu gefährden. Diese reduzierten Abhängigkeiten tragen dazu bei, die Fehlertoleranz sowie die Flexibilität zu erhöhen [Jos07]. Eine solche lose Kopplung erreicht man beispielsweise, indem Abhängigkeiten zwischen Services nicht fest verankert sind, sondern beim Start eines Services über einen zentralen Verzeichnisdienst vermittelt werden. Das gewünschte Maß an Unabhängigkeit bringt das Find, Bind, Invoke Prinzip, das in Abbildung 2.1 beschrieben ist, mit sich [W3C04] [Ora05] [Cha04]. Dieses sieht vor, dass ein Service als Service Provider seine Beschreibung in eine zentrale Service Registry veröffentlicht (publish). Ein anderer Service wiederum kann dann als Service Requestor die Service Registry fragen (find), ob ein Service die gewünschte Funktionalität anbietet. Ist das der Fall, übermittelt diese dem Service Requestor die entsprechende Schnittstellenbeschreibung, der diesen Service dann darüber aufrufen kann (bind & invoke) [Mí12] [Mel10].

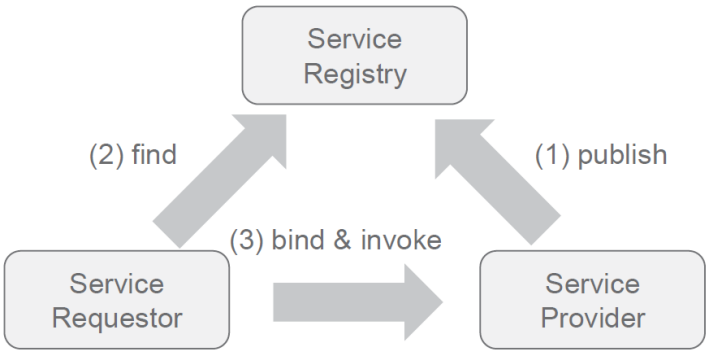

Abbildung 2.1: Das Find, Bind, Invoke Prinzip [Mí12]

Damit das vorangegangene Prinzip erfolgreich umgesetzt werden kann, ist es notwendig, dass die Beschreibung der Funktionalität der einzelnen Services abgerufen werden kann. Dieses Prinzip nennt man Auffindbarkeit. Es erfordert, dass Metainformationen jedes Services klar dokumentiert und an zentraler Stelle verwaltet werden. So soll vor allem verhindert werden, dass Redundanzen entstehen. Diese können auftreten, wenn im Laufe der Zeit eine Vielzahl an Services entstehen, über die der Über-

blick nur schwer zu behalten ist und deshalb ähnliche oder sogar gleiche Services mehrmals entwickelt und bereitgestellt werden. Diese Redundanzen widersprechen vollkommen der serviceorientierten Architektur, die das Prinzip der Wiederverwendbarkeit fordert. Die Wiederverwendbarkeit eines Services ist abhängig von der bereitgestellten Funktionalität und wie oft diese in den Geschäftsprozessen des Unternehmens genutzt werden kann. Dafür muss bereits bei der Abbildung eines Prozesses auf einen Service darauf geachtet werden, dass dieser Prozess wiederverwendet werden kann. Aber auch in der Bereitstellung müssen hierfür entsprechende Standards definiert werden.

2.2 Governance

Governance bezeichnet grundlegend die Leitung und Überwachung einer Einheit wie beispielsweise eines Staats, einer Gemeinde, oder einer privaten oder öffentlichen Organisation. Die Governance war eine Reaktion auf unethisches oder kriminelles Verhalten der Leitung oder Führung und sollte deren Handlungen transparenter gestalten [NHSS10]. In Zeiten der Wirtschaftskrise wurde dieses Thema verstärkt diskutiert. Inzwischen gibt es verschiedene Formen von Governance, die im folgenden näher beschrieben werden.

2.2.1 Corporate- und IT-Governance

Corporate-Governance bezeichnet die Definition von Regeln, Gesetzen, Richtlinien und Verordnungen, wie ein Unternehmen geführt werden soll und will sicherstellen, dass das Unternehmen ordnungsgemäß, effizient und verantwortungsvoll handelt [Dir13]. Ein weiterer Bereich, in dem Governance einen hohen Stellenwert erlangt hat, ist die Informationstechnologie (IT). Die IT-Governance gliedert sich in die in Corporate-Governance ein und umfasst die Verwaltung und Kontrolle der gesamten IT eines Unternehmens. Ziel der IT-Governance ist es, das Risiko von IT-Projekten zu minimieren und deren Geschäftsnutzen zu erhöhen.

2.2.2 SOA-Governance

Die SOA-Governance erweitert die klassische IT-Governance um serviceorientierte Aspekte. Während die IT-Governance sich mit der Organisation, Steuerung und Kontrolle der IT befasst, bezeichnet die SOA-Governance die Definition, Durchsetzung und Steuerung von organisatorischen Regeln, Richtlinien und Standards für eine serviceorientierte Architektur. Diese soll der Ausrichtung der Services und Geschäftsprozesse an die Unternehmensstrategie durch geeignete Steuerungs- und Kontrollmaßnahmen dienen [KEHZ08].

Kernaspekte der SOA-Governance

Im folgenden werden die Anforderungen an eine SOA-Governance von Königsberger et al. [KSM14] näher beschrieben.

Service Life Cycle Management Eines der am häufigsten genannten Aspekte, wenn es um SOA-Governance geht, ist das Management der Lebenszyklen von Services. Existieren nur wenige Services, ist es noch möglich, diese manuell zu verwalten. Mit der zunehmenden Anzahl und Komplexität der Services steigt die Notwenigkeit der Verwaltung und Überwachung, insbesondere bei steigender Wiederverwendung. Services durchlaufen immer einen bestimmten Lebenszyklus, der meist aus folgenden Phasen besteht: Identification, Specification, Implementation, Integration & Testing, Release, Sundowning, Retirement. Da in dieser Prozesskette alle Stakeholder zeitweise involviert sind, ist das Service Life Cycle Management zur Identifikation der Informationen für eine entsprechende Rolle äußerst wichtig. Befindet sich beispielsweise ein Service in der Phase Implementation, sind entsprechende Informationen im Dashboard für die Entwickler hilfreich. Befindet sich dieser Service dann irgendwann in Retirement, sollte dies den entsprechenden Service Konsumenten direkt kommuniziert werden.

Consumer Management Zwischen Service Provider und Service Consumer bestehen sogenannte Service Level Agreements (SLAs oder auch Contracts genannt), die nichtfunktionale Anforderungen an den Service stellen. Diese müssen von beiden Parteien eingehalten und überwacht werden. Für Provider sind Informationen dazu notwendig, um auf mögliche Ausfälle oder Engpässe möglich schnellst reagieren zu können. Aber auch andere Faktoren, wie beispielsweise die Zeit, die das Unternehmen benötigt, um auf Wünsche des Kunden zu reagieren, spielen hier eine große Rolle. Für Consumer dienen solche Informationen dazu zu prüfen, ob die Vereinbarungen eingehalten werden. Aus diesen Informationen lässt sich beispielsweise eine Tendenz der Zufriedenheit der Consumer ableiten. Diese sollte natürlich für alle Consumer so hoch wie möglich gehalten werden.

Meta Data Management In einer serviceorientierten Architektur fallen eine Vielzahl an strukturierten und unstrukturierten Daten an. Um diese effizient verwalten zu können, müssen sie in einem zentralen Repository gespeichert und gepflegt werden. Meta Daten sind in diesem Fall jegliche Informationen über Services, Service-Versions, Business Objects und so weiter, die über den kompletten Lebenszyklus anfallen. Die Dokumentation dieser Informationen soll dazu führen, dass die Kommunikation in jeglichen Bereichen erleichtert wird. Möchte man beispielsweise einen bestehenden Service nutzen, kann man über ein solches Repository Informationen, wie beispielsweise Entwurfsdokumente oder die Schnittstellenbeschreibung für verschiedene Service-Versionen erlangen. Außerdem bekommt man sehr schnell Aufschluss darüber, wer der geeignete Ansprechpartner ist und muss nicht nach diesem erst suchen. Auch heute ist die Kommunikation in Unternehmen ein großes Problem, was mit einem guten Meta Daten Management reduziert werden soll.

Organizational Structure Um eine bereichsübergreifende Projektarbeit zu erleichtern, muss mit der Einführung einer serviceorientierten Architektur eine neue Organisationsstruktur eingeführt werden. Hierfür müssen Aufgaben und Verantwortlichkeiten so definiert werden, dass Mitarbeiter aus den verschiedenen Fachbereichen wissen, was sie zu tun haben und an wen sie sich wenden können, wenn sie Fragen haben. Außerdem können sogenannte Boards entstehen, die aus verschiedenen Experten bestehen und die Aufgabe haben, bestimme Lösungen zu erörtern und gegebenenfalls Entscheidungen zu treffen.

Portfolio Management Das Portfolio besteht aus verschiedenen Domänen und Services. Es beinhaltet alle näheren Informationen, die benötigt werden, um sich einen Überblick über die Geschäftsprozesse, die von einem Service abgebildet werden, zu schaffen. Mit Hilfe des Portfolio Managements soll verhindert werden, dass beispielsweise ein Service neu entwickelt wird, der so oder so ähnlich bereits existiert. Aber nicht nur existierende Services, sondern auch bereits aufgesetzte Plattformen oder Werkzeuge sollten von Entscheidungsträgern stets berücksichtigt werden. Das Portfolio sollte deshalb stets auf die Unternehmensstrategie angepasst sein. So kann es auch vorkommen, dass sich herausstellt, dass ein bereits entwickelter Service nicht (wieder-)verwendet wird oder werden kann und somit aus dem Portfolio gelöscht wird.

Architectural Standards Neue Services oder Business Objects müssen so entwickelt werden, dass sie in die bestehende IT-Landschaft integriert werden können. Um dies zu erleichtern, ist es sinnvoll, Architektur Standards zu definieren. Ein einheitlicher Standard von Enterprise Service Bus oder Schnittstellen erleichtert das Entwerfen und Entwickeln von Services. So können Risiko und Kosten gesenkt werden, indem die Anzahl und Komplexität von Architekturentscheidungen reduziert wird.

Governance Hierarchy Neben der SOA-Governance gibt es in einem Unternehmen mit der IT- und Corporate Governance noch weitere Governance Ansätze, ohne die die SOA-Governance alleine nicht funktionieren würde. Da die SOA-Governance, anders als die IT-Governance, auch Geschäftsprozesse mit einschließt, muss sie mit der Einführung einer serviceorientierten Architektur in die anderen Formen der Governance integriert werden. Um dies zu gewährleisten, müssen die Ziele aller Governance-Konzepte übereinstimmen.

Funding Model Mit der Erschließung neuer Geschäftsfelder, wie beispielsweise dem Verkauf von Service Nutzungsrechten an andere Unternehmen, müssen neue Bezahlmodelle eingeführt werden. Für einen externen Nutzer eines Services kann eine fixe oder unlimitierte Anzahl an Aufrufen in den Service Contracts festgelegt werden. Diese Problem stellt sich allerdings nicht nur nach außen hin. Auch innerhalb des Unternehmens muss ein neues Finanzierungsmodell eingeführt werden, sodass Kosten nicht nur auf einer Abteilung lasten, sondern verteilt werden, falls ein Service von einer anderen Abteilung wiederverwendet wird.

Service Monitoring Das Service Monitoring nimmt ebenfalls eine wichtige Rolle in der SOA-Governance ein. Hauptziel ist es, Metriken wie die Verfügbarkeit, die Performance oder den Durchsatz eines Services zu dokumentieren oder in Echtzeit zu überwachen. Außerdem stellt das Service Monitoring die Grundlage für das Service Level Management dar. So können diese Daten mit den Anforderungen der Service Level Agreements verglichen werden. Liegen die Werte von Performance und Verfügbarkeit unter ihren festgelegten Werten, muss darauf entsprechend reagiert werden. Mit Hilfe der Daten zum Durchsatz kann beispielsweise ermittelt werden, ob es sich lohnt, weiter Kunden anzuwerben, um einen höheren Gewinn zu erzielen.

Maturity Measurement Da eine SOA ein sich kontinuierlich wandelnder Prozess ist, der nicht mit der Einführung abgeschlossen ist, müssen regelmäßig Prüfungen stattfinden. Hierbei muss geprüft werden, ob sich die Einführung lohnt. Dabei muss überwacht werden, ob die gesetzten Ziele nicht erreicht werden, um gegebenenfalls frühst möglich Maßnahmen ergreifen zu können. So spielen hier Faktoren wie verwendete Budgets, aufgewendete Zeit oder die Wiederverwendbarkeit von Services eine große Rolle, um zu messen, ob die serviceorientierte Architektur das Unternehmen vorantreibt, oder eher blockiert.

Business Object Management Business Objects sind Datenmodelle, wie beispielsweise eine Bestellung oder ein Kunde und dienen zur Entkopplung von Service und Datenmodell. Sie durchlaufen in ihrer Entwicklung, wie Services auch, einen bestimmten Lebenszyklus. Die Business Objects werden unternehmensweit standardisiert und können dann von verschiedenen Services genutzt werden. Dies erleichtert vor allem den Informationsfluss innerhalb und zwischen den Geschäftsprozessen.

SOA-Governance Meta-Model

Königsberger et al. erarbeiteten anhand dieser Anforderungen ein Meta-Model für ein Repository zur Unterstützung der SOA-Governance Aktivitäten in einem Unternehmen, welches in Abbildung 2.2 zu sehen ist. Dieses wurde in vier grundlegende Bereiche unterteilt: Service Provider, Service Consumer, Oganizational Structure und Business Object. Bereich A beinhaltet hauptsächlich den Service, der verschiedene Service-Versionen besitzen kann. Diese Service-Versionen verfügen sowohl über einen Lebenszyklus-Status als auch eine Service-Beschreibung, beispielsweise in Form einer Web Services Description Language (WSDL). Zudem kann eine Service-Version mehrere Endpunkte besitzen, welche als Schnittstelle für mögliche Service-Konsumenten dienen. Bereich B beinhaltet hauptsächlich den Konsumenten, der wiederum mit einer Service-Version verknüpft ist. Des weiteren besteht zwischen Konsument und Endpunkt der Service-Version ein Vertrag, der verschiedene Vertragseigenschaften, wie beispielsweise die Vertragsdokumente, besitzt. Bereich C beinhaltet das Rollen und Organisationsmodell. Die Rolle ist mit dem Konsument, dem Vertrag, dem Service, der Service-Version und der Business Object-Version verknüpft. Außerdem können jeder Rolle verschiedene Personen oder Verantwortlichkeiten angefügt werden. Bereich D beinhaltet hauptsächlich das Business Object, das verschiedene Business Object-Versionen besitzen kann. Diese haben ebenso wie Service-Versionen einen Lebenszyklus-Status.

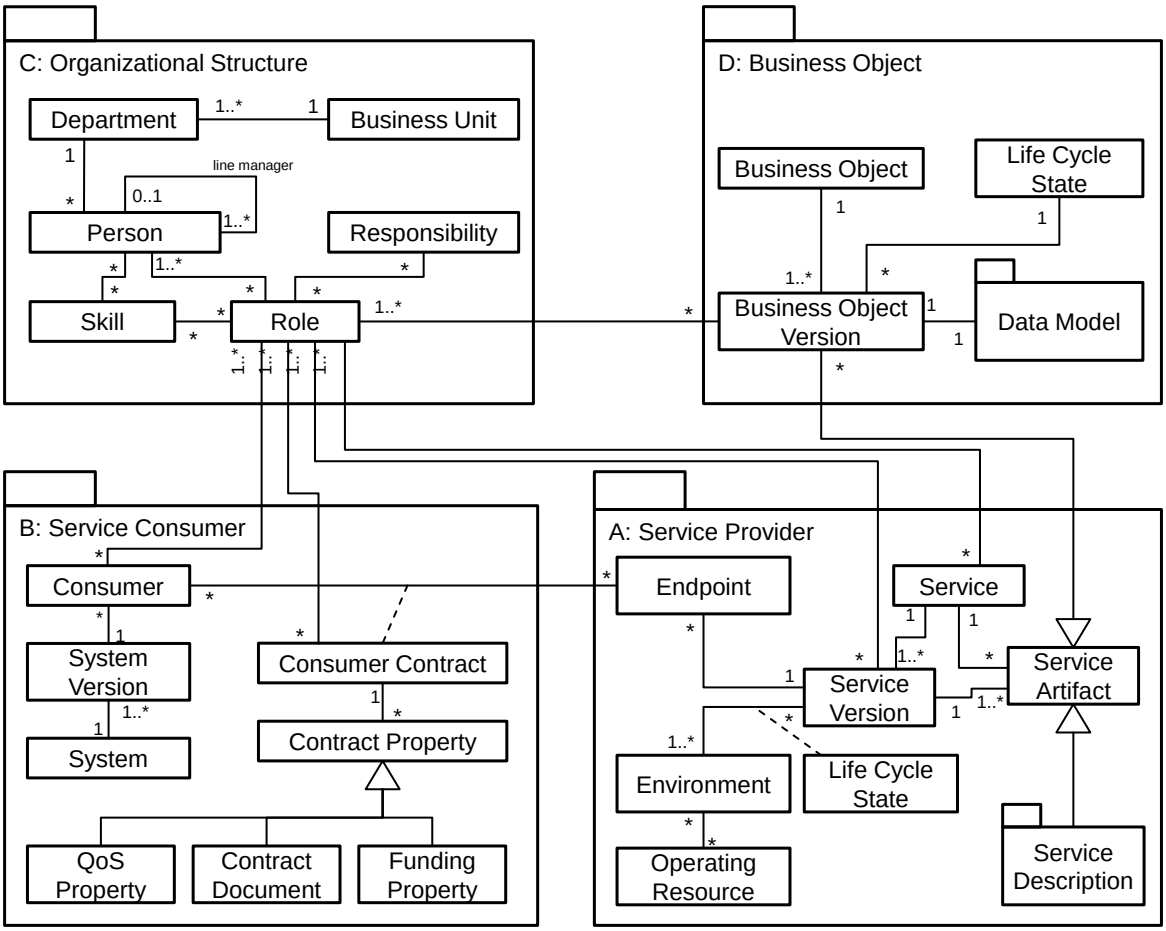

Abbildung 2.2: SOA-Governance Meta-Model [KSM14]

2.3 Dashboards

Ein Dashboard (englisch für Armaturenbrett) in einem Auto ist die Anzeige des aktuellen Betriebszustands die dem Fahrer die aktuelle Geschwindigkeit, den Kilometerstand, die Tankfüllung usw. optisch darstellt. Dabei wird auf unnötige textliche Erklärungen verzichtet und dafür die Darstellung so intuitiv wie möglich gestaltet. Diese Anzeige hat der Fahrer zu jeder Zeit in seinem Blickfeld. Zudem wird er auch, vor allem in neueren Autos, darauf hingewiesen, wenn ein kritischer Zustand erreicht wird. So blinkt beispielsweise ein Lämpchen auf, sobald die Tankfüllung ein gewisses Minimum unterschritten hat. Der Fahrer kann dann darauf reagieren und bei der nächsten Tankstelle halten [RC04].

In der IT ist ein Dashboard eine visuelle Darstellung der wichtigsten Informationen, zusammengefasst und angeordnet auf einer Seite, sodass die Information auf einen Blick erfasst werden kann. Damit ist es, wie in einem Auto, eine Art Kommunikationsinstrument, das dem Nutzer helfen soll, Korrelationen zu identifizieren, Trends, Ausreißer (Anomalien) oder Muster zu erkennen. Gerade in der Zeit, in

der Dashboards auch in der Softwarebranche bekannt wurden, waren diese ausschließlich für das Management gedacht. Das hat sich im Laufe der Zeit drastisch geändert. So sind heute Dashboards in allen möglichen Variationen für unterschiedlichste Benutzergruppen vorhanden. Die Stärke eines Dashboards liegt dabei weniger in der technischen Umsetzung als vielmehr in der Darstellung. Dabei hat sich gezeigt, dass Informationen grafisch sehr viel effizienter und aussagekräftiger kommuniziert werden können, als durch einen Text.

2.3.1 Anforderungen

Grundsätzlich gelten für Dashboards die gängigen Anforderungen der Softwareentwicklung. Dazu zählen beispielsweise Bedienbarkeit, Sicherheit und Wartbarkeit.

Few [Few06] definiert zudem eine Reihe an Anforderungen, welche im folgenden näher beschrieben werden. In einem Dashboard sollen Informationen angezeigt werden, um bestimmte Ziele zu erreichen. Um das zu erreichen, muss in den meisten Fällen eine Vielzahl an nicht zusammenhängenden Informationen aus unterschiedlichsten Quellen herangezogen werden. Dabei sollten die Informationen im Normalfall nicht zu detailliert, sondern zusammengefasst, um einen Überblick zu schaffen. In einem kritischen Fall hingegen sind Detailinformation durchaus angebracht. Das Dashboard muss zudem in der Lage sein, schnell Daten hervorzuheben, die die Aufmerksamkeit des Nutzers erfordern. Gerade in Fällen, in denen beispielsweise eine kritische Grenze unterschritten wird, sollte durch eine geeignete Darstellung ein schnelles Handeln ermöglicht werden. Dabei ist es wichtig, dass das Dashboard sich auf eine Seite beschränkte. Genauer gesagt, sollte es sich sogar nur auf einen Bereich beschränken, den der Nutzer ganz überblicken kann, um alle Informationen auf einen Blick zu sehen. Scrollen kann dabei in manchen Fällen noch vertretbar sein, das Wechseln zwischen verschiedenen Seiten hingegen sollte dringlichst vermieden werden. Zudem spielt die Platzierung des Dashboards eine wichtige Rolle. Der Browser bietet dafür eine gute Möglichkeit, da hierfür keine neue Software installiert werden muss und das Dashboard damit von überall abrufbar ist. Zudem können Daten über das Internet jederzeit aktualisiert werden. Natürlich kommt es dennoch darauf an, um was für ein Dashboard es sich hierbei handelt. So kann es natürlich auch Sinn machen, das Dashboard in bereits vorhandene Systeme zu integrieren.

Malik [Mal05] nennt des weiteren eine Reihe an Anforderungen, welche im folgenden näher beschrieben werden. Insgesamt sollen die angezeigten Informationen spezifisch für den Verantwortungsbereich, die Privilegien oder andere Kriterien des Nutzers sein. Andere Aspekte der Personalisierung, wie Sprache und visuelle Präferenzen, sollten für eine bessere Benutzerfreundlichkeit zur Verfügung stehen. Zudem sollte es möglich sein, dass der Nutzer einige Informationen hervorhebt. Diese sollen ihm dann in Zukunft gesondert angezeigt werden. Um das Vertrauen der Nutzer zu erlangen, müssen die angezeigten Informationen korrekt sein. Dafür müssen diese genau getestet und validiert werden. Zudem sollten Informationen über einen längeren Zeitraum gespeichert werden, um dem Nutzer die Möglichkeit zu bieten, anhand von Vergleichswerten Muster oder Anomalien zu erkennen. Um dem Nutzer zu ermöglichen, Details über die Informationen zu erlangen, muss eine Interaktion mit den Darstellungen möglich sein. Dadurch kann beispielsweise schneller auf auf Probleme reagiert werden, wenn der Nutzer die Detailinformationen auswerten kann.

2.3.2 Umsetzung

Malik beschreibt in seinem Buch *Enterprise Dashboards - Design and Best Practices for IT* [Mal05] eine bewährte Herangehensweise für die Umsetzung von Dashboards. Bei diesem Prozess stellen sich drei grundlegende Fragestellungen:

1. **Welche** Informationen sollen angezeigt werden?
2. **Wem** sollen die Informationen angezeigt werden?
3. **Wie** sollen die Informationen angezeigt werden?

Zunächst geht es darum, Meta-Daten aus Informationen zu ermitteln. Meta-Daten sind Informationen über Informationen. Diese müssen entsprechend dokumentiert werden. Hier ist sinnvoll, bereits diese Informationen in Bereiche wie Verkauf, Marketing usw. zu gruppieren, da in den meisten Fällen später die Informationen anhand solcher Bereiche zusammengefasst werden. Beispiele für den Verkauf wären die durchschnittlichen Verkaufszahlen oder die Anzahl der nicht verkauften Einheiten. Für das Marketing wäre es beispielsweise die Auswirkung der Werbekosten auf den Umsatz. Für diese Informationen muss dann geprüft werden, ob die notwendigen Datenquellen bereits vorhanden sind und wenn nicht, ob diese für eine weitere Auswertung hinterlegt werden können. Anschließend muss die Berechnung der Informationen, die angezeigt werden sollen, definiert werden. Für diese Berechnung sollte dann noch, sofern es Sinn macht, die Varianz angegeben werden, was bedeutet, dass festgelegt wird, ab welchem Ergebnis diese beispielsweise gut, normal oder schlecht ist. Dafür können zum Beispiel absolute Werte oder auch Bereiche angegeben werden.

