Westfälische
Wilhelms-Universität
Münster

Web-orientierte Architekturen

Eine Methode zur Konzeption, Planung, Umsetzung und Bewertung

Gunnar Thies

Web-orientierte Architekturen: Eine Methode zur Konzeption, Planung, Umsetzung und Bewertung

Inauguraldissertation

zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften
durch die Wirtschaftswissenschaftliche Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von
Gunnar Thies
aus Karlsruhe

Münster 2011

Dekan:	Prof. Dr. Thomas Apolte
Erster Gutachter:	Prof. Dr. Gottfried Vossen
Zweiter Gutachter:	Prof. Dr. Jörg Becker
Tag der mündlichen Prüfung:	06. Mai 2011

Gunnar Thies

Web-orientierte Architekturen

MV WISSENSCHAFT

Wissenschaftliche Schriften der WWU Münster

Reihe IV

Band 4

Gunnar Thies

Web-orientierte Architekturen

Eine Methode zur Konzeption, Planung, Umsetzung und Bewertung

MV WISSENSCHAFT

Wissenschaftliche Schriften der WWU Münster
herausgegeben von der Universitäts- und Landesbibliothek Münster
http://www.ulb.uni-muenster.de

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Dieses Buch steht gleichzeitig in einer elektronischen Version über den Publikations- und Archivierungsserver der WWU Münster zur Verfügung.
http://www.ulb.uni-muenster.de/wissenschaftliche-schriften

Gunnar Thies
„Web-orientierte Architekturen: Eine Methode zur Konzeption, Planung, Umsetzung und Bewertung“
Wissenschaftliche Schriften der WWU Münster, Reihe IV, Band 4

Die Reihe „Wissenschaftliche Schriften der WWU Münster“ erscheint im Verlagshaus Monsenstein und Vannerdat OHG Münster
www.mv-wissenschaft.com

ISBN 978-3-8405-0042-8 (Druckausgabe)
URN urn:nbn:de:hbz:6-34429578685 (elektronische Version)

Satz: Gunnar Thies
Umschlag: MV-Verlag
Druck und Bindung: MV-Verlag

Geleitwort

Das Schlagwort „Service-orientierte Architektur" (SOA) wird seit vielen Jahren als ein äußerst viel versprechendes Konzept zur Organisation von IT-Landschaften gesehen. Es verallgemeinert im Grunde das seit langem verwendete Client-Server-Konzept auf weitgehend beliebige Komponenten. Bei Client-Server erfolgt die Kommunikation zwischen einem Dienstnutzer (Client) und einem Diensterbringer (Provider oder Server) ausschließlich über sog. Request-Reply-Paare. Angewendet wurde es zunächst für bestimmte Hardware-Funktionen (z. B. Fileserver, Druckerserver), bald aber auch auf Software ausgedehnt. SOA verallgemeinert dies auf beliebige Granularitäten; Software soll so gekapselt werden, dass ihre Komponenten einzeln oder in bestimmten Konfigurationen als Dienste (Services) angeboten werden können. Diese lassen sich im Prinzip beliebig skalieren, vom vollständigen Anwendungspaket bis hinunter zur einzelnen Funktion. Dies erlaubt es ferner – zumindest im Prinzip – komplexe Applikationen aus einfachen zusammenzusetzen und die verwendeten Komponenten auch in anderen Kontexten wieder zu verwenden. Bereits diese Eigenschaften haben vor rund 10 Jahren zu einem wahren Hype um das Thema SOA geführt, denn es sah nun so aus, als hätte man den Schlüssel zu fehlerfreier Software gefunden. Es ergaben sich allerdings kleinere Randbedingungen, von denen hier nur die wichtigsten erwähnt seien: Als Dienste bereitgestellte Software musste auffindbar sein; dazu dient der UDDI-Standard, und der Traum war ein globales UDDI-Verzeichnis für alle Dienste; dieser ist nie Wirklichkeit geworden. Idealerweise muss derartige Software auch herstellerübergreifend zusammensetzbar sein; das hat bis heute nicht wirklich funktioniert. Schließlich wünschte man sich Konzepte wie transaktionale Garantien und ähnliches, die sich eben auch in monolithisch organisierter Software bereits bewährt hatten. Eine Folge war die Entwicklung einer Vielzahl von Standards für die unterschiedlichsten Funktionen und Konzepte im Rahmen einer SOA, eine andere eine Inflation von Vorschlägen zur Entwicklung einer SOA oder zur Migration auf eine SOA. Dies ging einher mit einer Abschottung der Hersteller gegeneinander, was insgesamt dazu geführt hat, dass SOA bis heute nicht wirklich als Erfolgskonzept gilt. Natürlich gibt es einzelne Erfolgsstories, aber auf breiter Front ist nicht das entstanden, was man sich gewünscht oder was man erwartet hat.

Parallel dazu sind im Rahmen der technologischen Entwicklungen, die zum sog. „Web 2.0" beigetragen haben, immer mehr Anbieter von Software-as-a-Service (SaaS) dazu übergegangen, ihre Dienste nicht nur im Browser nutzbar, sondern auch in andere Software einbettbar zu machen. Dazu stellt man als Anbieter eine Schnittstelle, ein sog. Application Programming Interface (API), bereit, das es, einem entfernten Prozeduraufruf (RPC) nicht unähnlich, ermöglicht, Funktionalität unter einer bestimmten Webadresse mit Namen und I/O-Parametern aufzurufen; eines der bekanntesten Beispiele ist Google Maps. Die Angebote solcher APIs sind heute sehr zahlreich und vielfältig in ihren Funktionen; sie verzichten weitgehend auf zentrale Verzeichnisdienste, sondern ein Client bzw. Nutzer muss das API kennen, um es in seinen Kontext einbinden zu können. Durch die fortschreitende Web-Basierung von Anwendungen hat dieses Konzept in den letzten Jahren erheblich mehr Furore gemacht als SOA, und das Ergebnis der Anwendung dieses Konzeptes wird nun in naheliegender Weise als Web-orientierte Architektur (WOA) bezeichnet. Festzustellen ist jedoch, dass sich Entwickler in diesem Bereich wenig um Grundlagen neuer Konzepte kümmern, sondern – dem rasanten Tempo der Entwicklung des Webs folgend – Funktionalität möglichst schnell an den Markt bringen wollen. Ebenso beschäftigt sich die Forschung bisher erstaunlich wenig mit dem Thema WOA, und dass, obwohl naheliegende Fragen keineswegs leicht zu beantworten sind: Wie kann ich bei der Entwicklung einer WOA vorgehen? Sind hier Vorschläge aus dem SOA-Umfeld wiederverwendbar oder sind Neuentwicklungen erforderlich? Wie lässt sich eine einmal konzipierte WOA realisieren, was ist dabei zu beachten? Und vor allem: Wann lohnt sich die Entwicklung einer oder der Umstieg auf eine WOA aus betriebswirtschaftlicher Sicht bzw. aus der Sicht eines IT-Controllers?

Diesen und weiteren Fragen ist die vorliegende Dissertation von Gunnar Thies gewidmet; sie gehört damit zu den weltweit ersten, die sich aus wissenschaftlicher Sicht mit dem für derzeitige Unternehmens-IT-Landschaften hochrelevanten Themenkomplex WOA beschäftigt. Ziel ist die Entwicklung einer praktikablen Methode zur Konzeption, Planung, Umsetzung und Bewertung einer WOA-Entwicklung. Outsourcing, Konzentration auf Kernkompetenzen, Web 2.0 und Cloud Computing stellen Unternehmen hinsichtlich der Organisation ihrer IT ständig vor neue Probleme, und dies alles vor dem oben geschilderten Hintergrund, dass das SOA-Konzept bis heute keinen durchschlagenden Erfolg gebracht hat. Angegangen werden diese

Aspekte in der Literatur zumeist entweder aus IT- oder aus betriebswirtschaftlicher Sicht, nur selten jedoch in Kombination oder angemessener Abwägung beider Sichtweisen. Ich sehe es daher als einen Glücksfall an, dass sich gerade Gunnar Thies mit dem Thema WOA so intensiv auseinandergesetzt hat, denn er ist die perfekte Mischung aus einem Informatiker, der Programmierung exzellent beherrscht und daher softwaretechnische Machbarkeitsfragen fast immer aus dem Stand beantworten kann, und einem Wirtschaftsinformatiker, der die Anwendbarkeit von IT-Konzepten in Unternehmenskontexten auch aus ganz anderen Blickwinkeln beurteilen und bewerten kann. Diese Fähigkeiten nutzt er in seiner Dissertation in überzeugender Weise aus. Die genannten Fragen und Zielsetzungen werden umfassend beantwortet bzw. erreicht. Ich wünsche dieser interessanten und gelungenen Arbeit daher eine breite Leserschaft.

Münster, im Mai 2011 Prof. Dr. Gottfried Vossen

Danksagung

Zunächst möchte ich meinem Doktorvater Prof. Dr. Gottfried Vossen danken, der mir bei der Entstehung der Arbeit immer mit fachlichem Rat unterstützend zur Seite stand, ein motivierendes Arbeitsklima am Lehrstuhl schuf und es mir ermöglichte, meine Forschungen auch bei diversen internationalen Konferenzen vorzustellen. Darüber hinaus möchte ich Prof. Dr. Jörg Becker für die Bereitschaft danken, als Zweitgutachter meiner Arbeit zu fungieren. Ebenso danke ich Prof. Dr. Thomas Langer für den Beisitz als Drittprüfer bei meiner Disputation.

In meiner Zeit am Lehrstuhl habe ich mit vielen Kollegen über meine Arbeit und angrenzende Themen gesprochen und dabei wertvolle Meinungen ausgetauscht sowie Standpunkte diskutiert. All dies hat ebenfalls einen entscheidenden Anteil zum Gelingen der Arbeit beigetragen. Ein großer Dank geht daher an Till Haselmann und Dr. Stephan Hagemann, die fachliche wie auch „TEXnische" Hilfestellungen gaben. Danken möchte ich darüber hinaus auch meinen Kollegen Christian Forster, Dr. Jens Lechtenbörger, Dr. Joachim Schwieren, Jens Sieberg und Florian Stahl. Ebenfalls herzlich danken möchte ich Barbara Wicher, die immer wieder bürokratische Angelegenheiten geregelt hat und Ralf Farke, der bei technischen Problemen stets ein offenes Ohr hatte.

Spezieller Dank gilt Stefan Reimers, der sich viele meiner Ideen an diversen Wochenenden anhören durfte, Teile der Arbeit Korrektur gelesen hat und über aktuelle fachliche Themen mit mir diskutierte.

Abschließend danke ich meinen Eltern für die immer schnelle und sehr akribische Arbeit als Korrekturleser meiner Arbeit in der nahen Vergangenheit und die mentale Unterstützung während meiner gesamten Dissertationszeit. Darüber hinaus danke ich meiner gesamten Familie für das nunmehr 30 Jahre anhaltende harmonische Familienklima, welches einen großen Beitrag für mein Durchhaltevermögen während der Arbeit an der Dissertation geschaffen hat.

Münster, im Mai 2011 Gunnar Thies

Inhaltsverzeichnis

Abkürzungsverzeichnis

AO	Abgabenordnung
API	Application Programming Interface
Atom	Atom Syndication Format
BAM	Business Activity Monitoring
BEEP	Blocks Extensible Exchange Protocol
BMBF	Bundesministerium für Bildung und Forschung
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
CA	Certificate Authority
CBM	Component Business Modeling
CORBA	Common Object Request Broker Architecture
COSMO	Conceptual Service Modelling
CRM	Customer Relationship Management
DaaS	Database-as-a-Service
DBMS	Datenbankmanagementsystem
DFN	Deutsches Forschungsnetz
EBNF	erweiterte Backus-Naur Form
ECC	Elastic Compute Cloud
EPK	ereignisgesteuerte Prozesskette
ERM	Entity Relationship Model
ERP	Enterprise Resource Planing
ESB	Enterprise Service Bus
ETL	Extraktion, Transformation, Laden
FDD	Feature Driven Development
FPS	Flexible Payment Service
FTP	File Transfer Protocol
GUI	Graphical User Interface
HGB	Handelsgesetzbuch
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPAuth	HTTP-Authentifizierung
HTTPS	verschlüsseltes HTTP
IaaS	Infrastructure-as-a-Service
IEEE	Institute of Electrical and Electronics Engineers
IKT	Informations- und Kommunikations-Technologie
IP	Internet Protocol
ISDL	Interaction System Design Language
ISO	International Organization for Standardization
IT	Informationstechnologie
ITIL	IT Infrastructure Library
JBI	Java Business Integration
JMS	Java Message Service
JSON	JavaScript Object Notation
KB	Kilobyte

KPI	Key Performance Indicator
MD5	Message-Digest Algorithm 5
MOM	Message-oriented Middleware
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OLAP	Online Analytical Processing
OMG	Object Management Group
OSI	Open Systems Interconnection
OWL	Web Ontology Language
PaaS	Platform-as-a-Service
PDF	Portable Document Format
PKI	Public Key Infrastructure
RDF	Resource Description Framework
REST	Representational State Transfer
RMI	Remote Method Invocation
ROA	Ressourcen-orientierte Architektur
ROI	Return on Investment
RPC	Remote Procedure Call
RSS	Really Simple Syndication
RUP	Rational Unified Process
SaaS	Software-as-a-Service
SCA	Service Component Architecture
SDLC	Systems Development Life Cycle
SeCSE	Service Centric Systems Engineering
SHA1	Secure Hash Algorithm
SLA	Service Level Agreement
SLR	Service Level Requirement
SMTP	Simple Mail Transfer Protocol
SOA	Service-orientierte Architektur
SOAP	ursprünglich Simple Object Access Protocol
SOMA	Service-oriented Modeling and Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SSL	Secure Socket Layer
SWOT	Strength, Weakness, Opportunities and Threats
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UC	Underpinning Contract
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
UNSPSC	United Nations Standard Products and Services Code
URI	Uniform Resource Identification
URL	Uniform Resource Locator
VOFI	vollständiger Finanzplan
W3C	World Wide Web Consortium
WAC	Web Architecture Controller
WADL	Web Application Description Language
WE	Workflow Engine
WOA	Web-orientierte Architektur
WP	Web-Prozedur
WPC	Web Procedure Call
WS-AT	WS-AtomicTransaction
WS-BA	WS-BusinessActivity

WS-CAF	WS-Composite Application Framework
WSA	Web Service Architecture
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
WWU	Westfälische Wilhelms-Universität
WWW	World Wide Web
XaaS	Everything-as-a-Service
XML	Extensible Markup Language
XP	eXtreme Programming
XSD	XML Schema Definition
YAGNI	You ain't gonna need it

1 Einleitung

In einer hochgradig technologieabhängigen Welt, in der der kommerzielle Erfolg eines Unternehmens oft stark von der informationstechnologischen Unterstützung der Geschäftsabläufe abhängt, ist die Qualität von Hard- und Software ein Schlüsselfaktor. Die Entwicklung von Softwaresystemen erfolgt damals wie heute meist auf den aktuellen Paradigmen der Softwarebranche. Schon früh wurde der Nutzen verteilter Systeme begriffen und in Systemarchitekturen berücksichtigt. Tanenbaum definiert ein verteiltes System als Zusammenschluss mehrerer unabhängiger Computer, die durch den Benutzer als ein System wahrgenommen werden (vgl. [AST07]). Diese Art von Vernetzung wurde durch die Entwicklung des Internets am CERN, der europäischen Organisation für Kernforschung, durch Tim Berners-Lee Ende der 90er Jahre leichter umsetzbar.

Die globale Weltwirtschaft ist heute nicht mehr ohne das Internet und die dadurch vorhandenen Kommunikationsmöglichkeiten von Menschen und Maschinen denkbar. So kamen u. a. durch diese Entwicklung Spezifikationen für die Maschine-zu-Maschine-Kommunikation verteilter Systeme auf. Die ersten Ansätze lagen mit dem Aufrufen entfernter Prozeduren Mitte der 70er Jahre vor (vgl. [Whi76]). Diese wurden dann Ende der 80er Jahre wieder unter dem Begriff *Remote Procedure Call* (RPC) aufgegriffen (vgl. [Net88]). Ebenso traten zu dieser Zeit Konzepte auf wie die *Common Object Request Broker Architecture* (CORBA) der *Object Management Group* und die Mitte der 90er Jahre als Alternative angekündigte *Remote Method Invocation* (RMI) von SUN. Die Auswirkungen auf die Struktur von Computerprogrammen und auf den Systemaufbau waren nur mäßig zu spüren, da Schnittstellen zu anderen Systemen lediglich vereinzelt implementiert wurden.

Als neues Paradigma der Softwareentwicklung verteilter Systeme entstand Mitte der 90er Jahre das Konzept der Service-orientierten Architektur (SOA), das zunächst die Art der Organisation und der Implementierung eines Softwaresystems stark beeinflusste (vgl. [SN96, Sch96]). Die grundlegende Idee ist dabei das Kapseln von Programmfunktionalitäten in Form von modularen

Services, die dann als Dienstleistung bereitgestellt und von anderen Systemen aufgerufen werden können. Welche fachliche Logik dabei in einem Service enthalten ist, orientiert sich an den zugrunde liegenden Geschäftsprozessen eines Unternehmens und folgt der Zielsetzung einer hohen Wiederverwendbarkeit. Durch ein konsequentes Service-orientiertes Design der IT-Landschaft lassen diese sich durch die Verknüpfung und Koordinierung (Orchestrierung) mehrerer Services realisieren und im Bedarfsfall flexibel anpassen. Die Komplexität der Vernetzung unterschiedlicher IT-Systeme soll hierbei durch die Verwendung standardisierter Schnittstellen und Technologien reduziert und damit u. a. eine Kostenreduktion herbeigeführt werden. Dabei ist das Konzept einer SOA an sich unabhängig von der konkreten technologischen Umsetzung, wobei sich in den Software-Produkten namhafter Hersteller meist dieselben Technologien und Standards wiederfinden. Bis heute sind SOA umstritten und setzen sich als IT-Systemumsetzung nur langsam durch, obwohl viele anerkannte Standards für die Komponenten einer SOA existieren. Dies mag unter anderem an der komplexen Technologie, den hohen Kosten der Umsetzung und der Nichterreichung der ursprünglich versprochenen Ziele liegen.

Neuer Aufschwung im Bereich der verteilten Systeme kommt seit einigen Jahren durch die Entwicklung sogenannter „Web 2.0"-Applikationen zustande. Diese Revolution der privaten Nutzung des Internets mit der Entstehung zahlloser sozialer Netzwerke und *Rich Internet Applications* (RIA) und dem dadurch verbundenen Wunsch nach einfacher Systemvernetzung verhilft Konzepten wie dem *Representational State Transfer* und *JavaScript Object Notation* (JSON) als „einfaches" Datenaustauschformat zu hoher Aufmerksamkeit. So werden Schnittstellen für Softwaresysteme, beispielsweise von den großen E-Commerce-Anbietern und „Big Playern" des Internets, mittlerweile nicht nur als Web Services, sondern zusätzlich auch als REST-Services oder einfache Web APIs angeboten. Dies führt bereits zu großer Akzeptanz dieser Schnittstellenumsetzung in vielen Bereichen der IT-Branche. In der Tat scheinen Architekturen auf Basis von REST-Services einige der Nachteile von SOA auszugleichen und durchaus eine ernst zu nehmende Konkurrenz zu werden.

Neue Konzepte wie das einer Web-orientierten (WOA) oder Ressourcenorientierten Architektur (ROA) rücken dabei stärker in den Fokus, da diese vor allem die einfache Kombination von Daten und Prozessen ermöglichen sollen (vgl. [RR07, Ove07a, Hin08]). Diese Ansätze werden zusätzlich durch aktuelle Entwicklungen des *Cloud Computing* und durch das Entstehen vieler Anbieter von Everything-as-a-Service (XaaS) gestärkt, so dass sich umfangreiche Mög-

lichkeiten für Unternehmen bieten, Teile ihrer IT-Landschaft auszulagern (vgl. [Cus10, Cof09]).

1.1 Problemstellung und Motivation

Für die Realisierung einer SOA gibt es diverse Vorgehensmodelle[1], die die Umsetzung des Paradigmas schrittweise beschreiben und zu einer funktionierenden Systemarchitektur führen sollen. Es existieren auch unbestreitbare Technologiestandards im SOA-Umfeld, wie beispielsweise *Web Services* für die Kommunikation zwischen Softwarekomponenten, *Universal Description, Discovery and Integration* (UDDI) als Verzeichnisdienst für Services und *SOAP* als XML-basiertes Nachrichtenformat. Darüber hinaus wurden allein im Bereich der Web Services mehr als 70 zusätzliche Standards[2] definiert, die Lösungsansätze für möglichst viele Aufgaben in einem verteilten System bereitstellen sollen. Trotzdem hat sich bis heute keines der Vorgehensmodelle als Standard etabliert und die Existenz der diversen WS-Standards verkompliziert eher die Realisierung einer SOA-Lösung, als diese zu vereinfachen. Hinzu kommt zum einen, dass bereits ein zentraler Teil, wie *UDDI*, de facto kaum wirklich benutzbar ist (vgl. [HLV07]). Zum anderen erinnert der Overhead für Code und Bandbreite, der durch die strikte Verwendung des SOAP-Formats und der damit zusammenhängenden Verwendung des Datenformats XML auftritt, hierbei an die Komplexität von CORBA, die als Middleware-Spezifikation in den 90er Jahren durch zu komplexe Strukturen nicht überzeugen konnte (vgl. [Hen08]).

Im Gegensatz zur SOA stützen sich Konzepte wie das hier betrachtete einer WOA oder einer ROA für die Realisierung eines verteilten Systems auf einige wenige Standards, wie beispielsweise das *Hypertext Transfer Protocol* (HTTP), welches u. a. die Grundlage des Internets darstellt. Dennoch sind für die Umsetzung dieser Softwarearchitekturen bisher noch keine anerkannten Vorgehensmodelle vorhanden. Dabei ist das Internet mit den darin enthaltenen Hypertext-Verknüpfungen eine unbestrittene Erfolgsgeschichte, so dass der Aufbau eines verteilten Anwendungssystems unter Verwendung derselben Standards nahe liegt. Als Basis für die Realisierung und den Betrieb einer Softwarearchitektur kann der *Systems Development Life Cycle* (SDLC) dienen,

[1] Siehe Kapitel 4.