Anschließend muss ermittelt werden, für welche Art von Nutzern die Informationen angezeigt werden sollen. Sollte der Fall auftreten, dass alle Nutzer die gleichen Informationen angezeigt bekommen sollen, können folgende Überlegungen diesbezüglich übersprungen werden. Da dieser Fall allerdings eher selten auftritt und die unterschiedlichen Informationen verschiedenen Nutzern angezeigt werden sollen, muss bei diesem Schritt im Detail untersucht werden, welche Nutzer gruppiert werden können. So kann es in dem Fall, dass 50 Mitarbeiter des Verkaufs die selben 20 Informationen angezeigt bekommen sollen, sinnvoll sein, diese zu der Gruppe Verkauf zusammenzufassen.

Abschließend müssen dann geeignete Diagrammtypen und Farbgebungen ermittelt werden. Die Auswahl geeigneter Diagrammtypen muss anhand der gegeben Informationen erfolgen. Dabei kommt es auch darauf an, welche Art von Information kommuniziert werden soll. So wäre beispielsweise ein Vergleich verschiedener Werte in einem Kreisdiagramm am besten umzusetzen. Die Farbgebung spielt sowohl bei der Darstellung der Informationen als auch bei der Umsetzung der Komponenten eine Rolle. Dabei können zum einen Signalfarben für Informationen verwendet werden und dezente Farben für Hintergrund oder Tabellen. Zusätzlich muss ein Layout ermittelt werden, in das dann die Informationen eingearbeitet werden können.

3 Anforderungen

In diesem Kapitel werden zunächst die grundlegenden Anforderungen eines SOA-Governance Dashboards untersucht. Dabei wird zunächst untersucht, welche beteiligten Personen einer serviceorientierten Architektur das Interesse haben geeignete Informationen in einem SOA-Governance Dashboard anzuzeigen. Diese Personen werden dann, abhängig von der gewünschten Informationen, in Benutzergruppen gegliedert. Um diese zu ermitteln, wird die hierfür vorhandene Literatur untersucht. Anhand dieser Resultate wird dann zunächst ein Rollenmodell entworfen. Anschließend werden die Informationen, die dargestellt werden sollen, diesen Rollen zugeordnet. Damit soll eine zielgruppengerechte Darstellung der Informationen erreicht werden.

Für die Analyse der Stakeholder und Kennzahlen wurden folgende Quellen herangezogen:

- Dirksen: *SOA Governance in Action* [Dir13]
- Open Group: *SOA Governance Technical Standard* [The08]
- Marks & Bell: *Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology* [MB06]
- Biske: *SOA Governance The key to successful SOA adoption in your organization (Biske)* [Bis08]
- Bieberstein et al.: *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap* [BBF+05]
- Brown et al.: *SOA Governance: Achieving and Sustaining Business and IT Agility* [ZM05]
- Everware-CBDI: *SOA Governance: Challenge or Opportunity?* [All08]
- Oracle: *SOA Governance: Framework and Best Practices* [Afs07]
- IBM: *Web Services project roles* [ZM05]
- Marks: *Service-oriented architecture governance for the services driven enterprise* [Mar08]
- Derler & Weinreich: *Models and Tools for SOA Governance* [DW07]
- ZapThink: *SOA Governance: IT Governance in the Context of Service Orientation* [Blo04]
- Software AG: *SOA Governance - Beherrschen Sie Ihre SOA* [Sof07]
- Erl et al.: *SOA Governance - Governing Shared Services On-Premise and in the Cloud* [EBC+11]
- Rieger & Bruns: *SOA-Governance und -Rollen: Sichern des Mehrwerts einer serviceorientierten Architektur* [RB08]

- Königsberger et al.: *SOA-GovMM: A Meta Model for a Comprehensive SOA Governance Repository* [KSM14]
- Windley: *SOA Governance: Rules of the Game* [Win06]
- Bernhardt & Seese: *A Conceptual Framework for the Governance of Service-Oriented Architectures* [BS09]
- SOA CoE Core Team 2008-2009: *Service Oriented Architecture (SOA) Governance Model* [SOA10]
- ITIL: *Business Relationship Management* [ITI16]
- Schröpfer: *Das SOA-Management-Framework* [Sch10]
- Software AG: *What's the Business Value of SOA? Show It with KPIs* [Sof10]
- McKendrick: *Justifying SOA: 12 key metrics to keep tabs on* [McK09]
- St-Cry: *Baby steps to SOA - step two: measure it* [SC13]
- PNMsoft: *KPI Examples* [PNM16]
- Kohnke & Scheffler & Hock: *SOA-Governance - Ein Ansatz zum Management serviceorientierter Architekturen* [KSH08]
- Will & Köppen: *Zentrales, standardisiertes Monitoring als Grundlage des Service Level Managements in flexiblen SOA-Lösungen* [WK12]

3.1 Stakeholder und Rollen

Als Stakeholder werden alle Personen oder Gruppen bezeichnet, die ein Interesse am Verlauf oder Ergebnis eines Prozesses oder Projektes haben und damit direkt oder indirekt an den Aktivitäten eines Unternehmens beteiligt sind. So sind beispielsweise interne Stakeholder, wie Geschäftsführer, Manager oder Entwickler, direkt und externe Stakeholder wie, Lieferanten oder Kunden indirekt an den Aktivitäten eines Unternehmens beteiligt.

Neben den klassischen Stakeholdern eines Unternehmens kommen mit der SOA-Einführung einer serviceorientierten Architektur eine Reihe neuer Stakeholder hinzu. Dafür müssen zusätzlich spezielle Rollen definiert werden. Dies ist besonders für eine funktionierende SOA-Governance wichtig, da nur so Aufgaben und Verantwortlichkeiten klar abgesteckt werden können. Da diesen Prozess jedes Unternehmen bei einer Einführung durchläuft, existieren inzwischen eine Vielzahl an vordefinierten Stakeholdern in der Literatur. Dennoch müssen diese in jedem Unternehmen individuell an die eigenen Bedürfnisse angepasst werden. Um nun ein allgemeines Rollenmodell aufstellen zu können, müssen diese Stakeholder untersucht und geeignet in Rollen zusammengefasst werden.

Grundlegend kann man zwischen Business- und IT-Stakeholdern unterscheiden. Business-Stakeholder sind sowohl Geschäftsführer als auch Angestellte aus den Fachabteilungen. Diese haben das Ziel, die Qualität und Effizient der Geschäftsprozesse im Hinblick auf die technische Realisierung zu definieren oder zu optimieren, um damit vor allem Kundenerwartungen bestmöglich erfüllen zu können. Auf der anderen Seite steht die IT mit ihren klassischen Stakeholdern wie Projektleitern,

Architekten, Entwicklern oder Testern. Diese sind speziell für die Entwickelung der einzelne Services verantwortlich.

Die Unterscheidung zwischen Business- und IT-Stakeholdern reicht allerdings für ein umfassendes SOA-Governance-Werkzeug nicht aus, um alle Vorteile auszunutzen. Der Hauptnutzen eines solchen Werkzeugs sollte es sein, die Kommunikation und damit die Effektivität innerhalb eines Unternehmens zu steigern. Dies kann erreicht werden, indem für die verschiedenen Stakeholder die wichtigsten Informationen in einem Dashboard dargestellt werden. Da es allerdings nicht sinnvoll ist, für jeden einzelnen Stakeholder eine eigene Dashboardansicht zu entwickeln, ist es vorteilhaft, diese so zu verschiedenen Rollen zusammenzufassen, dass die Anforderungen an den Informationsgehalt in einem Dashboard übereinstimmen.

Eine gewisse Schwierigkeit stellt die unterschiedliche Auffassung der verschiedenen Stakeholder im Unternehmensumfeld dar. Es kann vorkommen, dass die Aufgaben oder Verantwortlichkeiten beispielsweise eines Projektleiters von Unternehmen zu Unternehmen verschieden sind. Deswegen ist es besonders wichtig, dass Stakeholdern mehrere Rollen zugeordnet werden können. Eine flexible Implementierung eines Rollen-Managements ist in einem solchen SOA-Governance-Werkzeug daher unumgänglich.

3.1.1 Analyse

Die Tabellen 3.1, 3.2 und 3.3, zeigen alle Stakeholder, die aus verschieden Quellen zusammengetragen wurden. Hierfür wurde vor allem die Literatur im Zusammenhang mit SOA oder SOA-Governance näher untersucht, sowohl aus Büchern als auch aus akademischen Paper oder Artikeln aus Fachzeitschriften. Aber auch diverse Whitepaper von Softwareherstellern wie Oracle, IBM oder der Software AG wurden berücksichtigt. Die Tabellen greifen Kapitel 3.1.2 vor, da die erst dort beschriebenen Rollen in den Tabellen bereits als Gruppierung verwendet werden.

Zur Übersichtlichkeit wurden die Stakeholder ins Englische übersetzt, da die Mehrheit der Stakeholer bereits englischsprachig war, oder die Übersetzung ins Englische mit aufgeführt wurde. Zudem wäre eine Übersetzung ins Deutsche bei vielen englischen Begriffen, die keine genaue oder nur schwammige Übersetzung bieten, unnötig verwirrend. Besonders auffällig war bei der Analyse, dass die Open Group eine Vielzahl exotischer Stakeholder aufzählte, die in dieser Analyse nicht zwingend in die Auflistung aufzunehmen waren. Exotisch vermutlich deshalb, da die Stakeholder der Open Group aus unterschiedlichsten Unternehmen zusammengetragen wurden. Ansonsten war auffällig, dass teilweise nicht klar spezifiziert wurde, welche Aufgaben und Verantwortlichkeiten die genannten Stakeholder haben. Dies erschwerte eine Gruppierung.

Rolle	Stakeholder	Quelle
Director	Chief Executive Officer	Dirksen
	Chief Technical Officer	Open Group, Marks & Bell
	Chief Information Officer	Open Group
Manager	IT Leader	Marks & Bell
	Technical Lead	Biske
	Center of Excellence Leader	Bieberstein et al.
	IT Services Lead	Marks & Bell
	SOA Director	Open Group
	SOA Governance Lead	Brown et al., Everware-CBDI
	Project Management Process Responsible	Open Group
	Project Manager	Open Group, Oracle, Bieberstein et al., IBM, Marks
	IT Manager	Biske
	Platform Manager	Biske
	Service Manager	Biske
	Sales Manager	Dirksen
	Development Manager	Dirksen
	Product Manager	Derler & Weinreich
	Service Level Manager	Everware-CBDI
Enterprise Architect	Chief Enterprise Architect	Open Group
	Chief SOA Architect	Open Group, Brown et al.
	Enterprise Architect	Open Group, ZapThink, Marks, Oracle, Biske, Erl et al.
	SOA Architect	IBM, ZapThink
	Enterprise Service Architect	Everware-CBDI
Architect	Chief Architect	Open Group, Marks & Bell
	Lead Service Architect	Brown et al.
	Service Architect	Brown et al., Software AG, Bieberstein et al., Oracle, Erl et al.
	Solution Architect	Open Group, Biske
	Architect	Oracle, Bieberstein et al., Marks & Bell, IBM
	Information Architect	Biske
	Technical SOA-Architect	Rieger & Bruns

Tabelle 3.1: Stakeholder I

Rolle	Stakeholder	Quelle
Business Developer	Business Analyst	Open Group, Oracle, Bieberstein et al., IBM, Marks, Biske
	Business Champion	Software AG
	Business Leader	Marks & Bell
	Business Service Champion	Brown et al.
	Process Engineer	Open Group
	SOA Business Analyst	Brown et al.
	Process Flow Designer	Bieberstein et al., IBM
	Service Designer	Bieberstein et al., Brown et al., Software AG
	Business Service Leader	Bieberstein et al.
	Service Analyst	Oracle, Erl et al.
	Business Process Analyst	Bieberstein et al.
	Business Process Developer	Bieberstein et al.
	Operational Process Management Responsible	Open Group
	Lead Analyst	Dirksen
	Business Architect	Open Group, Software AG, Rieger & Bruns
Technical Developer	Service Modeler	IBM, Rieger & Bruns
	Service Developer	Bieberstein et al., Brown et al., Derler & Weinreich, IBM, Oracle, Erl et al., Rieger & Bruns
	Developer	Open Group, Oracle, Marks & Bell, IBM, Software AG, Franchise Tax Board
	SDLC Responsible	Open Group
	Integration Specialist	Open Group, Bieberstein et al.
Tester	Interoperability Tester	Rieger & Bruns, IBM
	Service Integration Tester	Bieberstein et al.
	Test Strategist	Open Group
	Tester	Open Group, Software AG, Bieberstein et al., IBM, Franchise Tax Board

Tabelle 3.2: Stakeholder II

Rolle	Stakeholder	Quelle
Security	Security Specialist	Bieberstein et al., IBM
	SOA Security Specialist	Oracle, Erl et al.
	Security Architect	Open Group, Biske
Administrator	Administrator	Software AG, Derler & Weinreich
	Database Administrator	Open Group, Bieberstein et al., IBM
	SOA Systems Administrator	Bieberstein et al.
	System Administrator	Open Group, Bieberstein et al., IBM
	Metadata-Administrator	Rieger & Bruns
	Service Administrator	Oracle, Erl et al.
	Cloud Resource Administrator	Oracle, Erl et al.
	Service Registrar	Brown et al., Bieberstein et al., IBM
	Services Governor	Bieberstein et al.
	Technical Communications Specialist	Oracle
	Operations Architect	Open Group
	Deployer	Bieberstein et al., IBM
Support	IT Support	Marks & Bell
	Product Support Manager	Dirksen
	Consumer	Rieger & Bruns, Königsberger et al.
	Service Owner	Oracle, Software AG, Königsberger et al.
	Policy Owner	Windley, Bernhardt & Seese
	Business Owner	Franchise Tax Board
	Business Domain Owner	Rieger & Bruns, ZapThink, Open Group
	Business Relationship Manager	ITIL
Partner	Outsourcing Partner	Marks
	Executive Sponsor	Brown et al.
	SOA Business Sponsor	Open Group

Tabelle 3.3: Stakeholder III

3.1.2 Rollenmodell

Die ermittelten Stakeholder wurden im folgenden geeignet zu Rollen zusammengefasst, die ähnliche Aufgaben und Verantwortlichkeiten haben. Dabei wurden 13 Rollen definiert. Alle diese Rollen sollten vor allem in Unternehmen vorhanden sein, die eine serviceorientierte Architektur eingeführt haben. Sie sind so allgemein wie möglich gehalten, damit sie für unterschiedlichste Unternehmen übernommen werden können und somit das Dashboard für die unternehmensspezifischen Stakeholder stets geeignet ist.

Director

Die Geschäftsführung repräsentiert die höchste Entscheidungsebene eines Unternehmens. In den untersuchten Quellen gehören im Zusammenhang mit einer serviceorientierten Architektur *Chief Executive Officer (CEO)*, *Chief Technical Officer (CTO)* oder *Chief Information Officer (CIO)* zu den involvierten Geschäftsführern. Der *CEO* definiert die strategische Ausrichtung des Unternehmens und trägt die Verantwortung für die Unternehmenszahlen, wie beispielsweise Umsatz und Gewinn. Technischer orientiert sind hingegen *CIO* und *CTO*. Diese sind neben dem operativen Geschäft zuständig für technische Verbesserungen, die das Potential haben, für das Unternehmen einen Marktvorteil zu erlangen. Obwohl die einzelnen Aufgaben sich untereinander unterscheiden, sind die Ziele recht ähnlich. Ihr Ziel ist es, das Unternehmen gewinnbringend voranzutreiben. Dazu gehören unter anderem personelle, finanzielle und organisatorische Entscheidungen.

Manager

Als Manager werden alle diejenigen Stakeholder zusammengefasst, die die Kontrolle und den Überblick über die serviceorientierte Architektur oder Teilprojekte daraus haben. Manager gibt es natürlich in den verschiedensten Sparten. Am gängigsten sind wohl *Project Manager*, die für spezielle Projekte verantwortlich sind. Zudem gibt es beispielsweise in einer SOA *Service Manager*, die für spezielle Services verantwortlich sind. Darüber hinaus gibt es auch Stakeholder auf einer höheren Ebene, wie den *SOA-Governance Lead*, der die SOA und dessen Governance steuert und überwacht. Grundlegend sorgen alle für das operative Geschäft. Sie definieren Ziele, überwachen Abläufe und kommunizieren zwischen den unterschiedlichen Teams. Sie berichten meist direkt an die Geschäftsführung und sind das Bindeglied zu allen anderen Angestellten.

Enterprise Architect

In dieser Rolle werden diejenigen Stakeholder zusammengefasst, die maßgeblich die Einführung der serviceorientierten Architektur durchführen und überwachen. Meist sind es die *Enterprise Architects* oder die *SOA-Architects*, die das Große und Ganze der Unternehmensarchitektur betrachten. Ihre zentrale Aufgabe ist es, Architekturentscheidungen, beispielsweise für Infrastruktur oder Schnittstellen von Services, zu treffen, damit über einheitliche Standards Services leicht entwickelt und integriert werden können. Dazu gehört auch das Erstellen von Konzepten für verschiedene Vorgehensweisen, wie zum Beispiel eines einheitlichen Authentifizierungsmodells für Services. Solche Standards müssen

dann natürlich überwacht werden. Zusätzlich zählt zu ihren Aufgaben, die generelle SOA-Vision in allen Bereichen des Unternehmens zu etablieren, Akzeptanz zu schaffen und Unstimmigkeiten zu klären.

Architect

Im Gegensatz zu der Rolle Enterprise Architect werden hier die Architekten zusammengefasst, die sich im Detail mit den Architekturen von Services, Lösungen oder Informationen befassen. So trifft also ein *Service Architect* zum einen Architekturentscheidungen und zum anderen erstellt er die Anforderungsanalyse für einen speziellen Service. Etwas abstrakter betrachtet der *Solutions Architect* eine Lösung, die beispielsweise aus verschiedenen Services orchestriert wird. Dabei plant er entweder die Integration bestehender Services, oder erstellt eine Anforderungsanalyse für einen neuen Service. Der *Information Architect* plant und überwacht den Informationsfluss zwischen verschiedenen Services. Außerdem gehört die Speicherung der Daten und Informationen zu seinen Aufgaben. All diese Architekten arbeiten eng mit den fachlichen und technischen Entwicklern zusammen.

Business Developer

In dieser Rolle werden diejenigen Stakeholder zusammengefasst, die sich mit den fachlichen Aspekten der SOA beschäftigen. Hierzu gehören klassisch die *Business Process Analysts*, die zunächst die Geschäftsprozesse des Unternehmens analysieren und optimieren. Zusammen mit den *Business Architects* entscheiden die *Service Analysts*, welche Geschäftsprozesse als Services abgebildet werden sollen. Diese werden dann von *Service Designern* in einer einheitlichen Beschreibungssprache, wie zum Beispiel der Business Process Model and Notation (BPMN), formuliert. Auch im laufenden Betrieb kontrollieren die Stakeholder dieser Gruppe die Abläufe und passen sie bei Änderung der Geschäftsprozesse entsprechend an.

Technical Developer

Stehen alle Anforderungen an einen Service, werden diese von technischen Entwicklern realisiert. Hierzu gehören die *Service-Modellierer*, die zunächst den Entwurf schreiben, der dann von den *Service-Entwicklern* implementiert wird. Diese arbeiten eng mit den *Testern* zusammen, da das Qualitätsmanagement kein abgeschlossener Prozess ist, sondern sich von Anfang bis zum Ende eines jeden Projektes zieht.

Tester

Bevor ein Service in Betrieb gehen kann, muss er ausführlich auf seine funktionalen und nichtfunktionalen Anforderungen getestet werden. Zudem muss die Kommunikation mit anderen Services betrachtet werden, um sicherzustellen, dass das Zusammenspiel von Service Provider und Service Consumer, das eventuell über einen ESB läuft, keine neuen Probleme aufwirft. Aber auch während dem Betrieb müssen Tests durchgeführt werden, um eine möglichst gute Qualität der Services und

damit eine hohe Kundenzufriedenheit erreichen zu können. Das Testen muss dafür auch stets auf die aktuell verwendeten Komponenten, wie Infrastruktur, Datenbanken oder Servern, ausgeweitet werden.

Security

Das Thema Security wird mit einer serviceorientierten Architektur abteilungsübergreifend. Das bedeutet, es reicht oft nicht mehr aus, einzelne Anwendungen zu sichern, da Services von verschiedenen Anwendungen verwendet werden können. Außerdem entstehen durch die lose Kopplung der einzelnen Services, besonders wenn diese für externe Kunden außerhalb des eigenen Netzwerks hin verfügbar sind, noch mehr Angriffspunkte. Dafür müssen geeignete Sicherheitsanforderungen von *Security-Architects* definiert und dokumentiert werden. Weiterhin überwachen sie den Betrieb der Services, der Infrastruktur sowie der Server und Datenbanken. Ausfälle, beispielsweise durch Distributed Denial of Service (DDoS) Attacken, oder durch unbefugte Zugriffe auf Services, können verheerende Folgen für das gesamte Unternehmen haben.

Administrator

In heutigen Unternehmen gibt es meist eine Vielzahl an Administratoren. Zum einen gibt es, vor allem in größeren Unternehmen, technische Administratoren, die sich um Infrastruktur, Server und Datenbanken kümmern. Zum anderen gibt es fachliche Administratoren, wie den *Metadata Administrator* oder den *Service Registrar*, die dafür sorgen, dass alle Informationen bestmöglich dokumentiert und aktuell gehalten werden. Dabei pflegen und überwachen sie maßgeblich den Betrieb des Repositories. Zusätzlich sorgen die Deployer dafür, dass Services in die bestehende Infrastruktur eingespeist werden, damit diese ohne Komplikationen genutzt werden können.

Support

Der Support nimmt in einer serviceorientierten Architektur ebenfalls eine wichtige Rolle ein. Gerade für den Fall, dass ein Service für einen oder mehrere Kunden angeboten wird und nicht nur unternehmensintern genutzt wird, müssen Probleme möglichst effizient gelöst werden. Der Support ist hierfür Ansprechpartner. Treten Probleme auf, kann der Support Anfragen an die entsprechenden Verantwortlichen weiterleiten, sofern er sie nicht selbst lösen kann. Aber auch unternehmensintern können Probleme auftreten. Wird ein Service von verschiedenen Abteilungen genutzt, können Fragen an die Abteilung, die den Service entwickelt hat, weitergeleitet werden. Dies soll sowohl die Kommunikation im Unternehmen als auch nach außen verbessern, um so die Qualität und damit die Kundenzufriedenheit zu erhöhen.

Consumer

Als Consumer werden diejenigen Stakeholder bezeichnet, die einen Service nutzen, beispielsweise als Kunde oder Entwickler einer anderen Anwendung. In den meisten Fällen ist der *Service Consumer* oder sein Produkt, zumindest bedingt, abhängig von der Funktionalität und der Zuverlässigkeit des Services, den er nutzt. *Service Consumer* sollten deshalb immer auf dem Laufenden gehalten werden, wenn sich etwas ändert oder Probleme auftreten. Diese Rolle dient vor allem der Steigerung der Kommunikation im Unternehmen oder zum Kunden. Dabei ist es eher unüblich, dass ein Stakeholder außschließlich diese Rolle einnimmt. Es kann der Fall eintreten, dass ein Stakeholder der einen anderen Service nutzt Entwickler eines anderen Services ist. Dann bekommt er neben der Rolle des Consumers ebenfalls die Rolle des Technical Developers für einen bestimmten Service zugeteilt.

Owner

Als Owner werden die Verantwortlichen eines Services bezeichnet. Meist handelt es sich hierbei um einen Projektleiter oder Architekten, der als für diese Aufgabe ausgewählt wurde. Verantwortlich kann man für einen Service, eine einzelne Version eines Services, für ein Business Object oder für einen Contract sein. Treten Probleme auf, ist der Owner der Ansprechpartner, der entsprechend dem Problem weiterführende Aktionen einleitet. Genauso wie der Rolle der Consumer ist es eher unüblich, dass ein Stakeholder nur diese Rolle einnimmt.

Partner

Partner gehören in der heutigen Zeit der Globalisierung zu fast jedem größeren Unternehmen. Hierzu gehören zum Beispiel Kollaborationen mit anderen Unternehmen. Diese können in die Entwicklung mit einbezogen sein oder haben Interesse an der aktuellen Entwicklung des Unternehmens. Gerade Partner, die eventuell sehr gute Beziehungen zum Unternehmen pflegen, haben oft einen tieferen Einblick als andere. Ähnliche Interessen haben zudem große Sponsoren eines Unternehmens.

3.2 Kennzahlen

Kennzahlen oder auch Metriken in einer serviceorientierten Architektur werden zur Messung der Effektivität verwendet. Sie geben Aufschluss darüber, was wie geplant umgesetzt wurde, oder wo eventuell nachgebessert werden muss. Besonders wichtige Kennzahlen werden oft als Key Performance Indicators (KPIs) bezeichnet.

3.2.1 Analyse

Die Tabellen 3.4, 3.5 und 3.6 zeigen alle Kennzahlen, die aus verschiedensten Quellen zusammengetragen wurden. Hierfür wurde vor allem Literatur im Zusammenhang mit SOA oder SOA-Governance

näher untersucht, sowohl aus Büchern als auch akademischen Paper oder Artikeln aus Fachzeitschriften. Aber auch diverse Whitepaper von Softwareherstellern wie Oracle, IBM oder der Software AG wurden berücksichtigt.

Die Kennzahlen wurden nach den Aspekten der Anforderungen an eine SOA Governance gruppiert. Zudem wurden Kennzahlen generalisierst, sofern diese nicht allgemein gehalten, sondern Beispiele waren.