[2] Ein Großteil der Standards stammt von der *Organization for the Advancement of Structured Information Standards* (OASIS), siehe: `http://www.oasis-open.org/specs/`

der grundlegende Planungs-, Analyse-, Entwicklungs-, Test- und Wartungsschritte erläutert (vgl. [Uni03, BF08]). Diese finden sich auch in den meisten Softwareentwicklungsmodellen wieder. Als grobe Gliederung wird im Kontext dieser Arbeit eine Einteilung des SDLC in die folgenden drei Phasen vorgenommen: Analyse und Planung, Realisierung sowie Betrieb.

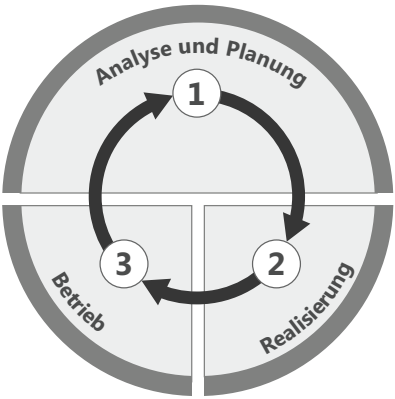


Abbildung 1.1: Grundlegende Phasen eines Softwareentwicklungsmodells.

Das Kernkonzept einer WOA, wie es in dieser Arbeit definiert wird, verwendet möglichst einfache und schon bekannte Web-Standards zur Umsetzung eines verteilten Systems. Dabei sollen vor allem bereits existierende Service-Angebote eingesetzt werden. Eine der Zielsetzungen bei einer archetypischen WOA ist der Verzicht auf eigene Hardware für das IT-System und die komplette Auslagerung der Komponenten, Funktionalitäten und Daten des Unternehmens in das Internet. Dieser Ansatz wird bei SOA-Entwicklungsmodellen lediglich rudimentär oder gar nicht betrachtet. Um die Realisierung einer WOA zu erläutern, bedarf es daher einerseits der Entwicklung eines Vorgehensmodells, das alle Phasen von der Analyse und Planung über die Entwicklung und das Testen bis zum Betrieb einer WOA ausführlich definiert. Dazu gehört auch die Betrachtung einer Migration bereits bestehender IT-Systeme auf eine WOA. Andererseits muss auch eine wirtschaftliche Bewertung erfolgen, die darlegt, welche Kostenvorteile bei einer WOA- gegenüber einer SOA-Realisierung zu erwarten sind. Ein wichtiger Faktor für die Auslagerung von Services sowie

die Kostenbetrachtung ist hierbei das stetig wachsende Angebot an XaaS-Anbietern, die das Outsourcing von Unternehmensaufgaben im IT-Bereich in großem Maße ermöglichen.

Die Fragestellungen, die innerhalb dieser Arbeit bearbeitet und erläutert werden, befassen sich daher zunächst mit dem grundlegenden Architekturstil einer WOA und der damit einhergehenden Abgrenzung gegenüber dem einer SOA. Darüber hinaus werden die Modellierung einer WOA, die tatsächliche technologische Ausgestaltung und Umsetzung sowie die wirtschaftliche Betrachtung im Rahmen einer Totalkostenberechnung behandelt. Einige wichtige Fragen sollen im Verlauf der Arbeit beantwortet werden:

1. Worin liegen die Unterschiede zwischen Service-, Ressourcen- und Web-Orientierung, und warum lohnt sich die Auseinandersetzung mit WOA?

2. Lassen sich verteilte Softwaresysteme mit einer WOA umsetzen? Dabei muss beispielsweise die Komplexität einer WOA mit der einer SOA verglichen werden. Des Weiteren müssen Aspekte der Sicherheit und komplexe Prozesskonstellationen berücksichtigt und umgesetzt werden.

3. Lässt sich ein generelles Vorgehensmodell zur Planung und Umsetzung einer WOA aus bekannten Modellen ableiten und wie kann eine Durchführung des Vorgehensmodells konkret ausgestaltet sein? Dabei müssen auch Unterschiede der Methode zwischen Unternehmen mit und ohne bestehender Systemlandschaft verglichen werden. Ein weiterer wichtiger Aspekt betrifft den Betrieb und die Wartung einer WOA nach deren Realisierung.

4. Wie lassen sich die Kosten einer WOA schätzen bzw. denen einer SOA gegenüberstellen? Wichtig ist hierbei, einen Kostenvergleich zwischen einer WOA- und einer SOA-Umsetzung zu erreichen, der beispielsweise auf der Berechnung der *Total Cost of Ownership* beruhen kann. Darüber hinaus sind auch Aspekte wie Outsourcing-, Planungs-, Umsetzungs- und Betriebskosten einer WOA von Interesse und müssen untersucht werden.

Durch das Phänomen des Web 2.0 und der schnell wachsenden Menge an Web-Applikationen, die teilweise nur ungenügend durchdacht sind,

haftet WOA und ROA der Ruf an, schnell programmierte Software mit „Quick'n'Dirty"-Ansätzen zu sein, ohne dabei ein eindeutiges Architekturkonzept zu verfolgen. Daher wird innerhalb dieser Arbeit einer WOA ein klares Architekturkonzept zugrunde gelegt und eine Vorgehensmethode vorgestellt, die eine moderne, verteilte Systemarchitektur als schlankes Modell (im Gegensatz zu einer reinen SOA-Lösung) umsetzt. Zusätzlich wird der Outsourcing-Gedanke ausgeweitet, denn viele Dienste erlauben das Konzentrieren auf die eigenen Stärken und die Verwendung von fertigen Systemkomponenten durch die Nutzung von XaaS (vgl. [BLH08]).

Abbildung 1.2: Das Rahmenkonstrukt einer WOA.

Das Rahmenkonstrukt einer WOA umfasst die drei Kernbereiche **Geschäftsprozesse** (*was*), **Architektur** (*womit*) und **Methoden** (*wie*) und ist in Abbildung 1.2 dargestellt. Die Reihenfolge der Nennung soll hierbei verdeutlichen, dass die Prozesse, also die organisatorische Sicht auf ein Unternehmen, als wichtigster Antrieb für eine Systemumsetzung angesehen werden. Erst an zweiter Stelle steht die Architektur, die zwar die Basis eines funktionierenden Unternehmens darstellt, jedoch zum Zwecke der Geschäftsprozessunterstützung realisiert wird. Der dritte zentrale Aspekt befasst sich mit den Methoden zur Umsetzung einer WOA. Wichtig ist hierbei, dass es für die Umsetzung einer WOA nicht nur einen einzigen passenden Weg gibt, sondern durchaus diverse Vorgehensmodelle und Werkzeuge zum Einsatz kommen können. Im Verlauf

dieser Arbeit werden zwar gewisse Ansätze bevorzugt und später auch ausgeführt, aber das hier entwickelte Vorgehensmodell definiert bewusst spezielle Anforderungen und Zwischenergebnisse, die sich leicht für die Verwendung in anderen Methoden adaptieren lassen.

1.2 Vorgehensweise und Aufbau der Arbeit

Für die Bearbeitung wissenschaftlicher Fragestellungen in der Wirtschaftsinformatik existiert ein breites Methodenspektrum, wie eine Studie aus dem Jahr 2007 feststellt (vgl. [WH07, WH06]). Darin werden verschiedene Forschungsmethoden beschrieben, die sich für die Untersuchung eines Themas eignen. Diese beziehen sich dabei entweder auf ein konstruktionswissenschaftliches (*Design Science*) oder ein verhaltenswissenschaftliches Paradigma (*Behavioral Science*). Während sich das Erste mit der konkreten Entwicklung von IT-Systemen unter Zuhilfenahme von Modellen, Methoden und Systemen widmet (vgl. [HMPR04]), befasst sich das Zweite mit der Analyse der Auswirkung bestehender Technologien und IT-Systemen auf deren Umwelt, die Menschen und die Märkte. Die vorliegende Arbeit folgt einem konstruktionswissenschaftlichen Ansatz und bedient sich für die Konzeption einer WOA und dem damit verbundenen Vorgehensmodell der Forschungsmethode der argumentativen Analyse. Diese enthält sowohl induktive Teile, ausgehend von Beobachtungen des Web 2.0, als auch deduktive Teile, beispielsweise die Betrachtung bestehender Vorgehensmodelle für SOA und die Verwertung einzelner Teile davon für die WOA-Methode. Durch die beispielhafte Realisierung eines Ausschnittes einer WOA sowie einer wirtschaftlichen Kostenbetrachtung erfolgt eine Evaluation, die zu weiterem Erkenntnisgewinn herangezogen wird.

Die Arbeit gliedert sich in sieben Kapitel, die im Folgenden beschrieben werden (vgl. Abbildung 1.3). Nach der Einleitung widmet sich das zweite Kapitel den Grundlagen des Themas. Zu Beginn wird darin eine Einführung in verschiedene Vorgehensmodelle der Softwareentwicklung gegeben (Kapitel 2.1). Dabei werden traditionelle wie auch agile Methoden vorgestellt. Anschließend werden verteilte Systeme und die Standards des Internets erläutert (Kapitel 2.2). Im letzten Abschnitt werden zwei Architekturarten beschrieben und voneinander abgegrenzt: zum einen die SOA und zum anderen die ROA. Abgerundet wird dieser Teil durch die Betrachtung funktionaler Klassifizierungen von Service-Arten (Kapitel 2.3).

Das dritte Kapitel befasst sich mit dem Konzept einer WOA. Zu Beginn werden darin die Motivation und die Aufgabe des Architekturmodells vorgestellt (Kapitel 3.1). Anschließend werden die Eigenschaften und Kernkonzepte einer WOA ausgeführt (Kapitel 3.2). Darin wird zunächst auf die Topologie und die unterschiedlichen Ausprägungen einer solchen Architektur eingegangen. Wie jede Softwarearchitektur besitzt auch eine WOA ein Schichtenmodell, das vorgestellt wird. Um die Steuerung von Services zu bewerkstelligen, existiert das Konzept des *Web Architecture Controller* (WAC), dessen Funktionalität beschrieben wird. Nachdem die architektonischen und fachlichen Randbedingungen einer WOA geklärt sind, wird auf die technologische Umsetzung eingegangen. Den Abschluss des Kapitels bildet eine Übersicht der Stakeholder, die in einer WOA-Realisierung und deren Betrieb eine Rolle spielen (Kapitel 3.3).

Verwandte Arbeiten, die sich mit der Umsetzung von verteilten Systemen befassen, meist einer SOA, werden im vierten Kapitel vorgestellt und von dem Ansatz dieser Arbeit abgegrenzt. Dabei wird eine Einteilung in Architekturmodelle (Kapitel 4.1) sowie partielle (Kapitel 4.2) und vollständige (Kapitel 4.3) Vorgehensmodelle vorgenommen.

Im fünften Kapitel wird eine ausführliche Beschreibung des Vorgehens zur Umsetzung und zum Betrieb einer WOA dargelegt. Da hierbei viele Modellierungsaufgaben zu erledigen sind, wird zunächst eine Auswahl über die in diesem Kapitel benötigten Modellsprachen getroffen und deren Verwendung skizziert. Im Anschluss wird eine Übersicht der generischen Phasen des Vorgehensmodells beschrieben, die sich einteilen lassen in *Analyse und Planung*, *Realisierung* und *Betrieb* (Kapitel 5.1). Es wird der Ablauf jeder einzelnen Sub-Phase kurz erläutert und die daraus resultierenden Dokumente genannt. Nach der Festlegung einiger Annahmen im Abschnitt zur Unternehmensstrategie (Kapitel 5.2) folgt der Einstieg in die Analyse- und Planungsphase (Kapitel 5.3). Hierin werden drei generische und drei WOA-spezifische Arbeitsschritte im Detail erläutert. Anschließend wird die Realisierungsphase beschrieben, in der die Umsetzung des geplanten Systems vollzogen wird (Kapitel 5.4). Dieser folgt die Erläuterung der Betriebsphase, in der auch auf die Kostenkontrolle einer WOA eingegangen wird (Kapitel 5.5). Zu diesem Zeitpunkt ist das Vorgehensmodell zur Erstellung einer WOA für ein Unternehmen ohne bestehendes IT-System vollständig beschrieben. Es folgt ein Abschnitt mit den Unterschieden im Vorgehen, wenn das Unternehmen ein bestehendes IT-System einzu-

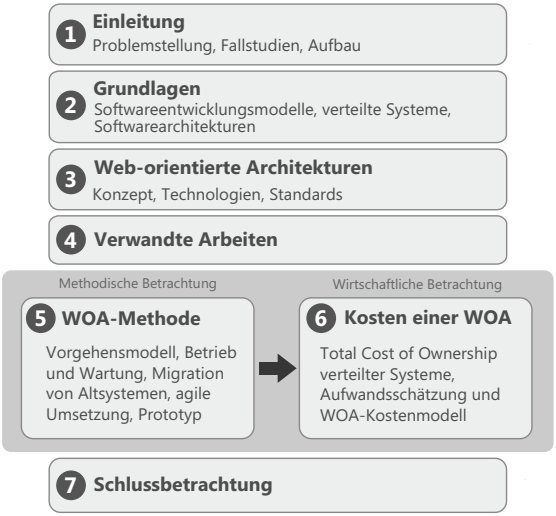


Abbildung 1.3: Aufbau der Arbeit.

binden hat (Kapitel 5.6). Anschließend wird die konkrete Umsetzung des zuvor vorgestellten Vorgehensmodells anhand der agilen Methode *Feature Driven Development* (FDD) beschrieben (Kapitel 5.7). Hierin werden die Vorzüge einer agilen Entwicklung und deren Vorteile für die später folgende Kostenberechnung einer WOA-Erstellung hervorgehoben. Den Abschluss des Kapitels bildet die Beschreibung einer tatsächlichen Umsetzung der zentralen Komponenten des WAC in einer WOA durch XaaS-Dienste des Internets (Kapitel 5.8). Darin wird gezeigt, dass die Umsetzung einer WOA keine Utopie, sondern ein erreichbares Ziel ist.

Um sich den Kosten einer WOA zu widmen, wird im sechsten Kapitel zunächst die Berechnungsmethode *Total Cost of Ownership* (TCO) für IT-Systeme erläutert (Kapitel 6.1). Dass die TCO-Methode auch mit einigen Mängeln behaftet ist, wird in einer Kritik aufgearbeitet, die auch zusätzliche Kostentreiber nennt und begründet. Im Anschluss wird ein Teil einer IBM-Fallstudie in einer konkreten SOA-Umsetzung vorgestellt. Dabei werden zwei Varianten berechnet: eine mit IBM- sowie eine mit Open-Source-Produkten. Daraufhin werden die Unterschiede der TCO-Berechnung für eine WOA erläutert, dieselbe Fall-

studie für eine WOA-Realisierung betrachtet und daran ein Kostenvergleich vorgenommen (Kapitel 6.2). Neben den Hard- und Softwarekosten muss auch die tatsächliche Realisierung berechenbar sein. Daher wird hier eine mathematische Betrachtung eines WOA-Kostenmodells für die Umsetzung beschrieben, welches es ermöglicht, auf Basis der agilen Methode FDD die Softwareentwicklungskosten einer WOA abzuschätzen. Eine vor allem durch diverse Geschäftsmodelle der XaaS-Anbieter begründetes Kostenmodell befasst sich daraufhin mit den Betriebskosten einer archetypischen WOA, die durch einen Service-Mix zusammengestellt ist. Zu diesem Teil der Kostenberechnung wurde darüber hinaus auch eine Web-Applikation als Prototyp erstellt, die es ermöglicht, verschiedene Szenarien zur Betriebskostenberechnung zu erstellen und WOA-Varianten miteinander zu vergleichen (Kapitel 6.3).

Im siebten Kapitel folgt eine Schlussbetrachtung, die neben einer Zusammenfassung der Ergebnisse der Arbeit auch einen Ausblick auf zukünftige Forschungsbereiche gibt. Darüber hinaus wird eine Stärken- und Schwächenanalyse der WOA skizziert, die zu einer Handlungsempfehlung für den Einsatz einer WOA in Unternehmen überleitet (Kapitel 7.2).

Zum Abschluss dieser Arbeit wird ein vollständiges Vorgehensmodell für die Planung und Realisierung einer WOA für Unternehmen definiert sein. Dabei wird auch berücksichtigt werden, ob ein Unternehmen bereits ein bestehendes IT-System besitzt, welches ggf. mit eingebunden werden muss. Durch die Betrachtung der Umsetzung mittels einer agilen Methodik wird dabei eine möglichst einfache Realisierungsmethode erläutert worden sein, die gerade für Start-up-Unternehmen eine relevante Option ist. Eine Evaluation der Umsetzbarkeit einer WOA wird anhand einer XaaS-Umsetzung des Web Architecture Controllers gezeigt und die Praktikabilität einer WOA-Umsetzung im Hinblick auf die Kostenreduzierung einer verteilten Systemarchitektur untersucht worden sein. Diese Arbeit versucht damit, den Missstand des Fehlens einer konkreten WOA-Definition und eines darauf zugeschnittenen Vorgehensmodells zu beheben. Dabei soll auch gezeigt werden, dass Web-basierte Anwendungen, die durch die Software-Applikationen des Web 2.0 teilweise einen schlechten Ruf haben, nicht grundsätzlich schnell und unsauber programmierte Software repräsentieren. Es soll dargestellt werden, dass eine WOA einem konkreten Architekturkonzept folgt und sie damit als produktives IT-System eines Unternehmens einsetzbar ist. Auch eine einfache Betriebskostenbetrachtung diverser Service-Nutzungsszenarien, die hier anhand eines Kostenmodells sowie

eines Web-Prototyps gezeigt werden soll, existiert bisher für eine WOA noch nicht. Die Idee des Outsourcings von Hard- und Software wird hierin konsequent soweit verfolgt, dass bei einer archetypischen WOA keinerlei eigene Infrastruktur mehr im Unternehmen für das fachliche IT-System existieren muss. Lediglich die Arbeitsplätze der Mitarbeiter müssen vorhanden und an das Internet angeschlossen sein. Neu ist hierbei nicht die komplette Auslagerung, die bereits bei vielen Unternehmen durch die Verlagerung der Server in Rechenzentren besteht, sondern die Umsetzung möglichst aller fachlichen Funktionalitäten durch diverse XaaS-Angebote des Internets. Durch aktuelle Entwicklungen des Cloud Computing wird dieser Aspekt der Arbeit zusätzlich unterstützt und motiviert.

1.3 Beispielszenarien

Im Verlauf der Arbeit werden zur Erläuterung einzelner Aspekte, insbesondere bei der Darlegung des Vorgehensmodells und der Kostenberechnung, des Öfteren Beispiele angeführt. Um für diese einen Rahmen zu schaffen, werden im Folgenden zwei Szenarien vorgestellt, die dann als fortlaufende Beispiele dienen. Das erste Szenario beschreibt ein Start-up-Unternehmen, das noch kein eigenes IT-System besitzt und das Ziel verfolgt, eine WOA vollständig durch Dienste des Internets abzubilden. Das zweite Szenario beschreibt ein Einzelhandelsunternehmen, das bereits ein umfassendes IT-System besitzt, welches bisher zum Teil als SOA-Lösung implementiert ist. Das Unternehmen ist aber dazu bereit, große Teile der IT-Struktur als WOA zu realisieren. Die Vorgehensweise zur Umsetzung einer WOA unterscheidet sich bei Unternehmen mit und ohne vorhandenem IT-System in einigen Punkten. Dies wird im Verlauf der Arbeit erläutert.

Start-up-Unternehmen

Das Start-up-Unternehmen *Boxed Business Intelligence (BBI) GmbH* besteht aus fünf IT-Spezialisten, die eine Web-basierte Modellierungssoftware für die Verwendung im Browser entwickelt haben, die sie als Software-as-a-Service (SaaS) im Internet anbieten wollen. Diese Software ermöglicht das Erstellen und Verwalten von Modellen verschiedener Sprachfamilien unter Berücksichtigung der modellspezifischen Semantik. Die aktuell implementierte Bandbreite umfasst Petri Netze, Entity Relationship-Modelle sowie Organigramme und

einige UML-Diagrammarten. Durch die modulare Programmierung der Software können nach und nach weitere Sprachfamilien hinzugefügt werden. Darüber hinaus bieten sie Beratungsleistungen im Bereich Business Intelligence und Data Warehousing an und führen Schulungen rund um grundlegende IT-Themen durch. Um eine möglichst günstige IT-Infrastruktur für die Neugründung zu etablieren, wollen sie ihr SaaS-Angebot ohne die Anschaffung und das Betreiben eigener Hardware umsetzen und damit die gesamte für das operative Geschäft benötigte Infrastruktur des Unternehmens als WOA ins Internet auslagern. Die folgende Liste nennt die von den Jungunternehmern identifizierten Anwendungsarten:

- Office-Software (Dokumentenverarbeitung, Präsentationssoftware, Tabellenkalkulation, etc.) für den alltäglichen Umgang mit Dokumenten.
- Projektmanagement-Software für die Planung, Verwaltung und Kontrolle der eigenen IT-Beratungsprojekte.
- Bugtracking-Software sowie Versionierungsverwaltung für die eigene Weiterentwicklung des SaaS-Projekts.
- Ticket-System, das von den Kunden des SaaS-Dienstes für Anfragen und Hilfestellungen verwendet werden kann.
- E-Mail-Server für das Verwalten der Mail-Konten der Mitarbeiter.
- Wiki-System als Basis für Dokumentationen und Wissenstransfer zwischen den Mitarbeitern und ggf. als Nachschlagewerk für Kunden.
- Überwachungssoftware für den eigenen SaaS-Dienst, der die Mitarbeiter benachrichtigt, wenn Teile des Systems fehlerhaft arbeiten oder ausgefallen sein sollten.
- Intelligente Telefonschaltung / Callcenter-Lösung für das Anbieten eines automatisierten Kundensupports und dadurch das Erreichen einer Entlastung der Mitarbeiter.
- *Customer Relationship Management* (CRM)-System für die Verwaltung von Kundendaten und Abwicklung der Kundenakquise.