Zum Beispiel nennt Schröpfer die Kennzahlen: Anzahl der Service Versionen und Anzahl der außer Betrieb genommenen Service Versionen. Diese wurden zu *Services und Service-Versionen in bestimmter Lebenszyklusphase* generalisiert. Daraus lassen sich dann im Detail die entsprechenden Kennzahlen wie beispielsweise die Anzahl der Services in oder außer Betrieb, aber auch das Verhältnis dieser zueinander, leicht ableiten und auf die Bedürfnisse des jeweiligen Unternehmens anpassen.

Bereich	**Kennzahl**	**Quelle**
Service Life Cycle Management	Services and service-versions in certain life cycle states	Schröpfer
Consumer Management	SLA compliance	Schröpfer, Software AG
	Service availability	Software AG, McKendrick
	Time to integrate new customers	Software AG
	Support requests	St-Cyr
	Customer satisfaction in terms of functionality, performance and quality of services	Schröpfer
	Service usage regarding to the location of consumers	Dirksen
	Percentage of customer issues that were solved by phone call	PNMsoft
Meta Data Management	Services managed with repository	Software AG
	Quality and actuality of the metadata in the repository	Schröpfer
	Documentation of the services in the repository	Dirksen
Organizational Structure	Projects and services that go through the SOA governance process	Marks
	Established service domains	Schröpfer

Tabelle 3.4: Kennzahlen I

Bereich	Kennzahl	Quelle
Portfolio Management	Services per domain	Schröpfer
	Services / -Versions planned	Marks
	Services / -Versions used	Kohnke & Scheffler & Hock
	Time to Web-enable existing assets	Software AG
Architectural Standards	Change frequency of service interfaces	Schröpfer
	Projects compliant with architecture standards	Marks
	Services / Business Objects compliant with policies and standards	Marks
	Exceptions requested/granted	Marks
	Percentage of new defects introduced by change	St-Cyr
	Project cost	St-Cyr
Funding Model	Investment in new services / domains	Schröpfer
	Cost of ownership of architectural stack	Marks
	Revenue per service	McKendrick
	Availability, performance, throughput, interruptions of a service	Will & Köppen
Service Monitoring	Response time of a service	Dirksen
	Service requests	Schröpfer, Dirksen
	Security violations	Marks, Dirksen
	Cycle time (time to market)	Software AG, Open Group

Tabelle 3.5: Kennzahlen II

Bereich	Kennzahl	Quelle
Maturity Measurement	Development time	Software AG, McKendrick
	Total cost of new process rollout	Software AG
	IT operational expenses to company revenue ratio	Software AG
	Maintenance(/Repair) costs/time	Software AG, McKendrick, St-Cyr
	Revenue-to-expense ratio	Software AG
	Time to integrate services	Software AG
	Development efficiency	Schröpfer
	Degree of reuse	Schröpfer, Kohnke & Scheffler & Hock, Software AG, Marks
	Timeliness of milestones in the SOA implementation	Schröpfer
	Services retired	Marks
	Reduction in project and maintenance costs	McKendrick
	Return on investment per service	McKendrick

Tabelle 3.6: Kennzahlen III

3.2.2 Kennzahlen im Detail

Die Kennzahlen, die in vorangegangenem Kapitel zusammengetragen und teilweise generalisiert wurden, werden nun im folgenden näher beschrieben. Zusätzlich werden für die generalisierten Kennzahlen exakte Beispiele für Kennzahlen zur Überwachung der SOA-Governance gegeben.

Service Life Cycle Management

Um einen Überblick über die Lebenszyklusphasen von Services und deren Versionen zu bekommen (*services and service-versions in certain life cycle states*), dienen Kennzahlen, wie die Anzahl dieser, die sich in entsprechenden Lebenszyklusphasen befinden. Diese können auch im Verhältnis zueinander als Kennzahl dienen, beispielsweise um schnell zu erkennen, wie hoch das Verhältnis der Services im Betrieb zu denen ist, die noch entwickelt werden müssen oder die außer Betrieb genommen wurden.

Consumer Management

Der wohl bedeutendste Punkt im Consumer Management ist die Überwachung der SLAs. Hinter dem Begriff SLA-Einhaltung (*SLA compliance*) verbergen sich eine Vielzahl von Kennzahlen. So kann man

hier die Werte, die im SLA festgelegt wurden, wie beispielsweise der durchschnittlichen Antwortzeit, entsprechend mit den aktuellen Werten vergleichen und bekommt so einen Überblick darüber, ob die Vereinbarungen eingehalten werden oder nicht. Durch Kennzahlen wie die Service-Betriebszeit (*application uptime*) oder die Wartungs-Ausfallzeit (*maintenance down time*), lässt sich feststellen, wie oft ein Service benutzbar ist und wie oft er für Wartungsarbeiten außer Betrieb genommen wurde. Generalisiert ist die Service-Verfügbarkeit (*service availability*). Da neue Kunden oder Partner möglichst schnell der Zugriff auf einen Service gewährt werden sollte, spielt die Zeit, die es dauert neue Kunden zu integrieren (*time to integrate new customers*), eine entscheidende Rolle. Möchte man beispielsweise aus Marketinggründen untersuchen, aus welchen Regionen ein Service benutzt wird (*service usage regarding to the location of consumer*), kann eine solche Kennzahl Aufschluss darüber geben. Um die Qualität eines Services zu messen, helfen Kennzahlen wie die Anzahl der Support-Anfragen (*support requests*) oder der Prozentsatz der Kunden-Probleme, die im Telefongespräch erledigt wurden (*percentage of customer issues that were solved by phone call*). Darüber hinaus können solche Kennzahlen auf eine Tendenz der Kundenzufriedenheit schließen lassen. Genaue Angaben dazu können aber ohne eine ausführliche Kundenbefragung oft nicht gemacht werden. Daher scheint die Kennzahl der Kundenzufriedenheit bezüglich der Funktionalität, Performanz und Qualität eines Services (*customer satisfaction in terms of functionality, performance and quality of services*) als schwer messbar.

Meta Data Management

Um sicherzustellen, dass das Repository seine gewünschten Vorteile, wie beispielsweise die Erleichterung der Kommunikation innerhalb der serviceorientierten Architektur, erbringen kann, kann anhand von Kennzahlen wie der Qualität und Aktualität der Meta-Daten (*quality and actuality of the metadata in the repository*) und der Dokumentation der Service (*documentation of the services in the repository*) überwacht werden. Zudem kann die Kennzahl der Anzahl der Services, die mit Hilfe des Repositorys verwaltet werden (*number of services managed with repository*), mit der Anzahl der Services im Portfolio verglichen werden, um festzustellen, ob überhaupt alle Services im Repository hinterlegt sind.

Organizational Structure

Mit Hilfe der Kennzahl Projekte und Services, die einen SOA Governance Prozess durchlaufen (*projects and services that go through the SOA governance process*) kann überwacht werden, welche Abteilungen sich an dem Governance-Prozess eines Projektes oder eines Services beteiligen. Betrachtet man ein spezielles Projekt oder Service sind im Optimalfall alle daran beteiligt, wenn nicht, kann über diese Information Aufschluss gewonnen werden, wo eventuell nachgebessert werden muss. Andersrum kann die Anzahl der Projekte und Services, die einen SOA-Governance-Prozess durchlaufen, mit allen Projekten und Services abgeglichen werden, um über das Verhältnis festzustellen, wie weit die serviceorientierte Architektur bereits im Unternehmen angekommen ist und unterstützt wird. Oft werden Organisations-Strukturen anhand etablierter Service-Domänen (*established service domains*) ausgerichtet.

Portfolio Management

Das Portfolio ist ein Repertoire aus allen Domänen inklusive deren Services. Über Kennzahlen wie die Services pro Domäne (*services per domain*) oder der geplanten Services (*services planned*) lässt sich darüber schnell ein Überblick gewinnen. Dies soll die Arbeit derer unterstützen, die zum einen nach einem existierenden Service suchen, oder derer, die zum anderen die Entscheidung zu treffen haben, ob ein neuer Service entwickelt wird. Dabei können unterstützend die Kennzahlen der bereits genutzten Services (*services used*), die Zeit, die für Wartungsarbeitet aufgewandt werden muss (*maintenance times*) oder die Zeit, die benötigt wird, um dem Kunden bestehende Assets zur Verfügung zu stellen (*time to web-enable existing assets*), herangezogen werden, um so zu entscheiden, ob bestehende Services überhaupt genutzt werden oder wie lange es dauert, diese dem Kunden zur Verfügung zu stellen. Auch hier fällt wieder auf, dass viele dieser Kennzahlen nur sehr schwer automatisch ermittelt werden können.

Architectural Standards

Um zu überprüfen, ob die gewählten Architektur-Standards richtig umgesetzt wurden, dienen Kennzahlen wie die Projekte, die sich an Architektur-Standards halten (*projects compliant with architecture standards*). Diese können sowohl als Anzahl, als auch im Verhältnis zu allen Projekten betrachtet werden. Das gleiche gilt natürlich für Services (*services compliant with policies and standards*) und Datenelemente (*percentage of data elements standardized*). Zusätzlich werden die Architektur-Standards an sich geprüft, in dem beispielsweise die Rate, wie oft das Interface eines Services geändert werden muss (*change frequency of service interfaces*), gemessen wird. Müssen oft Änderungen vorgenommen werden oder ist die Anzahl der Anfragen und Erlaubnisse, ob ein bestehendes Interface geändert werden kann (*number of exceptions requested/granted*) sowie der Anteil der hinzukommenden Fehler pro Änderung (*percentage of new defects introduced by change*) entsprechend hoch, müssen die Architektur-Standards näher betrachtet und gegebenenfalls überarbeitet werden.

Funding Model

Finanzierungsmodelle müssen gerade im dynamischen Unternehmen regelmäßig überprüft werden. So können Kennzahlen wie die Projektkosten (*project costs*) und die Investitionen in neue Services oder Domänen (*investment in new services or domain*) mit den Einnahmen eines Services (*revenue per service*) verglichen werden um festzustellen, ob und inwiefern das für einen bestimmten Service oder eine Domain gewählte Finanzierungsmodell wirtschaftlich ist. Zusätzlich müssen die Anschaffungs- und Betriebskosten von beispielsweise Hardware (*cost of ownership of architectural stack*) regelmäßig überwacht werden, um gegebenenfalls Kosteneinsparungen in Betracht zu ziehen.

Service Monitoring

Mit Hilfe des Services Monitorings können Kennzahlen wie die Verfügbarkeit, Performanz, Durchsatz und Unterbrechungen eines Services (*availability, performance, throughput, interruptions of a service*) überwacht werden. Gerade im Bezug auf die Service Level Agreements sind diese Kennzahlen sehr

wertvoll. Auch kann so überprüft werden, ob ein Service so funktioniert, wie er soll. Zusätzlich kann anhand von Kennzahlen wie die Antwortzeit eines Services (*response time of a service*) und die Anzahl der Serviceanfragen (*service requests*) erörtert werden, ob eine Investition in Entwicklung oder Hardware wirtschaftlichen Vorteil bringen würde. Anhand der Anzahl an Security-Verletzungen (*security violations*), wie beispielsweise der Anzahl an fehlerhaften Login-Versuchen oder der Anzahl an unbekannten Serviceanfragen, können versuchte Angriffe erkannt und im besten Fall schnell abgewehrt werden.

Maturity Measurement

Während der Einführung einer serviceorientierten Architektur sollte stets überwacht werden, ob die Meilensteine eingehalten werden (*timeliness of milestones in the SOA implementation*), damit diese zügig vonstattengeht und möglichst schnell ihre gewünschten Vorteile erbringen kann. Um zu überwachen, wie gut die eingeführte SOA funktioniert, gibt es dann eine Vielzahl an Kennzahlen. Die am häufigsten genannte Kennzahl ist der Wiederverwendungsgrad (*degree of reusability*), der sich aus der Anzahl der Systeme, die einen Service nutzen, dividiert durch die Anzahl der Systeme, die einen Service nutzen könnten, berechnen lässt. Diese Kennzahlen sind aber auch einzeln interessant, wenn man beispielsweise wissen möchte, welche Services am häufigsten bzw. welche am wenigsten häufig wiederverwendet werden (*services re-used*) oder welche bereits außer Betrieb genommen wurden (*services retired*). Am häufigsten genannten Kennzahlen sind hier die verschiedensten Bilanzen, wie der Ertrag des investierten Kapitals eines Services (*return on investment per service*) oder das Verhältnis der IT-Kosten zum Gesamtertrag des Unternehmens (*IT operational expenses to company revenue ratio*). Daraus lassen sich dann wieder Kennzahlen wie das allgemeine Umsatz-Kosten-Verhältnis (*revenue-to-expense ratio*) ablesen. Aber auch Kosteneinsparungen, wie die Reduzierung der Projekt- und Wartungskosten (*reduction in project and maintenance costs*), spielen hier eine Rolle. Die Gesamtkosten einer neuen Prozess-Einführung (*total cost of new process rollout*) oder die Zeit, um auf Änderungen am Markt regieren zu können (*cycle time (time to market)*), können ebenfalls Aufschluss darüber geben, inwiefern das Unternehmen von der eingeführten SOA profitiert. Außerdem sollen, durch die serviceorientierte Architektur sowohl die Entwicklungszeiten (*development time*) reduziert und die Entwicklungseffizienz (*development efficiency*) erhöht werden, als auch die Kosten und die Zeit für Wartungen (*maintenance costs/time*) oder das Integrieren von Services (*time to integrate services*) reduziert werden.

Zusammenfassend muss man sagen, dass die identifizierten Kennzahlen teilweise schwer zu messen sind. Gerade wenn es um Kundenzufriedenheit geht, lassen sich solche Kennzahlen nicht aus vorhandenen Metriken der serviceorientierten Architektur berechnen. Diese könnten allerdings über andere Verfahren, wie beispielsweise im Fall der Kundenzufriedenheit über Umfragen, ermittelt und geeignet hinterlegt werden. Prinzipiell müssen all diese Kennzahlen natürlich von jedem Unternehmen auf die eigenen Ansprüche angepasst werden, da sie oft von den unternehmerischen Zielen abhängen. Nichtsdestotrotz gibt es einige Kennzahlen, wie die Projektkosten, die in jedem Unternehmen vorhanden sein sollten.

3.3 Zuordnung der Kennzahlen zu den Rollen

Nachdem ein Rollenmodell aufgestellt und mögliche Kennzahlen einer serviceorientierten Architektur identifiziert wurden, müssen diese einander zugeordnet werden. Das bedeutet, dass für die gegebenen Kennzahlen untersucht werden muss, für welche Rollen sie interessant sind und diese einsehen können dürfen. Des weiteren haben unterschiedliche Rollen auch unterschiedliche Ansprüche auf ein und dieselbe Kennzahl. Aus diesem Grund reicht es nicht aus zu sagen, Ja, diese Kennzahl sollte der Rolle angezeigt werden oder Nein, diese Kennzahl sollte der Rolle nicht angezeigt werden. Die Tabellen 3.7 und 3.8 zeigen diese Zuordnung. In diesen wurden aus Platzgründen Abkürzungen eingeführt, die deutlich machen sollen, welche Ansprüche die Rollen an die jeweiligen Kennzahlen haben. Die verwendeten Abkürzungen sind dabei jeweils unterhalb der Tabelle beschrieben. Ein Beispiel wäre die Kennzahl *Services and service-versions in certain life cycle states*. Stakeholder der Rolle Director haben den Anspruch, das Verhältnis (V) der Lebenszyklusphasen aller Services oder Service-Versions zu betrachten. Stakeholder der Rolle Manager hingegen interessiert nur eine Übersicht (Ü) der eigenen (*) Services oder Service-Versions. Für Stakeholder der Rolle Owner sind dann nur noch die eigentlichen Lebenszyklusphasen für den Service oder die Service-Version , für die sie verantwortlich sind, wichtig. Für alle anderen Stakeholder, für die ein Minuszeichen eingetragen ist, bedeutet das, dass sie kein Interesse oder keinen Anspruch auf entsprechende Kennzahlen haben.

Kennzahl	D	M	E	AT	B	TR	T	SY	AR	SP	C	O	P
Services and service-versions in certain life cycle states	V	Ü*	-	-	-	Ü*	-	-	-	-	*	*	-
SLA compliance	V	D*	-	D*	-	-	-	-	-	D*	D*	D*	-
Service availability	Ü	Ü*	Ü	V*	-	-	Ü*	-	-	-	D*	D*	-
Time to integrate new customers	Ø	Ø*	Ø	Ø*	-	-	-	-	-	Ø*	-	-	-
Support requests	-	#*	-	-	-	-		-	-	#	#*	#*	-
Customer satisfaction in terms of functionality, performance and quality of services	Ü	Ü*	-	-	-	-	-	-	-	-	-	Ü*	-
Service usage regarding to the location of cunsumers	Ü	Ü*	-	-	Ü	-	-	-	-	-	-	-	-
Percentage of customer issues that were solved by phone call	-	V*	-	-	-	-		-	-	ØV*	-	V*	-
Services managed with repository	V	V*	V	-	-	V*	-	-	V	-	-	-	-
Quality and actuality of the metadata in the repository	-	-	V	-	-	V*	-	-	A	-	-	V*	-
Documentation of the services in the repository	-	-	V	-	-	V*	-	-	A	-	-	-	-
Projects and services that go through the SOA governance process	-	V	V	-	-	-	-	-	-	-	-	-	V
Established service domains	A	-	-	-	A	-	-	-	-	-	-	-	A
Services per domain	-		-	-	Ü	-	-	-	-	-	-	-	-
Services / -Versions planned	-	A	A	-	A	-	-	-	A	-	-	*	A
Services / -Versions used	-	A	-	-	A	-	-	-	-	-	-	*	
Time to Web-enable existing assets	Ø	Ø	-	-	-	-	-	-	-	Ø*	-	-	-
Change frequency of service interfaces	-	Ø	A	Ø	-	-	-	-	-	-		-	-
Projects compliant with architecture standards	-	-	A	*	-	-	-	D	-	-	-	-	-

Tabelle 3.7: Zuordnung der Kennzahlen zu den Rollen I

V = Verhältnis, Ü = Übersicht, # = Anzahl, D = Detailliert, Ø = Durchschnitt, A = Auswahl, * = Eigene
D = Director, M = Manager, E = Enterprise Architect, AT = Architect, B = Business Developer, TR = Technical Developer, T = Tester, SY = Security, AR = Administrator, SP = Support, C = Consumer, O = Owner, P = Partner

Kennzahl	D	M	EA	Ar	BD	TD	T	Se	Ad	Su	C	O	P
Services / Business Objects compliant with policies and standards	-	-	A	*	-	-		D	-	-	-	*	-
Exceptions requested/granted	-	-	A	*	-	-	-	D	-	-			-
Percentage of new defects introduced by change	-	-	Ø	-	-	*		-		-		*	-
Project cost	A	*	-	-	-	-	-	-	-	-	-	-	-
Investment in new services / domains	A												
Cost of ownership of architectural stack	A	*	A	-	-	-	-	-	-	-	-	-	-
Revenue per service	A	-	Ø	-	D	-	-	-	-	-	-	-	-
Availability, performance, throughput, interruptions of a service	-	-	-	-	-	-	-	-	D	-	-	*	-
Response time of a service	-	-	Ø	*	-	-	-	-		-	-	*	-
Service requests	-	-	-	*	-	-	-	-	-	-	-	*	-
Security violations	-	Ø	Ø	-	-	-	-	D	-	-	-	-	-
Cycle time (time to market)	Ø	-	-	-	-	-	-	-	-	-	-	-	-
Development time	-	V	-	-	-	-	-	-	-	-	-	-	-
Total cost of new process rollout	#	*	#*	-	Ø	-	-	-	-	-	-	-	-
IT operational expenses to company revenue ratio	V	-	-	-	-	-	-	-	-	-	-	-	V
Maintenance(/Repair) costs/time	-	Ø	Ø D	-	-	-	-	-	-	-	-	*	-
Revenue-to-expense ratio	V	-	-	-	-	-	-	-	-	-	-	-	-
Time to integrate services	-	Ø	Ø D	-	-	-	-	-	-	-	-	-	-
Development efficiency	-	*	V	-	-	-	-	-	-	-	-	-	-
Degree of reuse	D	-			*	-	-	-	-	-	-	-	-
Timeliness of milestones in the SOA implementation	V	D	-	-	-	-	-	-	-	-	-	-	-
Services retired	#			-	A	-	-	-	-	-	*	-	-
Reduction in Project and Maintenance Costs	V	V	-	-	-	-	-	-	-	-	-	-	-
Return on investment per domain / service	D	-	-	-	D	-	-	-	-	-	-	-	-

Tabelle 3.8: Zuordnung der Kennzahlen zu den Rollen II

V = Verhältnis, Ü = Übersicht, # = Anzahl, D = Detailliert, Ø = Durchschnitt, A = Auswahl, * = Eigene
D = Director, M = Manager, E = Enterprise Architect, AT = Architect, B = Business Developer, TR = Technical Developer, T = Tester, SY = Security, AR = Administrator, SP = Support, C = Consumer, O = Owner, P = Partner

4 Konzept

Grundsätzlich gilt für Dashboards wie auch für viele andere Bereiche der Softwareentwicklung, dass es kein Patentrezept gibt und deshalb jedes Dashboard auf Basis der gegebenen Bedürfnisse entworfen werden muss. Schließlich müssen in den meisten Fällen eine Vielzahl an Stakeholdern mit unterschiedlichen Kenntnissen mit der Benutzung gut zurechtkommen. Das Dashboard soll deshalb sowohl intuitiv als auch möglichst informativ sein. Nachdem im vorangegangenen Kapitel ein Rollenmodell entworfen wurde, mögliche Kennzahlen identifiziert und untersucht wurde, welche Kennzahlen für welche Rollen angezeigt werden sollen, stellt sich natürlich die Frage, wie dem Nutzer eine Kennzahl möglichst effektiv kommuniziert werden kann, sodass er die Aussagekraft der Kennzahl optimal erfassen kann. Obwohl Stakeholder verschiedener Rollen unterschiedliche Interessen an diesem Dashboard haben, sind die grundlegenden Anforderungen die selben. Aus diesem Grund wird im Folgenden ein Konzept für ein solches SOA-Governance Dashboard entworfen. Dafür wird zunächst der Aufbau des Dashboards beschrieben, da dieser keine unerhebliche Rolle spielt. Anschließend werden verschiedene Darstellungsmöglichkeiten für Kennzahlen ermittelt und untersucht. Abschließend wird dann noch geklärt, wie das Rechte-Management der Kennzahlen für die verschiedenen Stakeholdern realisiert werden könnte.

4.1 Aufbau

Einen möglichen Aufbau des Dashboards zeigt Abbildung 4.1. Das Dashboard ist hier in zwei größere Bereiche aufgeteilt. Zum einen in den Bereich für mögliche Benachrichtigungen und zum anderen in den Bereich für Kennzahlen. An erster Stelle können verschiedenste Benachrichtigungen angezeigt werden. Benachrichtigungen sind wichtige Ereignisse innerhalb des Systems, die für den Nutzer informativ sind. So wäre beispielsweise eine Benachrichtigung sehr hilfreich, die einem Service Consumer mitteilen, dass die Service-Version, die er nutzt, nicht mehr gewartet oder bald abgestellt wird. In dem Fall kann er sich frühst möglich überlegen, ob er eine andere Service-Version nutzen möchte oder nicht. Aber auch für andere Stakeholder ergeben sich hierfür Anwendungsfälle, wie beispielsweise der Fall, dass ein Service von niemandem mehr genutzt wird oder der Fall, dass eine neue Service-Version für einen bereits genutzten Service vorhanden ist. Für all diese Fälle sollen solche Informationen so effektiv wie möglich kommuniziert werden. Diese Benachrichtigungen sind ebenfalls, wie die Kennzahlen, benutzerdefiniert und werden nur den Nutzern angezeigt, für die sie bestimmt sind. Da die Kennzahlen dennoch im Vordergrund stehen sollten, nehmen diese auch hier den größten Teil des Dashboards ein. Aus diesem Grund folgt unter den Benachrichtigungen der Block mit den Kennzahlen, die für ihn bestimmt sind. Würde das Dashboard nur für eine bestimmte Zielgruppe entworfen werden, würde es ausreichen, die Kennzahlen sinnvoll anzuordnen und das Dashboard damit statisch zu implementieren. Da hier allerdings ein rollenbasiertes Dashboard entworfen werden

soll, ist dies keine Option. So muss die Anordnung der Kennzahlen dynamisch sein. Dies bringt den Vorteil mit sich, dass der Nutzer sich Kennzahlen aussuchen kann, die er angezeigt bekommen möchte und diese dann verschieden anordnen kann. Aus diesem Grund ist ein kleiner Bereich unterhalb der Kennzahlen notwendig, über den Einstellungen an den Kennzahlen vorgenommen werden können und so der Nutzer die Möglichkeit hat, seine Ansicht zu personalisieren. Dieser kleinere Bereich zählt deshalb zu dem der Kennzahlen.