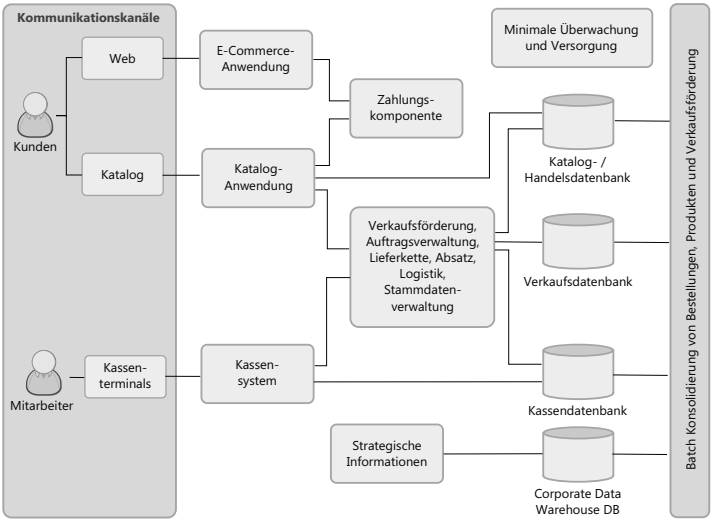
Eine Anforderung der *BBI GmbH* an ihr IT-System ist der Betrieb einer flexibel skalierbaren Serverplattform für den eigenen SaaS-Dienst sowie die Möglichkeit, diese zu überwachen und zu steuern. Dabei soll keine eigene Hardware benötigt werden. Darüber hinaus soll auch die komplette Softwareausstattung für den Beratungsbetrieb und die Standardaufgaben der Mitarbeiter durch ausgelagerte Software realisiert werden. Ein besonderes Augenmerk legen sie hierbei auf SLAs, die eine hohe Verfügbarkeit der einzelnen Software-Komponenten garantieren, da sonst ihr SaaS-Produkt sowie die produktiven Prozesse der Beratung und Schulung durch Ausfallzeiten seitens der verwendeten Dienstleister stark beeinträchtigt wären. Als IT-Infrastruktur möchte das Unternehmen selbst nur Laptops sowie eine Breitbandverbindung zum Internet bereitstellen. Das erklärte Ziel der *BBI GmbH* durch Einsatz einer WOA ist daher, eine günstige, ausgelagerte IT-Plattform zu implementieren, die sich leicht erweitern lässt und die Gründung des Unternehmens ohne viel Startkapital ermöglicht. Eine Integration bestehender Systeme ist hierbei nicht notwendig und vereinfacht dadurch die Erstellung und Wartung der Systemarchitektur.

Das größte Risiko der Unternehmer liegt vor allem in der Abhängigkeit von XaaS-Anbietern sowie der Verfügbarkeit der Breitbandverbindung zum Internet. Für das Betreiben eines eigenen SaaS-Angebots muss eine garantierte Erreichbarkeit angestrebt werden. Diese sollte durch geeignete SLAs mit den Anbietern von Serverinfrastruktur klar definiert sein. Darüber hinaus lassen sich beispielsweise Teile der IT-Struktur redundant aufbauen, so dass das Risiko eines Ausfalls minimiert werden kann. Zusätzlich haben die Mitarbeiter der *BBI GmbH* die Möglichkeit des mobilen Zugriffs auf das Internet, so dass ein Ausfall der stationären Breitbandverbindung in den Räumlichkeiten des Unternehmens kurzfristig kompensiert werden kann.

Einzelhandelsunternehmen mit existierender SOA-Umsetzung

Als Szenario für die Migration bestehender Systeme und die Verwendung einer WOA in kleinen und mittleren Unternehmen (KMU) wird eine Fallstudie von IBM aus [KBD+08] verwendet. Darin wird das fiktive Einzelhandelsunternehmen *JKHL Enterprises* untersucht.

Das Ziel der IBM-Studie war dabei das Aufzeigen von SOA-Lösungsstrategien, die zu einer Steigerung der Prozesseffizienz und der Kundenzufriedenheit, einer Verringerung der Produkteinführungszeit und



Quelle: nach [KBD+08]

Abbildung 1.4: Systemarchitektur der Verkaufssysteme der *JKHL Enterprises*.

der Kosten führen soll. Das Unternehmen betreibt bereits einen Web-Shop sowie 900 Ladengeschäfte und hat die Zielsetzung, dass die Kunden über mehrere Kanäle, online und offline, Produkte kaufen können. Das in der Fallstudie aufgezeigte Architekturmodell einer SOA wird im Verlauf dieser Arbeit als Beispielarchitektur für die Entwicklung einer WOA in KMU dienen. Die Annahme ist hierbei, dass *JKHL Enterprises* mit dem SOA-Ansatz aufgrund der hohen Kosten, der aufwendigen Wartung und der umständlichen Skalierbarkeit nicht zufrieden ist und sich für eine WOA-Umsetzung interessiert. Eine komplette Auslagerung des Systems und der Daten des Unternehmens ist aus Gründen des Datenschutzes nicht angestrebt. Es sollen aber große Teile der IT-Infrastruktur durch XaaS-Dienste ersetzt werden. Im Gegensatz zum zuvor vorgestellten Start-up-Unternehmen zielt man mit der Migration hierbei nicht primär auf eine Kostenreduktion ab, sondern versucht eine Entlastung der eigenen IT-Abteilung durch die Vereinfachung der IT-Landschaft und die Nutzung von Cloud-Angeboten zu erreichen. In Abbildung 1.4 ist ein Überblick über die verwendeten Bestandteile des IT-Systems dargestellt. Es werden hierbei zwei Akteure identifiziert: der Kunde, der über das Internet oder einen Katalog direkt Waren bestellen kann, sowie der Mitarbeiter in einem Ladengeschäft, der über ein Kassensystem Waren verkauft. Des Weiteren sind die dahinter liegenden Systeme und verknüpften Datenbanken aufgelistet.

Die gezeigte Architektur beschreibt nur den Teil des IT-Systems von *JKHL Enterprises*, der für den Verkauf und Vertrieb von Produkten relevant ist. Dieser genügt aber als Szenario für den Anwendungsfall der WOA in einem KMU mit bestehender Infrastruktur. Eine Gesamtbetrachtung wäre auf der einen Seite sehr detailliert und nicht mehr geeignet als Beispiel für die Migration auf eine WOA. Auf der anderen Seite würde man bei der Umsetzung selten direkt ein gesamtes IT-System betrachten und migrieren, sondern nur einen Teilbereich. In diesem Fall werden zum einen Teile des Systems durch XaaS-Angebote ersetzt, zum anderen verbleiben manche Altsysteme im Unternehmen und werden in eine WOA integriert. Die zu erreichenden Vorteile für das Unternehmen bestehen in der Reduzierung eigener Hardware und von Personal im technischen Wartungsbereich sowie der möglichen Kostensenkung für den Betrieb des IT-Systems. Die Gefahren liegen hierbei vor allem in der Sicherheit von Daten in der Cloud und in der Komplexität des Gesamtsystems bei der Realisierung von IT-Strukturen innerhalb und außerhalb des Unternehmens.

2 Grundlagen

Dieses Kapitel stellt die der Arbeit zugrunde liegenden relevanten Themenbereiche der Informationstechnologie (IT) vor. Begonnen wird bei einer Übersicht über Softwareentwicklungsmodelle, wobei insbesondere klassische Vorgehensmodelle von agilen Methoden abgegrenzt werden. Daraufhin folgt eine Einführung in die architektonischen Umsetzungsmöglichkeiten verteilter Systeme im Bereich der Informations- und Kommunikations-Technologie (IKT). Darin wird insbesondere das größte verteilte System der Welt, das *World Wide Web* (WWW), vorgestellt und es werden die darin hauptsächlich verwendeten Standards erläutert. Der größte Teil der Grundlagen widmet sich Softwarearchitekturen und stellt zunächst zwei mögliche Vertreter vor, die für diese Arbeit relevant sind, die Service-orientierte Architektur (SOA) sowie die Ressourcenorientierte Architektur (ROA). Anhand sinnvoller Kategorien werden die beiden Ansätze anschließend miteinander verglichen und bewertet. Die Einführung einer der beiden vorgestellten Architekturen beinhaltet grundsätzlich die ausführliche Auseinandersetzung mit internen und externen IT-Services sowie deren architektonischer Ausgestaltung. Diese werden daher abschließend zunächst aus architektonischer und daraufhin aus fachlicher Sicht klassifiziert.

2.1 Übersicht über Softwareentwicklungsmodelle

Jedes professionell betriebene Softwareprojekt sollte einem Prozessmodell für die Entwicklung folgen, welches die Phasen innerhalb der Softwareerstellung genau definiert. Es existieren hierfür diverse Vorgehensmodelle, die jeweils ihren speziellen Anwendungsfokus haben und sich in vielerlei Hinsicht unterscheiden können. Jedes Modell sollte die Reihenfolge des Arbeitsablaufs, die darin durchzuführenden Aktivitäten, die Definition von Teilprodukten, die Fertigstellungskriterien, die Mitarbeiterqualifikationen, die Verantwortlichkeiten und Kompetenzen sowie die anzuwendenden Standards, Methoden, Richtlinien und Werkzeuge definieren (vgl. [Bal98]). Drei grundlegende Paradigmen der Softwareentwicklungsmodelle lassen sich identifizieren: sequenzielle, ite-

rative oder leichtgewichtige, agile Modelle. Die sequenziellen (traditionellen) Verfahren haben sich schon früh im IT-Bereich etabliert und bieten seit den 70er Jahren ein ingenieurartiges Herangehen an die Softwareerstellung. Im Folgenden werden zunächst zwei traditionelle Verfahren erläutert: Das bekannteste Modell in diesem Zusammenhang ist das Wasserfallmodell, an das viele spätere Modelle anknüpfen. Dies wurde anfangs streng sequenziell verwendet, später jedoch in einer überarbeiteten Version durch Iterationszyklen verbessert. Ein im deutschen Raum ebenfalls sehr bekanntes Modell ist das V-Modell, welches für staatliche Projekte als Standard verwendet wird. Es basiert auf dem Wasserfallmodell und versucht dieses durch entsprechende Erweiterungen zu verbessern. Dem gegenüber stehen die jüngeren, agilen Vorgehensmodelle, die versuchen, eine weniger formale, dafür aber mehr an den Kundenwunsch angelehnte, Verfahrensweise zur Erstellung von Software umzusetzen. Agile Vorgehensmodelle unterscheiden sich im Vergleich zu traditionellen Varianten in vielen Punkten. Das agile Manifest[3], welches im Jahre 2001 von namhaften Vertretern[4] der agilen Methoden verfasst wurde, enthält die folgenden vier Grundsätze, auf die sich fast alle agilen Vorgehensmodelle berufen:

1. **Individuen und Interaktionen** werden höher bewertet als Prozess und Werkzeuge.

2. **Funktionierende Software** wird höher bewertet als ausgedehnte Dokumentation.

3. **Zusammenarbeit mit dem Kunden** wird höher bewertet als Vertragsverhandlungen.

4. **Reaktion auf Veränderung** wird höher bewertet als Planverfolgung.

Im Kontext dieser Arbeit wird zunächst auf *eXtreme Programming* (XP) eingegangen, welches viele Ansätze der agilen Methoden definiert. Anschließend wird ein weiteres konkretes Vorgehensmodell – das Feature Driven Development – im Detail vorgestellt. Auf diesem basiert später auch das Vorgehensmodell zur Umsetzung einer Web-orientierten Architektur (WOA).

[3] Vgl. `http://agilemanifesto.org/`

[4] Autoren des agilen Manifests: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland und Dave Thomas.

2.1.1 Wasserfallmodell

Das Wasserfallmodell wurde in seiner ersten Form[5] von Royce im Jahre 1970 vorgestellt (vgl. [Roy70]). Das grundlegende Charakteristikum eines solchen sequenziellen Vorgehensmodells ist es, dass jede Phase vollständig durchgeführt werden muss, bevor die nächste begonnen werden darf. Das Wasserfallmodell, nach Balzert, teilt die Entwicklungsschritte des Softwareentwurfs in die folgenden sieben Hauptschritte ein: (vgl. [Bal01]):

- **System- und Softwareanforderungen**: In den ersten beiden Phasen des Wasserfallmodells werden die Anforderungen an das gesamte System und die Umgebung sowie die Software im Einzelnen ermittelt. Anforderungen lassen sich in drei Arten unterteilen: funktionale und qualitative Anforderungen sowie Randbedingungen (vgl. [PR10]).

- **Analyse und Entwurf**:

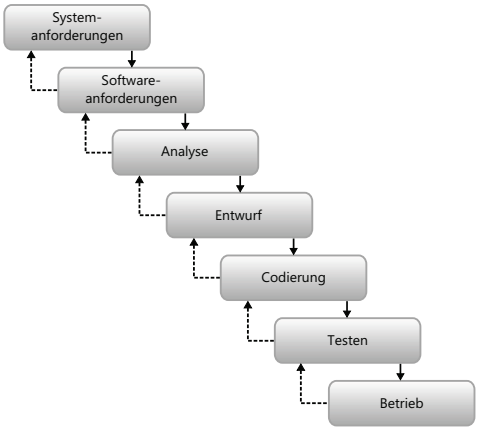
  Diese Phasen widmen sich zunächst dem Erstellen eines Konzepts der Software, welches die grundlegenden fachlichen Funktionen des Systems festlegt. Dabei entsteht in den meisten Fällen eine Übersichtsdokumentation; in diesem speziellen Fall ein Lastenheft. Der Entwurf definiert im Anschluss an das Konzept die tatsächliche Architektur der Software, beispielsweise welche Komponenten wie zu implementieren sind. Dabei entsteht eine ausführliche Dokumentation – das Pflichtenheft – in dem alle Funktionen, Daten und Schnittstellen der Software detailliert aufgeschlüsselt und erläutert werden. Einen vollständigen Entwurf zu definieren, kann durchaus viel Zeit in Anspruch nehmen, da etliche Details bereits hierin fixiert werden. Für den fachlichen Entwurf stehen diverse Beschreibungssprachen zur Verfügung. Sehr verbreitet sind in diesem Bereich die *Unified Modeling Language* (UML) (siehe [HKKR05]) sowie, vor allem für Prozessbeschreibungen, immer mehr die *Business Process Modeling Notation* (BPMN) (siehe [Wes07]).

- **Codierung und Testen**: In der Implementierungsphase werden alle zuvor festgelegten Funktionen implementiert und in Programmcode umgesetzt. Hierbei wird exakt nach der Spezifikation des Pflichtenhefts vorgegangen. Nachträgliche Änderungen oder Verbesserungen an der zu-

[5] Basierend auf einer Veröffentlichung von BENINGTON im Jahre 1956 (vgl. [Ben83], eine Neuauflage der Arbeit aus den 60er Jahren).

vor festgelegten Funktionalität sind hierbei, im Standardmodell, nicht vorgesehen. Nach Abschluss der Programmierung werden Tests durchgeführt. Hierbei reicht die Spannweite von einzelnen Unit-Tests, die eine Klasse oder eine Funktion prüfen, über Modul-Tests, die einen Funktionsbereich testen, bis hin zu System-Tests, die das System im Gesamten überprüfen.

- **Betrieb**: Wenn alle Tests erfolgreich absolviert wurden, kann das System installiert und in die Zielumgebung eingeführt werden.



Quelle: vgl. [Bal01]

Abbildung 2.1: Das Wasserfallmodell.

Das Wasserfallmodell ist ein Dokumenten-getriebenes Vorgehensmodell, so wird nach jeder Phase ein Dokument erstellt, welches für die darauf folgende Phase verwendet wird. Potenzielle Benutzer des Systems und Auftraggeber werden innerhalb der Definitionsphase in den Entwurfsprozess einbezogen. In späteren Phasen werden diese nicht mehr gehört. Das normale Wasserfallmodell sieht keine Rückkopplungen zwischen den einzelnen Phasen vor, was es zu einem sehr starren Modell macht. Royce bemängelt dies auch in seinem Artikel und schlägt zum einen Rückkopplungen vor, so dass man beispielsweise aus der Codierung auch wieder in die Entwurfsphase zurückkehren darf. Außerdem werden iterative Zirkel eingeführt, die das Modell weiter dynamisieren.

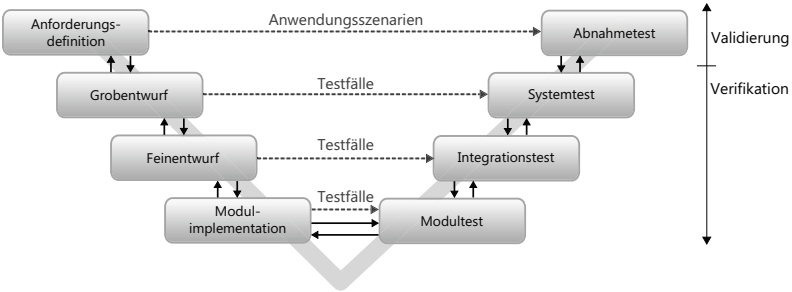
In Abbildung 2.1 ist das Wasserfallmodell mit Rückkopplungsschritten dargestellt. Bis heute wird das Wasserfallmodell als klassisches Vorgehensmodell des Softwareentwurfs vermittelt, wobei die Praxisanwendung aber aufgrund der Unflexibilität deutlich zurückgegangen ist. Auch Balzert nennt die Problematik, dass die Dokumentation manchmal wichtiger wird als das eigentliche System und dass es nicht immer sinnvoll ist, alle Entwicklungsschritte vollständig durchzuführen, bevor zur nächsten Phase übergegangen wird (vgl. [Bal01]).

2.1.2 V-Modell

Das V-Modell stellt eine Erweiterung des Wasserfallmodells dar, welches zum ersten Mal im Jahre 1992 veröffentlicht wurde. Diese Vorgehensweise firmiert seit 2005 unter dem Namen „V-Modell®XT" – XT steht hierbei für *Extreme Tailoring*, also die erweiterte Anpassbarkeit des Modells – und wird für IT-Projekte des Bundes als Entwicklungsstandard zwingend vorgeschrieben. Das Modell beschränkt sich allerdings nicht nur auf Softwareerstellung. Das V-Modell XT wird durch ein 842 Seiten starkes Dokument[6] im Detail beschrieben. Im Vergleich zum Wasserfallmodell wurden vor allem die Testphasen dabei deutlich hervorgehoben, um qualitativ hochwertige Software zu erzeugen. So wird jeder zu entwerfenden Tätigkeit eine testende gegenübergestellt (siehe Abbildung 2.2). Das Modell teilt die Qualitätssicherung in zwei Stufen auf: zum einen in die *Verifikation*, welche überprüft, ob ein zu den Anforderungen passendes, korrektes System entwickelt wird, und zum anderen in die *Validierung*, welche sicherstellt, dass ein richtiges System entwickelt wird.

Das V-Modell XT umfasst mehrere Submodule: Dies sind die Systemerstellung, die Qualitätssicherung, das Konfigurationsmanagement und das Projektmanagement. Bei der Entwicklung von Software können Systeme im V-Modell in einzelne Segmente unterteilt werden, die wiederum Software- und Hardware-Einheiten enthalten können. Darunter angeordnet sind ggf. Software-Komponenten und auf unterster Ebene Software-Module. Ein solches IT-Modell kann im Rahmen des V-Modells beliebig komplex werden, wobei jegliche Aktivität innerhalb des Prozesses durch spezielle Rollen ausgeführt werden darf. Das V-Modell als Standardverfahren des Bundes ist, wie

[6] Weitere Informationen zum V-Modell XT unter : `http://www.v-modell-xt.de/`. Das Dokument ist verfügbar unter der URL: `http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.3/V-Modell-XT-Gesamt.pdf`



Quelle: angelehnt an [Bal01].

Abbildung 2.2: Das V-Modell.

man an dieser Stelle schon anhand der kurzen Einführung erahnen kann, sehr detailliert beschrieben und durchorganisiert.

Die Vorteile des V-Modells sieht Balzert zunächst in der integrierten Darstellung der vier zuvor genannten Submodule. Auch die standardisierte Abwicklung und die besonders gute Eignung für große Projekte – insbesondere für das Erstellen von eingebetteten Systemen – wird hier angeführt. Dem entgegen stehen jedoch einige Nachteile (vgl. [Bal01]):

- Das für eingebettete Systeme sinnvolle Vorgehensmodell wird unkritisch auf andere Systemarten angewendet.
- Für kleine und mittlere Software-Projekte führt das V-Modell schnell zu einem hohen Maß an unnötiger Bürokratie.
- Ohne geeignete Software-Werkzeuge ist das V-Modell nicht handhabbar.
- Die 25 vorgesehenen Rollen sind für ein Softwareprojekt definitiv zu viel und unrealistisch.

Zusätzlich ist der Ablauf des Modells noch immer streng sequenziell, wie bei dem Wasserfallmodell, und erfolgt in aufeinanderfolgenden Phasen, so dass Rücksprünge zu vorherigen Phasen schwierig sind.

2.1.3 eXtreme Programming

Das Vorgehensmodell XP wurde im Jahre 2000 von Beck entwickelt (vgl. [Bec04]). Es handelt sich hierbei um ein agiles Vorgehensmodell, welches das Zusammenspiel von bereits bekannten agilen Praktiken beschreibt und zusammenfasst. Das XP baut im Kern auf fünf Werten auf (vgl. [BW08]):

1. **Kommunikation**: Im Gegensatz zu Dokumenten-getriebenen Vorgehensmodellen steht in agilen Methoden die Kommunikation zwischen den Projektbeteiligten klar im Fokus. Das Ziel ist, dass Bedürfnisse zwischen Auftraggeber und Auftragnehmer ebenso wie zwischen Projektleiter und Programmierer deutlich ausgesprochen werden, um eine möglichst optimale Lösung zu erhalten.

2. **Einfachheit**: Softwareerstellung ist niemals wirklich einfach; dies ist durch die implizite Komplexität von Software gegeben. Doch innerhalb agiler Softwaremethoden wird versucht, alle Faktoren zu beseitigen, die ein Software-Projekt komplizierter gestaltet, als es ohnehin schon ist. Dieser Wert wirkt sich primär auf die Ebenen *Technik* (nur das implementieren, was benötigt wird), *Organisation* (wenige Rollen und Überbau im Projekt) und *Methodik* (verständliches Vorgehen und einfache Standards) innerhalb des Projektes aus.