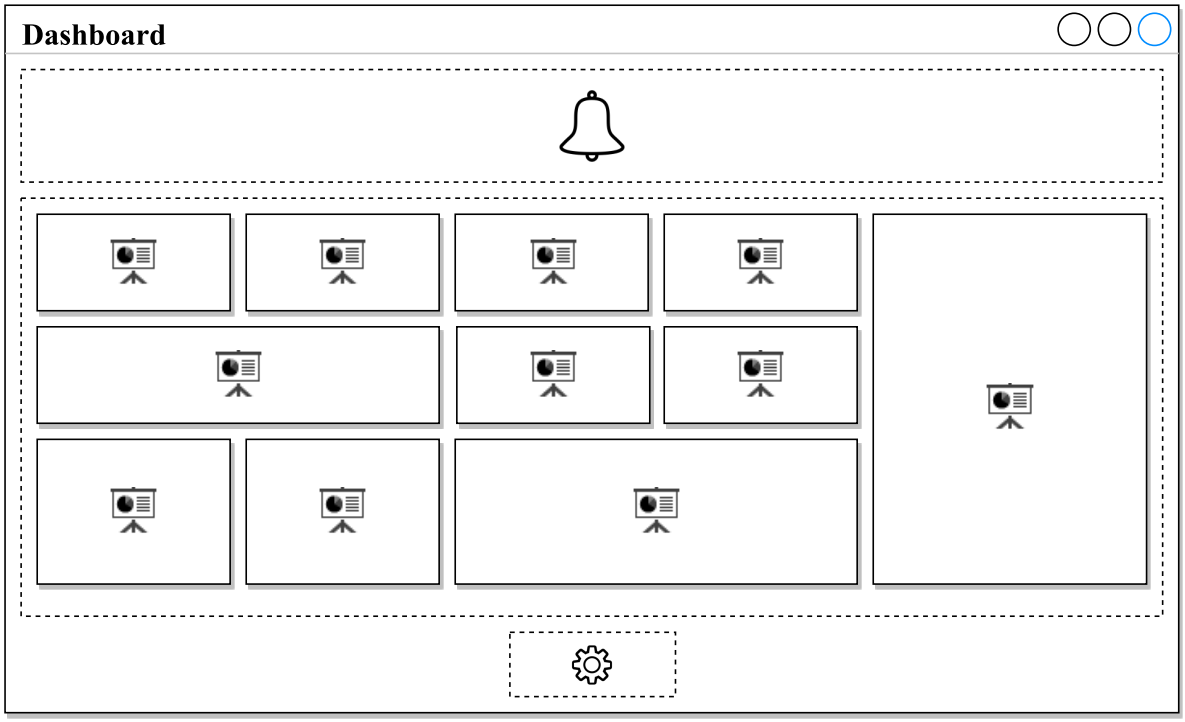

Abbildung 4.1: Aufbau des Dashboards

4.1.1 Benachrichtigungen

Benachrichtigungen dienen dazu, den Nutzern wichtige Statusmeldungen zu präsentieren. Abbildung 4.2 zeigt, wie diese Benachrichtigungen umgesetzt werden können. Da es verschiedene Typen von Benachrichtigungen, wie beispielsweise Infos, Warnungen oder Fehler, gibt, sollten dementsprechend diese Benachrichtigungen auch ihre Botschaften durch spezifische Farbgebungen und den Einsatz von Icons darstellen. Für eine Warnung bietet sich die Hintergrundfarbe Gelb und ein Icon mit einem Ausrufezeichen an, für eine Info die Farbe Blau und ein Icon mit einem kleinen i und für einen Fehler die Farbe Rot und ein Icon mit einem Kreuz oder einem Stopp Symbol. Die genannten Icons haben sich allgemein durchgesetzt und werden in den verschiedensten Kontexten benutzt.

Bekommt der Nutzer eine oder mehrere solcher Benachrichtigungen angezeigt, hat er die Möglichkeit diese zur Kenntnis zu nehmen und anschließend über einen kleinen Löschen-Button auf der rechten Seite als gelesen zu markieren. Die Benachrichtigung wird dann ausgeblendet und bei folgenden

Besuchen nicht wieder angezeigt. Damit soll verhindert werden, dass der Nutzer mit Benachrichtigungen überhäuft wird. Besonders störend ist es, wenn eine solche Vielzahl an Benachrichtigungen untereinander aufgelistet wird und der Rest des Dashboards somit in den Hintergrund rückt. Dies kann auftreten, wenn der Nutzer mehrere Wochen das Dashboard nicht besucht hat und dabei sehr viele Benachrichtigungen angefallen sind. Dem Problem sollte deshalb vorgebeugt werden. Da der Platz, den die Benachrichtigungen einnehmen können, begrenzt ist, muss eine Möglichkeit geschaffen werden, eine Vielzahl an Benachrichtigungen sinnvoll unterzubringen. Wie in Abbildung 4.2 zu sehen, kann dies realisiert werden, indem die Benachrichtigungen auf mehrere kleine Seiten aufgeteilt werden, durch die man dann mit Hilfe von Vor- und Zurück-Buttons navigieren kann. Des weiteren sollte ab einer bestimmten Anzahl an Benachrichtigungen dem Nutzer die Möglichkeit gegeben werden, alle Benachrichtigungen auf einmal zu löschen, da es mühsam wird, jede Benachrichtigung einzeln zu löschen.

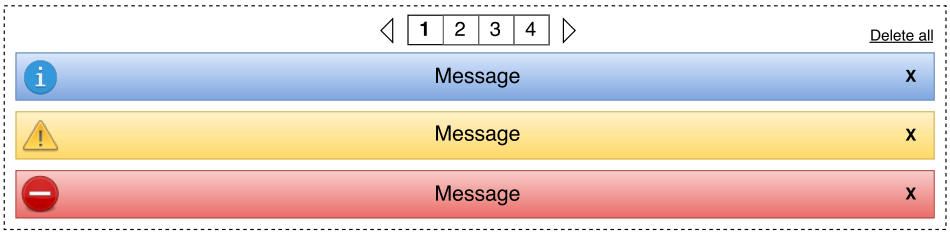

Abbildung 4.2: Benachrichtigungen

Zusätzlich ist es für einige Benachrichtigungen interessant, diese mit einem Link zu versehen. Dieser Link verweist dann auf weiterführende Informationen, wie beispielsweise in dem Fall, dass eine neue Service-Version bereitgestellt wurde. Der Link führt dann direkt zu dieser Version. So muss der Nutzer diese Service-Version nicht extra im System suchen, sondern kann diese Informationen, sofern sie für ihn interessant sind, direkt einsehen. Es gibt verschiedene Möglichkeiten, dies umzusetzen. Zum einen könnte im Beispielfall der Name der neuen Service-Version mit einem Link unterlegt sein. Zum anderen könnte hinter der Benachrichtigung standardisiert ein Link mit dem Namen *See details...* angezeigt werden. Letzteres ist für den Nutzer deutlicher sichtbar und wäre für alle Benachrichtigungen konsistent. Ein Nachteil dagegen wäre, dass bei einer Standardisierung kein genauer Aufschluss über die Informationen gegeben wird, auf die der Link verweist. Aus diesem Grund erscheint die erste Methode sinnvoller, da der Informationsgehalt stärker gewertet werden sollte, als die Konsistenz. Außerdem gewöhnt sich der Nutzer schnell an diese Methode und kann direkt entscheiden, ob die weiterführenden Informationen von Belang sind.

4.1.2 Kennzahlen

Typischerweise werden die entsprechenden Kennzahlen mit einem sogenannten Widget-System realisiert. Ein Widget ist in diesem Fall ein Panel, in dem eine Kennzahl dargestellt wird. Ein solches Widget, wie beispielhaft in Abbildung 4.3 zu sehen, hat im Titel den Namen der Kennzahl und im Inhalt des Widgets die entsprechend aufbereiteten Daten. Das hat den Vorteil, dass durch die Widgets die Kennzahlen klar voneinander getrennt sind und somit besser an einzelne Nutzer gebunden werden können. Eine weiter Möglichkeit wäre gewesen, mehrere Kennzahlen in einem Widget

zusammenzufassen, was allerdings das Gegenteil bewirkt hätte und die Zuordnung zu den Nutzern unnötig erschwert hätte. Dadurch, dass nur eine Kennzahl pro Widget dargestellt wird, erleichtert man die Zuordnung der Kennzahlen an die Nutzer. Betrachtet man das Widget im Abbildung 4.3 näher, fällt auf, dass im rechten oberen Eck einige Funktionalitäten zur Verfügung stehen. Ein Widget sollte zum einen minimiert und gelöscht werden können und zum anderen über einen Button verschiedene weitere Funktionalitäten, wie beispielsweise das Drucken der Darstellung oder Einstellungen zur Darstellung, anbieten. Dies wird am besten realisiert, indem man hierfür ein Menü implementiert. Wechselt man auf die Einstellungen des Widgets, wird die Darstellung ausgeblendet und es erscheinen, wie in Abbildung 4.3, die entsprechenden Einstellungen samt Wiederherstellen- und Speichern-Button. So wird dem Nutzer noch mehr Freiraum für eine personalisierte Ansicht seines Dashboards gegeben, indem er zwischen verschiedenen Darstellungsmöglichkeiten wählen, die Anzahl der Services bestimmen oder gar den Zeitraum für einen Datenvergleich festlegen kann.

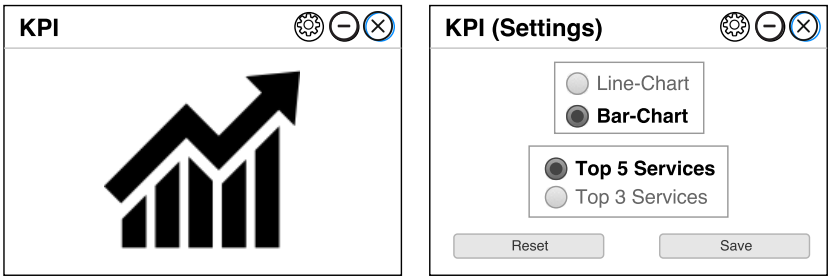

Abbildung 4.3: Beispiel eines Widgets mit dessen Einstellungsmöglichkeiten

Das Widget-System besteht dann aus einer Vielzahl an solchen Widgets, die sich beliebig per Drag & Drop verschieden anordnen lassen. Diese Anordnung der Widgets wird als Konfiguration bezeichnet. Da im Normalfall die Konfiguration des Dashboards nicht sehr oft geändert wird, macht es Sinn, verschiedene Modi für dieses Widget-System einzuführen. So gibt es einen normalen Modus und einen Bearbeitungsmodus. Diese Einstellungen lassen sich im dafür vorgesehenen Bereich, wie in Abbildung 4.4 deutlich wird, bedienen. So kann zwischen den zwei Modi mittels eines Schalters gewechselt werden. Die darunterliegenden Buttons für das Speichern, Wiederherstellen und Hinzufügen von Widgets sollten dementsprechend ausgegraut sein oder betätigt werden können.

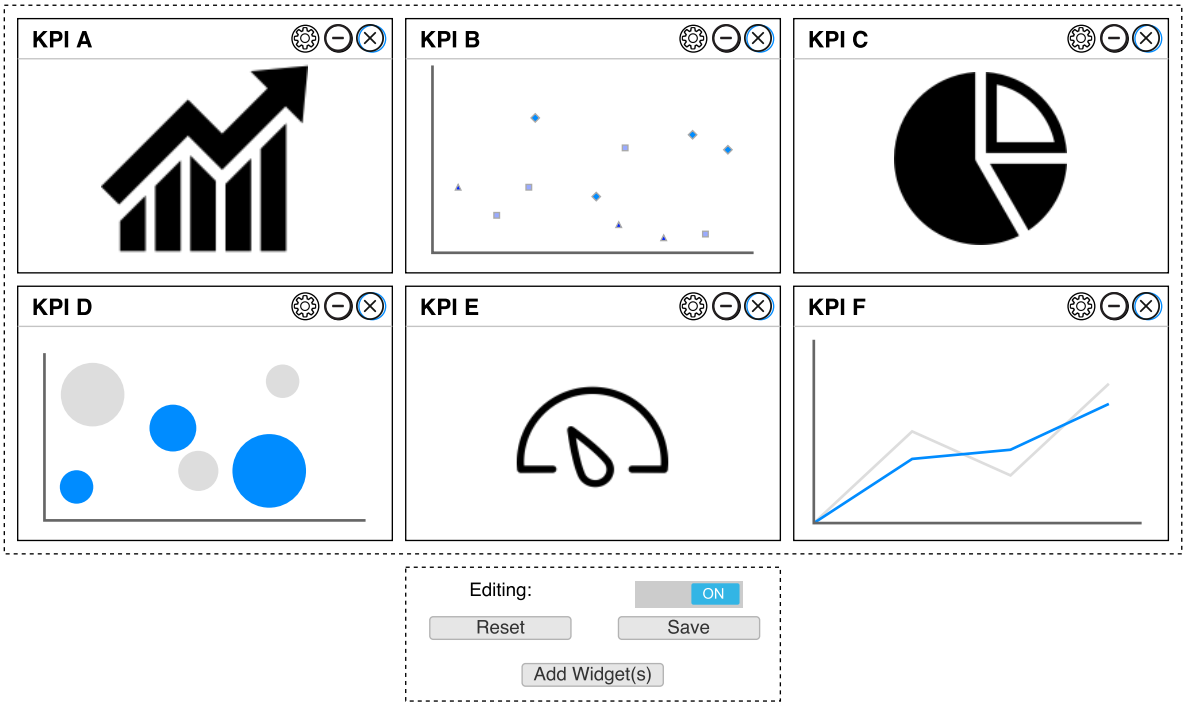

Abbildung 4.4: Widget-System im Editiermodus

Im normalen Modus, in dem sich das Dashboard standardmäßig befindet, lässt sich die Konfiguration der Widgets nicht verändern. Das hat den Sinn, dass der Nutzer beim Arbeiten mit dem Dashboard nicht aus Versehen die Anordnung durcheinanderbringt oder gar Widgets löscht. In diesem normalen Modus hat der Nutzer dann ausschließlich die Möglichkeit, ein Widget zu minimieren, um die Kennzahlen so hervorzuheben, wie er es aktuell benötigt. Zusätzlich stehen ihm Funktionen wie Drucken oder Einstellungen zur Verfügung.

Entscheidet sich der Nutzer, in den Bearbeitungsmodus zu wechseln, steht ihm die Möglichkeit zur Verfügung, Widgets zu löschen, zu verschieben oder neue hinzuzufügen. Da der Nutzer die Größe der einzelnen Widgets beliebig verändern kann und somit nicht jedes Widget gleich groß sein muss, sollte der Nutzer diese per Drag & Drop verschieden anordnen können. Somit wird die Benutzerfreundlichkeit erheblich gesteigert. Wird die Größe eines Widgets geändert, sollte sich die Darstellung der Kennzahl natürlich entsprechend ändern und sollten sich die übrigen Widgets dem Layout anpassen. Damit soll der Nutzer das Dashboard so personalisieren können, dass ein übersichtliches Layout für seine Zwecke entsteht. Fängt er allerdings an, das Dashboard zu bearbeiten und merkt er dann, dass es davor besser war, sollte ihm die Möglichkeit gegeben werden, über einen Zurücksetzen-Button, diese Änderungen rückgängig zu machen.

Wird ein Widget gelöscht, verschwindet es aus dem Dashboard. Dieses kann dann in einem separaten Bereich wieder hinzugefügt werden. Das Widget Bereich bietet die Möglichkeit, bereits gelöschte oder neu hinzugekommene Kennzahlen in das Dashboard mit aufzunehmen. Kennzahlen können hinzukommen, indem die Rechte des Nutzers auf entsprechende Kennzahlen hinzugefügt werden. Das gleiche gilt natürlich für Kennzahlen, für die der Nutzer die Rechte entzogen bekommen hat. Diese

werden ihm dann in Zukunft nicht mehr angezeigt. Um mögliche Missverständnisse auszuschließen, ist es hilfreich, wenn vor dem Löschen eines Widgets ein Bestätigungs-Pop-up erscheint, was den genauen Sachverhalt erläutert und erst bei einer Bestätigung das Widget schließt. Hierbei kann dem Nutzer durch eine Checkbox die Möglichkeit gegeben werden, dieses Pop-up in Zukunft nicht mehr anzuzeigen.

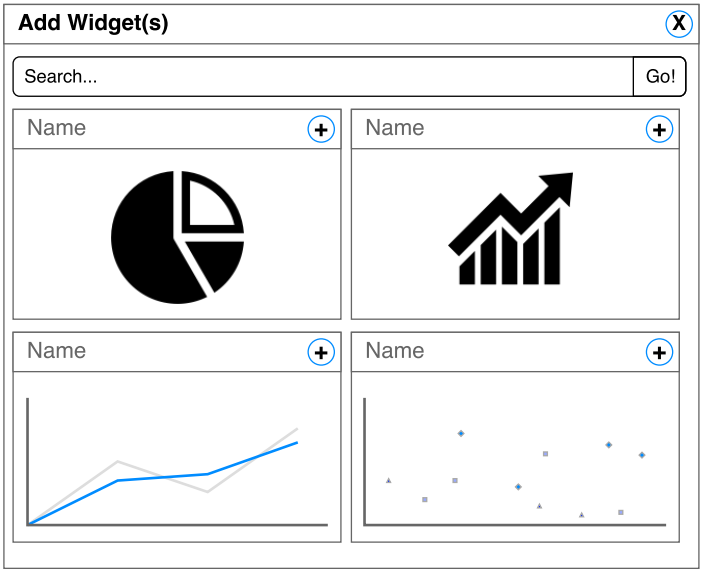

Abbildung 4.5: Bereich zum Hinzufügen von Kennzahlen

Das Aus- und Einblenden dieses Bereiches kann über einen dafür vorgesehenen Button gesteuert werden. Für eine Umsetzung dieses Bereiches gibt es verschiedene Möglichkeiten. Der Bereich kann entweder unter oder neben dem Dashboard liegen oder aber sich über ein Pop-up öffnen lassen. Da in den meisten Fällen sehr viele Kennzahlen zur Verfügung stehen, wird die Seite sehr lang und unübersichtlich, sofern sich diese weiteren Kennzahlen darunter ausklappen lassen. Werden diese weiteren Kennzahlen an der Seite des Dashboards ausgeklappt, müssten die aktuellen Kennzahlen Platz machen und somit kleiner werden oder nach unten rutschen, was ebenfalls den Effekt hat, dass es sehr unübersichtlich wird. Als am besten geeignet ist daher ein Pop-up, wie es beispielhaft in Abbildung 4.5 zu sehen ist. In diesem erscheinen dann die Widgets mit einer entsprechenden Vorschau der Darstellung. Das hat den großen Vorteil, dass der Nutzer bereits vor dem Hinzufügen erkennen kann, ob die Kennzahl für ihn wirklich interessant ist. Er muss es dafür nicht erst durch das Hinzufügen ausprobieren. Das Hinzufügen von Widgets lässt sich auf zwei unterschiedliche Arten realisieren. Die beste Möglichkeit ist, dass sich die Wigets aus dem separaten Bereich direkt per Drag & Drop an die gewünschte Stelle im Dashboard verschieben lassen. Das hat den großen Vorteil, dass der Nutzer direkt weiß, wo das neu hinzugefügte Widget platziert ist. Zum anderen gibt es die Möglichkeit, über einen Hinzufügen-Button an den einzelnen Widgets diese in das Dashboard mit aufnehmen. In beiden Fällen sollten allerdings die hinzugefügten Widgets aus diesem separaten Bereich verschwinden und im Dashboard auftauchen. Diese Widgets sollten dann zudem im Dashboard kurzzeitig farblich markiert werden, um schnell erkennen zu können, welche Kennzahlen neu hinzugefügt wurden.

Da sehr viele Kennzahlen zur Verfügung stehen können, sollte im oberen Teil des Pop-ups eine Suchfunktion dem Nutzer die Möglichkeit geben, schnell eine bestimmte Kennzahl zu finden.

Sobald der Nutzer mit der Bearbeitung seines Dashboards fertig ist, sollte ihm die Möglichkeit gegeben werden, die Konfiguration zu speichern. Das System speichert diese dann beispielsweise in einer Datenbank und fragt sie bei zukünftigen Starts wieder ab. Mit dem Speichern sollte das Dashboard dann wieder in den normalen Modus wechseln, da dem Nutzer so unnötige Interaktionen erspart bleiben. Im anderen Fall, dass der Nutzer seine alte Konfiguration wiederherstellen möchte, sollte das Dashboard weiterhin im Bearbeitungsmodus bleiben, da der Nutzer beispielsweise ein falsches Widget gelöscht haben könnte und es nicht extra suchen möchte, um es wieder in das Dashboard aufzunehmen und stattdessen ein anderes Widget zu löschen.

4.2 Darstellung der Kennzahlen

Grundlegend gilt, dass es für eine Kennzahl mehrere Darstellungsmöglichkeiten gibt, die jeweils andere Stärken und Schwächen haben und damit auch eine andere Aussagekraft haben können. So kann die Kennzahl der Service-Verfügbarkeit zum einen für einen Service und zum anderen für mehrere Services gelten. Handelt es sich um mehrere Services, eignet sich besonders ein Flächendiagramm, in dem man sehr viel Informationen zusammengefasst bekommt. Interessiert allerdings der zeitliche Verlauf weniger und geht es primär darum, ob ein Service aktuell verfügbar ist oder nicht, so reicht ein Ampel-Modell aus. Aus diesem Grund muss für jede Kennzahl entschieden werden, welche Information diese kommunizieren soll welche Darstellung hierfür am besten geeignet ist. Im folgenden werden deshalb verschiedene Darstellungsmöglichkeiten untersucht und entsprechende Anwendungsfälle beschrieben.

In einigen Fällen sollte man die Darstellungsform der Kennzahl dem Nutzer selbst überlassen. So wird in einem solchen Fall eine Darstellung zunächst gewählt, die dann in den Einstellungen eines Widgets geändert werden kann. Diese verschiedenen Darstellungen müssen natürlich entsprechend vorbereitet werden. Der Nutzer kann sich dann aus der Liste der vorbereiteten Darstellungen eine aussuchen.

4.2.1 Ampel

Ein beliebtes Modell für die Darstellung einer Kennzahl ist das Ampel-Modell, also eine Graphik, die eine Ampel zeigt. Diese kann Grün, Gelb oder Rot signalisieren. Zeigt die Ampel Grün, ist alles in Ordnung, zeigt sie Gelb, könnte etwas nicht stimmen und zeigt sie Rot, stimmt etwas sicher nicht. Natürlich handelt es sich hierbei nicht um ein klar definiertes Modell, allerdings scheinen fast alle Nutzer das Ampel-Modell richtig interpretieren zu können und eignet sich deshalb besonders gut. Gerade eine richtige Interpretation solcher Darstellungen ist besonders wichtig, um mögliche Missverständnisse auszuschließen. Eine große Rolle bei einem Ampel-Modell spielt dabei die Farbgebung, die den Nutzer schnell das Richtige interpretieren lässt. Diese Farbgebung kann auch für andere Darstellungen, wie beispielsweise in Abbildung 4.7 zu sehen, äußerst hilfreich sein. Das Ampel-Modell reicht allerdings oft nicht aus, da seine Aussagekraft meist nicht sehr hoch ist. Zur Darstellung der Kennzahlen helfen

daher meist Grafiken wie Balkendiagramme, Säulendiagramme, Kurvendiagramme, Punktdiagramme oder Strukturdiagramme. Aber auch Schaubilder oder Icons erreichen oft das gewünschte Ergebnis.

4.2.2 Icon

In den meisten Fällen werden mit Icons Vergleichswerte dargestellt, also beispielsweise ein Ist- und ein Soll-Wert. Am simpelsten wäre es, die Werte in Textform getrennt durch ein /-Zeichen zu präsentieren, also beispielsweise im Fall der Projektkosten links der aktuelle Stand und rechts die geplanten Kosten. Hierbei helfen auch oft Icons, wie im Fall von Abbildung 4.6, in dem die aktuelle Entwicklungszeit mit derjenigen der letzten Monate verglichen wird. Da hierbei eine geringere Anzahl besser ist, ist das Dreieck, das eine Verbesserung signalisieren soll, grün und mit der Spitze nach oben gerichtet. Hätte sich die Zeit verlängert, wäre das Dreieck rot und die Spitze würde nach unten zeigen. Andere Anwendungsfälle für diese Darstellung könnte die Zeit sein,die benötigt wird, um neue Kunden zu integrieren oder die Änderungshäufigkeit von Service-Schnittstellen. Insgesamt geht es bei dem Einsatz von Icons darum, die nackten Zahlen ansehnlicher aufzubereiten und bei der Interpretation zu unterstützen. Icons können dabei verschiedenste Formen, wie beispielsweise Dreiecke oder Pfeile annehmen.

Abbildung 4.6: Beispiel einer Darstellung mittels einem Icon

4.2.3 Fortschrittsanzeige

Die Darstellung mittels einem Icon kann in manchen Fällen ausreichen, ist allerdings weniger aussagekräftig. Deutlicher hingegen ist, wie im Fall der Dokumentation von Services in Abbildung 4.7 zu sehen, der Einsatz von Fortschrittsanzeigen. Hier wird schnell sichtbar, inwiefern die Services bereits dokumentiert sind. Zusätzlich wird hier der Fortschritt, wie im oben beschriebenen Ampel-Modell, farblich gekennzeichnet. Ist der Service ausreichend dokumentiert, ist der Balken grün, besteht Verbesserungsbedarf, ist er gelb und gibt es erhebliche Lücken, ist er rot. Zudem wird auch stets die aktuelle Prozentzahl angegeben, um nicht nur wie bei der Farbgebung einen Richtwert zu lesen, sondern auch den tatsächlichen Wert.