3. **Rückkopplung**: Die Rückkopplung steht für die Einbindung des Kunden in das agile Vorgehen. Anders als bei traditionellen Vorgehensmodellen wird der Kunde während der gesamten Laufzeit des Projektes für fachliche Fragen herangezogen. Ebenso werden die Kundenwünsche kontinuierlich während des Wachsens des Systems über Rückkopplung aufgenommen und möglichst direkt umgesetzt.

4. **Mut**: Dieser Wert mutet etwas merkwürdig an, ist aber durchaus berechtigt. Denn bei der Anwendung von agilen Methoden stößt man häufig auf Skepsis und Widerstände gegenüber den Praktiken. Hier ist ein mutiger Umgang mit den Skeptikern und ein offenes Ansprechen von Reibungspunkten gefordert. So muss dem Kunden beispielsweise erklärt werden, dass seine dauernde, beratende Mitwirkung in der Implementierungsphase zwar zeitintensiv ist, die Planungsphase zu Beginn des Projektes aber verkürzt.

5. **Respekt**: Um offen und ehrlich miteinander reden zu können, was der Wert Kommunikation fordert, müssen sich alle Projektbeteiligten gegenseitig respektieren. Nur durch ein faires Miteinander können Probleme und Missstände konstruktiv und ohne Angst besprochen werden.

Neben den fünf Werten basiert XP auf 14 Prinzipien, die das Wesen dieser agilen Vorgehensweise beschreiben. Diese Werte und Prinzipien sind in Abbildung 2.3 dargestellt.

Abbildung 2.3: Die Werte und Prinzipien des XP.

Um diese Werte und Prinzipien im Softwareerstellungsprozess umzusetzen, existieren Praktiken („Best Practices"), die sich in Primär- und Folgepraktiken einteilen lassen. Die Primärpraktiken sind hierbei die wichtigen, umzusetzenden Prozesse, die sich später durch Folgepraktiken ergänzen oder erweitern lassen. Eine Übersicht über die Primärpraktiken ist in Abbildung 2.4 dargestellt. Da eine vollständige Erläuterung aller Praktiken an dieser Stelle nicht zielführend ist, werden die wichtigsten Eckpfeiler hervorgehoben und erläutert.[7]

Die Anforderungen an die Software werden in informellen **Geschichten** (use stories) niedergeschrieben, im besten Fall vom Kunden direkt. Diese Art der Anforderungsaufnahme ersetzt das explizite Definieren von Lasten- und Pflichtenheft und die damit sehr umfangreiche Analyse- und Designphase zu Beginn eines Softwareprojekts. Um diese Geschichten in der Implementierungsphase umzusetzen, teilen sich zwei Mitarbeiter einen Arbeitsplatz und

[7] Erläuterungen aller Praktiken finden sich in [BW08].

Programmieren in Paaren. Durch zwei involvierte Personen, die sich ständig gegenseitig kontrollieren, soll die Qualität der Software steigen und das Entwurfskonzept ständig überprüft und vereinfacht werden. Ein vor allem im XP wichtiger Ansatz ist die **testgetriebene Entwicklung**. Hierbei werden für jede Geschichte zunächst etliche Testfälle programmiert, bevor die Implementierung der eigentlichen Geschäftslogik erfolgt. Dabei wird so lange entwickelt, bis alle Testfälle ohne Fehler durchlaufen werden. Der **inkrementelle Entwurf** der Software führt dazu, dass die Software schrittweise entlang der Anforderungen implementiert wird. Hier gilt die Prämisse, dass keine Funktionalität auf Verdacht programmiert wird, sondern nur aktuell geforderte. Damit soll ein Aufblähen des Codes durch solche Programmteile vermieden werden, die ggf. später gar nicht zum Einsatz kommen[8]. Der letzte hier zu erwähnende Punkt betrifft die **kontinuierliche Integration** (continuous integration), bei der jeder Codeblock, der funktioniert und getestet ist, sofort in die gemeinsame Codebasis überführt wird. Dies geschieht bestenfalls mehrmals am Tag, spätestens jedoch nach Beendigung der Programmierung an einer Geschichte. So lassen sich frühzeitig Integrationsprobleme zwischen verschiedenen Code-Teilen bemerken und auflösen.

Quelle: angelehnt an [BW08].

Abbildung 2.4: Agile Praktiken bei XP.

[8]Dieses Konzept ist in der agilen Programmierung als *You ain't gonna need it* (YAGNI) bekannt.

Auch das XP birgt Gefahren, die den Erfolg eines Projektes beeinflussen können. Kritiker bemängeln vor allem oft die unrealistischen oder zumindest teils naiven Annahmen des XP (vgl. [McB02, BT03]): beispielsweise die hohe Einbindung des Kunden für die Konzeption jeder Geschichte, wenn diese weiter verfeinert werden muss, der Mangel an Dokumentation des Projekts und auch der Nachteil des Behebens von Fehlern und/oder Designänderungen erst spät im Projekt.

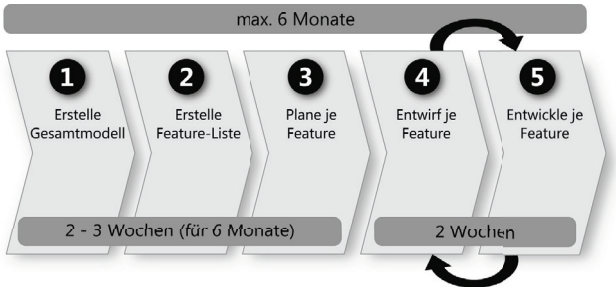
2.1.4 Feature Driven Development

Das FDD ist ein Vorgehensmodell, welches agile Methoden mit klassischen Vorgehensmodellen wie beispielsweise dem Wasserfallmodell vereint, um möglichst das „Beste aus beiden Welten" zu vereinen (vgl. [PFP02]). Der wichtigste Unterschied zu den Mitstreitern im agilen Bereich ist die Existenz einer Planungsphase des Systems, bevor die Implementierung beginnt. Im Gegensatz zu manchen sequenziellen, etablierten Vorgehensmodellen lässt sich FDD aber auf wenigen Seiten dokumentieren und ist damit ideal als leichtgewichtiges, unkompliziertes Vorgehensmodell.[9] FDD ist normalerweise für Projekte mit einer Laufzeit von bis zu 6 Monaten ausgelegt (bei längeren Projekten werden die Phasen wieder von vorne begonnen) und geht von einer 2-3 Wochen umfassenden Planungsphase aus. Man unterscheidet daher die Planungs- und die Umsetzungsphase, die im Folgenden erläutert werden (siehe Abbildung 2.5):

Planungsphase

In der ersten Phase des FDD wird ein Gesamtmodell des IT-Systems erarbeitet. Dabei wird auf oberster Ebene begonnen und die einzelnen Domänen werden in Form von UML-Klassendiagrammen fachlich dokumentiert und beschrieben. In dieser Phase sind Domänenexperten (meist der Kunde), Chefprogrammierer und Chefarchitekten anwesend. Hierbei können bestehende Dokumentationen, wie Use-Case-Modelle, Daten- und Objektmodelle und vorhandene Anforderungen aus einem Pflichtenheft, eingebracht werden. In kleinen Gruppen (von 3 Personen) werden die vom Domänenexperten genannten Domä-

[9] Die 10-seitige Einführung in FDD von Jeff De Luca ist unter `http://www.nebulon.com/articles/fdd/latestprocesses.html` zu finden.



Quelle: angelehnt an [BW08].

Abbildung 2.5: Phasen des Feature Driven Development.

nen weiter verfeinert und später im Team besprochen, um dann ein weiteres Mal verfeinert zu werden. Die hierbei entstehenden Modelle können mit Notizen annotiert werden, um komplexe Modelle oder Modellierungsalternativen zu dokumentieren. Am Schluss dieses Schrittes liegen Klassendiagramme vor, die die Klassen in der Domäne, deren Abhängigkeiten und vorhandene Einschränkungen und Bedingungen dokumentieren. Bereits an dieser Stelle identifizierte Methoden und Attribute werden an die Klassen geschrieben.

Innerhalb der zweiten Phase folgt die fachliche Dekomposition der aus der ersten Phase resultierenden Klassenmodelle, um alle Funktionalitäten (Features) des IT-Systems zu bestimmen. Die Dekomposition unterteilt hierbei in die Bereiche: *Domäne* → *Fachgebiet* → *Geschäftsaktivität* → *Schritte* (≈ *Features*). Hierbei sollen von den Teilnehmern der Planungsphase Funktionalitäten in der Form `<Aktion><Ergebnis><Objekt>` festgeschrieben werden. Beispielsweise: *„Berechne den Umsatz im Januar durch den Verkauf von i7-Core-Prozessoren"*. Die Granularität eines Features sollte der Faustregel entsprechen, dass kein Feature mehr als zwei Wochen Bearbeitungszeit für die Detailplanung und Implementierung benötigt. Sollte ein Feature nach Aufwandsschätzung mehr Zeit benötigen, ist es weiter zu unterteilen. Zum Abschluss der Phase liegt eine Liste von Fachgebieten vor, in denen Geschäftsaktivitäten und die zur Erfüllung der Aktivität notwendigen Schritte beschrieben werden.

Die dritte Phase stellt die Planung der auf der Feature-Liste enthaltenen Elemente nun in ihrem Ablauf und der Zuordnung zu Programmierern oder Teams auf. Hierbei ist es notwendig, Abhängigkeiten zwischen den Features

zu beachten, da ggf. Services oder Programmlogik auf einem anderen Feature fachlich aufbauen und daher erst anschließend ausgeführt werden können. Für jedes Feature bzw. die Geschäftsaktivität wird ein Fertigstellungsdatum festgelegt, welches sich auf den Monat und das Jahr beschränkt. In der gesamten Ablaufplanung werden die folgenden Aspekte beachtet:

- Abhängigkeiten zwischen den Features beachten (u. a. durch die involvierten Klassen zu erkennen).
- Verteilung der Arbeitsauslastung auf die Programmierer optimieren.
- Komplexität der Implementierung eines Features beachten.
- Risikobehaftete oder komplexe Geschäftsaktivitäten an den Anfang setzen.
- Angesetzte/geplante Meilensteine, Beta-Versionen, Kontrollpunkte und die dafür benötigten Features in Einklang bringen.

Diese Phase wird von Projekt-, Entwicklungsmanagern und den Chefprogrammierern ohne das Hinzuziehen von Domänenexperten durchgeführt. Zum Abschluss dieser Phase liegt eine Entwicklungsplanung des umzusetzenden IT-Systems vor, welches die verantwortlichen Programmierer je Geschäftsaktivität, die jeweiligen Fertigstellungszeitpunkte der Geschäftsaktivitäten sowie, daraus abgeleitet, die Fertigstellung der Implementierung eines Fachgebiets enthält.

Umsetzungsphase

Innerhalb der Umsetzungsphase des FDD werden zwei Arbeitsschritte iterativ durchlaufen. Im ersten Teil wird ein zu programmierendes Feature vom zuständigen (Chef-) Programmierer auf ein Team verteilt und mit einem Domänenexperten erörtert. Dabei werden alle verfügbaren Dokumente zum vorliegenden Feature (Memos, Designvorschläge, System- und Schnittstellenspezifikationen, etc.) gelesen. Anschließend werden Ablaufdiagramme (z. B. in Form von Sequenz- oder Aktivitätsdiagrammen) für das Problem erstellt und diskutiert. Zum Abschluss der Phase liegt eine Beschreibung des Features, zusätzliche Dokumente, Ablaufdiagramme, Objektmodelle und ein Plan für die Durchführung der einzelnen Aufgaben zur Fertigstellung des Features im Team vor.

Nachdem der Entwurf eines Features durchgeführt wurde, kann dieses implementiert werden. Hier werden üblicherweise Testfälle programmiert, die garantieren sollen, dass ein Feature, für sich genommen, funktioniert. Nach der Implementierung und der Überprüfung des Codes durch einen Chefprogrammierer kann der Code in die laufende Version des IT-Systems eingespielt werden und das zugeordnete Team kann sich um das nächste Feature kümmern. Während der gesamten Umsetzungsphase werden die Fortschritte des Projekts mittels *Feature Burndown*- und *Parking Lot*-Diagrammen dokumentiert. Ein Feature Burndown-Diagramm ermöglicht eine schnelle Übersicht über die bereits erledigten Aufwandspunkte eines Projektes und zeigt an, ob man sich noch auf der nach Aufwand prognostizierten Zeitlinie befindet. Ein Parking Lot-Diagramm wird für jede Gruppe von Features (*Feature Set*) erzeugt und fasst die wesentlichen Daten zusammen. In Kapitel 5 werden diese beiden Diagrammtypen noch im Detail vorgestellt.

Vor- und Nachteile des Feature Driven Development

Nach dieser kurzen Einführung zu FDD sollen die Vorteile eines solchen Vorgehensmodells stichpunktartig genannt werden:

- Eine Projektplanung (und somit ebenfalls eine frühe Kostenabschätzung aufgrund der Komplexität der Features) liegt bereits nach maximal 3 Wochen vor. So lässt sich hier bereits eine Entscheidung zur Durchführung oder den Abbruch des Projektes fällen (mit relativ geringen Kosten für die Planung im Vergleich zu traditionellen Vorgehen).

- Eine Aufwandsschätzung wird hierbei auf Feature-Ebene von erfahrenen Programmierern und ggf. auch Domänenexperten vorgenommen, was die Annahme nahelegt, dass die Ergebnisse dadurch relativ genau ausfallen. Wie in jedem Projekt gilt, dass eine solche Schätzung zu Beginn eines Projektes unabhängig vom gewählten Vorgehensmodell immer nur eine grobe Schätzung darstellt.

- Das Vorgehensmodell FDD ist auf wenigen Seiten beschreibbar und daher wenig komplex. So ist eine schnelle Einarbeitung in das Vorgehensmodell möglich.

- Im Gegensatz zu manchen traditionellen Methoden lässt sich mittels FDD der „Big Bang" in der Einführungsphase vermeiden. Es kann hier-

bei eine schrittweise Einführung der Software-Komponenten erfolgen. Ein modulares IT-System, wie ein verteiltes System, ist dafür bestens geeignet.

- Durch FDD ist eine schnelle Reaktion auf Änderungen der Anforderungen an das System möglich, da nicht alle Details feststehen und geplant sind.

- Details einzelner Geschäftsaktivitäten (genaue Prozessabläufe, Implementierungsdetails, Rechtestrukturen) werden erst entworfen, wenn sie gebraucht werden. Dies ist gerade dann sinnvoll, wenn sich die Anforderungen im Verlauf des Projekts ändern können.

- Die häufigen Release-Zyklen führen zu hoher Kundeneinbindung in einem frühen Stadium. So lassen sich schon zu Beginn des Releases diverse Probleme erkennen und beseitigen. Außerdem liegt dadurch zu jeder Zeit ein lauffähiges System vor.

- Eine „klassische" Planungsphase mit der Domänen-Dekomposition und dem Planen des „Big Pictures" der IT-Architektur findet trotz agilen Verständnisses am Anfang des Projektes statt.

- Generell ist FDD unabhängig von spezifischen Modellierungssprachen (auch wenn hier meist von UML-Modellierung ausgegangen wird). Es lassen sich die wichtigsten Modellarten, wie Objekt-, Aktivitäts- oder Sequenzdiagramme durchaus mit Petri-Netzen, BPMN-Modellen oder auch ereignisgesteuerten Prozessketten (EPK) realisieren. Wichtig ist allerdings, dass alle Beteiligten die Werkzeuge beherrschen und die darin enthaltenen Informationen verwerten können.

- Der Projektfortschritt ist beispielsweise durch die Verwendung von *Feature-Burndown*- und *Parking-Lot*-Diagrammen sehr genau zu verfolgen und liefert auch der Projektleitung und dem Management wochenaktuelle, leicht zu verstehende Planungsstände.

- Die Rollenaufteilung in FDD liegt sehr nahe bei real existierenden Organisationsstrukturen in Unternehmen und erleichtert somit die Einführung des Verfahrens.

- Aufgrund der klaren Rollentrennung von Programmierer über *Class Owner* bis hin zu Chefprogrammierer skaliert das FDD auch für größere Softwareprojekte.

Neben den vielen Vorteilen von FDD lassen sich aber auch klare Nachteile eines solchen Vorgehens identifizieren:

- Es wird keine vollständige Planung und Dokumentation des IT-Systems zu Beginn der Projektlaufzeit durchgeführt.

- Der Kunde bekommt eine zentrale und aufwendige Rolle in der Entwicklung des Systems. Im Gegensatz zu klassischen Verfahren, bei dem der Kunde das Pflichtenheft lediglich abnehmen und bestätigen muss, und daraufhin auf das erste fertige Release wartet, ist bei FDD eine dauernde Projektbetreuung notwendig. Nicht nur in der Anfangsphase (2-3 Wochen), sondern auch bei jeder Planung eines Features, muss ein Domänenexperte (vom Kunden) die Planung betreuen und die Details der Geschäftsaktivität erläutern.

- Das Vorgehensmodell erfordert die Disziplin der Mitarbeiter, da es nicht jedes Detail exakt vorschreibt. Daher muss durch den Projektleiter und die Chefprogrammierer die Einhaltung im Team festgelegter Praktiken zur Beschreibung der Arbeit und der Dokumentation von Code und Features vorgelebt und durchgesetzt werden.

- Einzubindende und selbst zu erstellende Services innerhalb von Prozessen (Features) werden teilweise erst in der Feature-Planungsphase bekannt.

2.2 Verteilte Systeme und das World Wide Web

Eine der Grundlagen für global erfolgreich agierende Unternehmen ist in der heutigen Zeit ein sinnvoller Einsatz von IKT-Systemen, die die Erledigung des Kerngeschäfts des Unternehmens effizient unterstützen oder gar erst ermöglichen. Hierbei kommt der Umstand zum Tragen, dass die Menge an Daten und Dokumenten jeglicher Art ohne digitale Unterstützung kaum mehr handhabbar wären. Daten und deren Verarbeitung fallen aber zumeist nicht an einem

Ort an, sondern sind an diversen Stellen innerhalb und außerhalb eines Unternehmens zu finden. Aus diesem Grund ist die Verbindung von Ressourcen und deren ständige Verfügbarkeit an den Orten, an denen sie gebraucht werden, seien es nun Computerprogramme, Datenbanken oder Dokumentensammlungen, ein wichtiger Aspekt für den Unternehmenserfolg. Ein untereinander verbundenes Netzwerk von IT-Ressourcen und IKT-Systemen wird in der Informatik als *verteiltes System* bezeichnet, welches in verschiedenen Ausprägungen ausgestaltet sein kann. Im Folgenden werden die gängigsten Umsetzungen vorgestellt und erläutert. Im Anschluss daran wird das größte verteilte System der Welt, das Internet, und die wichtigsten darin verwendeten Standards vorgestellt, da sich die vorliegende Arbeit konkret mit Systemarchitekturen des Internets auseinandersetzt.

2.2.1 Verteilte Systeme

Service-, Ressourcen- und Web-orientierte Architekturen basieren im Kern auf dem grundlegenden Gedanken, dass bestimmte Funktionalitäten einer Software als Komponenten vorhanden sind, die jeweils einen bestimmten funktionalen Bestandteil der Software kapseln und diese Funktionsbausteine anderen IT-Systemen zur Verfügung stellen. Die dafür benötigte Basis bezeichnet man als verteiltes System. Diese werden mit dem Ziel der gemeinsamen Ressourcen-Nutzung erstellt. Tanenbaum beschreibt ein verteiltes System als eine Menge an miteinander verbundenen Computern, die der Nutzer als ein System wahrnimmt (vgl. [TS07]). Wobei als Ressourcen eines verteilten Systems im weiteren Sinne auch beispielsweise Drucker und weitere Peripherie-Geräte verstanden werden können. Als bei dem Aufbau eines verteilten Systems zu berücksichtigende Aspekte nennt Coulouris die Heterogenität, Offenheit, Sicherheit, Skalierbarkeit, Fehlerverarbeitung, Nebenläufigkeit sowie Transparenz (vgl. [CDK02]).

Es lässt sich hierbei eine grundlegende Unterscheidung der Integrationsstile feststellen, die die Netzwerkarchitektur eines verteilten Systems und die Art der Verbindung zwischen einzelnen Systemkomponenten beschreiben (vgl. [HW03]):

- Nutzen einer gemeinsamen Datenbank (*Shared Database*)
- Aufrufen entfernter Funktionen (*Remote Procedure Calls*)

- Verwendung eines zentralen Nachrichtenkanals (*Message Bus*)
- Direktes Verschicken von Dateien (*File Transfer*)

Diese vier Integrationsstile werden im Folgenden genauer erläutert.

Gemeinsame Datenbank

Ein verteiltes System lässt sich unter Verwendung einer gemeinsamen Datenbank realisieren. Bei diesem Integrationsstil greifen die beteiligten Komponenten auf eine gemeinsame Datenbank zu. Dies können Server, Endnutzer-PCs oder auch mobile Geräte sein. In dieser Datenbank werden unter anderem gemeinsam genutzte Datenstrukturen gespeichert und verwaltet. Wie in Abbildung 2.6 gezeigt, sind die Komponenten (hier als Komponenten A bis C bezeichnet), nicht direkt untereinander verbunden, sondern besitzen eine bidirektionale Verbindung zur Datenbank. Gemeinsame Inhalte oder Austausch zwischen den einzelnen Systemkomponenten erfolgt daher durch den Umweg über die Datenbank. Der Vorteil einer solchen Architektur besteht in der Unabhängigkeit der Systemkomponenten voneinander, da diese nicht direkt miteinander kommunizieren müssen. So lassen sich die unterschiedlichsten Systemarten miteinander kombinieren. Dennoch muss ein Datenbankkonnektor bestehen, der mit dem verwendeten Datenbankmanagementsystem (DBMS) arbeiten kann. Ein Nachteil dieses Stils ist jedoch in der Realisierung von Funktionsaufrufen zu sehen. Möchte Komponente A einen Funktionsaufruf von Komponente B starten, so lässt sich dies zwar auf einigen Wegen[10] implementieren, ist aber nicht der eigentliche Verwendungszweck eines DBMS. Es ist also keine direkte synchrone oder asynchrone Kommunikation zwischen Komponenten des verteilten Systems möglich. Die Größenordnung der Zeitverzögerung bei solchen Aufrufen hängt jeweils von der Konfiguration der einzelnen Systeme ab. Man muss sich hierbei aber im Klaren sein, dass bei vielen Komponenten innerhalb eines Systems die Datenbank den Flaschenhals bildet und man dementsprechend für den skalierbaren Einsatz in einem Netzwerk über eine Architektur mit mehreren Datenbanken nachdenken muss. Dafür bieten sich Master-Slave-Lösungen an, die Replikationen der Datenbanken im DBMS

[10] Bspw. durch das Ausführen eines Triggers, der einen Socket-Aufruf der gewünschten Komponente ausführt, oder das Abfragen der Datenbank auf eingegangene Nachrichten für eine Komponente in bestimmten Intervallen.

ermöglichen, um beispielsweise Lesezugriffe auf mehrere Instanzen der Datenbank zu verteilen.

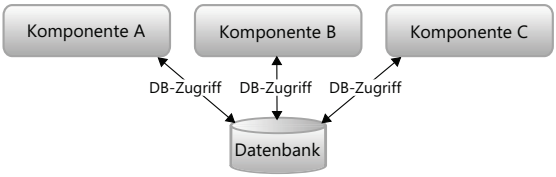


Abbildung 2.6: Nutzung einer gemeinsamen Datenbank.

Entfernte Funktionsaufrufe

Die Integration durch entfernte Funktionsaufrufe – auch RPCs genannt – sieht eine direkte Verbindung jeder Komponente miteinander vor. Diese Methode des Programmierens wurde 1984 von [BN84] vorgestellt. Dabei definiert jede Komponente eine Anzahl von Schnittstellen, die für den Aufruf bereitgestellt werden. Ein Aufruf wird initiiert durch eine zielgerichtete Anfrage einer Komponente (*request*), auf die die angefragte Komponente antwortet (*response*). Ob hierbei eine synchrone oder asynchrone Benachrichtigung erfolgt, ist erst einmal zweitrangig. Diese direkte Verbindung erfordert aber, dass innerhalb des verteilten Systems zumindest ein geeignetes Kommunikationsprotokoll verwendet wird, welches von allen Komponenten des Systems verstanden wird. Dies ist vor allem bei der Einbindung von Alt-Systemen schwierig, die mitunter proprietäre Protokolle benötigen, und somit nicht ohne Weiteres durch alle anderen Komponenten angesprochen werden können. Aus diesem Grund wurden beispielsweise für das WWW das Transportprotokoll HTTP oder für SOA das Nachrichtenprotokoll SOAP entwickelt. Bei komplexen Systemen bleibt es aber nicht aus, auch die Transformation von Nachrichtenformaten und -protokollen zu berücksichtigen, um diverse Komponenten miteinander zu verbinden. Im Gegensatz zum Einsatz eines zentralen Punktes (wie bei der gemeinsamen Datenbank) folgt dieser Integrationsstil der Topologie eines vollvermaschten Netzwerks, wie in Abbildung 2.7 zu sehen ist. Dabei wird jede Komponente mit jeder anderen verbunden, was die Komplexität der Vernetzung drastisch erhöht. Die meisten moderneren IT-Systeme (und natürlich

auch das Internet) bilden entweder diesen Stil oder eine Art Bus-System ab, welches im nächsten Abschnitt erläutert wird.

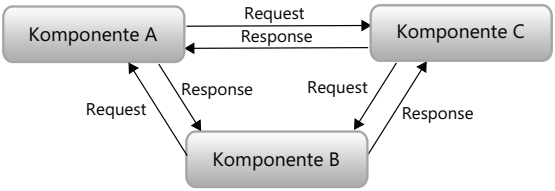


Abbildung 2.7: Nutzung entfernter Funktionsaufrufe.

Zentraler Nachrichtenkanal

Das Implementieren eines verteilten Systems unter Verwendung des Konzepts eines zentralen Nachrichtenkanals ist spätestens seit der Entwicklung von CORBA ein gängiges Vorgehen. Diese Methode wird auch als Nachrichtenorientierte Middleware bezeichnet (vgl. [KBS07]). Die Verwendung eines zentralen Bus-Systems ist ebenfalls in den meisten SOA wieder zu finden, dort meist als *Enterprise Service Bus* (ESB) bezeichnet. Alle Komponenten des Systems werden hierbei direkt an den Nachrichtenkanal angeschlossen und schicken und empfangen Nachrichten über diesen. Das Zusenden der Nachrichten an den Empfänger übernimmt hierbei das Bussystem. Oft werden auch weitere Funktionen, wie beispielsweise Fehlerbehandlung, Nachrichtenwarteschlangen und Überwachung von Komponenten durch das Bussystem angeboten. Die Besonderheit hierbei ist, dass der Nachrichtenkanal jede Nachricht in ein normalisiertes Format umwandeln kann und damit eine Weiterverarbeitung in beliebige andere Formate ermöglicht. So muss man für das Anbinden einer Komponente, welche ein spezielles Daten- oder Protokollformat erfordert, lediglich einmal eine Schnittstelle für den Bus zu programmieren (im Fall eines ESB auch als *Binding Component* bezeichnet). Damit lassen sich alle ähnlichen Komponenten, die dasselbe Protokoll verwenden, ohne zusätzlichen Aufwand an den Bus anbinden. Dies ist im Gegensatz zu einem vollvermaschten Netz gerade in einem sehr heterogenen Netzwerk von großem Nutzen, da man nicht für jede Komponente eine Schnittstelle zu jedem anderen Gegenpart implementieren muss. Die Umwandlung des ggf. proprietären

Formats in eine normalisierte Nachricht wird einmalig implementiert und dem Bus zur Verfügung gestellt. Demgegenüber steht der Nachteil, wie bei jeder zentralisierten Netzwerk-Architektur, dass bei Ausfall oder Störung des Bus-Systems das gesamte verteilte System betroffen ist. Aber auch dafür gibt es entsprechende Ausfallsicherheitsstrategien, die beispielsweise im redundanten Betrieb von Bussen besteht, oder auch im Partitionieren von Bereichen auf mehrere Bussysteme (teilweise als *Federated Service Bus Infrastructure* bezeichnet, vgl. [KAB+04]). In Abbildung 2.8 ist die grundlegende Struktur dieses Integrationsstils dargestellt.

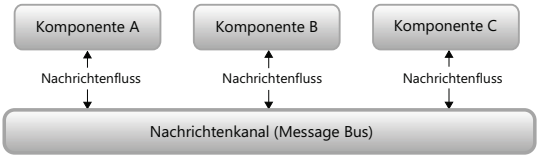


Abbildung 2.8: Nutzung eines zentralen Nachrichtenkanals.

Zentraler Dateispeicher

Das Verschicken von Dateien ist nicht mit der Verwendung von entfernten Funktionsaufrufen gleichzusetzen, bei denen ebenfalls Dateien mitgeschickt werden können. Vielmehr ist bei diesem Integrationsstil ein gemeinsamer Ordner (*shared folder*) auf einem zentral zugänglichen Netzlaufwerk vorgesehen, auf dem die Komponenten des verteilten Systems Dateien ablegen und lesen können. Dieser Integrationsstil ähnelt der Topologie bei der Verwendung einer gemeinsamen Datenbank. Die Verwendung einer solchen Netzwerkarchitektur ist sinnvoll, wenn die angeschlossenen Komponenten große Datenmengen schreiben und lesen. Die meisten Nachrichtenprotokolle erlauben zwar das Versenden von Dateien als Anhang bzw. Nutzlast einer Nachricht (*workload*), sobald es sich aber um größere Datenmengen handelt – beispielsweise Datenbank-Backups oder abzuarbeitende Jobs für automatische Workflows – ist es sinnvoll, auf dafür vorgesehene Transportprotokolle zurückzugreifen. Hier ist beispielsweise eher *File Transfer Protocol* (FTP) als HTTP zum Versenden von Dateien zu präferieren. In Abbildung 2.9 ist das Datei-basierte verteilte System dargestellt.

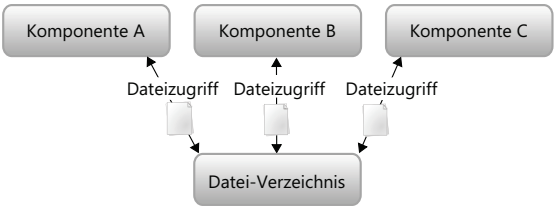


Abbildung 2.9: Nutzung eines gemeinsamen Datei-Verzeichnisses.

2.2.2 Standards des World Wide Web

Das World Wide Web ist zusammen mit E-Mail einer der bekanntesten Dienste des Internets. Neben den Standards *Hypertext Markup Language* (HTML) als Dokumentenbeschreibungssprache und *Uniform Resource Locator* (URL) für die Angabe eindeutiger Adressen basiert es auf der Erfindung des Transportprotokolls HTTP. Obwohl die Begriffe Internet und WWW oft synonym verwendet werden, ist das WWW nur als Teilmenge des Internets zu verstehen. Im Folgenden wird der Begriff Internet für die Gesamtheit des weltweit verteilten Netzwerks verwendet. In Fällen, in denen explizit der HTTP-basierte Teil gemeint ist, wird der Begriff WWW benutzt. Ähnlich verhält es sich mit der Begrifflichkeit URL und *Uniform Resource Identification* (URI): Während URI einen Identifikator für eine allgemeine Adresse des Internets bezeichnet, definieren URLs eine Teilmenge davon und geben den genauen Ort der Ressource an. Im Folgenden wird hier jedoch der gebräuchlichere Begriff URL verwendet. Darüber hinaus sind auch weitere Standards, gerade im Bereich der sicheren Datenverbindungen, in einer auf dem Internet basierenden Softwarearchitektur relevant.

Hypertext Transfer Protocol

Das *Hypertext-Transfer-Protokoll*[11] (HTTP) ist ein Protokoll zur Übertragung von Daten über ein Netzwerk (vgl. [BLFF96, FGM+99]). Das gesamte Internet basiert auf HTTP und nutzt dabei *Transmission Control Protocol* (TCP)-Verbindungen. Das TCP (genauer: TCP/IP) ist hierbei für die Verbindung von Servern und PCs verantwortlich, deren Details für den Programmierer und erst

[11] HTTP ist als *RFC 2616* unter `http://www.ietf.org/rfc/rfc2616.txt` zu finden.

recht für den Nutzer des Internets nicht sichtbar sind. Die Software-Schichten, die für Übertragungen von Daten im Internet notwendig sind, lassen sich als *Open Systems Interconnection* (OSI)-Modell darstellen. Besser bekannt als das ISO/OSI-Modell (vgl. [Zim80]). Die sieben Schichten sind in Abbildung 2.10 dargestellt. Die Schichten 1 bis 4 werden vom Betriebssystem bereitgestellt und daher im Folgenden als gegeben angesehen. Interessant für den Nutzer und Entwickler von verteilten Systemen sind daher die Schichten 5 bis 7, die sich als Anwendungsschicht zusammenfassen lassen. Auf dieser Ebene sind Protokolle wie HTTP für das Übertragen von Web-Inhalten, FTP für das Übertragen von Dateien oder auch *Simple Mail Transfer Protocol* (SMTP) für das Senden von E-Mails zu finden.

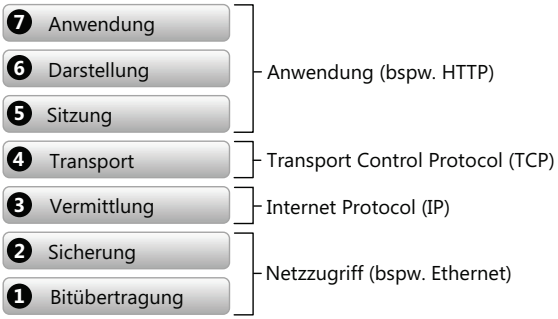


Abbildung 2.10: ISO/OSI-Modell mit Fokus auf verteilte Systeme.

HTTP ist ein zustandsloses Protokoll, welches sich aus einem Nachrichtenkopf (*message header*) und einem Nachrichtenkörper (*message body*) zusammensetzt. Zustandslos bedeutet hierbei, dass alle für eine Nachricht relevanten Daten in der Nachricht mitgesendet werden müssen. Im Nachrichtenkopf werden die HTTP-Methode und die Zieladresse angegeben. Die Zieladresse wird in Form einer *Internet Protocol* (IP)-Adresse oder einer URL angegeben. Der Nachrichtenkörper enthält, je nach aufgerufener Methode, weitere Parameter des Aufrufs. Die hierfür verfügbaren Methoden in der Version 1.1 von HTTP sind in Tabelle 2.1 aufgelistet.

Die Verwendung von HTTP im Internet ist meist nur auf `POST` und `GET` beschränkt. Viele Browser und auch Server unterstützen in der Standardkonfiguration lediglich diese beiden Methoden des HTTP-Standards. Eine beispiel-

Tabelle 2.1: HTTP-Methoden und deren Verwendung.

HTTP-Methode	Verwendung[a]
`GET`	Die `GET`-Methode ermöglicht das Abrufen einer Ressource. Parameter, die durch `GET` übergeben werden sollen, werden durch ein Fragezeichen eingeleitet und jeder Parameter als Schlüssel-Wert-Paar – durch ein kaufmännisches Und getrennt – an die URL angefügt.
`POST`	Schickt Daten an einen Server (die angegebene URL), die beispielsweise durch Schlüssel-Wert-Paare im Nachrichtenkörper geschickt werden. Hiermit wird eine Ressource angelegt.
`PUT`	Hiermit lässt sich eine bestehende Ressource mittels mitgeschickter Datei oder Parametern überschreiben.
`DELETE`	Diese Methode löscht die angegebene Ressource vom Server.
`HEAD`	Eine Anfrage mittels `HEAD` ist ähnlich einer `GET`-Anfrage, prüft aber nur die Existenz einer Ressource und gibt den Nachrichtenkopf zurück. Der Nachrichtenkörper wird nicht verschickt.
`TRACE`	Diese Methode liefert die an den Server gestellte Anfrage direkt zurück. Dies ist beispielsweise für die Fehlersuche sinnvoll.
`OPTIONS`	Schickt man `OPTIONS` an den Server, bekommt man eine Liste aller unterstützten HTTP-Methoden zurück.
`CONNECT`	Diese Methode ist reserviert für die Nutzung von Proxyservern, die beispielsweise einen SSL-Tunnel anbieten.[b]

[a] Die Interpretation der `GET`- und `POST`-Methode durch gängige Web-Server weicht hiervon meist ab.
[b] Wird im Kontext dieser Arbeit nicht betrachtet.

hafte GET-Anfrage (*request*) ist in Listing 2.1 dargestellt. Diese wird durch die Angabe des Pfads in der zweiten Zeile und des Host-Namens in der vierten Zeile an die URL `www.domain.de/kunde/12/rechnungen` geschickt. Es wird hierbei der Parameter *parameter1* mit dem Wert *wert1* mitgeschickt.

```
1 # Methode und Pfad der URL
2 GET /kunde/12/rechnungen?parameter1=wert1 HTTP/1.1
3 # Ziel-Domain der Anfrage
4 Host: www.domain.de
```

Listing 2.1: Beispiel einer GET-Anfrage.

Eine Anfrage mittels POST-Methode sieht ähnlich aus (vgl. Listing 2.2). Hierbei werden jedoch die Parameter als Nachrichtenkörper mitgeschickt. Dazu werden die Art (Zeile 6) und Länge (Zeile 8) des Nachrichtenkörpers angegeben. Dies ist hier beispielsweise das Ergebnis aus einem Web-Formular (*application/x-www-form-urlencoded*) mit der Länge von 41 Zeichen. Die Parameter bestehen hier aus einem String, der aneinandergereiht die Parameter *aktion* und *kundenname* enthält. Binärdaten, die im Nachrichtenkörper einer POST-Nachricht versendet werden, können im Prinzip beliebig groß werden. Sollen Dateien über einen Web-Browser ins Internet geladen werden, wird daher meist die POST-Methode des HTTP-Standards verwendet.

```
1  # Methode und Pfad der URL
2  POST /kunde/12/rechnungen HTTP/1.1
3  # Ziel-Domain der Anfrage
4  Host: www.domain.de
5  # Inhalt der Anfrage
6  Content-Type: application/x-www-form-urlencoded
7  # Zeichen im Nachrichtenkörper
8  Content-Length: 41
9  # Nachrichtenkörper
10 aktion=erstelle&kundenname=Peter Schuster
```

Listing 2.2: Beispiel einer POST-Anfrage.

Der HTTP-Standard gibt in einer Antwort auf eine Anfrage im Kopfelement der Nachricht immer einen Statuscode zurück, der dem aufrufenden Programm wichtige Informationen zu der aufgerufenen Ressource zukommen lässt. Darüber hinaus kann der Nachrichtenkörper im Falle eines GET-

Tabelle 2.2: Statuscodes des HTTP.

HTTP-Statuscodes	Bedeutung
1xx	Diese Statuscodes teilen dem Aufrufer mit, dass die Bearbeitung der Anfrage noch andauert.
2xx	Durch die Rückgabe eines 200er-Status wird das erfolgreiche Abarbeiten der Anfrage mitgeteilt.
3xx	300er-Nummern teilen mit, dass eine Umleitung der Anfrage auf Serverseite notwendig war.
4xx	Sollte ein Fehler auftreten, signalisieren die 400er-Fehler, dass der Fehler in den Zuständigkeitsbereich des Clients fällt.
5xx	Diese Codes signalisieren einen Server-Fehler.

Aufrufs die Repräsentation der Ressource enthalten. Es gibt fünf verschiedene Statuscode-Arten, die jeweils durch eine 3-stellige Zahl dargestellt werden, und die weitere Subtypen besitzen (siehe Tabelle 2.2).

Es existieren über 50 Statuscodes innerhalb der fünf Familien, die an dieser Stelle nicht alle erläutert werden. Die wichtigsten sind hierbei sicherlich die Statusmeldungen **200**, **403**, **404** und **500**. Bei einer **200** (*OK*) wurde die Anfrage erfolgreich bearbeitet und die Antwort im Nachrichtenkörper übertragen. Eine **403** (*Forbidden*) wird zurückgegeben, wenn der Aufruf dieser URL nicht erlaubt ist, was beispielsweise bei geschützten Webseiten eingesetzt wird. Ein häufiger Fehlercode ist die **404** (*Not Found*), der darauf hinweist, dass die URL nicht gefunden wurde. Die **500** (*Internal Server Error*) steht schließlich für einen Server-Fehler, der nicht erwartet wurde. Eine HTTP-Antwort, die eine erfolgreiche Abarbeitung meldet und im Nachrichtenkörper eine Textrepräsentation der angefragten Ressource trägt, ist in Listing 2.3 dargestellt. In der zweiten Zeile steht die HTTP-Version sowie der Antwortcode, gefolgt vom aktuellen Datum des Sendens der Antwort (Zeile 4) und der letzten Änderung an der Ressource (Zeile 6). Die Sprache sowie die Codierung des Nachrichtenkörpers ist im Beispiel in Zeile 8 und 10 angegeben. Darauf folgt die Repräsentation der Ressource.

```
# Antwortcode des Servers
HTTP/1.1 200 OK
# Zeitstempel der Antwort
Date: Mon, 22 Jun 2010 17:11:18 GMT
# Letzte Änderung der Ressource
Last-Modified: Tue, 29 Dec 2009 11:21:40 GMT
# Sprache des Inhalts
Content-Language: de
# Codierung des Inhalts
Content-Type: text/html; charset=utf-8

Dies ist eine Repräsentation der Ressource.
...
```

Listing 2.3: Beispiel einer korrekt ausgeführten HTTP-Antwort.

Sicheres HTTP

Die Sicherheit von Datenübertragungen in einem Netzwerk ist seit Langem ein zentrales Thema der IT. Bereits in den 70er Jahren wurden grundlegende Algorithmen der Kryptografie wie der Diffie-Hellman-Schlüsselaustausch und der RSA-Algorithmus entwickelt, die noch heute in der Nachrichtenverschlüsselung Verwendung finden (vgl. [DH76, RSA78]). Ein internationaler Standard für die Sicherheit in Netzwerken wurde 1989 durch die *International Organization for Standardization* (ISO) in der Richtlinie *ISO 7498-2* festgelegt (vgl. [Int89]). Für die Sicherung des Transportwegs zwischen zwei Kommunikationspartnern, die per HTTP Daten austauschen, existiert *verschlüsseltes HTTP* (HTTPS). Dieser Standard verwendet *Transport Layer Security* (TLS) (vgl. [DA99]) und das normale HTTP. Hierbei werden die ausgetauschten Nachrichten durch ein Public-Key-Verfahren verschlüsselt, so dass ein zufällig mitlesender Dritter keine Informationen erhält. Das Verfahren, welches hier konkret entwickelt wurde, nennt sich *Handshake*-Protokoll (vgl. [Eck09]). In Abbildung 2.11 ist der Ablauf gezeigt: Zunächst stellt der Client, meist mittels Browser, eine Verbindungsanfrage an den Server. Dieser teilt dem Client mit, dass die Ressource nur über eine verschlüsselte Verbindung erreichbar ist und schickt seinen öffentlichen Schlüssel an den Client. Im dritten Schritt erzeugt der Client nun eine Zufallszahl, verschlüsselt diese asymmetrisch mit dem öffentlichen Server-Schlüssel und schickt im vierten Schritt dieses Datenpaket zum Server. Auf Server-Seite kann nun die Zufallszahl anhand des privaten Schlüssels des Ser-

vers wieder entschlüsselt werden. Nun besitzen beide Kommunikationsparteien einen durch die geheime Zufallszahl begründeten Schlüssel, mit dem alle weiteren Datenpakete auf beiden Seiten symmetrisch verschlüsselt werden. Durch dieses an sich einfache Verfahren ist nun die Verbindung von Client zu Server relativ sicher.