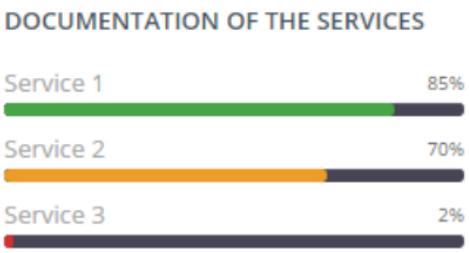

Abbildung 4.7: Beispiel einer Darstellung mittels einer Fortschrittsanzeige

4.2.4 Kreisdiagramm

Kreisdiagramme eignen sich beispielsweise besser, um das Verhältnis mehrerer Werte zueinander vergleichen zu können. Diese haben, obwohl es hier weniger auf die Farbgebung ankommt, ebenfalls eine sehr hohe Aussagekraft. Als Beispiel wurde in Abbildung 4.8 das Verhältnis der Lebenszyklusphasen von allen Service-Versionen dargestellt. Wichtig ist vor allem die Legende, die zeigt, welche Farbe für welche Lebenszyklusphase steht. Zusätzlich hilfreich sind Angaben innerhalb der gefärbten Kreisbereiche. Diese sollten entweder die entsprechenden Prozentangaben, die absoluten Werte oder eine Kombination aus beiden anzeigen.

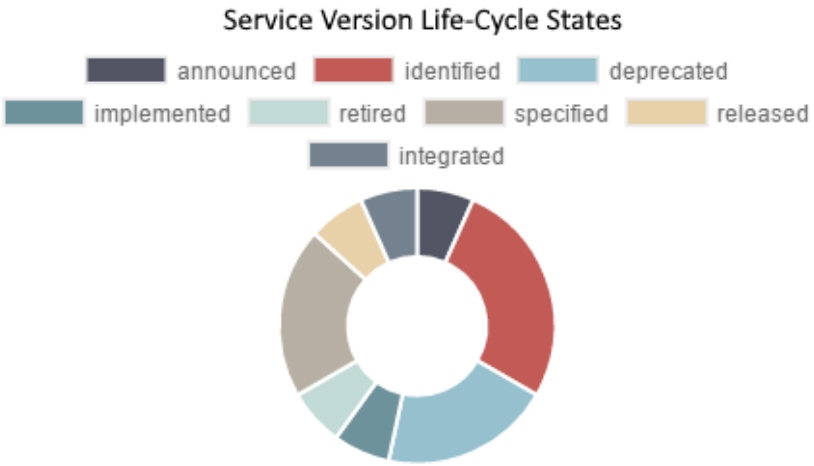

Abbildung 4.8: Beispiel einer Darstellung mittels einem Kreisdiagramm

4.2.5 Flächendiagramm

Flächendiagramme eignen sich dafür, das Verhältnis mehrerer Werte in Kombination mit anderen Werten darzustellen. Ein Beispiel ist der Vergleich der Anzahl von Services, Business Objects und Consumer über einen bestimmten Zeitraum in Abbildung 4.9. Dabei wird deutlich, wie sich die Anzahl der verschiedenen Elemente über einen bestimmten Zeitraum zueinander verhält. Das hat zum einen den Vorteil, dass die absoluten Werte der Elemente bereits enthalten sind und zum anderen überblickt werden kann, wie die absoluten Werte im Verhältnis zueinander stehen und sich im Jahresverlauf verändern.

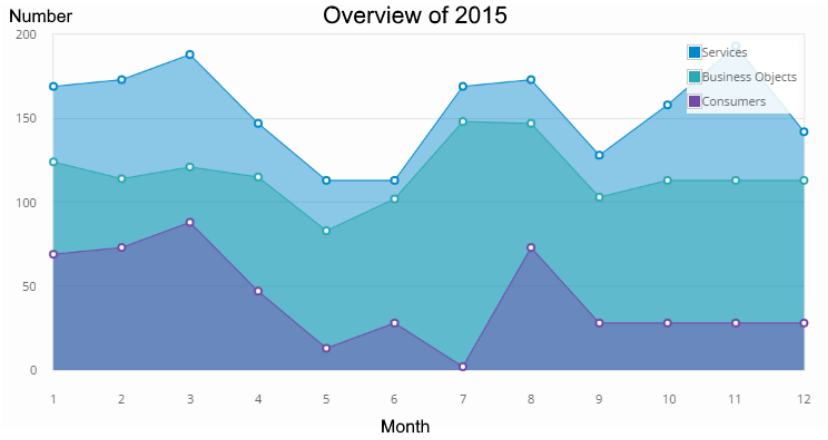

Abbildung 4.9: Beispiel einer Darstellung mittels einem Flächendiagramm

4.2.6 Liniendiagramm

Betrachtet man das vorangegangene Beispiel, erkennt man, dass es sinnvoll sein kann, den Verlauf nur eines Wertes zu verfolgen. Dafür eignen sich besonders Liniendiagramme. Wie in Abbildung 4.10 zu sehen, wird dabei die Anzahl der Verkäufe eines Artikels im Jahresverlauf gezeigt. Diese Darstellung eignet sich beispielsweise auch für die Anzahl der Support-Anfragen oder die Verfügbarkeit eines oder mehrerer Services, eben überall dort, wo ein Wert über einen gewissen Zeitraum betrachtet werden soll. Das ist allerdings nicht der einzige Anwendungsfall eines Liniendiagramms. Die X-Achse muss natürlich nicht zwangsläufig die Dimension Zeit sein, sondern kann auch andere Werte annehmen. Gleiches gilt für die Y-Achse.

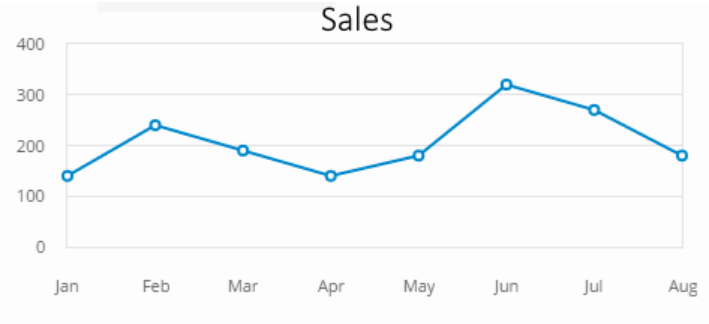

Abbildung 4.10: Beispiel einer Darstellung mittels einem Liniendiagramm

4.2.7 Balkendiagramm

Oft ist allerdings auch ein Vergleich von Werten verschiedener Objekte gefordert. Hier sind Balkendiagramme aussagekräftiger als Liniendiagramme. Wie in Abbildung 4.11 zu sehen, wird hier die Anzahl von Objekten wie Services, Business Objects, Consumer usw. miteinander verglichen. Diese

Darstellungsform hat den Vorteil, dass zum einen die absoluten Werte, als auch das Verhältnis der einzelnen Werte deutlich wird.

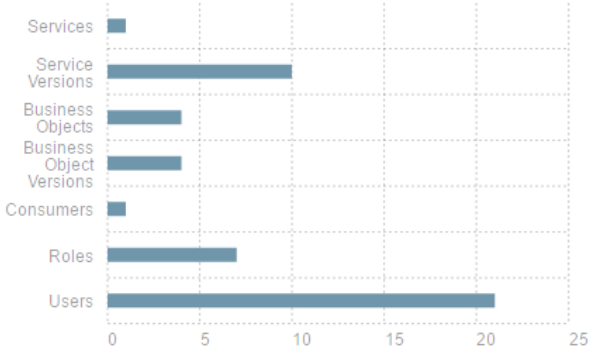

Abbildung 4.11: Beispiel einer Darstellung mittels einem Balkendiagramm

4.2.8 Tachodiagramm

Handelt es sich bei einer Kennzahl um Echtzeit-Werte, wird häufig ein Tachodiagramm verwendet. Abbildung 4.12 zeigt eine Beispieldarstellung für die Anzahl der Service-Aufrufe pro Stunde. Hierbei schwenkt der Zeiger immer auf die aktuell gemessene Anzahl. Diese Echtzeit-Darstellung eignet sich besonders für Kennzahlen im Bereich des Verkaufs. Es kann aus diesem Diagramm abgelesen werden, ob der Service die gewünschte Anzahl an Aufrufen erreicht. So können durch Farbgebung für jede Kennzahl verschiedene Bereiche definiert werden. In dem Beispielfall sind zwischen 70 und 210 Service-Aufrufe pro Stunde vertretbar. Alle Aufrufe darüber könnte die Server überlasten und alle Werte darunter könnten finanzielle Verluste mit sich bringen. Der gelbe und grüne Bereich sind für interne Zwecke nochmals hälftig. Mehr Aufrufe bedeutet in diesem Fall mehr Umsatz. Eine mögliche Einstellungsmöglichkeit wäre hier, den Zeitraum ändern zu können, sprich die Service-Aufrufe pro Minute, pro Stunde oder pro Tag anzuzeigen.

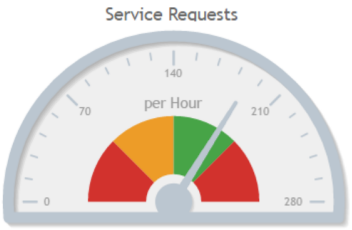

Abbildung 4.12: Beispiel einer Darstellung mittels einem Tachodiagramm

4.2.9 Echtzeitdiagramm

Wie in dem vorangegangenen Abschnitt tritt der Fall, dass Echtzeit-Werte dargestellt werden sollen, öfters auf. Hierbei helfen sogenannte Echtzeitdiagramme. In den meisten Fällen sind im Themengebiet

Service-Monitoring die Kennzahlen in Echtzeit erforderlich. In Abbildung 4.13 wird die Server-Gebrauch in einem Flächendiagramm animiert dargestellt. Dabei bewegt sich die Fläche nach links entsprechend der aktuell erfassten Werte. Natürlich lässt sich diese Art der Darstellung mit den meisten Diagrammtypen erzeugen. Des weiteren gilt zu entscheiden, in welchen Abständen ein neuer Wert in das Diagramm aufgenommen wird. So kann es zum einen ein fließender Übergang sein oder zum anderen ein Intervall von einer Sekunde, einer Stunde oder jeden beliebigen anderen Zeitabstand. Auch hier bietet sich eine Einstellungsmöglichkeit sehr gut an.

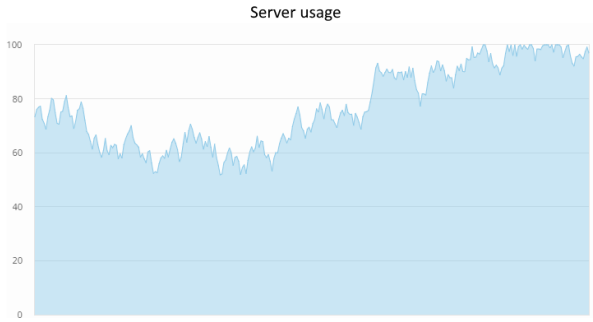

Abbildung 4.13: Beispiel einer Darstellung mittels einem Echtzeitdiagramm

4.2.10 Kartenmaterial

Eine weitere interessante Darstellung lässt sich mittels Kartenmaterial realisieren. Als Beispiel ist in Abbildung 4.14 das Verhältnis dargestellt, aus welchem Land wie viele Service-Aufrufe pro Tag kommen. Diese Kennzahl hilft zum einen Administratoren zur Überwachung der Ressourcen und zum anderen dem Marketing zur Bewertung des Markterfolgs. Natürlich müssen entsprechende Farben in einer Legende erläutert werden. Es müssen die Länder jedoch nicht zwangsläufig eingefärbt sein. Entsprechende Daten können auch mittel Kreisdiagrammen oder ähnlichem, innerhalb der einzelnen Länder dargestellt werden.

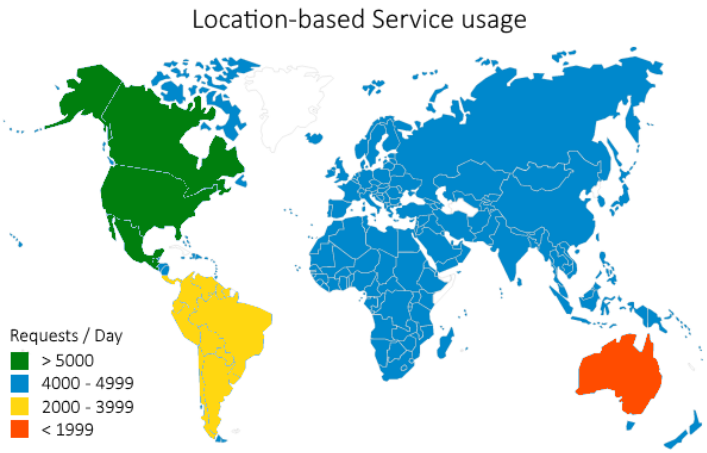

Abbildung 4.14: Beispiel einer Darstellung mittels Kartenmaterial

4.3 Rechte-Management der Kennzahlen

In den meisten Fällen hat nicht jeder Nutzer das gleiche Interesse oder die gleiche Berechtigung, jede Kennzahl einsehen zu können. Es muss also definiert werden, welche Kennzahlen für welchen Nutzer bestimmt sind. Da es gerade bei einer sehr hohen Kennzahlen- und Nutzer-Anzahl wenig Sinn macht, für jeden einzelnen Nutzer die jeweiligen Berechtigungen zu vergeben, sollte dies im Kontext eines Rollen-Systems realisiert werden. So werden also verschiedene Rollen definiert, die einem Nutzer zugeordnet werden. Für diese Rollen werden dann die entsprechenden Berechtigungen auf die Kennzahlen gesetzt. Somit hat ein Nutzer, der eine oder mehrere Rollen haben kann, automatisch die entsprechenden Berechtigungen. Um das zu realisieren, benötigt man ein flexibles Rechte-Management. Flexibel deshalb, weil Berechtigungen einem Nutzer jederzeit hinzugefügt oder genommen werden können, wenn sich beispielsweise dessen Verantwortlichkeiten oder Aufgaben ändern.

Wird das Dashboard in ein bestehendes System integriert, muss zunächst geprüft werden, ob bereits ein entsprechendes Rechte-Management implementiert ist und ob es für diesen Zweck erweitert werden kann. In dem Fall, dass das Dashboard Teil einer Neuentwicklung ist, kann hier bereits Einfluss darauf genommen werden, um später das flexibles Management der Rechte ermöglichen zu können. Für jede Kennzahl sollte eine Berechtigung implementiert sein, die definiert, ob ein Nutzer diese Kennzahl anzeigen darf. Dafür müssen Rollen erstellt werden können, die aus verschiedenen Berechtigungen bestehen. Diese Rollen können dann einem Nutzer zugeordnet werden. Über diese Rollen hat der Nutzer dann die Berechtigungen auf gewisse Kennzahlen. So sollte im System die Möglichkeit bestehen, alle Rollen und alle Nutzer zu verwalten. Es müssen also sowohl Rollen als auch Nutzer erstellt, bearbeitet und gelöscht werden können.

Das Dashboard sollte dann vor der Instanziierung prüfen, für welche Kennzahlen der Nutzer eine Berechtigung hat. Loggt sich der Nutzer das erste Mal ein, werden alle Kennzahlen, für die er eine Berechtigung hat, in das Dashboard mit aufgenommen. Für alle folgenden Logins muss für jede Kennzahl erneut geprüft werden, ob entweder eine Berechtigung hinzugekommen ist, oder eine entfernt wurde. Wir dem Nutzer eine neue Berechtigung erteilt, erscheint diese in dem bereits in Kapitel 4.1.2 beschriebenen Bereich *Add Widget(s)*. Da dieser Bereich nicht immer sichtbar sein muss, macht es Sinn, ihm zusätzlich eine Benachrichtigung mit dem Inhalt: *You now have the permission to show the KPI: [Name of the KPI] in your Dashboard* anzuzeigen, da es sonst passieren könnte, dass er nur aus Zufall auf die neu hinzugekommene Kennzahl stößt. Mit diesem Hinweis kann er dann, wenn er möchte, diese Kennzahl in die Konfiguration seines Dashboards mit aufnehmen. Wird dem Nutzer allerdings eine Berechtigung genommen, ist es nicht zwangsläufig notwendig, dies in einer Benachrichtigung zu übermitteln. Aus Zwecken der Konsistenz wäre eine Benachrichtigung dennoch benutzerfreundlicher, unter anderem um Missverständnisse zu vermeiden, wie beispielsweise in dem Fall, dass der Nutzer sich nicht daran erinnert, die Kennzahl in seine Konfiguration mit aufgenommen zu haben und diese in dem Bereich *Add Widget(s)* vergeblich sucht.

Zusammengefasst müssen die Berechtigungen auf Kennzahlen also in zwei Bereichen geprüft werden. Zum einen im Bereich der dargestellten Kennzahlen in der aktuellen Konfiguration des Nutzers und zum anderen im Bereich *Add Widget(s)*.

5 Prototyp

Das im vorangegangen Kapitel konzeptionell entworfene SOA-Governance Dashboard soll abschließend in einem bereits existierenden Prototyp für ein SOA-Governance Repository umgesetzt. In diesem Kapitel wird deshalb zunächst der zu erweiternde Prototyp beschrieben. Hierfür wird sowohl auf seine Funktionalität als auch seine Architektur eingegangen. Anschließend erfolgt eine Beschreibung der Implementierung der grundlegenden Komponenten, die als Vorbereitung für die folgenden Implementierungsschritte dient. Abschließend werden dann die im Konzept beschriebenen Funktionalitäten für Benachrichtigungen und Kennzahlen umgesetzt. Hierfür erfolgt jeweils eine Implementierung als auch eine Beschreibung der umgesetzten Anwendungsfälle, die beliebig erweitert werden können.

5.1 Governance Repository

Im Zuge eines Studienprojekts wurde ein SOA Governance Repository auf Grundlage des *SOA-Governance Meta-Model* [KSM14] an der Universität Stuttgart entwickelt. Dabei wurde dieses Meta-Model prototypisch umgesetzt. Hierbei wurde eine JavaEE-Webanwendung mittel JavaServer Faces (JSF) [Ora15] implementiert, die auf einem Application Server läuft und über einen Internetbrowser abgerufen werden kann. Diese Anwendung wurde im Zuge einer Masterarbeit erweitert. Hierbei wurde die bestehende relationale Datenbank durch einen RDF Triplestore ersetzt.

In Abbildung 5.1 ist ein Screenshot des Prototypen nach erfolgreichem Login zu sehen. Die Oberfläche ist in verschiedene Tabs eingeteilt. Der erste Tab ist der für das Dashboard. Dieses ist ebenfalls in der gegebenen Abbildung zu sehen. Zum einen werden hier die System-Events in einer Liste und zum anderen die Informationen über das gegeben System wie den Host, die Version und die Anzahl der Artefakte wie Services, Business Objects usw. angezeigt. Man könnte diese Informationen bereits als Kennzahlen betrachten. Dieses Dashboard erfüllt allerdings nicht die gewünschten Anforderungen für ein solches Governance Repository. Es soll deshalb entsprechend dem im letzten Kapitel vorgestellten Konzept ausgebaut werden.

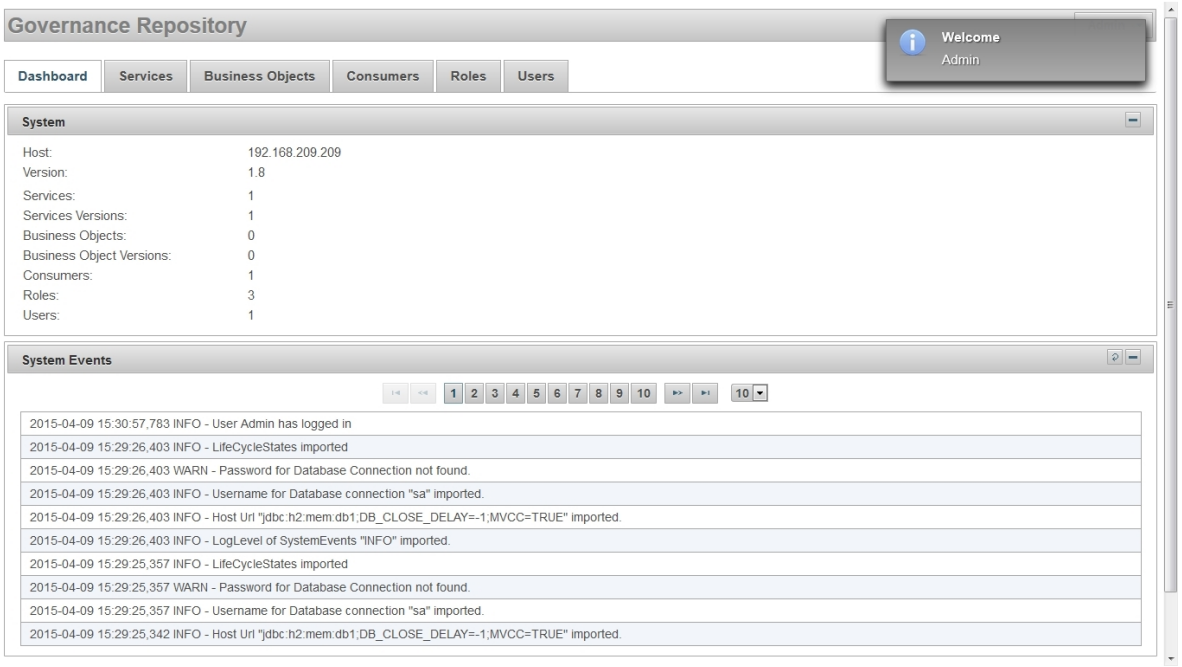

Abbildung 5.1: Screenshot des Governance Repositories

5.1.1 Funktionalität

Das Governance Repository hat den Zweck, Meta-Daten einer serviceorientierten Architektur zu verwalten. Meta-Daten sind in diesem Fall Informationen über Services, Business Objects oder Consumer. So kann beispielsweise ein Consumer angelegt werden, für den dann verschiedene Contracts mit Service-Versionen dokumentiert werden können. Es soll dabei helfen, eine serviceorientierte Architektur besser überwachen zu können, Daten zu verwalten und die Kommunikation zwischen den beteiligten Stakeholdern der SOA zu fördern. Für diesen Zweck können verschiedene Nutzer angelegt werden, denen unterschiedliche Rollen zugeordnet werden können. Bei diesen Nutzern handelt es sich um Stakeholder einer SOA. Besonderheit des Repositories ist ein feingranulares Rechte-Management, über das Rollen anhand von Berechtigungen erstellt werden können. So kann beispielsweise ein Nutzer für einen Entwickler angelegt werden, der das Recht hat, neue Services einschließlich verschiedener Versionen anzulegen. Dieser kann dann einem anderen Nutzer, beispielsweise einem Tester, das Recht einräumen, diesen erstellten Service einzusehen, aber nicht zu ändern.

5.1.2 Architektur

Die Architektur des Governance Repositories basiert auf dem Model View ViewModel (MVVM) Prinzip, welches eine Abwandlung des von JSF genutzten Model View Controller (MVC) Prinzips ist. Wie in Abbildung 5.2 zu sehen, wurden neben den klassischen MVVM-Komponenten zwei weitere Komponenten, nämlich einen Datenbank-Controller sowie ein RDF Triplestore, über klar definierte Schnittstellen gekapselt. Die View-Komponente enthält ausschließlich die Ansicht. Die ViewModel-Komponente

enthält die Anzeigelogik und dient hier als Vermittlerrolle zwischen Ansicht und Geschäftslogik. Die Model-Komponente enthält ausschließlich die Geschäftslogik. Das hat den Vorteil, das diese Komponenten voneinander losgekoppelt sind. Analog verhält es sich hier zu Datenbank-Controller und Datenbank. Diese Kapselung soll eine bessere Wartbarkeit ermöglichen und erleichtert beispielsweise den Austausch bestehender Komponenten oder deren Wiederverwendbarkeit.

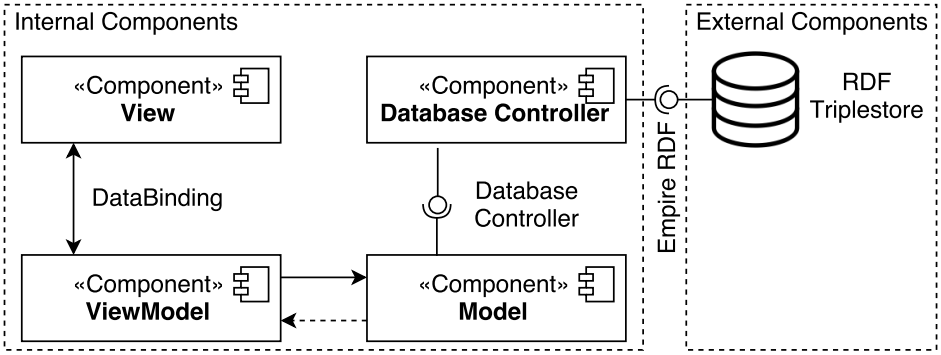

Abbildung 5.2: Komponentendiagramm des Governance Repositories

Die für diese Arbeit relevanten Komponenten werden im folgenden näher beschrieben.