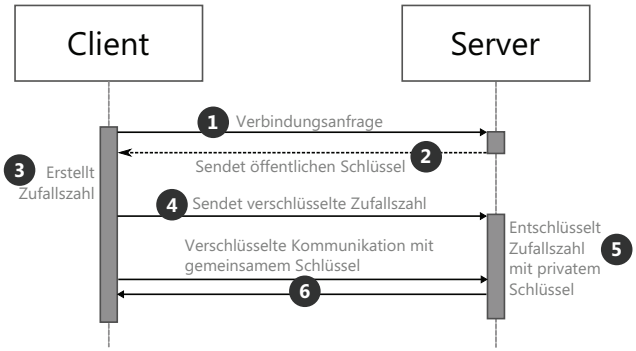


Abbildung 2.11: Das Handshake-Protokoll zum Aufbau einer sicheren HTTP-Verbindung.

Public-Key-Verschlüsselung

Es existieren zwei gängige Verschlüsselungsverfahren, die für den Aufbau sicherer Verbindungen (sei es gesicherte Webseitenaufrufe oder auch verschlüsselte Web Service-Aufrufe) verwendet werden. Dies sind die symmetrische und die asymmetrische Verschlüsselung (vgl. [BSW95]). Bei der symmetrischen Verschlüsselung existiert ein Schlüssel, der den beiden Parteien bekannt ist. Mit diesem Schlüssel ist die Ver- und Entschlüsselung einer Nachricht möglich. Bei der asymmetrischen Verschlüsselung existiert für jeden Kommunikationspartner ein Schlüsselpaar, welches aus öffentlichem und privatem Schlüssel besteht. Eine Nachricht wird hierbei durch die Verwendung des öffentlichen Schlüssels verschlüsselt. Um die Nachricht wieder zu entschlüsseln, benötigt man den privaten Schlüssel. Bei dem asymmetrischen Verfahren ist es durch die dahinter liegenden komplexen mathematischen Verfahren kaum möglich, eine Nachricht ohne den privaten Schlüssel zu nutzen. In diesem Zusammen-

hang muss auch *Hashing* genannt werden. Dies sind Einweg-Funktionen, die nur in eine Richtung funktionieren. Beim einem Hash-Wert handelt es sich um eine Art Prüfsumme für ein beliebiges Datum. Hierbei wird (normalerweise) kein Schlüssel benötigt, sondern ein bestimmtes mathematisches Verfahren auf ein Datum angewandt, um eine Prüfsumme zu errechnen. Der resultierende Wert kann beispielsweise für die Prüfung der Datenintegrität verwendet werden. Sehr verbreitete Verfahren sind in diesem Zusammenhang die Algorithmen *Message-Digest Algorithm 5* (MD5) und *Secure Hash Algorithm* (SHA1).

Zertifikate

Um standardisierte Schlüssel für die kryptografischen Verfahren zur Verfügung zu stellen, verwendet man Zertifikate. Das bekannteste Beispiel im Zusammenhang mit dem Internet ist wohl das X.509-Zertifikat (vgl. [HPFS02], aktuellste Version 3.0), welches für den Aufbau einer sogenannten *Public-Key-Infrastruktur* verwendet wird. Ein solches Zertifikat besteht aus einem öffentlichen und einem privaten Anteil. Die meisten SSL-Server-Zertifikate entsprechen dem X.509-Standard. Ein solches Zertifikat enthält u. a. die Informationen zum Aussteller des Zertifikats (*Certificate Authority* (CA)), zur Gültigkeit (Ablaufdatum) und zum Zertifikatinhaber, sowie die Daten des zur Erstellung des Schlüssels benutzten Algorithmus. Die Idee bei X.509-Zertifikaten ist, dass es eine Hierarchie an Zertifizierungsstellen gibt. Jedes Zertifikat wird von der ausstellenden CA signiert. Eine CA kann selbst durch eine höhere CA zur Ausstellung von Zertifikaten autorisiert sein. Damit lassen sich komplette hierarchische Systeme aufbauen. So werden beispielsweise X.509-Zertifikate der Westfälischen Wilhelms-Universität von der WWU selbst, dem darüber liegenden Zertifikat des Deutsches Forschungsnetz (DFN) und darüber von dem Wurzelzertifikat der Deutschen Telekom signiert (siehe Abbildung 2.12). Eine solche Kette sagt zwar nicht direkt etwas über die Vertrauenswürdigkeit des signierten Zertifikats aus, bedeutet aber in jedem Fall, dass eine autorisierte Stelle das Zertifikat ausgestellt hat.

X.509-Zertifikate bieten grundsätzlich zwei Anwendungsmöglichkeiten: Es lassen sich Nachrichten signieren und/oder verschlüsseln. Beim Signieren einer Nachricht wird für den zu signierenden Inhalt ein Prüfwert (Hashwert) mittels des privaten Schlüssels der Zertifikate erstellt. Anschließend lassen sich Nachricht und Prüfwert an eine weitere Partei übermitteln. Anhand des öffent-

Abbildung 2.12: Eine CA-Kette am Beispiel der WWU.

lichen Schlüssels kann nun der Empfänger prüfen, ob der Prüfwert zu der geschickten Nachricht passt. Auf diese Weise kann man zum einen ermitteln, ob die Nachricht auch tatsächlich von dem vermeintlichen Absender stammt und zusätzlich, ob die Nachricht auch unverändert beim Empfänger ankam. Denn nur in diesem Fall ergibt die Prüfung ein positives Ergebnis. Der zweite Fall ermöglicht (wie bereits beim SSL-Verfahren erläutert) das Ver- und Entschlüsseln von Nachrichten mittels des privaten und öffentlichen Schlüssels. Dies findet zum einen im E-Mail-Austausch als auch bei Client-Server-Kommunikation oft Verwendung.

2.3 Softwarearchitekturen

Dem Begriff Architektur wird in der Informatik häufig eine zentrale Rolle beigemessen. Er tritt dabei üblicherweise in unterschiedlichen Kombinationen auf, beispielsweise System-, Netzwerk- oder eben auch Softwarearchitekturen. Eine Definition von Softwarearchitektur aus [RH08, S. 1] lautet folgendermaßen (vgl. auch [IEE01]):

> „Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung, sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen."

Zwei grundlegende Softwarearchitekturen, SOA und ROA, die sich für den Aufbau verteilter Systeme eignen, werden im folgenden Kapitel vorgestellt und im Anschluss daran miteinander verglichen. Abschließend werden *Services* nach technologischen und fachlichen Aspekten klassifiziert.

2.3.1 Service-orientierte Architekturen

Das Konzept einer Service-orientierten Architektur ist zum ersten Mal ungefähr zur Mitte der neunziger Jahre in Wissenschaft und Praxis diskutiert worden[12] und ist seitdem Gegenstand lebhafter Diskussionen um Tragfähigkeit und Nutzen für die Unternehmens-IT. Die Eigenschaften, die eine SOA auszeichnen, sind zum einen die Verwendung von Services, die Funktionalitäten von (Dritt-)Systemen kapseln, atomar und zustandslos sind, und zum anderen der Einsatz von wohl definierten Schnittstellen, die einen Service fachlich sowie technisch beschreiben. Eine SOA soll hierbei vor allem eine hohe Wiederverwendbarkeit, eine Kosten- und Komplexitätsreduzierung und ein hohes Maß an Flexibilität erzielen. Ein wichtiger Punkt wird hierbei immer wieder hervorgehoben: SOA darf nicht nur als technisch-getriebenes Projekt, sondern vielmehr als eine Strategie gesehen werden, welche sich bei korrekter Einführung auch auf die Organisationsstruktur eines Unternehmens auswirkt (vgl. [Erl05, KBS07, Tre09]).

Der Begriff SOA wird von vielen großen Software-Herstellern oder Gremien in einer eigenen Definition zusammengefasst, darunter beispielsweise SAP, IBM, Microsoft, Oracle, OMG und viele weitere. In [MLM+06, S. 29] findet sich die folgende prägnante Definition des Standardisierungs-Gremiums *Organization for the Advancement of Structured Information Standards* (OASIS), dem einige der oben genannten Unternehmen angehören:

> „Service Oriented Architecture is a paradigm for organizing and utilizing **distributed capabilities** that may be under the control of **different ownership** domains. It provides a **uniform** means to offer, discover, interact with and use capabilities to produce **desired effects** consistent with **measurable** preconditions and expectations."

Die wichtigen Kernpunkte sind darin die verteilten Komponenten (*distributed capabilities*), die nicht unter der eigenen Kontrolle stehen müssen (*different ownership*) und dabei einen einheitlichen Zugriff (*uniform*) für die Verwendung der Komponenten bieten. Dabei sollen die gewünschten Effekte (*desired effects*) mit messbaren Vorbedingungen und Erwartungen (*preconditions and expectations*) auftreten. Obwohl das Konzept einer SOA für das Feld der IKT bereits als alt einzustufen ist, wurde erst im Oktober 2009 in Rotterdam

[12] Das Marktforschungsunternehmen GARTNER verwendete den Begriff im Jahre 1996 zum ersten Mal.

von einigen einschlägigen SOA-Spezialisten[13] ein Versuch unternommen, die unterschiedlichen Auffassungen und Ansichten der Service-Orientierung und SOA zu präzisieren. Das dabei entstandene Dokument wurde, in Anlehnung an das agile Manifest (siehe 2.1), als **SOA-Manifest** bezeichnet und definiert die grundlegenden Werte und Prinzipien einer SOA (vgl. [Jos09][14]). Die Autoren betonen, dass dieses Manifest nur als richtungsweisende Grundsätze und nicht als genaue Anleitung für SOA zu verstehen ist. Die vorgenommene Priorisierung der Werte des Konzepts verdeutlicht damit das allgemeine Verständnis einer SOA (zumindest das der Autoren). Die folgenden sechs Werte gelten unter der Prämisse, dass von den beiden genannten Aspekten der zuerst genannte (fett gedruckt) als wichtiger angesehen werden:

1. „**Business value** over technical strategy": Bei der Einführung einer SOA soll darauf geachtet werden, dass es bei dem Verfolgen einer technischen Strategie immer wichtiger ist, dass mit der Einführung einer SOA auch ein Nutzen stiftender Effekt erzielt wird.

2. „**Strategic goals** over project-specific benefits": Da SOA grundsätzlich auch als strategisches Instrument im Bereich der IKT angesehen wird, soll der Gesamtstrategie mehr Wert beigemessen werden, als dem projektspezifischen Nutzen (der ggf. nur lokal auf ein Projekt begrenzt ist).

3. „**Intrinsic interoperability** over custom integration": Die Interoperabilität zwischen Systemen innerhalb einer SOA ist immanenter Bestandteil des Konzepts. Natürlich müssen auch beispielsweise Alt-Systeme angebunden werden, was in vielen Fällen als maßgeschneiderte Integration zu bezeichnen ist, jedoch ist die grundlegende Fähigkeit der Interoperabilität wichtiger.

4. „**Shared services** over specific-purpose implementations": Die Wiederverwendung von Services ist ein wichtiger Aspekt der SOA und sollte grundsätzlich höher bewertet werden als zweckgebundene Implementierungen.

[13] Die Autoren des SOA-Manifests: Ali Arsanjani, Grady Booch, Toufic Boubez, Paul C. Brown, David Chappell, John deVadoss, Thomas Erl, Nicolai Josuttis, Dirk Krafzig, Mark Little, Brian Loesgen, Anne Thomas Manes, Joe McKendrick, Steve Ross-Talbot, Stefan Tilkov, Clemens Utschig-Utschig und Herbjörn Wilhelmsen.

[14] Der englische Originaltext des SOA-Manifests findet sich unter: `http://www.soa-manifesto.org/`.

5. „**Flexibility** over optimization“: Ohne Zweifel ist die Optimierung eines IT-Systems für den Gebrauch in einem Unternehmen obligatorisch. Jedoch soll bei der Umsetzung einer SOA mehr Wert auf Flexibilität gegenüber Optimierung gelegt werden.

6. „**Evolutionary refinement** over pursuit of initial perfection“: Der letzte Punkt betrifft das Vorgehen zur Einführung einer SOA in hohem Maße. Hierin wird das Schritt-für-Schritt-Vorgehen höher bewertet als der Versuch, ein anfänglich perfektes System aufzubauen. Dies könnte man als Zugeständnis zu agilen Methoden werten, die als Wert grundsätzlich das Verfeinern eines Systems in Iterationen befürworten.

Diese Werte sind als Ansatz zu sehen, wie sich Experten das Umsetzen einer SOA vorstellen. Sie lassen sich bestenfalls für das Treffen von strategischen Entscheidungen während der Planung und Umsetzung und ggf. auch Wartung einer SOA heranziehen. Kritiker dieses Manifests, die sich hauptsächlich in Blogs darüber äußern, bemängeln beispielsweise, dass die Erkenntnisse des Manifests entweder schon seit Jahren so gelten und im Grunde nichts Neues sind oder teilweise nur Allgemeingültiges wiedergeben.[15] Grundsätzlich kann man in dem SOA-Manifest aber den Versuch der Experten erkennen, eine gemeinsame Sichtweise des SOA-Ansatzes zu verdeutlichen.

Architekturprinzipien und Konzept

Das zentrale Konzept in einer SOA ist die Service-Orientierung und damit auf technischer Seite der Service; also eine Dienstleistung. An dieser Stelle muss darauf geachtet werden, dass der Service-Begriff genauer definiert wird, da er mit diversen Bedeutungen versehen ist. Eine im IKT-Bereich verbreitete und anerkannte Sammlung von Ansätzen und „Best Practices“ des IT-Managements ist *IT Infrastructure Library* (ITIL). So wird in [SZ08, S. 138] ein IT-Service definiert als „*ein oder mehrere IT-Systeme, die einen Geschäftsprozess ermöglichen oder unterstützen und vom Kunden als zusammenhängendes Ganzes wahrgenommen werden.*“. Darüber hinaus werden IT-Services in [Olb08, S. 12] die folgenden Eigenschaften zugesprochen: “*Dienstleistungen*

[15] Siehe hierzu:
http://soa-today.blogspot.com/2009/11/soa-manifesto-value-statement-critique.html,
http://ralfw.blogspot.com/2009/10/le-manifeste-du-jour-was-will-uns-die.html,
http://www.deltalounge.net/wpress/2009/10/soa-symposium-next-generation-soa/

sind immateriell, unteilbar, zeitlich begrenzt, individuell, standortbezogen und können nicht zurückgerufen werden.". Um den Service-Begriff über das generelle Verständnis hinaus in einem SOA-Kontext zu definieren, soll folgende (Web) Service-Definition aus dem Glossar des W3C (vgl. [W3C04d]) dienen:

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

Sinngemäß übersetzt ist ein (Web) Service eine Softwarekomponente für Maschine-zu-Maschine-Kommunikation über ein Netzwerk, der darüber hinaus eine maschinenlesbare Beschreibung (hier WSDL) besitzt. Die Interaktion wird gemäß dieser Definition meist mittels des HTTP und in Form von SOAP-Nachrichten vorgenommen. Dabei werden die *Extensible Markup Language* (XML)[16] und weitere Web-Standards verwendet. Erl schreibt einem Service, unabhängig von der verwendeten Technologie, die folgenden Eigenschaften zu (vgl. [Erl05]).

- **Lose Kopplung**: Dem Benutzer eines Services müssen keinerlei Details der Implementierung bekannt sein, die der Service kapselt. Nur die Information über dessen Endpunkt und die Art und Ausführung des Aufrufs sind relevant. So lassen sich gleichartige Services ohne Probleme austauschen, da keine Abhängigkeit zu der internen Logik der gekapselten Funktion aufgebaut wird.

- **Service-Vertrag**: Ein Service wird grundsätzlich durch eine Dokumentation beschrieben, dessen Inhalt nicht nur die Funktionalität, sondern zusätzlich auch die „Vertragsbedingungen" für einen Aufruf spezifiziert. Diese sogenannten *Service Level Agreements* sind ein wesentlicher Bestandteil von Services, um diese in einem Unternehmen einzusetzen.

[16] Eine generelle Einführung in XML findet sich in [Min02] und im Standard des W3C (vgl. [BPSM+08]). XML im SOA-Kontext wird betrachtet in [Erl04]

- **Autonomität**: Der Service kapselt einen Teil spezieller Systemfunktionalität und hat die Kontrolle über diese Funktionalität.
- **Abstraktion**: Ähnlich zur losen Kopplung ermöglicht die Abstraktion, dass die internen Abläufe des Services nicht von Belang sind. Der Service abstrahiert von der konkreten Funktionalität und ist für den Benutzer des Services wie eine Art *Blackbox*. Der Service wird mit bestimmten Parametern aufgerufen und gibt ggf. ein Ergebnis zurück. Die Abläufe innerhalb des Services werden nicht preisgegeben.
- **Wiederverwendbarkeit**: Im Idealfall ist der Service derart gestaltet, dass er sich in verschiedenen Kontexten wiederverwenden lässt. Der Grad der Wiederverwendbarkeit hängt hierbei stark von der Domäne und dem Systemdesign ab.

  17

- **Möglichkeit des Zusammensetzens**:

  Services lassen sich miteinander koppeln und ermöglichen so, eine größere Service-Einheit zu gestalten. Aus so zusammengesetzten Services werden dadurch Service-Bausteine, die mehrere Aktionen ausführen und sich ebenfalls wieder miteinander koppeln lassen. Dadurch werden komplexe Abläufe und Prozesse abbildbar (sogenannte *Workflows*).
- **Zustandslosigkeit**: Ein Service sollte keinen Zustand kennen und wird bei jedem Aufruf mit allen Parametern versorgt, die er benötigt. Dadurch lassen sich Abhängigkeiten zu anderen Softwarekomponenten verhindern, die sonst Fehlverhalten provozieren könnten.
- **Auffindbarkeit**: Ein Service wird durch Dokumente so beschrieben, dass er für Service-Nutzer auffindbar ist. Dies zielt auf einen Verzeichnisdienst wie UDDI ab, in dem ein Nutzer nach passenden Services suchen kann.
- **Einfachheit**: Eines der Versprechen der SOA ist die häufig genannte Einfachheit. Man muss aber aufpassen, dass man diese Aussage im richtigen Kontext deutet (vgl. [DJI05]). Hiermit ist nicht die Einfachheit

[17] Je sorgfältiger der Softwareerstellungsprozess bei bestehenden Prozessen durchgeführt wird, desto eher lassen sich identische Services erkennen und in einer Implementierung umsetzen.

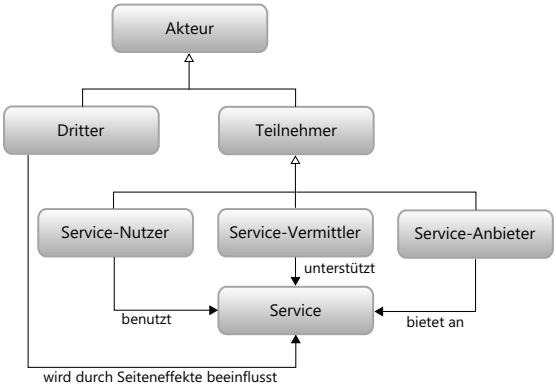
der technologischen Umsetzung einer SOA gemeint. Vielmehr geht es darum, dass die bei der Umsetzung einer SOA verwendbaren Konzepte einfach sind. Bei einer genaueren Betrachtung der technologischen Umsetzungsmöglichkeiten für eine SOA wird dies schnell expliziert (vgl. [VZT08]).

Das einer SOA zugrunde liegende Konzept lässt sich grob in zwei Aspekte einteilen: das **Rollenmodell**, das die Akteure einer SOA beschreibt, und das **Ebenenmodell**, das den Aufbau einer Unternehmens-SOA darstellt. Beide Aspekte werden in den nächsten Unterabschnitten näher betrachtet.

SOA-Rollenmodell

Eine Übersicht über die Akteure innerhalb einer SOA beschreibt das Stakeholder-Modell (vgl. [MELT08]), welches in Abbildung 2.13 dargestellt ist. Hierin werden vier verschiedene Rollen von Akteuren einer SOA identifiziert. **Service-Nutzer** können IT-Systeme sein, die Services in ihre Architektur einbinden, wie auch menschliche Nutzer, die einen Service „von Hand" ausführen. Jegliche Art von gekapselter Funktionalität, die über eine Schnittstelle aufgerufen werden kann, lässt sich als Service durch einen **Service-Anbieter** bereitstellen. Dieser ermöglicht damit Dritten, Teile seines IT-Systems zu nutzen. Die Schnittstellen sollten hierbei eine Dokumentation bereitstellen, die technische und organisatorische Informationen enthalten. Technische Dokumentation zu Web Services können beispielsweise durch *Web Service Description Language* (WSDL)-Dokumente beschrieben werden. Organisatorische Informationen werden dabei unter dem Begriff *Service Level Agreement* (SLA) zusammengefasst. Die Rolle des **Service-Vermittlers** wird in einer SOA meist durch ein Service-Verzeichnis, das UDDI, ausgefüllt. Dieses Verzeichnis ermöglicht es, den SOA-Teilnehmern Services bereitzustellen und zu suchen. Dabei sind diverse Informationsebenen vorgesehen, die organisatorische wie auch technische Informationen umfassen. Über den Service-Vermittler können Service-Nutzer für ein Nutzungsszenario nach passenden Services suchen. Die als **Dritte** bezeichneten Akteure sind keine aktiven Teilnehmer des SOA, hängen aber von der Nutzung oder Bereitstellung von Services ab. Dies kann z. B. eine Rechnungsstelle innerhalb eines Unternehmens sein, die Nutzungsstatistiken von Services benötigt, um innerbetriebliche Rechnungen zu stellen.

Um den generellen Ablauf der Kommunikation zwischen den Akteuren innerhalb einer SOA zu verdeutlichen, lässt sich das Modell auf die drei Akteu-



Quelle: angelehnt an [MELT08]

Abbildung 2.13: Akteure in einer SOA.

re **Service-Nutzer**, **Service-Anbieter** und **Service-Vermittler** bzw. **Service-Verzeichnis** vereinfachen. Die Kommunikation bei der Service-Nutzung zwischen Anbieter und Nutzer vollzieht sich in fünf aufeinanderfolgenden Schritten (ausführlich beschrieben als *Mechanisms for Awareness* in [MELT08], und dargestellt in Abbildung 2.14). Im Folgenden wird von der Nutzung von Web Services als technische Umsetzung der Kommunikation ausgegangen.

1. Der Anbieter registriert zunächst seine Dienste bei dem Service-Verzeichnis. Dabei werden diverse technische wie auch organisatorische Informationen zu den angebotenen Diensten hinterlegt.

2. Der Service-Nutzer sucht über eine entsprechende Oberfläche oder Schnittstelle im Service-Verzeichnis nach einem für sein IT-Problem passenden Service.

3. Das Service-Verzeichnis gibt ein entsprechendes Suchergebnis zurück, welches einen zur Anfrage passenden Service und dessen Beschreibung (z. B. ein WSDL-Dokument) enthält.

4. Der Service-Nutzer ruft dann unter Verwendung der technischen Beschreibung den Service auf, indem er eine der Schnittstelle entsprechenden SOAP-Nachricht an die Aufrufadresse (*Endpoint*) des Services schickt.

5. Der Service-Anbieter (bzw. der Service) schickt eine Antwort-Nachricht an den Service-Nutzer zurück.

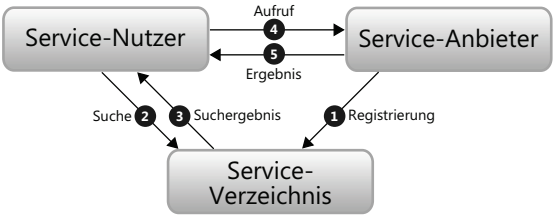


Abbildung 2.14: Das SOA-Ablaufmodell.

Ebenenmodell

Neben dem Rollen-Modell, das die Abläufe zwischen den Parteien einer SOA beschreibt, ist auch die Sicht auf die IT-Architektur einer SOA-Anwendung relevant. In der Literatur wird eine SOA meist durch ein Ebenenmodell mit vier horizontalen und mehreren weiteren vertikalen Schichten charakterisiert (vgl. [AGA+08, KBS07, Erl07]). Die unterste horizontale Ebene beinhaltet dabei (Alt-) Systeme, Datenbanken sowie Drittapplikationen und wird daher als **Applikationsschicht** bezeichnet. Darüber angeordnet ist die **Serviceschicht**, in der Services angeboten werden, deren Funktionalität der Applikationsschicht gekapselt zur Verfügung gestellt wird. Um Services zu Prozessen zusammenzusetzen, ist darüber die **Prozessschicht** angeordnet. Diese dient der sogenannten „Orchestrierung" von Services, bei der gekapselte Funktionen aus der Serviceschicht miteinander zu einem wertschöpfenden Prozess verknüpft werden. Zuoberst existiert die **Präsentationsschicht**, die für die Interaktion mit Menschen zuständig ist. In Abbildung 2.15 sind zusätzlich zwei vertikale Ebenen abgebildet: zum einen die **Integrationsschicht**, die beispielsweise mittels ESB die horizontalen Ebenen miteinander verbindet; zum anderen die Schicht für Qualitätsmanagement, Sicherheit, etc., die hier als generische **Steuerungsschicht** abgebildet ist. Diese steht hier exemplarisch für alle Steuerungsaufgaben (*Governance*) innerhalb einer SOA.

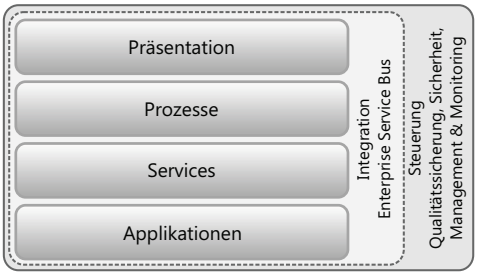


Abbildung 2.15: Das Ebenenmodell einer SOA.

Organisatorische Herausforderungen

Dass Service-Orientierung nicht nur eine Frage der technischen Umsetzung, sondern zusätzlich ein organisatorisches Unterfangen ist, wird spätestens dann klar, wenn die Umsetzung einer SOA über den Prototypbetrieb hinaus größeren Umfang anstrebt. Durch die Kapselung von Funktionalitäten und der damit verbundenen – und auch explizit durch die Definition einer SOA geforderten – Eigenschaft, dass eine verteilte Funktionalität nicht unter eigener Kontrolle stehen muss, dürfen organisatorische Aspekte nicht unberücksichtigt bleiben. Gerade in größeren Unternehmen mit diversen Fachabteilungen, die idealerweise ihre Kernkompetenzen in Form von Services zur Verfügung stellen, müssen die verschiedenen Rollen innerhalb der SOA verwaltet werden. Neben den Rollen gilt es hierbei aber auch die SOA als Ganzes unter Kontrolle zu behalten, was sich durch die Natur eines verteilten Systems als eine durchaus komplexe Aufgabe herausstellen kann. Ein mittlerweile geläufiger Begriff ist hier die *SOA-Governance*, die stellvertretend für Themen der Kontrolle, Steuerung, Überwachung und weiterer organisatorischer Aufgaben innerhalb einer SOA steht. Das Marktforschungsinstitut Gartner [Mal06] empfiehlt dazu Folgendes:

> „SOA governance isn't optional – it's imperative. Without it, return on investment will be low and every SOA project out of pilot phase will be at risk."

Der Meinung, dass SOA-Governance untrennbar mit dem Erfolg einer SOA-Einführung verbunden ist, stimmen viele Autoren direkt zu (vgl. [Afs07, BMT06, Jos08, NL04]). Die konkrete Ausgestaltung der Governance ist dabei jedoch in noch viel höherem Maße durch die jeweils „gelebte" Firmenkultur geprägt

als die technische Sichtweise. In [SIVE08] wurde daher eine Studie durchgeführt, um den größten gemeinsamen Nenner zu identifizieren. Daraus gingen zumindest zwei weitestgehend übereinstimmende Punkte bezüglich der SOA-Governance hervor:

- **Richtlinien**: Richtlinien (*policies*) sollen in einer SOA festlegen, welche Werkzeuge und Technologien verwendet werden, welche organisatorischen Eckpunkte (und dazu gehören auch Verantwortlichkeiten von Personen zu Services) zu beachten sind, und die Art, wie Web Services entwickelt und genutzt werden.

- **Leistungsmessung/-überwachung**: Die Leistungsmessung überwacht Web Services und jegliche angeschlossene Systeme der SOA und protokolliert deren Nutzung, auftretende Fehler sowie ggf. weitere Aspekte der Nutzung. Es gibt viele Gründe, die Leistung (*performance*) einer SOA zu überwachen und zu messen. Die Messung erlaubt später beispielsweise Abrechnungen zu erstellen, oder zusammen mit der Überwachung von Services innerhalb der SOA Dienste zu verbessern oder Fehler aufzudecken.

An dieser Stelle soll aufgrund der doch sehr großen Unterschiede in der detaillierten Umsetzung der Governance auf einen tieferen Einstieg in die Materie verzichtet werden. Das Thema wird jedoch an manchen Stellen im Verlauf der vorliegenden Arbeit wieder aufgegriffen.

Technologien und Standards

Um eine SOA umzusetzen, existieren diverse standardisierte Technologien. Die häufig verwendete Technologie für die Kommunikation und die Realisierung von Services sind die bereits in Abschnitt 2.3.1 erwähnten *Web Services*. Eine weitverbreitete Technologiekombination ist hierbei *SOAP* als Nachrichtenformat auf *Extensible Markup Language* (XML)-Basis und der Transport mittels HTTP über das Transportprotokoll TCP/IP[18]. Darüber hinaus wird, um einen Service und seine Methoden zu beschreiben, die *Web Service Description Language* (WSDL) verwendet. Diese Kombination von Standards bildet den **Kern** einer SOA-Umsetzung.

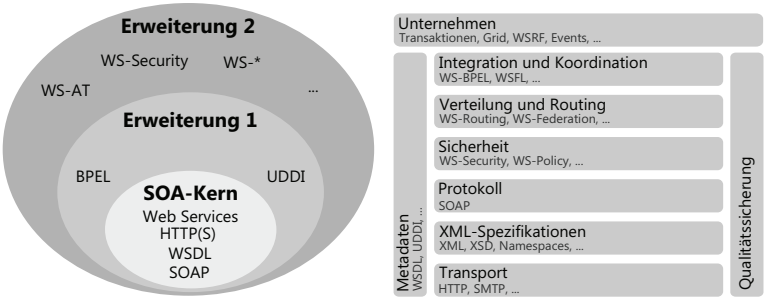
[18]Hier sind auch diverse andere Transportprotokolle einsetzbar: beispielsweise SMTP, FTP, etc.

Eine erste **Erweiterungsstufe** wird durch das Einsetzen eines Verzeichnisdienstes, meist in Form des Standards UDDI erreicht. So lassen sich implementierte Web Services auch registrieren, suchen und finden. Der nächste sinnvolle Schritt innerhalb einer SOA besteht nun in der Verwendung einer Prozessabbildungssprache wie der *Business Process Execution Language* (BPEL), mittels derer komplexe Prozesse modelliert und ausgeführt werden können. Eine **zweite Erweiterungsstufe** beschäftigt sich daraufhin mit vielen weiteren Problemen verteilter Systeme, wie beispielsweise der Sicherheit von Nachrichten (*WS-Security*) oder den transaktionalen Garantien (*WS-Atomic Transaction*, *WS-Business Activity*, *WS-Composite Application Framework*).

Im Bereich der Web Services gibt es über 70 Standards (teilweise noch im Entwicklungsprozess), so dass im Verlauf der Arbeit nur auf einige wenige eingegangen wird. Eine grafische Übersicht über die technologischen Ebenen einer SOA und deren Ausbaustufen ist in Abbildung 2.16(a) als Schalenmodell dargestellt. Hierbei ist anzumerken, dass die Grafik den ESB als wichtigen Bestandteil einer konkreten Umsetzung innerhalb eines Unternehmens bewusst nicht enthält, da dieser in der ursprünglichen Definition nicht enthalten ist. Dennoch wird das Konzept des ESB als Rückgrat einer modernen SOA später noch erläutert. Sofern noch nicht geschehen, werden die hier als Kerntechnologien angesehenen Standards, sowie die der ersten Erweiterungsstufe, im Folgenden genauer erläutert. Details der zweiten Erweiterungsstufe werden im Folgenden jedoch nicht weiter vertieft. Eine zusätzliche Sichtweise auf die Verwendung von WS-Standards bei der Umsetzung einer SOA mit Web Services stellt der *Web Services Stack* dar. Dieser skizziert die aufeinander aufbauenden Stufen der Nutzung von Web- und WS-Standards (vgl. Abbildung 2.16(b)).

SOAP

Ursprünglich als *Simple Object Access Protocol* eingeführt, ist SOAP als leichtgewichtiges Nachrichtenaustauschformat für strukturierte Informationen konzipiert worden. SOAP verwendet dabei den Standard *XML* zur Darstellung der Protokollstruktur, was es unabhängig von der verwendeten Transportschicht macht. Im SOA-Kontext wird SOAP für die Kommunikation zwischen Web Services verwendet und benutzt hierfür meist HTTP als Transportprotokoll. Es ist aber ebenso denkbar, Nachrichten über die Protokolle FTP oder SMTP zu versenden.



(a) Schalenmodell einer SOA

(b) Web Services Stack

Quelle: b) angelehnt an [DJI05]

Abbildung 2.16: SOA-Standards als Schalen- und Stack-Modell.

Der grundlegende Aufbau von SOAP sieht ein Header- sowie ein Body-Element vor. Das Body-Element nimmt in diesem Zusammenhang den sogenannten *Payload* einer Nachricht auf: Hierbei handelt es sich um Nutzdaten des Nachrichtenaustausches, die zwischen einem Service-Nutzer und -Anbieter ausgetauscht werden. Das Header-Element einer Nachricht enthält Kontext-Informationen zur Nachricht. Hierunter fallen beispielsweise Informationen zu Nutzungsrechten und Authentifizierung oder Zeitstempel. Somit ermöglichen Daten im Header das Auswerten, Weiterleiten und Verifizieren von SOAP-Nachrichten, während die Body-Inhalte für den Aufruf einer Funktion als Parameter bzw. dessen Rückgabewert verwendet werden. In Listing 2.4 ist eine SOAP-Nachricht beispielhaft dargestellt. Der Header (Zeile 3-8) enthält dabei ein *WS-Security*-Element namens *UsernameToken*, welches das Mitsenden von Authentifizierungsdaten des Aufrufers ermöglicht. In diesem Beispiel werden der Benutzername (Zeile 5) und dessen Passwort als Klartext (Zeile 6) übertragen. Anschließend enthält der Body (Zeile 9-13) die Nutzlast der Nachricht: hier den Aufruf der Methode *getWeather* (Zeile 10) mit Angabe eines Parameters; der Stadt Münster (Zeile 11).

Auf Anbieterseite kann das Header-Element zunächst ausgewertet werden, was in einer Überprüfung des Benutzers und dessen Passwort bestehen würde. Anschließend wird die angegebene Methode (*getWeather*) aufgerufen und deren Rückgabewert in eine Antwort-Nachricht verpackt. Diese wird dann an den Aufrufer zurückgesendet. Die Antwort könnte (dem obigen minimalen Beispiel

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
    envelope" ...>
3  <env:Header>
4   <wsse:UsernameToken wsu:Id="Example">
5     <Username>Bob</Username>
6     <Password Type="wsse:PasswordText">meinGeheimesPasswort</
        Password>
7   </wsse:UsernameToken>
8  </env:Header>
9  <env:Body>
10  <m:getWeather xmlns:m="http://www.beispiel.de/Weather">
11   <m:city>Muenster</m:city>
12  </m:getWeather>
13 </env:Body>
14 </env:Envelope>
```

Listing 2.4: Aufbau einer SOAP-Anfrage (request).

folgend) wie in Listing 2.5 aussehen. Dabei wird kein Header-Element mitgeschickt, da dieses für die Antwort nicht relevant ist. Das Body-Element enthält den Rückgabewert der aufgerufenen Methode (Zeile 5), eingeschlossen von einem XML-Element mit dem Namen der aufgerufenen Methode (Zeile 4 und 6: *getWeatherResponse*).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
    envelope" ...>
3 <env:Body>
4   <m:getWeatherResponse xmlns:m="http://www.beispiel.de/Weather
      ">
5    <return xsi:type="xsd:string">Teilweise bewölkt, 23 Grad C</
       return>
6   </m:getWeatherResponse>
7  </env:Body>
8 </env:Envelope>
```

Listing 2.5: Aufbau einer SOAP-Antwort (response).

Diese Minimalbeispiele sollen verdeutlichen, wie der Aufbau einer SOAP-Nachricht aussieht. Die aktuelle SOAP-Spezifikation ist als Standard des *World Wide Web Consortium* (W3C) in Version 1.2 verfügbar (siehe [W3C07c]).

Web Service Description Language

Um die Schnittstellen von Web Services zu beschreiben, wurde die *Web Service Description Language* spezifiziert. Auch dieser Standard ist vom W3C entwickelt und liegt aktuell in Version 2.0 vor (siehe [W3C07d]). Wie viele der Standards im SOA-Umfeld basiert auch WSDL auf XML und ermöglicht durch den standardisierten Aufbau des Dokuments die automatische Verarbeitung und somit das automatische Aufrufen von Web Services durch den Aufrufer, wenn dieser das beschreibende Dokument kennt. Die wichtigen, in einem WSDL-Dokument beschriebenen, Informationsaspekte sind hierbei die folgenden:

- **types**: Hiermit werden die Datentypen beschrieben, die innerhalb des Services verwendet werden. Die Standardtypen aus dem W3C-Schema werden hierbei präferiert. Es ist aber auch möglich weitere Namensschemata (wie beispielsweise DTDs oder Relax NG) einzubinden.
- **interface**: Das Interface-Element beschreibt die abstrakten Methoden sowie Nachrichten, die für den Service benötigt werden.
- **binding**: Die Binding-Elemente beschreiben das Transportformat der Nachricht, somit z. B. die Art der Codierung des Payloads innerhalb der Nachricht.
- **service**: Dieser Eintrag zeigt schließlich auf den Zugriffspunkt des Services (Endpunkt), über den der Web Service aufgerufen werden kann.

Im Folgenden wird der Wetter-Service aus dem vorigen Beispiel einer ursprünglich Simple Object Access Protocol (SOAP)-Nachricht verwendet. Ein dazu passendes WSDL-Dokument ist in den nachfolgenden vier Listings beschrieben. Entgegen der Reihenfolge innerhalb der Beschreibungsdatei ist die logische Lesart eines WSDL-Dokuments von unten nach oben: Es wird ein *Service* beschrieben, welcher Methoden (*Bindings*) bereitstellt. Diese haben eine Methodensignatur (*Interface*), die beschreibt, welche Parameter für einen Methodenaufruf notwendig sind und wie die Rückgabeparameter definiert sind. Die einzelnen Parameter sind hierbei als Typen (*Types*) hinterlegt. Die Struktur eines WSDL-Dokuments (in Version 2.0) ist in Abbildung 2.17 schematisch dargestellt.

Innerhalb des Services werden zwei Typen benötigt: ein Aufrufparameter und ein Rückgabeparameter. In Listing 2.6 ist der Anfang des XML-Gerüsts

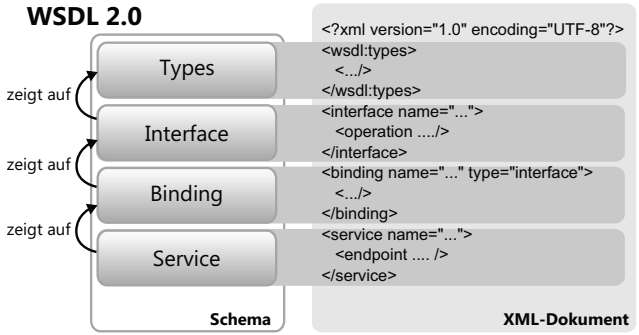


Abbildung 2.17: Die WSDL-Dokumentenstruktur.

eines WSDL-Dokuments dargestellt. Unter dem Knoten <*wsdl:types*> wird dabei ein Schema zur Verfügung gestellt, welches die beiden Elemente namens *getWeather* (Zeile 4) und *getWeatherResponse* (Zeile 5) beschreibt. Durch das Attribut *type* wird der Typ des Elements beschrieben. Um das Beispiel kurz zu halten, sind beide Parametertypen hierbei als Zeichenkette (String) definiert und stellen Standardtypen aus dem Namensraum *xs* dar.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:types>
3    <s:schema elementFormDefault="qualified" targetNamespace="
        http://www.beispiel.de/Weather/">
4       <xs:element name="getWeather" type="xs:string"/>
5       <xs:element name="getWeatherResponse" type="xs:string"/>
6    </s:schema>
7 </wsdl:types>
```

Listing 2.6: Typ-Definition in einem WSDL-Dokument.

Im Interface-Teil der Service-Beschreibung werden die Parameter der verfügbaren Methoden beschrieben. Im Listing 2.7 wird das Interface für den Endpunkt WeatherSoap (Zeile 1) beschrieben. Anschließend werden die darin enthaltenen Methoden (Operation) und deren Eingabe- sowie Ausgabeparameter erläutert. Der hier beschriebene Service enthält lediglich eine Methode namens *getWeather* (Zeile 2), der wiederum einen Eingabeparameter (Zeile 3) und einen Ausgabeparameter (Zeile 4) definiert. Hierbei ist zu sehen, dass das

Attribut *element* eines Parameters jeweils auf einen der bereits definierten Typen (auf *getWeather* und *getWeatherResponse*) verweist. Somit ist klar, dass der Eingabe- sowie der Rückgabeparameter dieser Methode ein String ist.

```
1 <interface name="WeatherSoap">
2    <operation name="getWeather" pattern="http://www.w3.org
        /2004/08/wsdl/in-out">
3       <input messageLabel="getWeatherSoapIn" element="
           s0:getWeather"></input>
4       <output messageLabel="getWeatherSoapOut" element="
           s0:getWeatherResponse"></output>
5    </operation>
6 </interface>
```

Listing 2.7: Interface-Definition in einem WSDL-Dokument.

Der dritte Bestandteil einer WSDL-Datei ist das *Binding*. Hierin werden die Angaben für Transport und Format eines Services beschrieben (siehe Listing 2.8). Zunächst wird per *name* (Zeile 1) auf den Service *WeatherSoap* verwiesen. Damit sind die folgenden Details für den entsprechenden Service vorgegeben. In Zeile 2 wird HTTP als Transportprotokoll definiert und der Stil der SOAP-Nachricht auf *document* gestellt. An dieser Stelle wäre auch die Möglichkeit gegeben, auf den RPC-Stil zurückzugreifen.[19] Anschließend werden für Eingabe- (*Input*, Zeile 6) sowie Ausgabenachricht (*Output*, Zeile 9) die Art der SOAP-Body-Codierung vorgenommen: in diesem Fall als *literal*.

Der letzte Teil eines WSDL-Dokuments ist die Service (bzw. Endpunkt)-Beschreibung. Hierin wird der Name des Services beschrieben (Listing 2.9, Zeile 1) und ein Endpunkt definiert, an dem dieser Service aufgerufen werden kann. In der Beschreibung des Endpunkts wird auf das verwendete *Binding* des vorigen XML-Abschnitts verwiesen (Zeile 2) und die Adresse des Endpunkts, hier eine URL `http://www.beispiel.de.de/Weather`, angegeben.

Durch diese vier Teile wird ein gängiger Web Service (auf SOAP-Basis) mittels WSDL beschrieben, um diesen automatisch aufrufbar zu gestalten. Generell kann man hieran schon erkennen, dass es relativ viel Aufwand erzeugt, einen so trivialen Service – mit lediglich einem Eingabe- und einem Ausgabeparameter – zu beschreiben. Darüber hinaus sind hierbei technische Beschreibungs-

[19] Hierbei unterscheidet sich die SOAP-Nachrichten dergestalt, dass im Document-Modus ein gesamtes Dokument übertragen wird, und im RPC-Modus nur die aufgerufene Methode und deren Parameter im SOAP-Body stehen.

```
<binding name="WeatherSoap" type="s0:WeatherSoap">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/
      http" style="document">
      <operation ref="getWeather">
         <soap:operation soapAction="http://www.beispiel.de/
            Weather?getWeather" style="document"/>
         <input>
            <soap:body use="literal"/>
         </input>
         <output>
            <soap:body use="literal"/>
         </output>
      </operation>
   </soap:binding>
</binding>
```

Listing 2.8: Binding-Definition in einem WSDL-Dokument.