View Die View-Komponente besteht aus verschiedenen sogenannten Facelets. Mit Hilfe dieser lassen sich dann die gewünschten Oberflächen im XHTML-Format erzeugen. Hierfür stellt JSF eine Reihe an Komponenten, wie beispielsweise Buttons, Tabellen oder InputFelder bereit. Zusätzlich zu diesen Komponenten wurde hier fast ausschließlich das Komponentenframework Primefaces [Pri16] verwendet. Dieses nutzt jQuery [jQu15] und erweitert die bestehenden JSF-Komponenten um nützliche Features wie beispielsweise das Sortieren oder Suchen innerhalb einer Tabelle. Grundsätzlich existiert für jeden Reiter in der Tab-Ansicht eine XHTML-Seite.

ViewModel Die ViewModel-Komponente besteht aus sogenannten ManagedBeans. Diese sind Java-Klassen, die zum einen die Attribute für das DataBinding über Getter & Setter für die View-Komponente bereitstellen und zum anderen über entsprechende Methoden bei Bedarf die Model-Komponente ansprechen. Eine solche ManagedBean hat einen sogenannten Gültigkeitsbereich (Scope), der die Lebensdauer der ManagedBean definiert. Grundsätzlich wurde für jedes Facelet eine eigene ManagedBean implementiert. Diese sind mit Session-Scope annotiert, was bedeutet, dass die Bean, mit all ihren Daten, für die gesamte Dauer einer Sitzung, in der der Nutzer mit der Anwendung verbunden ist, besteht.

Model Die Model-Komponente besteht aus Java-Klassen. Zum einen gibt es hier einen sogenannten Controller, der die Schnittstelle zur ViewModel-Komponente darstellt. Zum anderen gibt es Entitäten für das bestehende Datenbank-Schema, in denen auch die erforderliche Geschäftslogik implementiert ist. So ist hier beispielsweise das bereits erwähnte feingranulare Rechte-Management fest verankert.

5.2 Implementierung der Grundbausteine

Um die Funktionalitäten des Konzeptes implementieren zu können, müssen zunächst einige Grundbausteine umgesetzt werden. Dazu gehört einerseits die Erweiterung des Rollen-Managements um eine zielgruppengerechte Darstellung der Kennzahlen zu ermöglichen. Andererseits muss eine Möglichkeit geschaffen werden, entsprechende Funktionalitäten sowohl im Frontend als auch im Backend umzusetzen.

5.2.1 Erweiterung des Rollen-Managements

Das bestehende Rollen-Management eignet sich für die Umsetzung des Dashboard optimal. Es existieren zwei verschiedene Arten von Rollen: *SpecificRoles* und *GeneralRoles*. Beide Rollen verfügen über eine Vielzahl an unterschiedlichen Berechtigungen. *GeneralRoles* sind sehr allgemeine Rollen, die Berechtigungen wie das Erstellen von Services, Business Objects oder Consumer enthalten können. Diese Rollen können einem Stakeholder direkt zugeordnet werden und gelten dann für das gesamte System. Anders bei den *SpecificRoles*. Diese sind Rollen, die direkt an eine/n Service/-Version oder ein/e Business Object/-Version über sogenannte *PermissionRelations* gekoppelt sind. Diese können einem Stakeholder nicht direkt zugeordnet werden, sondern werden erstellt, wenn beispielsweise ein Stakeholder, der einen Service angelegt hat, einem anderen Stakeholder die Möglichkeit geben möchte, diesen bestimmten Service oder eine Service-Version anzusehen. Letzterem Stakeholder werden dann im System auch nur die Services oder Service-Versionen angezeigt, für die er entsprechende Berechtigungen hat. Entsprechend verhält es sich mit Business Objects, Consumer usw.

Da die Kennzahlen möglichst feingranular an die entsprechenden Rollen gekoppelt sein sollten, wurde hierfür eine weitere Unterteilung im Rechte-Management geschaffen, wie sie in Abbildung 5.3. Da für *SpecificRoles* bereits entsprechende Berechtigungen für jeden Stakeholder im System erstellt werden können, muss diese Möglichkeit für *GeneralRoles* umgesetzt werden. Hierfür wurden der Klasse *GeneralRole* entsprechende Berechtigungen auf die bereits existierende Kennzahlen, System-Events und Systemübersicht, hinzugefügt. Diese Klasse kann dann um weitere Berechtigungen für Kennzahlen beliebig erweitert werden. In dem Pop-up, in dem eine neue Rolle erstellt werden kann, wurde dafür eine Tab-Ansicht hinzugefügt, die zwischen den allgemeinen Berechtigungen und den Berechtigungen auf Kennzahlen wechselbar ist. Der Reiter KPIs ist nur dann sichtbar, wenn die Rolle, die angelegt werden soll, vom Typ *GeneralRole* ist.

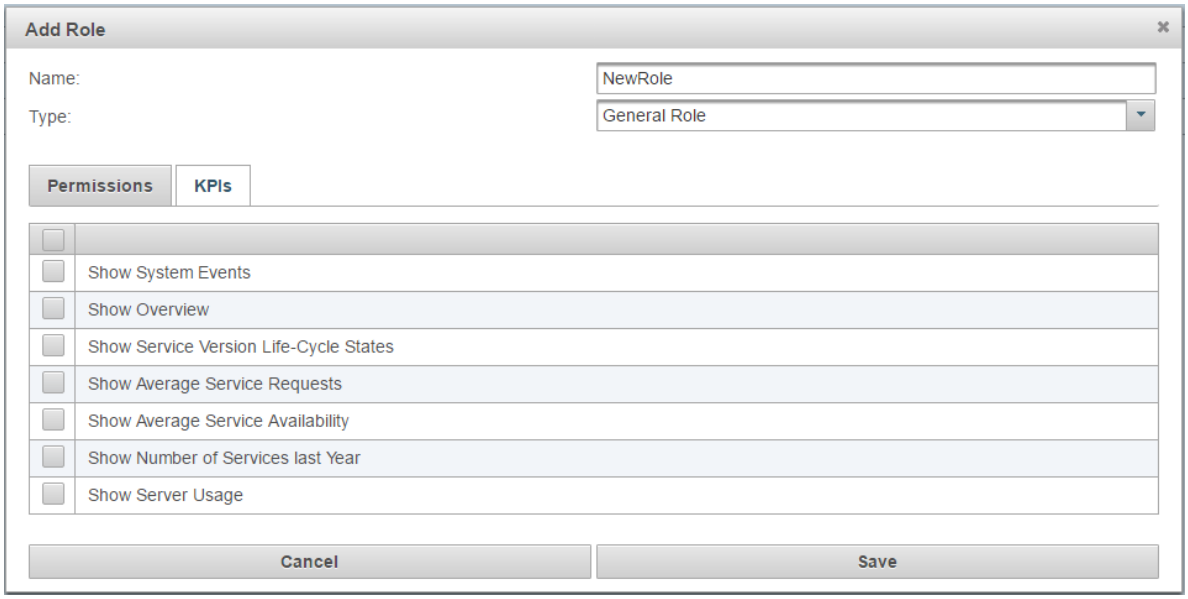

Abbildung 5.3: Implementierung des *Add Role*-Bereichs

5.2.2 Dashboard-Facelet und Dashboard-Bean

Grundlegend basiert die Benutzeroberfläche auf einem *Index*-Facelet. Dieses beinhaltet einen Header-Bereich, in dem beispielsweise das Nutzer-Menü, in dem der Nutzer Einstellungen ändern oder sich ausloggen kann, sowie die Tab-Ansicht, die für jeden Reiter ein entsprechendes Facelet inkludiert. Diese werden allerdings erst nach einem erfolgreichen Login, der über ein *Authentication* Pop-up erfolgt, bedienbar. Da das Pop-up nur einen kleinen Teil der Seite abdeckt, dürfen hier noch keine Daten sichtbar sein, was bedeutet, dass das zu implementierende Dashboard erst nach erfolgreichem Login erscheinen darf. Aus diesem Grund sind die Panels für Systemübersicht und System-Events zuvor eingeklappt. Diese befinden sich in dem dafür implementierten *Dashboard*-Facelet. Der Login sowie andere zentrale Operationen des Repositories werden über eine *MainBean* gesteuert. In dieser befinden sich ebenfalls die Daten für das aktuelle Dashboard. Da dieses Dashboard im Weiteren durch ein solches ersetzt werden soll, wie es in dieser Arbeit entworfen wurde, genügt diese Variante nicht mehr. Aus diesem Grund wurde die sogenannte *DashboardBean* erstellt. Diese hat den Sinn, alle Funktionalitäten des Dashboards bereitzustellen und Daten, die für Kennzahlen notwendig sind, zu erzeugen.

Um die folgende Beschreibung der Implementierung übersichtlicher zu gestalten, wird in Abbildung 5.4 bereits ein Klassendiagramm der implementierten DashboardBean, sowie deren Interaktionen mit der View- und Model-Komponente, gezeigt.

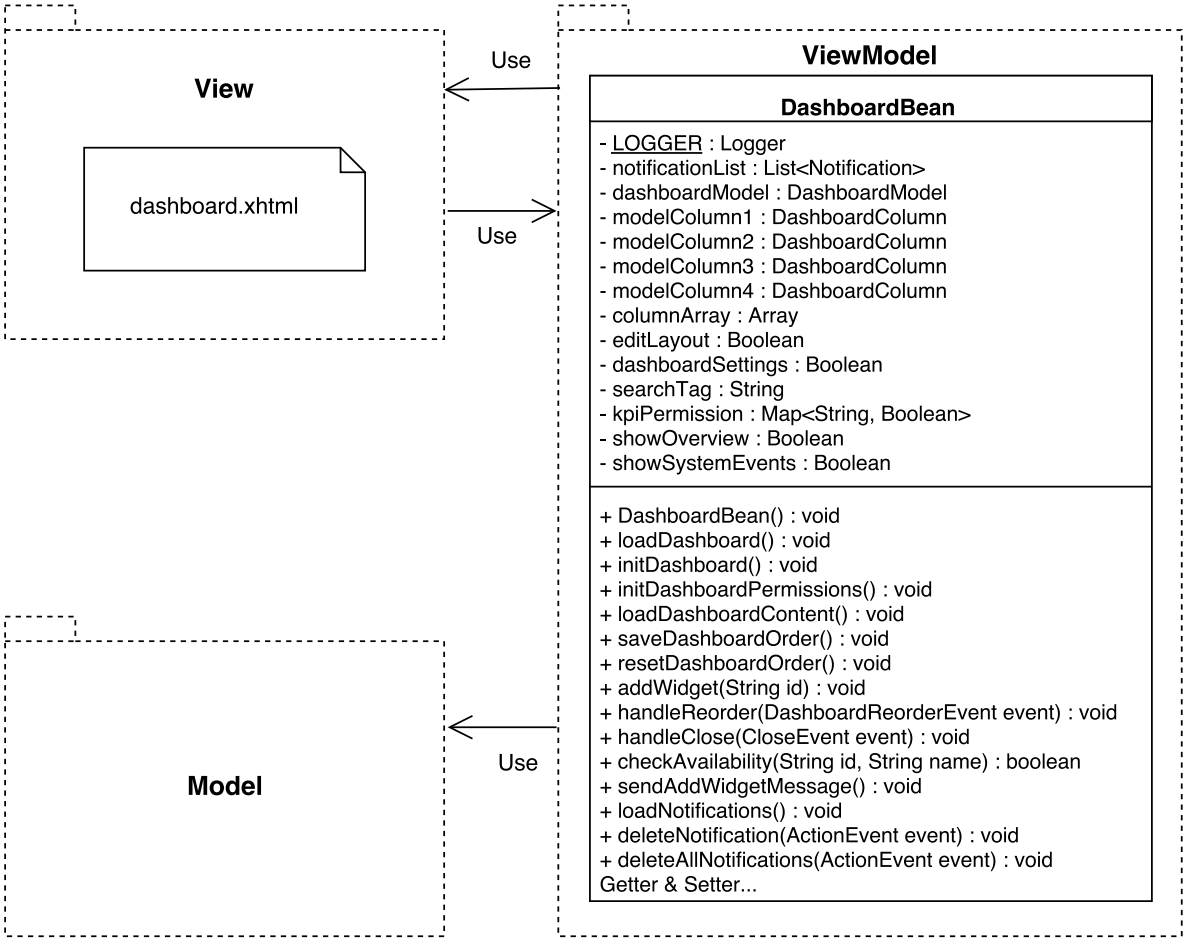

Abbildung 5.4: UML-Diagramm zu *Dashboard*-Facelet und *DashboardBean*

Da die zentrale Eigenschaft des Dashboards die ist, rollenbasiert zu sein, wurde bereits hier eine Methode geschaffen, die Berechtigungen für mögliche Kennzahlen abzufragen und die erforderlichen Daten zu laden. So soll nun in der weiteren Entwicklung für jede Kennzahl global ein privates Boolean-Attribut mit dem Namen der Kennzahl erstellt werden. Diese sind somit für verschiedenste Funktionalitäten über Getter & Setter vorhanden und werden mit der Instanziierung der *DashboardBean* auf false gesetzt. Mit einem erfolgreichen Login wird dann die Methode *initDashboardPermissions()* aufgerufen, die für den aktuellen Stakeholder prüft, auf welche Kennzahlen dieser eine Berechtigung hat. Falls ja, werden diese auf true gesetzt. Über diese Methode können dann auch noch im späteren Verlauf einer Session die Berechtigungen des Stakeholders aktualisiert werden. Nach einer erfolgreichen Instanziierung der Berechtigungen sollen dann ausschließlich die erforderlichen Daten geladen werden. Die dafür geschaffene Methode *loadDashboardContent()* fragt dann die global definierten Attribute für die entsprechenden Benachrichtigungen ab und lädt deren Inhalt. Diese Methoden sollen in der folgenden Entwicklung mit Kennzahlen erweitert werden. Außerdem sollen sie sicherstellen, dass ein einheitlicher Programmcode entsteht, der leicht gewartet werden kann und den Programmablauf deutlich macht.

Eine weitere wichtige Funktion der *DashboardBean* ist das Exception Handling. Exceptions können hier vor allem dann auftreten, wenn Verbindungsschwierigkeiten mit der Datenbank entstehen. Diese müssen dem Stakeholder natürlich sinnvoll übermittelt werden. Für diesen Zweck wurde bereits in der *MainBean* eine Methode geschaffen, Kurznachrichten dem Stakeholder anzuzeigen. Außerdem ist es wichtig, diese Exceptions zu dokumentieren. Hierfür wurde, wie in der restlichen Anwendung üblich, ein beanübergreifender Logger inkludiert, der diese Funktionalität vorsieht. Tritt eine Exception auf, soll also in der folgenden Entwicklung der *DashboardBean* eine sinnvolle Kurznachricht über die *MainBean* an den Stakeholder geschickt sowie die aufgetretene Exception mittels des Loggers dokumentiert werden.

5.3 Benachrichtigungen

Im folgenden werden eine Implementierung sowie geeignete Anwendungsfälle für die im Konzept beschriebenen Benachrichtigungen erläutert.

5.3.1 Implementierung

Für die Implementierung der Benachrichtigungen schien die Primefaces-Komponente *Messages* optimal geeignet. Dabei konnte man aus dem Backend sogenannte *FacesMessages* an das Frontend senden. Eine solche *FacesMessages* hat eine Fehlergrenzstufe sowie eine Nachricht, die dann im Frontend ihrer Fehlergrenzstufe entsprechend dargestellt wird. Folgende vier Fehlergrenzstufen stellt die Komponente bereit: *Info, Warning, Error* und *Fatal*. Eine weitere Besonderheit war jene, dass eine solche *FacesMessage* geschlossen werden konnte. Es stellte sich allerdings heraus, dass bei einer Vielzahl an *FacesMessages* gleicher Fehlergrenzstufe diese in einer *FacesMessage* zusammengefasst wurden und somit auch nur zusammen schließbar waren. Da das allerdings dem Konzept widersprach, musste über eine eigene Lösung nachgedacht werden. Abbildung 5.5 zeigt die Umsetzung in der Datenstruktur.

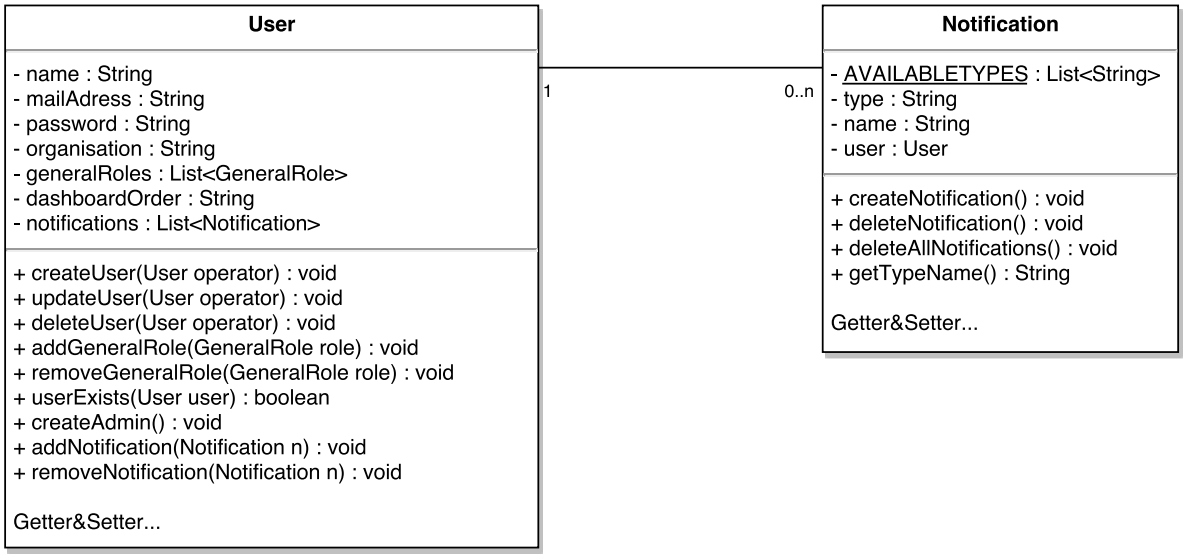

Abbildung 5.5: UML-Diagramm der Klassen *User* und *Notification*

Es wurde neben der bereits existierenden Entität *User* eine neue Entität für Benachrichtigungen (*Notification*) erstellt, die von der Klasse *DatabaseMappingBasicClass* erbt. Die Superklasse dient zur Nutzung des Datenbank-Controllers, indem sie für alle Entitäten eine eindeutige ID vergibt. Daher musste für die Entität *Notification* keine eigene ID implementiert werden. Deshalb erweitert diese ihre Superklasse vor allem um die Attribute *Type* und *Name*. Zusätzlich ist der User über eine @OneToOne-Relation (nach dem Keep it small and simple Prinzip) verknüpft. Der Typ stellt die Fehlergrenzstufe dar und der Name den Text der Benachrichtigung. Da nur bestimmte Typen zugelassen sind, wäre an dieser Stelle ein Enum angebracht gewesen, was allerdings Fehler verursachte. Um diese Fehler beheben zu können, hätte eine weitere Einarbeitung in RDF stattfinden müssen, welche dem Nutzen des Enums nicht gerecht geworden wäre. Deshalb wurde aus Gründen der besseren Wartbarkeit eine finale und statische Liste (*AVAILABLETYPES*) mit allen verfügbaren Typen erstellt. Der eigentliche Typ konnte dann als String gespeichert werden. Auf Grund dessen wird beim Setzen des Typs geprüft, ob dieser valide ist. Falls nicht, wird er per Default auf Info gesetzt. Im Prototyp wurden die Typen *INFO, WARN* und *ERROR* implementiert. Die Methode *getTypeName()* dient dazu, die entsprechenden Typen in lesbare Namen umzuwandeln, sprich Info, Warning und Error. Zusätzlich zu den Attributen wurden der Entität die Methoden *createNotification()*, *deleteNotification()* und *deleteAllNotifications()* hinzugefügt. Diese erstellen und löschen die entsprechende Benachrichtigung in der Datenbank anhand in der Superklasse enthaltenen ID.

Die Entität User wurde zum einen um ein neues Attribut für die Benachrichtigungen namens *notifications* erweitert. Hierbei handelt es sich um eine Liste, die über eine @OneToMany-Relation mit entsprechenden Benachrichtigungen verknüpft ist. Zum anderen wurde die Entität um die Methoden *addNotification(Notification n)* und *removeNotification(Notification n)* erweitert. Die Methode zum Hinzufügen einer Benachrichtigung fügt zunächst die Benachrichtigung, die sie als Parameter übergeben bekommt, dem User hinzu und erstellt die Benachrichtigung in der Datenbank. Anschließend wird das *User*-Objekt direkt in der Datenbank aktualisiert. Der Grund dafür wird im nächsten Absatz beschrieben. Dasselbe wurde in der Methode zum Löschen einer Benachrichtigung implementiert, mit

dem einzigen Unterschied, dass nach dem Löschen der Benachrichtigung aus der *notifications*-Liste der Stakeholder zunächst aktualisiert wird, bevor die Benachrichtigung aus der Datenbank endgültig gelöscht wird.

Eine Schwierigkeit stellte hier das implementierte Rechte-Management dar. So war es stets gedacht, bei jeder Operation auf der Datenbank zu prüfen, ob der Stakeholder dafür die notwendige Berechtigung hat. Dies konnte bei den Benachrichtigungen nicht angewandt werden. So muss das System bei einem Nutzer, der keine Berechtigung hat, einen anderen Stakeholder zu bearbeiten, ebenfalls die Möglichkeit haben, für einen oder mehrere dieser anderen Stakeholder eine Benachrichtigung zu erstellen und diesen damit zu bearbeiten. Aus diesem Grund wurde das Rechte-Management hier umgangen, indem keine Berechtigung für die Operationen Erstellen und Löschen von Benachrichtigungen abgefragt wurde, da diese vom System ausgeführt werden und der Stakeholder keinen direkten Einfluss darauf hat. Aus diesem Grund wird das User-Objekt in den Methoden zum Hinzufügen und Löschen direkt selbst in der Datenbank aktualisiert. In diesen Fällen ist es also nicht mehr notwendig, die Methode *updateUser(User operator)* auszuführen. Dieser Sachverhalt ist im JavaDoc der jeweiligen Methode beschrieben und soll so unnötige Verwirrung beim weiteren Arbeiten an dem Prototypen verhindern. Listing 5.1 zeigt beispielhaft, wie eine Benachrichtigung im System erstellt werden kann. So werden hier zunächst die Stakeholder ermittelt, die eine Benachrichtigung angezeigt bekommen sollen. Anschließend reicht es aus, die gewünschte *Notification* mit entsprechenden Parametern zu erstellen und diese über *addNotification(Notification n)* dem jeweiligen Stakeholder hinzuzufügen.

Listing 5.1 Erstellen einer Benachrichtigung im System

```
// send Notification to all users involved in the service
Communicator c = new Communicator();
for (UserRoleService uRS : c.getAllPermissionRelations(selectedService,UserRoleService.class)) {
      Notification n = new Notification("INFO", "There is a new Service Version (" +
          newServiceVersion.getName() + ") availiable for a Service (" +
          selectedService.getName() + ") you're involved in.");
      uRS.getUser().addNotification(n);
}
```

Um die Benachrichtigung im Frontend anzuzeigen, wurden hierfür die CSS-Klassen der bereits oben beschriebenen Primefaces-Komponente *Messages* übernommen. Loggt sich der Stakeholder in das System ein, wird für die Liste an Benachrichtigungen des Stakeholders entsprechend der jeweiligen Typen dargestellt. Bei einem Klick auf das Schließen-Symbol auf der rechten Seite wird die Benachrichtigung für den Stakeholder in der Datenbank direkt gelöscht und ihm so für zukünftige Logins nicht mehr angezeigt. Um zu verhindern, dass die Liste an Benachrichtigungen zu lange wird und damit die Kennzahlen zu weit nach unten gerückt werden, wurden die Benachrichtigungen in ein HTML-DIV-Element mit maximaler Höhe von 202 Pixeln, was vier Benachrichtigungen entspricht, eingebettet. Sollten mehr Benachrichtigungen angezeigt werden, wird die Liste scrollbar. Die Umsetzung einer Pagination wäre in der Realisierung deutlich aufwändiger gewesen und hätte zudem verhindert, dass der Stakeholder schnell durch die Benachrichtigungen scrollen kann. Des weiteren wurde ein Link zum Löschen aller Benachrichtigungen unterhalb dieser Liste implementiert, der nur dann zu sehen ist, wenn auch die Scrollbar sichtbar ist, also wenn mehr als vier Benachrichtigungen angezeigt werden. Diese Anzahl ist noch vertretbar, wenn man sich vorstellt, dass der Nutzer jede einzelne

Benachrichtigung schließen muss. Eine größere Anzahl würde die Handhabung erschweren und unter Umständen dazu führen, dass die Stakeholder gar keine Benachrichtigungen mehr schließen.