```
<service name="getWeather">
   <endpoint name="WeatherSoap" binding="s0:WeatherSoap">
      <soap:address location="http://www.beispiel.de/Weather"/>
   </endpoint>
</service>
```

Listing 2.9: Service-Definition in einem WSDL-Dokument.

aspekte definiert. Eine fachliche oder sogar semantische Beschreibung sucht man an dieser Stelle vergebens. Glücklicherweise erstellen viele Programmierumgebungen die WSDL-Beschreibungen zu programmierten Services automatisch, so dass hierbei kein manueller Aufwand entsteht.

Universal Description, Discovery and Integration

Je mehr Services innerhalb einer SOA existieren, desto leichter verliert man die Übersicht über die Service-Landschaft. Um großen Mengen an Services zu begegnen und nach Services suchen zu können, wurde das *Universal Description, Discovery and Integration* (UDDI) als Verzeichnis-Dienst in SOA entwickelt (vgl. [WCL+05]). Das Gremium *OASIS*, in dem viele „Big Player" der Softwarebranche mitarbeiten, hat diesen Standard, mittlerweile in der 3. Version, erarbeitet. OASIS ist neben dem W3C eine der Organisationen, die sich intensiv mit Web Service Standards beschäftigt. Die Grundidee bei UDDI ist das Registrieren von

Web Services innerhalb des Verzeichnisses und die Beschreibung auf verschiedenen Ebenen, um den Service für Nutzer auffindbar zu speichern. Dabei wird innerhalb des Verzeichnisses auf *White Pages*, *Yellow Pages* und *Green Pages* zurückgegriffen. Die White Pages (Basisinformationen) beinhalten Daten über die Identität eines Serviceanbieters. Dies sind Informationen über das Betätigungsfeld des Anbieters sowie Kontaktdaten. Die Yellow Pages (Servicekategorisierung) ordnen die Services in spezifische Kategorien ein. Die Kategorien orientieren sich hierbei an Standards wie dem internationalen Klassifikationssystem *United Nations Standard Products and Services Code*[20], welches ein Produkt bzw. eine Service-Leistung aus einem fünf-stufigen Code zusammensetzt. Die ersten vier Stufen sind hierbei standardisiert. Schließlich beinhalten die Green Pages die technischen Beschreibungen zu einem Service. Die hier abgelegten Daten werden in einem sogenannten *tModel* abgelegt. Um einen passenden Service zu finden, werden die Daten daraufhin bei einer Suche mit zu spezifizierenden Suchparametern abgeglichen. Das Datenmodell des UDDI-Verzeichnisses setzt sich daher aus fünf Hauptbestandteilen zusammen:

- *businessEntity*: Enthält Informationen über den Anbieter, beispielsweise Unternehmen, Name, Arbeitsgruppe, etc.
- *businessService*: Hierin werden der Service und seine aufrufbaren Methoden beschrieben. Dieser gehört eindeutig zu einem *businessEntity*.
- *bindingTemplate*: Umfasst technische Eigenschaften des Services.
- *publisherAssertion*: Beschreibt die Verbindung zweier *businessEntities* zueinander. Beispielsweise kann somit eine Eltern-Kind-Beziehung zwischen Unternehmensteilen ausgedrückt werden.
- *t-Modell*: Hierin werden Referenzen auf technische Anforderungen gespeichert. Dies kann u. a. auch ein WSDL-Dokument sein. Es handelt sich hierbei nicht um das oben genannte tModell.

Eine Untersuchung diverser Web Service Verzeichnisse und deren Eignung für das Auffinden von Web Services aus dem Jahre 2007 schloss mit den folgenden Worten (vgl. [HLV07]):

[20] Siehe: `http://www.unspsc.org/`.

> „The analysis shows that there are currently no successful Web service registries that base their strategy on being an access point for service search alone."

Dies zeigt eine gewisse Zurückhaltung gegenüber UDDI. Darüber hinaus wird dieses Ergebnis unterstrichen durch die Tatsache, dass das letzte öffentliche UDDI-Verzeichnis (immerhin betrieben von IBM, Microsoft und SAP) Anfang des Jahres 2006 abgeschaltet wurde. Die offizielle Aussage war, dass nach 5 Jahren erfolgreicher Laufzeit die Robustheit und Funktionsfähigkeit des UDDI-Verzeichnisses bestätigt wurde[21].

Eine mögliche Alternative zu UDDI im SOA-Umfeld ist *WS-Inspection* (vgl. [BBM+01, HLV07]). Bei dieser Methode wird in das Wurzelverzeichnis des Servers, auf dem die veröffentlichten Services liegen, eine Datei mit Namen `inspection.wsil` abgelegt, die eine Liste aller Services und deren WSDL-Beschreibungen enthält. Dieser Ansatz ermöglicht, im Gegensatz zu UDDI, keine zentrale Suche nach Services, sondern nur das lokale, auf Domänen bzw. Server beschränkte, Auffinden von Services. WS-Inspection sieht aber ein *Link*-Element vor, welches den Aufbau von Hierarchien der Beschreibungsdateien ermöglicht.

Ebenfalls seit 2009 als OASIS-Standard verabschiedet, ist *WS-Discovery* (vgl. [OAS09a]) eine technische Spezifikation für das Suchen nach Services per Multicast. Diese Spezifikation wurde im Jahre 2005 von BEA Systems, Canon, Intel, Microsoft und WebMethods entwickelt und ermöglicht das Suchen von Services in einem lokalen Netzwerk.

Business Process Execution Language

Die BPEL – offiziell als *WS-BPEL* bezeichnet – ist eine Beschreibungssprache für WS-basierte Geschäftsprozesse, die auf dem WSDL-Standard aufbaut (vgl. [OAS07]). Jede Aktivität innerhalb eines Prozesses wird hierbei durch einen Web Service repräsentiert. BPEL wurde im Jahre 2002 von IBM, BEA Systems und Microsoft eingeführt. Interessant ist, dass ein BPEL-Prozess selbst auch wieder als Web Service aufrufbar ist. Darüber hinaus ist im eigentlichen Standard keine Interaktionsmöglichkeit mit einem Menschen vorgesehen, die Prozesse verstehen sich daher als vollständig automatisiert. Möchte man Prozesse mit menschlicher Beteiligung modellieren, so kann man ergänzende Standards, wie beispielsweise *WS-BPEL4People* verwenden.

[21] Siehe: `http://uddi.microsoft.com/about/FAQshutdown.htm`.

Tabelle 2.3: Atomare Aktivitäten bei BPEL.

Aktivität	Bedeutung
`assign`	Zuweisen oder Ändern von Variablenwerten
`invoke`	Synchroner oder asynchroner Aufruf eines Web Services
`receive/reply`	Bereitstellen einer Web Service-Schnittstelle
`throw`	Ausgeben eines Fehlers
`wait`	Warten auf ein spezielles Ereignis oder eine angegebene Zeitspanne
`empty`	Platzhalter-Element für keinerlei Ausführungsanweisung

Die Sprache definiert zunächst atomare Aktivitäten (*Basic Activities*), die einfache Aktionen durchführen. Um Abläufe und logische Konstrukte, wie beispielsweise Schleifen oder Bedingungen, im Prozess auszudrücken, existieren strukturierte Aktivitäten (*Structured Activities*). Darin lassen sich wiederum atomare Aktivitäten verwenden. Darüber hinaus existiert das Konzept der Gültigkeitsbereiche (*Scopes*), mit deren Hilfe Aktivitäten gebündelt und zu transaktionalen Einheiten zusammengefasst werden können. In den Tabellen 2.3 und 2.4 werden zunächst die unterschiedlichen Aktivitäten aufgezählt.

In dem nachfolgenden Beispiel[22] (siehe Listing 2.10) ist eine Prozessbeschreibung für ein klassisches „Hello World"-Programm dargestellt.

Die ersten Zeilen (3-8) dienen der Definition von Namensräumen für den Prozess und dem Import von existierenden WSDL-Dokumenten. Damit lassen sich innerhalb des Prozesses Elemente aus den importierten Dokumenten referenzieren (siehe Zeile 12). Die sogenannten *partnerLinks* (Zeile 10-14) müssen am Anfang der Prozessbeschreibung definiert werden und stehen jeweils für einen Web Service, der im Verlauf des Prozesses aufgerufen werden soll. Diese enthalten einen Namen (*name*), einen Typ (*partnerLinkType*) und eine Rolle (*partnerRole*). In einem Prozess lassen sich auch Variablen definieren, denen später durch den Befehl `assign` ein Wert zugewiesen werden kann. In Zeile 16-19 wird die Variable *hello_world* definiert. Eine Variable muss hierbei einem Typ entsprechen, der durch *messageType* angegeben wird. Die Aktivi-

[22] Dieses Beispiel ist im Original zu finden unter: `http://www.eclipse.org/tptp/platform/documents/design/choreography_html/tutorials/wsbpel_tut.html`.

Tabelle 2.4: Strukturierte Aktivitäten bei BPEL.

Aktivität	Bedeutung
`sequence`	Folge von sequenziell abzuarbeitenden Aktivitäten
`while`	Ausführen einer Schleife, bis eine (boolesche) Bedingung erfüllt ist
`switch`	Bedingte Ausführung einer Aktivität aus einer Liste von Aktivitäten
`flow`	Angabe für beliebig ausführbare Reihenfolge bei einer Menge an Aktivitäten (durch *links* können Abhängigkeiten angegeben werden)
`pick`	Nicht-deterministische Wahl durch externe Ereignisse

tät `assign` weist anschließend der Variablen *hello_world* den Wert des Literals "Hello World" zu (Zeile 21-26). Der für die Prozessausführung im Kontext einer SOA interessanteste Teil ist in Zeile 28 zu finden: Durch den Befehl `invoke` wird ein zuvor definierter Web Service aufgerufen. Es muss der entsprechende *partnerLink*, die aufzurufende Methode des Services (*operation*) sowie die zu übergebende Variable bzw. Variablen (*inputVariable*) angegeben werden. Mit dem Aufruf des *printService* endet dieser BPEL-Prozess.

Für die Ausführung von BPEL-Prozessen kommen *Workflow Engines (WEs)* zum Einsatz, denen eine Prozessbeschreibung übergeben werden muss (als *deployment* bezeichnet). Einmal übergeben, lässt sich ein BPEL-Prozess dann beliebig oft ausführen. Die Anwendungsoberfläche der WE erlaubt dabei das Verwalten und Ausführen sowie die Überwachung der Prozesse.

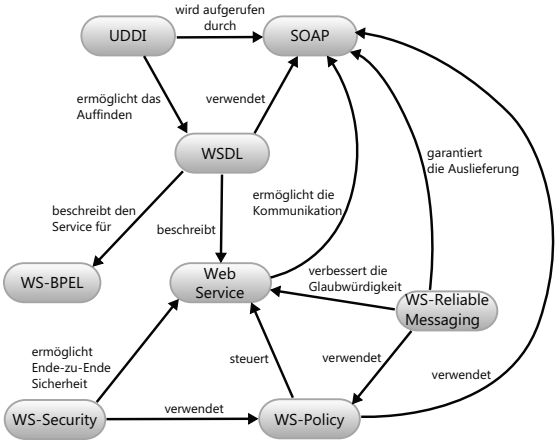
Die Tatsache, dass BPEL als XML-Dokument beschrieben wird, und kein Standard für eine grafische Notation vorliegt, führte dazu, dass die meisten Anbieter von BPEL-unterstützender Software eigene Modellierungsnotationen umgesetzt haben. Durch die Blockstruktur von BPEL entstehen hierbei zwar meist Modelle in der Art von Struktogrammen. Es bleibt jedoch genügend Interpretationsspielraum für eigene Erweiterungen. Der Austausch der Beschreibungen sollte hierbei zwar nicht eingeschränkt sein, sofern man sich an den BPEL-Standard hält, jedoch ist die grafische Darstellung und die damit einhergehende vereinfachte Erstellung von Prozessbeschreibungen durch grafische Oberflächen von Hersteller zu Hersteller verschieden. Obwohl BPEL als Quasi-

```
<?xml version="1.0" encoding="UTF-8"?>
<process
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:print="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
  <!--Hello World - Rudimentäres BPEL-Beispiel -->
  <import importType="http://schemas.xmlsoap.org/wsdl/"
    location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
    namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />

  <partnerLinks>
    <partnerLink name="printService"
        partnerLinkType="print:printLink"
        partnerRole="printService"/>
  </partnerLinks>

  <variables>
    <variable name="hello_world"
        messageType="print:PrintMessage" />
  </variables>

  <assign>
    <copy>
      <from><literal>Hello World</literal></from>
      <to>hello_world.value</to>
    </copy>
  </assign>

  <invoke partnerLink="printService" operation="print"
     inputVariable="hello_world" />

</process>
```

Listing 2.10: Hello World Beispiel mit BPEL.

Standard im Web Service-Umfeld zu sehen ist, erfreut sich die Modellierungssprache BPMN immer größerer Beliebtheit. In einem späteren Kapitel wird auf Prozessmodellierung mit BPMN im Zusammenhang mit dem Vorgehensmodell einer Web-orientierte Architektur (WOA) noch eingegangen.

Eine Ontologie für Web Services, die zur Steigerung der Übersichtlichkeit die hier bisher behandelten Komponenten in ihren Beziehungen darstellt, ist in Abbildung 2.18 skizziert.



Quelle: angelehnt an `http://www.soaspecs.com/ws.php`.

Abbildung 2.18: Ausgewählte WS-Standards und deren Assoziationen.

Enterprise Service Bus

Obwohl das hier vertretene Verständnis der Kerntechnologien einer SOA den Enterprise Service Bus nicht umfasst, ist diese Architekturkomponente für die IT-Infrastruktur einer Unternehmens-SOA durchaus relevant. Das zugrunde liegende Konzept ist grundsätzlich immer die Nutzung eines zentralen Nachrichtenkanals, über den alle Services miteinander in Verbindung treten können (siehe 2.2.1). In der Literatur werden verschiedene Begriffe für dieses Konzept verwendet: *Message Broker* (vgl. [ACKM04]), *Service Bus* (vgl. [CHKT05]) oder – meist auch als Bezeichnung von Softwareprodukten – Enterprise Service Bus (vgl. [NL04, Cha04, Kee04]). Das Konzept eines ESB wurde von der Unternehmensberatung Gartner im Jahre 2002 folgendermaßen beschrieben (vgl. [Sch02]):

> „A new form of **enterprise service bus** (ESB) infrastructure – combining **message oriented middleware**, **web services**,

> **transformation** and **routing intelligence** – will be running in the majority of enterprises by 2005. These high-function, low-cost ESBs are well suited to be the backbone for service oriented architectures and the enterprise nervous system."

Wenn man von der Tatsache absieht, dass die Prognose von Gartner so nicht eingetroffen ist, fällt diese Definition sehr treffend aus. Eine allgemein anerkannte Definition ist schwer zu finden, da der Begriff des Service Busses sehr vom Marketing großer Softwarefirmen geprägt ist. Die Grundmerkmale sind aber meist ähnlich den Anforderungen an Nachrichten-orientierte Middleware (*Message-oriented Middleware (MOM)*). Ein Unterscheidungsmerkmal ist jedoch, dass man bei der klassischen MOM ein Warteschlangenmodell einsetzt, bei dem eine Nachricht in einer Warteschlange eingereiht wird und diese wartet, bis sie abgeholt wird (vgl. [KBS07]). Die Vorstellung beim ESB ist normalerweise die, dass eine synchrone Kommunikation über einen zentralen Nachrichtenkanal realisiert wird. Die Hauptaufgabe eines ESB, mit Fokus auf dem Datenfluss, liegt hierbei in dem Entgegennehmen von Nachrichten, dem Suchen des Empfängers der Nachricht sowie der Übermittlung der Nachricht. Ebenso wie bei der Definition ist auch im geforderten Umfang der Fähigkeiten eines ESB keine klare Übereinstimmung zu finden. Die Literatur nennt aber die folgenden Kern-Anforderungen an einen ESB: (vgl. [CHKT05, Cha04, DJI05]):

- **Transformation**: Die Kommunikation zweier heterogener Komponenten über einen ESB erfordert, dass verschiedene Nachrichtenformate ineinander umgewandelt werden können. Denn nicht immer handelt es sich um Web Services, die beide mittels SOAP-Nachrichten kommunizieren. Intern sollen daher die Nachrichten in eine normalisierte Form gebracht werden (*normalized message*), so dass die Umwandlung in Zielnachrichtenformate vereinfacht wird. Der Vorteil ist hierbei, dass man eine Komponente für die Umwandlung (im Java-Umfeld des ESB als *Binding Component* bezeichnet) für ein Nachrichtenformat lediglich ein einziges Mal implementiert und damit immer wieder für Transformationen in eine normalisierte Nachricht verwenden kann.

- **Protokollunabhängigkeit**: Was durch die Transformation für das Nachrichtenformat erreicht wird, die Unabhängigkeit von der Art der Nachrichtencodierung, soll auch für das Protokoll des Versendens von

Nachrichten gelten. Dadurch soll erreicht werden, dass sich alle netzwerkfähigen Komponenten an den ESB anschließen können. Somit soll es möglich sein, beispielsweise Nachrichten per SMTP, FTP oder eben HTTP zu versenden.

- **Intelligentes Weiterleiten**: Das Weiterleiten einer Nachricht (*Routing*) soll dazu beitragen, dass die Nachricht vom Absender direkt zum Empfänger geschickt wird. Dabei soll diese Nachricht nicht an alle angeschlossenen Komponenten verschickt werden (*Broadcasting*), sondern exakt an einen Adressaten. Um zum Ziel „routen" zu können, soll der ESB intelligente Verfahren besitzen, um den Endpunkt der Nachricht einwandfrei zu ermitteln. Einige der ESB-Produkte besitzen daher ein internes UDDI-Verzeichnis, welches alle angeschlossenen Services (bzw. Komponenten) kennt und so nach dem korrekten Empfänger suchen kann.

- **Einfachheit**:

  Es wird hierbei gefordert, dass der ESB möglichst einfach sein soll. Abgesehen davon, dass Einfachheit in diesem Kontext nicht konkret messbar ist, erfordert dies hier Folgendes: Die zentrale Aufgabe, das Vernetzen von System-Komponenten, sollte mit möglichst wenig Aufwand erreicht werden können. Nach den Erfahrungen mit der hohen Komplexität von CORBA wollte man einen ESB möglichst einfach halten (vgl. [DJI05]). Dass dieses Ziel manchmal leider nicht erreicht werden kann, zeigt eine ESB-Umsetzung an der Westfälische Wilhelms-Universität (WWU) (vgl. [VZT08]).

In Abbildung 2.19 ist das typische Verständnis eines ESB als zentraler Nachrichtenkanal in einer SOA dargestellt.

Durch die weite Verbreitung der Programmiersprache Java im Umfeld von SOA, die u. a. durch robuste Enterprise-Server getrieben wurde, hat zumindest die Java-Community ein Standard für den ESB erarbeitet. Unter dem Namen *Java Business Integration (JBI)* werden darin die technischen Rahmenbedingungen und Spezifikationen einer Software erläutert, die als zentraler Nachrichtenvermittler in einer SOA seinen Dienst verrichtet (vgl. [THW05]). Der Standard – seit dem Jahre 2007 in Version 2.0[23] – wird durch viele An-

[23] Siehe: `http://www.jcp.org/en/jsr/proposalDetails?id=312`. Letzter Abruf 25.02.2011.

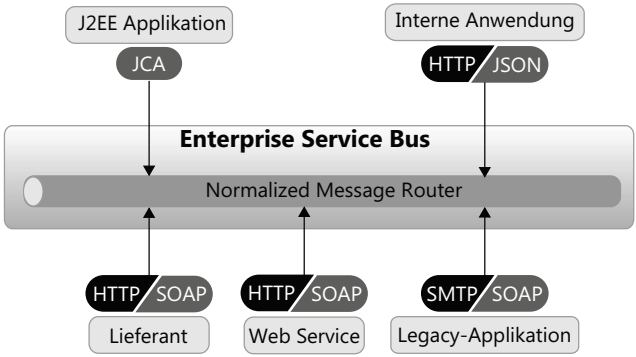


Abbildung 2.19: Die Vorstellung eines ESB.

bieter von SOA-Produkten und Nutzern getrieben und beschreibt den Nutzen mit den Worten: *„ [...] enable the creation of a Java business integration environment for the creation of Composite Applications involving such technologies as BPEL, Rules, XSLT and existing Java implementations [...]"*. Allerdings wurde die Arbeit an JBI eingestellt und wird zum aktuellen Zeitpunkt als inaktiv bezeichnet. Dennoch existieren einige Open Source-Produkte, die diesen Standard umsetzen.

2.3.2 Ressourcen-orientierte Architekturen

Das World Wide Web ist seit jeher ein großes Ressourcen-orientiertes Netzwerk. Jede Internetadresse - also jeder *Uniform Resource Locator* (URL) - kann als eine Ressource aufgefasst werden und wird beispielsweise bei Aufruf durch einen Browser an den Aufrufer zurückgeliefert. Roy Fielding, eines der ehemaligen Mitglieder der Arbeitsgruppe um Berners-Lee, beschrieb in seiner Dissertation im Jahre 2000 den auf HTTP basierenden Architekturstil *Representational State Transfer* (REST) und die damit vorhandene Möglichkeit zur Umsetzung einer verteilten Architektur mittels der gegebenen Standards (vgl. [Fie00]). Eine den REST-Stil verwendende Architektur wurde daher später öfter als ROA bezeichnet (vgl. [Sne04, Ove07a]). Die Umsetzung einer Softwarearchitektur mittels einer ROA ist aber immer noch selten. Durch das Phänomen des „Web 2.0" scheint sich dies aber langsam zu ändern: Denn in diesem

Kontext wird nach einfachen Mitteln zur Vernetzung von Web-Anwendungen gesucht und REST bietet sich hierfür an. Zusätzlich dazu gelangte auch JavaScript und das darauf basierende Datenaustauschformat JSON durch Douglas Crockford zu neuer Beachtung im Web-Kontext (vgl. [Cro06b]). Im Folgenden werden das Architekturprinzip und das