5.3.2 Anwendungsfälle

Beispielhaft wurden für die Implementierung einige Anwendungsfälle umgesetzt. Damit sollte gezeigt werden, welchen Nutzen die Benachrichtigungen haben und wie sich diese erstellen lassen. Dafür wurde das System betrachtet und überlegt, wo solche Benachrichtigungen Sinn machen. Da der größte Teil des Systems darauf beruht, Services und Business Objects zu verwalten, beschränken sich die folgenden Anwendungsfälle auf diese Bereiche. Es wurde pro Bereich für jeden Typ eine Benachrichtigung implementiert, also jeweils für Services und Business Objects eine Info-, eine Warning- und eine Error-Benachrichtigung. Als Info-Benachrichtigung erschien hier sinnvoll, wenn alle Stakeholder, die in einen Service involviert sind, eine Benachrichtigung erhalten, sobald eine neue Service-Version verfügbar ist. Hierfür wurde zunächst geprüft, welche Methode im Model aufgerufen wird, wenn eine Service-Version über die Oberfläche erstellt wird. Innerhalb dieser wurden dann alle Stakeholder abgefragt, die irgendwie in den Service, für den die Service-Version erstellt werden soll, involviert sind. Für all diese wurden dann, wie in Listing 5.1 zu sehen, folgende Benachrichtigung erstellt: *Info: There is a new Service Version ([Name]) availiable for a Service ([Name]) you're involved in.* [Name] dient hier und im Folgenden als Platzhalter für die entsprechenden Namen. Diese werden beim Erstellen entsprechend eingefügt. Als Error-Benachrichtigung wurde der Fall gewählt, dass eine Service-Version nicht mehr verfügbar ist. Dafür wurde analog zum vorherigen Fall folgende Benachrichtigung erstellt: *A Service Version ([Name]) you're involved in is no longer availiable.* Als Warning-Benachrichtigung wurde die Änderung eines Life-Cycle States gewählt. Dafür wurde folgende Benachrichtigung erstellt: *The Life-Cycle State of the Service Version you're involved in ([Name]) changed to [Name].* Abbildung 5.6 zeigt beispielhaft einige Anwendungsfälle, sowie die Scrollbar auf der rechten Seite.

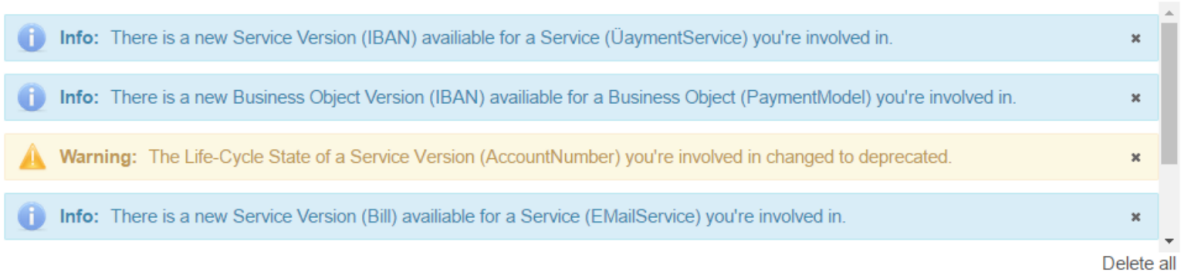

Abbildung 5.6: Screenshot des Benachrichtigungen-Bereichs

5.4 Kennzahlen

Im folgenden werden eine Implementierung sowie geeignete Anwendungsfälle für die im Konzept beschriebenen Kennzahlen erläutert.

5.4.1 Implementierung

Für die Implementierung der Kennzahlen wurde die Primefaces-Komponente *Dashboard* gewählt. Diese ermöglicht es, im Frontend mehrere Panel zu erstellen, die sich per Drag & Drop verschieden anordnen lassen. Diese Panel werden in diesem Zusammenhang als Widgets bezeichnet. Im Backend basiert die Komponente auf einem sogenannten *DashboardModel*. Diesem Model können beliebig viele Spalten, sogenannte *DashboardColumns*, hinzugefügt werden, welche eine Teilmenge der Widgets enthalten. Ein Widget wird dann im Frontend angezeigt, wenn es anhand der entsprechenden ID einer Spalte im Backend hinzugefügt wurde. Hierfür dient ein kurzer Blick auf Listing 5.2, in dem zu sehen ist, wie ein Widget, mit der ID *sampleKPI*, innerhalb der Dashboard-Komponente implementiert ist. Dieses Widget kann dann bei Bedarf in der *DashboardBean* einer *DashboardColumn* über die Methode *addWidget(ßampleKPI")* hinzugefügt werden. Primefaces durchsucht hierfür die im Dashboard Facelet implementierten Widgets daraufhin, ob ein Widget mit der gegeben ID vorhanden ist. In diesem Fall würde dieses Widget im Dashboard angezeigt werden. Im anderen Fall, also in dem Fall, dass ein Widget nicht über seine ID hinzugefügt wurde, erscheint es auch nicht im Dashboard. Analog kann es natürlich auch aus dem Model entfernt werden und wird dann folglich auch nicht mehr angezeigt.

Diese Widgets können entsprechend der Panel-Komponente von Primefaces angepasst werden, was bedeutet, dass Icons an der rechten oberen Seite des Widgets beliebig definiert werden können. So konnten die Panels mit einem Einstellungs-Button sowie Buttons für das Minimieren und das Schließen erstellt werden. Ein Nachteil dieser Komponente ist allerdings, dass die Größe der einzelnen Widgets nicht veränderbar ist, da die Komponente so implementiert ist, dass sie spaltenweise die Widgets untereinander rendert und sich somit zwei Spalten nicht überlappen können. Da aber die Breite der einzelnen Spalten, in denen sich die Widgets befinden, beliebig angepasst werden kann, kann man indirekt Einfluss auf die Größe der Widgets nehmen. Es muss also entschieden werden, wie groß die einzelnen Widgets sein sollen und dementsprechend die Anzahl und die Breite der Spalten definiert werden. Dennoch erschien der Einsatz für diesen Prototypen angemessen, da zum einen das Oberflächendesign damit konsistent ist und zum anderen eine eigene Implementierung dieser Aspekte nicht in der vorhandenen Zeit umgesetzt werden konnte. Es hat sich gezeigt, dass die optimale Größe eines Widgets dann erreicht wird, wenn alle Widgets auf vier Spalten aufgeteilt werden. Dies wurde entsprechend im Backend so implementiert, kann aber jederzeit geändert werden. Dabei muss beachtet werden, die Datenstruktur anzupassen, die entsprechend der bisher definierten Spaltenanzahl erstellt wurde. Diese lässt sich aber ebenfalls sehr leicht an die neue Spaltenanzahl anpassen.

Zunächst wurde ein dem Konzept entsprechendes Widget implementiert, das für jede Kennzahl wiederverwendet werden kann. Die Implementierung dessen ist in Listing 5.2 dargestellt. Diese kann dann beim Anlegen einer Kennzahl kopiert werden und muss dann nur noch angepasst werden. Dafür muss beispielsweise eine eindeutige ID definiert werden. Die Stellen, die mit einem Kommentar vermerkt sind, kann man dann nach Bedarf implementieren. So soll also beispielsweise an der Stelle des Kommentars *Implementation of Visualization* die Darstellung der Kennzahl eingefügt werden. Beispielhaft wurde zunächst, wie in Abbildung 5.7 zu sehen, die bereits vorhandene Kennzahl für System-Events umgesetzt.

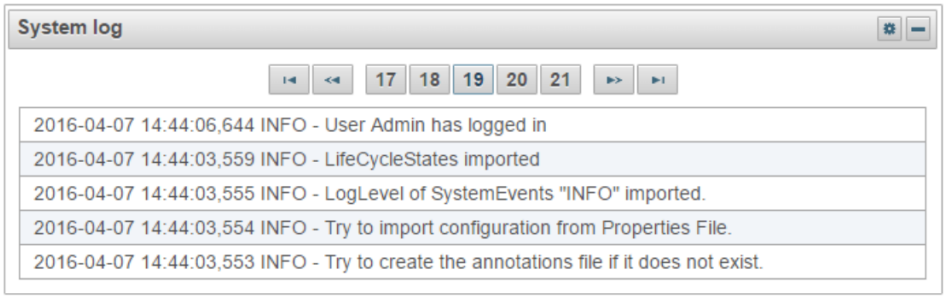

Abbildung 5.7: Screenshot eines Widgets

Listing 5.2 Implementierung eines Beispiel-Widgets

```
<p:dashboard id="dashboard" disabled="#{!dashboardBean.editLayout}"
    model="#{dashboardBean.dashboardModel}">
      <p:ajax event="reorder" listener="#{dashboardBean.handleReorder}" />

      <!-- SampleKPI -->
      <p:panel id="sampleKPI" header="Sample KPI" class="dashboardPanel" toggleable="true"
            closable="#{dashboardBean.editLayout}">
            <p:ajax event="close" listener="#{dashboardBean.handleClose}" />
            <f:facet name="options">
                  <p:menu>
                        <!-- Implementation of Options -->
                  </p:menu>
            </f:facet>
            <!-- Implementation of Visualization -->
      </p:panel>
</p:dashboard>
```

Anschließend wurden die zwei verschiedenen Modi des Konzepts umgesetzt. Hierfür wurde unterhalb der Dashboard-Komponente ein Einstellungs-Bereich, wie in Abbildung 5.8 zu sehen, implementiert, in dem die entsprechenden Funktionalitäten vorhanden sind. Um überprüfen zu können, in welchem Modus sich das Dashboard befindet, wurde in der *DashboardBean* das Boolean-Attribut *editLayout* erstellt. Beispielhaft ist in Listing 5.2 zu sehen, wie das disabled-Attribut der Dashboard-Komponente entsprechend gesetzt wird. Zunächst befindet sich das Dahboard im normalen Modus, was bedeutet, dass zum einen die Widgets nicht verschoben oder gelöscht werden können und die entsprechenden Funktionalitäten wie Speichern, Wiederherstellen oder das Hinzufügen von Widgets nicht zur Verfügung stehen. Dies wird durch das Ausgrauen der Buttons signalisiert. Zusätzlich ist, wie in Abbildung 5.7 zu sehen, der Löschen-Button im rechten oberen Eck des Widgets ausgeblendet. Wechselt der Stakeholder in den Editiermodus, erscheint dieser Löschen-Button, die Widgets können verschoben werden und die Funktionalitäten stehen dem Stakeholder nun zur Verfügung. Außerdem wurde für eine bessere Benutzerfreundlichkeit der Cursor für die Titelbereiche der Widgets auf das Move-Symbol gesetzt, sofern der Stakeholder den Editiermodus aktiviert hat. So wird schnell deutlich, anhand welches Bereiches die Widgets verschoben werden können.

Abbildung 5.8: Screenshot des Einstellungen-Bereichs

Der Bereich für neue Widgets wurde in einem Pop-up, wie in Abbildung 5.9 zu sehen, realisiert, das sich über den Button *Add Widget(s)* im Einstellungs-Bereich öffnen lässt. Darin sind alle Widgets implementiert, wie sie auch im Dashboad angezeigt werden würden, mit dem einzigen Unterschied, dass anstelle der Funktionalitäten im rechten oberen Bereich des Widgets außschließlich ein Button zum Hinzufügen des entsprechenden Widgets vorhanden ist. Ein Hinzufügen der Widgets mittels Drag & Drop konnte nicht realisiert werden, da die gewählte Primefaces-Komponente dies nicht fehlerfrei unterstützt. Aus diesem Grund wurde die Variante über einen Hinzufügen-Button gewählt. Die Widgets sind hier in zwei Spalten aufgeteilt, da mit jeder weiteren Spalte das Pop-up größer und damit unübersichtlicher werden würde. Zudem wurde eine Suchfunktion für Widgets implementiert, die bei Änderung des Suchbegriffes automatisch die Widgets aktualisiert. Wird das Pop-up geöffnet, wird anhand der Methode *checkAvailability(String id, String name)* innerhalb der DashboardBean geprüft, ob zum einen der Stakeholder die Berechtigung, hat das Widget seinem Dashboard hinzuzufügen und zum anderen, ob das Widget dem entsprechenden Suchkriterium entspricht. Widgets, die der Stakeholder bereits in sein Dashboard mit aufgenommen hat, werden hier nicht noch einmal angezeigt. Wird ein Widget über diesen Bereich hinzugefügt, verschwindet es und erscheint direkt im Dashboard in der ersten Spalte. Diese Lösung soll konsistent erscheinen und dem Stakeholder erleichtern, die hinzugefügten Widgets schneller zu finden, um diese gegebenenfalls zu verschieben. Zudem werden die hinzugefügten Widgets blau umrandet. Dieser Sachverhalt wird beim Schließen des Pop-ups in einer sogenannten *Growl-Message* am rechten oberen Rand des Systems erläutert.

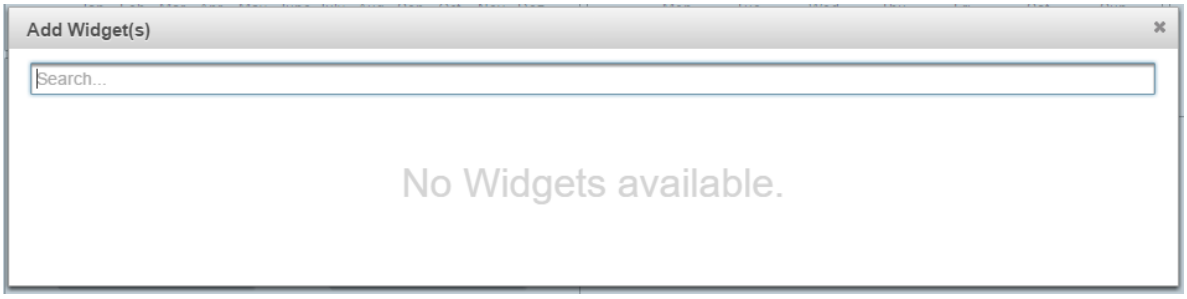

Abbildung 5.9: Screenshot des *Add Widget(s)*-Bereichs

Da die Dashboard-Komponente keine Möglichkeit bietet, die Dashboard-Konfiguration zu speichern, musste dafür eine eigene Datenstruktur geschaffen werden. Diese Konfiguration der Widgets muss

an die Entität User geknüpft sein. Da Widgets anhand ihrer ID sowohl im Front- als auch im Backend identifizierbar sind, bot sich an, diese auch in der Datenstruktur zu verwenden. Wie bereits beschrieben, sind die einzelnen Widgets an die Spalten gebunden. Aus diesem Grund wurde als Datenstruktur ein String namens *dashboardOrder* definiert, um welche die Entität User als Attribut erweitert wurde, wie in Abbildung 5.5 zu sehen ist. Dieser String enthält eine Serialisierung der Kennzahlen-IDs, die durch Kommas und Semikolons getrennt sind. Ein Komma trennt dabei die Widgets und ein Semikolon die Spalten. Beispielhaft wurde in Abbildung 5.10 der String für eine Konfiguration des Dashboards erzeugt. Dieser String kann dann für einen entsprechenden User in der Datenbank gelesen und geändert werden.

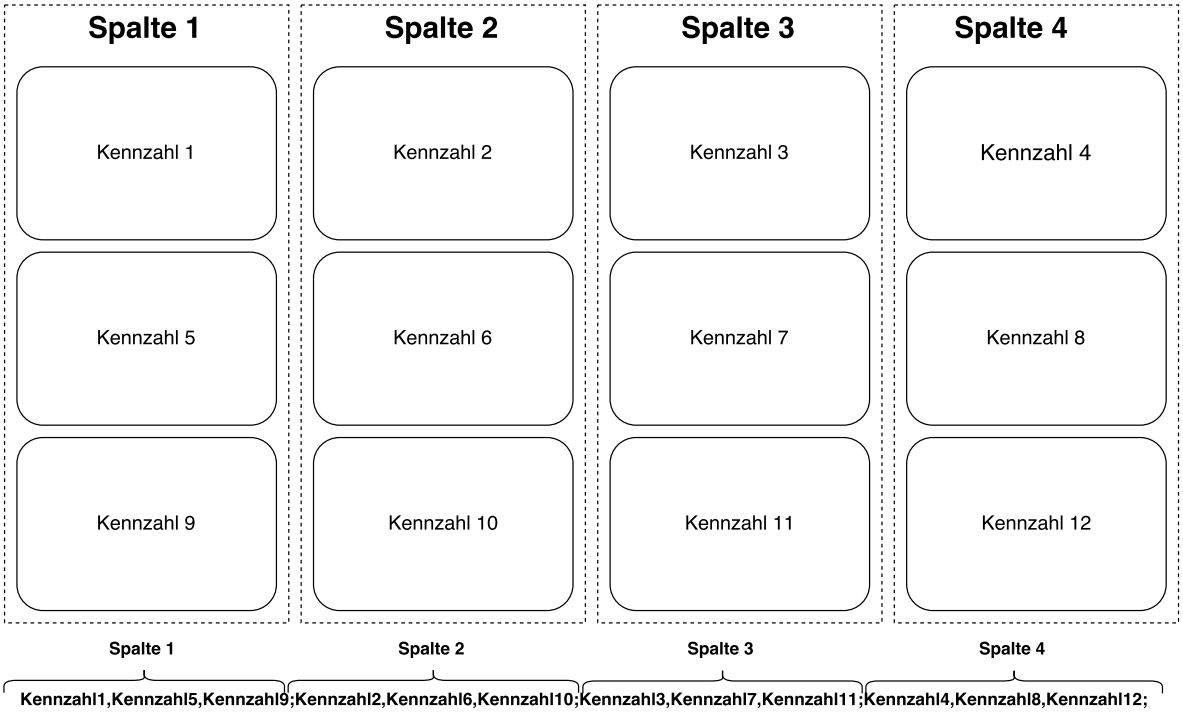

Abbildung 5.10: Datenstruktur der Kennzahlen-Konfiguration

Mit Hilfe dieser Datenstruktur konnten dann die Funktionalitäten zum Speichern und Wiederherstellen der Dashboard-Konfiguration implementiert werden. So wird beim Speichern durch die Spalten und deren Widgets durchiteriert und der String mittels einem StringBuilder anhand der Datenstruktur erstellt. Dieser String wird dann für den Stakeholder gesetzt und in der Datenbank aktualisiert. Anschließend wechselt das Dashboard in den normalen Modus mit der *Growl-Nachricht*, dass das Speichern erfolgreich war. Beim Wiederherstellen wird für den Stakeholder den gespeicherten String aus der Datenbank gelesen, geparst und anhand dessen die alte Dashboard-Konfiguration geladen.

Nachdem die Grundfunktionalitäten implementiert waren, wurde über verschiedene Möglichkeiten der Implementierung bezüglich der Darstellung der Kennzahlen untersucht. Um die Darstellung der Kennzahlen innerhalb der Widgets realisieren zu können, wurden dafür verschiedene Implementierungsvarianten untersucht. Inzwischen gibt es eine Vielzahl an JavaScript Bibliotheken, mit

denen sich schnell graphische Diagramme erstellen lassen. Dabei sollte das Hauptaugenmerk auf der Benutzung für ein solches SOA Governance Repository und dessen Kennzahlen liegen. Ein äußerst wichtiger Faktor dafür ist die Lizenz. Da in den meisten Fällen, vor allem im kommerziellen Bereich, dies ein sehr heikles Thema ist, werden im folgenden nur Open-Source Lösungen, die frei verwendet werden dürfen, betrachtet. Zusätzlich soll die Verwendung mit JSF untersucht werden.

Primefaces Charts - jqplot Hierbei handelt es sich um die Diagramm-Bibliothek von Primefaces. Diese benutzt und erweitert das Open-Source jQuery-Plugin jqplot um entsprechende JSF Funktionalitäten. So müssen die Diagramme nicht umständlich im Frontend mittels JavaScript zusammengebaut werden, sondern können anhand von Modellen im Backend erstellt werden. Das hat zum einen den Vorteil, dass die Daten nicht extra ins Frontend transportiert werden müssen, da sie direkt in der Bean verarbeitet werden können. Ein großer Nachteil dieser Bibliothek ist allerdings, dass die Darstellung nicht sehr ansprechend ist und veraltet wirkt. So werden die Diagramme innerhalb eines Containers gerendert, der um das eigentliche Diagramm eine schattierte Box erstellt. Mittels CSS ist die Anpassung der Darstellung zwar möglich, aber sehr aufwändig, da dies von den Primefaces-Entwicklern so nicht vorgesehen ist. Insgesamt stößt man vor große Schwierigkeiten, sobald man anfängt, in die Darstellung des Diagramms einzugreifen. Lizenz: Apache License 2.0

Chartist.js Hierbei handelt es sich um eine reine JavaScript Bibliothek, die sich vor allem durch ihr Responsive Design auszeichnet, was bedeutet, dass sich die Diagramme an ihren zur Verfügung stehenden Platz automatisch anpassen. Ein weiterer großer Vorteil ist, dass die Graphiken im SVG-Format gezeichnet werden, was die Bibliothek durchaus zukunftssicher erscheinen lässt. Zudem kommt mit der JavaScript Datei eine ausgelagerte CSS Datei, welche beispielsweise das Anpassen der Farben sehr viel angenehmer macht. Ein Nachteil hingegen ist die mangelhafte Unterstützung von Diagrammbeschriftungen bei Kreisdiagrammen, da diese zum Teil abgeschnitten werden, sofern sie zu lang sind. Hier bietet Chartist.js keine Möglichkeit, diese in einer separaten Legende samt der Farbgebung aufzulisten. Alle anderen Graphiken können allerdings ohne Probleme verwendet werden und bieten zudem eine relativ hohe Konfigurierbarkeit. Lizenz: MIT

Chart.js Hierbei handelt es sich ebenfalls um eine reine JavScript Bibliothek. Beim Erstellen von Graphiken tritt allerdings etwas Verwirrung auf, da nicht immer klar dokumentiert ist welche Parameter letztendlich angegeben werden müssen und welche nicht. Ein großer Pluspunkt ist allerdings, dass Chart.js bei Kreisdiagrammen eine sehr schöne Legende bereitstellt, mit dem einzigen Nachteil, dass die Legende ausschließlich oberhalb oder unterhalb und nicht seitlich des Kreisdiagramms angezeigt werden kann. Lizenz: MIT

jQuery Sparkline Hierbei handelt es sich um ein jQuery-Plugin für kleine Schaubilder, die sich in den Text integrieren lassen. Das ist vor allem dann von Vorteil, wenn eine Kennzahl mit viel Details gefüllt ist, beispielsweise wenn es um die Dokumentation der eigenen Services im Repository geht. So können hiermit die Schaubilder direkt mit in eine Tabelle oder Liste integriert werden. Lizenz: The BSD 3-Clause License

Da alle Bibliotheken ihre Stärken und Schwächen haben, sollte man sich nicht zwingend auf eine dieser beschränken. Vor allem deshalb nicht, da jede Bibliothek unterschiedliche Diagramme bereitstellt. So sollte im Falle einer Implementierung zunächst geprüft werden, welche der Bibliotheken die gewünschte Diagramm-Komponente anbieten und entsprechend der eigenen Anforderungen an die Darstellung, eine auswählen.

5.4.2 Anwendungsfälle

Beispielhaft wurden für die Implementierung einige Anwendungsfälle umgesetzt. Damit sollte gezeigt werden, welchen Nutzen die Kennzahlen haben und wie sich diese realisieren lassen. Dafür wurde das System betrachtet und überlegt, welche Kennzahlen Sinn machen. Da das Repository aktuell noch keinen Zugriff auf Laufzeitdaten einer SOA hat, wurden im Folgenden hauptsächlich Kennzahlen gewählt, die sich direkt auf die im Repository gespeicherten Daten beziehen. Insbesondere also Kennzahlen zu Services, Business Objects usw. Zunächst wurden die vorhandenen Kennzahlen System-Events und Systemüberblick, die bereits im vorangegangenen Kapitel beschrieben wurden, umgesetzt. Da die Komponente für ein Widget bereits definiert war, konnte diese hierfür verwendet werden. Es musste also nur noch eine sinnvolle Darstellung erstellt werden. Bei den System-Events ändert sich im Gegensatz zum Systemüberblick, welcher zu *Overview* umbenannt wurde, nichts. Die Kennzahl *Overview* wurde mit Hilfe eines Balkendiagramms umgesetzt, da hier verschiedene Werte, wie die Anzahl an Services, Business Objects, usw. miteinander verglichen werden können.

Als neue Kennzahl wurde dann ein Vergleich der Lebenszyklusphasen der Service-Versionen implementiert. Dies ist die einzige neue Kennzahl, die ihre Daten direkt aus dem System bezieht. Um diese Daten zu ermitteln, wurde die Anzahl der einzelnen Service-Versionen pro Lebenszyklusphase ausgewertet. Das Repository wurde so implementiert, dass die einzelnen Lebenszyklusphasen im System nicht hart codiert sind, sondern über eine Konfigurationsdatei eingelesen werden. Die hier implementierte Variante wurde deshalb mit einer HashMap realisiert, die den Namen der Lebenszyklusphase als Key- und die Anzahl der sich darin befindenden Service-Versionen als Value-Attribut beinhaltet. Dies stellte sich dann für die Implementierung der Darstellung als hinderlich heraus, da JSF keine Möglichkeit vorsieht, in Facelets durch HashMaps zu iterieren. Aus diesem Grund musste diese HashMap wiederum zu einer ArrayList transformiert werden. Für genau solche Fälle dient die *DashboardBean*, in der für Rohdaten geeignete Ansichtsdaten generiert werden können. Die Variante, das Diagramm über die Primefaces-Komponente Charts zu realisieren und damit dieses aus dem Backend zu erzeugen, war wegen der mangelhaften Implementierung des Kreisdiagramms keine Option. Chartist.js eignete sich für diesen Anwendungsfall nicht, da das Kreisdiagramm keine wirklich schöne Legende bereitstellt. So wurde für die Implementierung Chart.js verwendet, mit der es möglich war, eine schöne Legende zu erzeugen. Dieser Ablauf zum Erstellen einer Kennzahl im Dashboard wurde für alle weiteren Kennzahlen wiederholt und wird deshalb im folgenden nicht mehr im Detail beschrieben.

Da die Möglichkeiten ausgeschöpft waren, Daten des Repositories in sinnvolle Kennzahlen zu verpacken, gab es verschiedene Überlegungen dazu. Eine Möglichkeit wäre gewesen das Repository um entsprechende Daten zu erweitern. Beispielsweise hätte man jedem Service und dessen Service-Versionen einen Zeitpunkt, in dem er erstellt wurde, hinzufügen können. Hierfür wäre der Aufwand nicht sonderlich hoch gewesen und hätte so eine weitere Kennzahl ergeben. Da es allerdings nicht

sehr schön ist, den Zeitpunkt beim Erstellen eines Services manuell zu generieren, hätte hierfür die Oberfläche um eine weitere Eingabemaske erweitert werden müssen. Dies erschien dann wiederum aufwändiger. Ähnlich verhielt es sich bei allen weiteren Überlegungen, das Governance Repository so zu erweitern, dass neue Kennzahlen erstellt werden konnten. Aus diesem Grund wurde die Möglichkeit verworfen und festgelegt, dass alle weiteren Kennzahlen mit Beispieldaten erstellt werden. Dementsprechend wurden Kennzahlen gesucht, die den Nutzen des Dashboards verdeutlichen sollen. So sind Kennzahlen wie beispielsweise Anzahl der Services im Jahresrückblick mittels eines Liniendiagramms oder die durchschnittliche Anzahl an Service-Aufrufen pro Stunde und pro Woche mit Hilfe zweier Tachodiagramme umgesetzt worden.

Für jede der implementierten Kennzahlen wurde der *GeneralRole* eine entsprechende Berechtigung hinzugefügt. Diese können dann beim Erstellen einer solchen Rolle beliebig gesetzt werden. Da keine Rollen hart codiert werden sollten, blieb es bei dieser Möglichkeit. Dennoch ist es so gedacht, dass die Rollen des in dieser Arbeit definierten Rollenmodells im das Governance Repository erstellt werden, sofern diese mit den Anforderungen des Betreibers übereinstimmen.

Abbildung 5.11 zeigt eine mögliche Dashboard-Konfiguration. In diesem Fall hatte der Stakeholder für alle Kennzahlen die entsprechende Berechtigung und konnte sich so ein personalisiertes Dashboard erstellen.

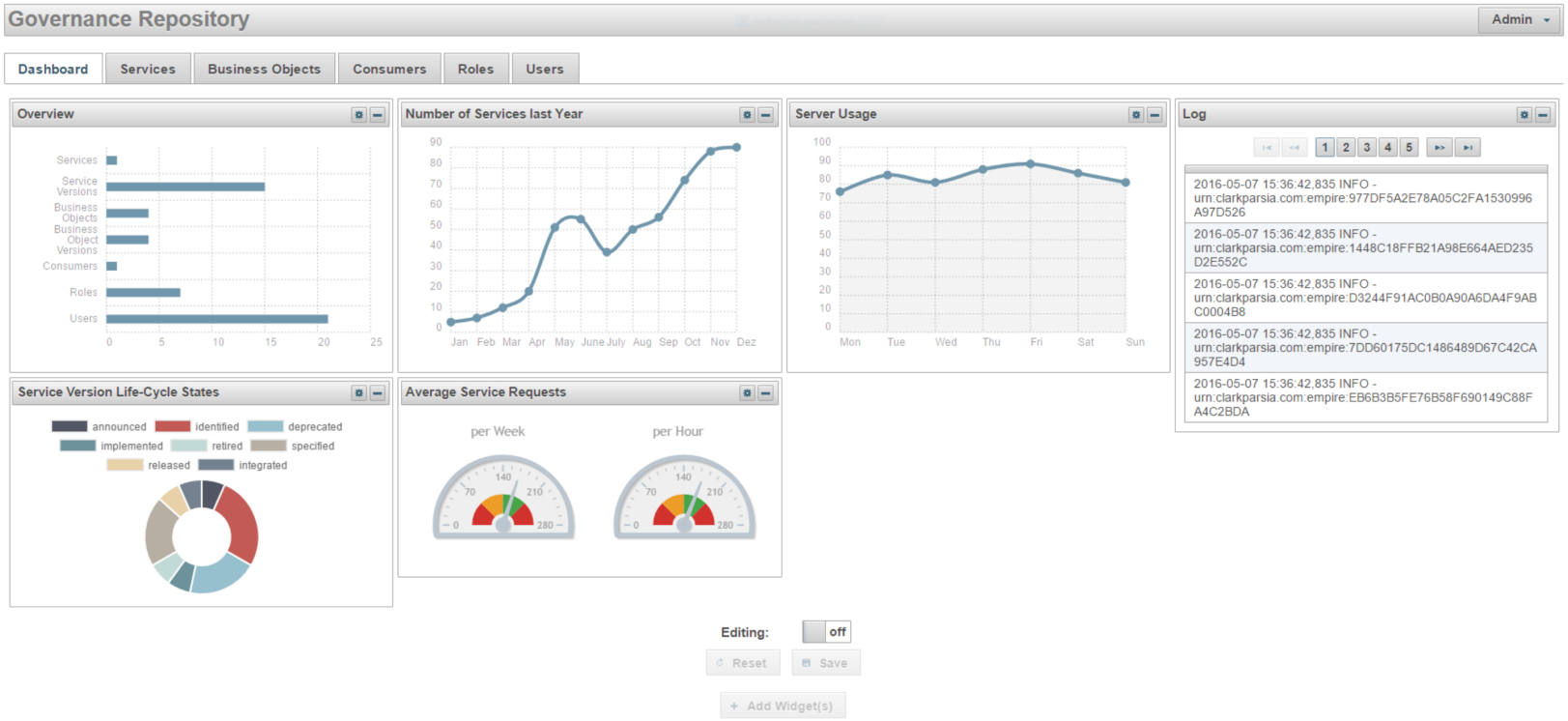

Abbildung 5.11: Beispiel einer möglichen Dashboard-Konfiguration

6 Zusammenfassung

Ziel dieser Arbeit war es, die Anforderungen an ein SOA-Governance Dashboard zu untersuchen, ein Konzept für dieses zu entwerfen und es prototypisch umzusetzen. Im Folgenden wird die Herangehensweise der Arbeit mit ihren Resultaten beschrieben. Zudem werden die aufgetretenen Schwierigkeiten beschrieben und abschließend ein Ausblick gegeben, an welchen Stellen der Arbeit angeknüpft werden kann.

6.1 Fazit

Die Anforderungen, die sich im speziellen auf verschiedene Rollen sowie unterschiedliche Kennzahlen einer serviceorientierten Architektur in Bezug auf die SOA-Governance beschränkten, wurden anhand einer tiefgehenden Quellenarbeit erarbeitet. Insgesamt ist aufgefallen, dass mehr Quellen zu Stakeholderen existieren wie zu Kennzahlen. Das könnte daran liegen, dass die Ermittlung solcher Kennzahlen zum einen sehr unternehmensspezifisch ist und zum anderen ungern in die Öffentlichkeit getragen wird. Diese Quellenarbeit war sehr mühselig, da die darin erwähnten Stakeholder oder Kennzahlen in den meisten Fällen nicht detailliert genug beschrieben waren, um eine genaue Abgrenzung zwischen den einzelnen Quellen ermitteln zu können. Dennoch konnten genug sinnvolle Kennzahlen identifiziert werden. Das erstellte Rollenmodell, das mit seinen 13 Rollen äußerst detailliert, aber dennoch generisch genug ist, um auf jedes Unternehmen angewandt werden zu können, diente als Grundlage für die Zuordnung der identifizierten Kennzahlen. Hierbei wurde für jede Kennzahl untersucht, welche Informationen für die Stakeholder der jeweiligen Rolle interessant sind. Grundlegend wurde Wert darauf gelegt, das Rollenmodell so allgemein wie möglich zu halten um ein breites Spektrum an Unternehmen abzudecken.

Für das Konzept wurde zunächst ein Aufbau entworfen, der das Dashboard grundlegend in die Bereiche für Benachrichtigungen und Kennzahlen unterteilt. Diese Bereiche wurden dann detailliert beschrieben und anhand von Mock-ups verdeutlicht. Das Konzept der Benachrichtigungen war im Gegensatz zu den Kennzahlen in diesem Zusammenhang neu, da sich die Benachrichtigungen nicht ausschließlich auf die Kennzahlen beziehen. Es wurde gezeigt, wie hilfreich solche Benachrichtigungen sind und wie sie das Dashboard zu einem noch mächtigeren Werkzeug machen. Das Konzept der Widgets ist bei Dashboards weit verbreitet, was deutlich macht, wie viele Vorteile diese mit sich bringen. Dieses Widget-System wurde dann entsprechend der Anforderungen an ein solches Dashboard erweitert. Es wurde erläutert, welchen Nutzen es hat, zwei verschiedene Modi einzuführen und wie diese umsetzbar sind. Diese unterschiedlichen Modi verhindern unnötige Probleme und erleichtern das Verändern der aktuellen Dashboard-Konfiguration. Dabei wurde zudem sehr detailliert beschrieben, wie eine optimale Umsetzung für das Zurücksetzen und das Speichern der Konfiguration sowie das Löschen und Hinzufügen neuer Kennzahlen aussieht. Anschließend wurde die Darstellung

der Kennzahlen entworfen. Dafür wurden eine Vielzahl an Darstellungsmöglichkeiten wie Icons, Diagramme oder Kartenmaterial untersucht und eine Empfehlung gegeben, welche Darstellung für welche Art von Kennzahlen am besten geeignet ist. Beispielhaft wurden hierfür die vorher identifizierten Kennzahlen verwendet. Anschließend wurde das Rechte-Management der Kennzahlen entworfen. Dieses sieht vor, dass Rollen des vorangegangen Kapitels über Berechtigungen verfügen. Eine solche Berechtigung erlaubt den Zugriff auf eine bestimmte Kennzahl. Da diese Rollen Nutzern zugeordnet werden, erreicht man dadurch ein feingranulares Rechte-Management der Kennzahlen, was die Grundlage für ein rollenbasiertes Dashboard ist.

Das entworfene Konzept wurde dann unter Beachtung der Anforderungen prototypisch umgesetzt. Die Umsetzung erfolgte in einem bereits vorhandenen Prototypen für ein SOA-Governance Repository, welches zunächst näher beschrieben wurde. Als erstes wurden für die Implementierung des Konzeptes notwendige Vorbereitungen getroffen. Dafür wurde das existierende Rechte-Management für das Dashboard und die Kennzahlen erweitert. Diese und andere Änderungen an dem bestehenden Prototypen, wie beispielsweise das Erstellen der notwendigen Bean, wurde vorgenommen, um für nachfolgende Implementierungen zum einen, eine sinnvolle Konsistenz und zum anderen eine bessere Wartbarkeit zu ermöglichen. Anschließend erfolgte die Implementierung der Benachrichtigungen des Widget-Systems und der Kennzahlen. Bei der Implementierung der Benachrichtigungen gab es im Gegensatz zu der Implementierung des Widget-Systems kaum Schwierigkeiten. Bei letzterem hingegen musste darauf verzichtet werden, Widgets unterschiedlicher Größe anzeigen zu können, da dies die genutzte Komponente nicht vorsah. Trotz einiger Herausforderungen bei der Implementierung wurden die meisten Anforderungen des Konzepts so umgesetzt wie sie angedacht waren. Als Anwendungsfälle wurde dann sowohl für Benachrichtigungen als auch für die Kennzahlen untersucht, welche Beispielfälle implementiert werden konnten. Grundlegend sah die prototypische Implementierung vor zu zeigen, wie solche Anwendungsfälle anhand der realisierten Implementierung umgesetzt werden können und welchen Nutzen sie für ein solches Dashboard haben. Diese können dann entsprechend den eigenen Bedürfnissen erweitert werden. Dafür wurde bereits während der Implementierung stets darauf geachtet, dass Erweiterungen problemlos möglich sind.

6.2 Ausblick

Ein möglicher Ausblick stellt die Erweiterung des Prototypen dar. Hierbei können weitere Anwendungsfälle implementiert werden, die vor allem die identifizierten Kennzahlen betreffen. Dafür muss eine Möglichkeit geschaffen werden, weitere Daten, die als Grundlage der Kennzahlen dienen, in das System einzupflegen. So könnte man beispielsweise einführen, dass für jedes Element wie Service, Service-Version, Business Object, Business Object-Version usw. ein Datum eingetragen werden kann, das angibt wann das gegebene Element erstellt wurde. Damit könnten dann Kennzahlen umgesetzt werden, wie die Anzahl der Services über eines bestimmten Zeitraumes verglichen werden. Ein weiterer großer Schritt würde gemacht werden, wenn das Governance Repository um eine Anbindung an einen Enterprise Service Bus (ESB) erweitert werden würde, der häufig in einer serviceorientierten Architektur verwendet wird um Nachrichten zwischen verschiedenen Services zu übertragen. Über diesen könnten dann vor allem Echtzeit-Daten, die sowohl für das Service-Monitoring, als auch für das Überwachen der SLAs notwendig sind, bezogen werden. Daraus würden sich eine Vielzahl an

Kennzahlen realisieren lassen, was zur Folge hätte, dass das Dashboard den Stakeholdern eine größere Menge an Kennzahlen anbieten könnte.

Grundlegend können natürlich auch die Einschränkungen, die vor allem den Aufbau des Dashboards betreffen, behoben werden. So könnte die verwendete Primefaces Komponente ausgetauscht werden, da sich in der Umsetzung gezeigt hatte, dass Mängel an dieser bestehen. Zum einen könnte hier untersucht werden ob es andere frei verfügbare Bibliotheken gibt oder eine eigenständige Implementierung vorgenommen werden. Damit könnte dann beispielsweise erreicht werden, dass Wigets in ihrer Größe veränderlich sind. Zudem könnte untersucht werden, ob es Sinn macht, dabei nicht nur die Größe der Darstellung anzupassen, sondern unterschiedliche Darstellungsformen für unterschiedliche Größen zu entwerfen. Das hätte den Vorteil, dass das Dashboard damit im Detail personalisiert werden kann. Mit dieser unterschiedlichen Größe der Widgets muss natürlich auch darauf geachtet werden, dass sich die restliche Anordnung der Widgets ebenso anpasst. Eine solche Umsetzung des Dashboards würde dann dem Konzept entsprechen.

Zudem kann das Konzept um weitere Funktionalitäten erweitert werden. So wäre beispielsweise eine Report-Funktion, die das exportieren der Kennzahlen vorsieht, äußerst hilfreich, um die Stakeholder einer SOA noch stärker in ihrer Arbeit zu unterstützen. Hierbei sollte es möglich sein, eine Zusammenfassung der gegeben Kennzahlen als druckbare PDF oder der einzelnen Kennzahlen als speicherbare Bilder für Dokumente oder Präsentationsfolien zu exportieren. Dafür wurde bereits beispielhaft gezeigt, wie eine solche Funktionalität für das exportieren eine Darstellung aussehen könnte. Die hierbei verwendete Primefaces Komponente, ist ähnlich zur Dashboard Komponente, nur für kleine Anwendungsfälle geeignet und sollte deshalb bei der Umsetzung einer solche Report-Funktion vermieden werden. Auch hier wäre eine eigene Implementierung vermutlich unumgänglich, wobei es bereits eine Vielzahl von frei verfügbaren Bibliotheken gibt, die zumindest das Erstellen einer PDF oder einer Grafik ermöglichen.

Literaturverzeichnis

[Afs07] M. Afshar. *SOA Governance: Framework and Best Practices.* Oracle Corporation, 2007. (Zitiert auf Seite 23)

[All08] P. Allen. *SOA Governance: Challenge or Opportunity?* CBDI Journal, 2008. (Zitiert auf Seite 23)

[BBF+05] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, R. Shah. *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap.* IBM Press developerWorks, 2005. (Zitiert auf Seite 23)

[Bel08] M. Bell. *Service-Oriented Modeling: Service Analysis, Design, and Architecture.* John Wiley and Sons, 2008. (Zitiert auf Seite 14)

[BHA04] B. Borges, K. Holley, A. Arsanjani. Delving into Service-Oriented Architecture. Online: http://www.developer.com/java/data/article.php/3409221/Delving-into-Service-Oriented-Architecture.htm, 2004. (Zitiert auf Seite 14)

[Bis08] T. Biske. *SOA Governance - The key to successful SOA adoption in your organization.* Packt Publishing, 2008. (Zitiert auf Seite 23)

[BKR11] J. Becker, M. Kugeler, M. Rosemann. *Process Management. A Guide for the Design of Business Processes.* Springer-Verlag Berlin Heidelberg, 2011. (Zitiert auf Seite 9)

[Blo04] J. Bloomberg. SOA Governance: IT Governance in the Context of Service Orientation. Online: http://www.zapthink.com/actions/download.php?id=WP-0134, 2004. (Zitiert auf Seite 23)

[BS09] J. Bernhardt, D. Seese. *A Conceptual Framework for the Governance of Service-Oriented Architectures.* Springer Verlag Berlin Heidelberg, 2009. (Zitiert auf Seite 24)

[Cha04] D. Chappell. Enterprise Service Bus. Online: http://www.oracle.com/technetwork/articles/javase/soa-142870.html, 2004. (Zitiert auf Seite 14)

[Dir13] J. Dirksen. *SOA Governance in Action.* Manning, 2013. (Zitiert auf den Seiten 15 und 23)

[DW07] P. Derler, R. Weinreich. *Models and Tools for SOA Governance.* Springer-Verlag Berlin Heidelberg, 2007. (Zitiert auf Seite 23)

[EBC+11] T. Erl, S. G. Bennett, B. Carlyle, C. Gee, A. T. M. Robert Laird, R. Moores, R. Schneider, L. Shuster, A. Tost, C. Venable, F. Santas. *SOA Governance - Governing Shared Services On-Premise and in the Cloud.* Prentice Hall, 2011. (Zitiert auf Seite 23)

[Few06] S. Few. *Information Dashboard Design.* O'Reilly, 2006. (Zitiert auf Seite 20)

[Goa07] L. Goasduff. Bad Technical Implementations and Lack of Governance Increase Risks of Failure in SOA Projects. Online: http://www.gartner.com/newsroom/id/508397, 2007. (Zitiert auf Seite 9)

[ITI16] ITIL. Business Relationship Management. Online: http://wiki.de.it-processmaps.com/index.php/Business_Relationship_Management#Business_Relationship_Manager, 2016. (Zitiert auf Seite 24)

[Jos07] N. M. Josuttis. *SOA in Practice.* O'Reilly, 2007. (Zitiert auf Seite 14)

[jQu15] jQueryFoundation. jQuery. Online: http://api.jquery.com/, 2015. (Zitiert auf Seite 59)

[Kai02] M. Kaib. *Enterprise Application Integration.* Deutscher Universitäts-Verlag, 2002. (Zitiert auf Seite 13)

[KEHZ08] M. Kuffner, G. Ember, S. Hirschberger, U. Zeithammer. SOA Know How - Serviceorientierte Architekturen. Online: http://www.soa-know-how.de/SOA, 2008. (Zitiert auf den Seiten 10 und 15)

[KSH08] O. Kohnke, T. Scheffler, C. Hock. *SOA-Governance - Ein Ansatz zum Management serviceorientierter Architekturen.* Wirtschaftsinformatik Vol. 5 2008, 2008. (Zitiert auf Seite 24)

[KSM14] J. Königsberger, S. Silcher, B. Mitschang. *SOA-GovMM: A Meta Model for a Comprehensive SOA Governance Repository.* Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration, 2014. (Zitiert auf den Seiten 7, 10, 11, 15, 19, 24 und 57)

[Lau08] W. Laurent. SOA Governance Dashboards. Online: http://www.dashboardinsight.com/articles/digital-dashboards/building-dashboards/soa-governance-dashboards-the-whole-picture.aspx, 2008. (Zitiert auf Seite 10)

[Mí12] J. Mínguez. *A Service-oriented Integration Platform for Flexible Information Provisioning in the Real-time Factory.* Universität Stuttgart, 2012. (Zitiert auf den Seiten 7 und 14)

[Mal05] S. Malik. *Enterprise Dashboards: Design and Best Practices for IT.* John Wiley and Sons, 2005. (Zitiert auf den Seiten 20 und 21)

[Mar08] E. A. Marks. *Service-oriented architecture governance for the services driven enterprise.* John Wiley and Sons, 2008. (Zitiert auf Seite 23)

[MB06] E. A. Marks, M. Bell. *Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology.* John Wiley and Sons, 2006. (Zitiert auf Seite 23)

[McK09] J. McKendrick. Justifying SOA: 12 key metrics to keep tabs on. Online: http://www.zdnet.com/article/justifying-soa-12-key-metrics-to-keep-tabs-on, 2009. (Zitiert auf Seite 24)

[Mee08] M. Meehan. SOA adoption marked by broad failure and wild success. Online: http://searchsoa.techtarget.com/news/1319609/SOA-adoption-marked-by-broad-failure-and-wild-success, 2008. (Zitiert auf Seite 9)

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services.* Spektrum Akademischer Verlag, 2010. (Zitiert auf Seite 14)

[NHSS10] M. Niemann, K. D. Hagedorn, S. Schulte, R. Steinmetz. *Typische Elemente von SOA-Governance-Ansätzen - Ein Survey.* Technische Universität Darmstadt, 2010. (Zitiert auf Seite 15)

[Nie05] K. D. Niemann. *Von der Unternehmensarchitektur zur IT-Governance.* Vieweg, 2005. (Zitiert auf Seite 13)

[Ora05] Oracle. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). Online: http://www.oracle.com/technetwork/articles/javase/soa-142870.html, 2005. (Zitiert auf Seite 14)

[Ora15] Oracle. JavaServer Faces. Online: https://javaserverfaces-spec-public.java.net/, 2015. (Zitiert auf Seite 57)

[PNM16] PNMsoft. KPI Examples. Online: http://www.pnmsoft.com/resources/bpm-tutorial/key-performance-indicators, 2016. (Zitiert auf Seite 24)

[Pri16] PrimeTek. Primefaces. Online: http://www.primefaces.org/docs/guide/primefaces_user_guide_5_3.pdf, 2016. (Zitiert auf Seite 59)

[RB08] I. Rieger, R. Bruns. *SOA-Governance und -Rollen: Sichern des Mehrwerts einer serviceorientierten Architektur.* OBJEKTspektrum, 2008. (Zitiert auf Seite 23)

[RC04] K. Rivard, D. Cogswell. *Using Analytical Dashboards to Cut Through the Clutter.* DM Review, 2004. (Zitiert auf Seite 19)

[SC13] J. St-Cry. Baby steps to SOA - step two: measure it. Online: https://theagilecoder.wordpress.com/2013/06/01/baby-steps-to-soa-step-two-measure-it, 2013. (Zitiert auf Seite 24)

[Sch10] C. Schröpfer. *Das SOA-Management-Framework.* GITO mbH Verlag, 2010. (Zitiert auf Seite 24)

[SN96] W. R. Schulte, Y. V. Natis. *'Service Oriented' Architectures, Part 1 and Part 2.* SSA Research Note SPA-401-068 and SSA Research Note SPA-401-069, 1996. (Zitiert auf Seite 9)

[SOA10] SOA CoE Core Team 2008-2009. *Service Oriented Architecture (SOA) Governance Model.* Franchise Tax Board, 2010. (Zitiert auf Seite 24)

[Sof07] Software AG. SOA Governance - Beherrschen Sie Ihre SOA. Online: http://www.softwareag.com/Corporate/Images/WP%20SOA%20Governance_tcm16-22130.pdf, 2007. (Zitiert auf Seite 23)

[Sof10] Software AG. *What's the Business Value of SOA? Show It with KPIs.* Software AG Business White Paper, 2010. (Zitiert auf Seite 24)

[The08] The Open Group. SOA Governance Technical Standard. Online: https://www.opengroup.org/soa/source-book/gov/index.htm, 2008. (Zitiert auf Seite 23)

[VLA09] G. Viering, C. Legner, F. Ahlemann. *The (Lacking) Business Perspective on SOA - Critical Themes in SOA Research.* 9th International Conference on Business Informatics, 2009. (Zitiert auf Seite 9)

[W3C04] W3C. Web Services Architecture. Online: http://www.w3.org/TR/ws-arch/, 2004. (Zitiert auf Seite 14)

[Win06] P. J. Windley. SOA Governance: Rules of the Game. Online: http://www.infoworld.com/d/architecture/governing-soa-221, 2006. (Zitiert auf Seite 24)

[WK12] L. Will, V. Köppen. *Zentrales, standardisiertes Monitoring als Grundlage des Service Level Managements in flexiblen SOA-Lösungen.* GI Jahrestagung, 2012. (Zitiert auf Seite 24)

[ZM05] O. Zimmermann, F. Mueller. *Web Services project roles.* IBM Press developerWorks, 2005. (Zitiert auf Seite 23)

Alle URLs wurden zuletzt am 01. 05. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift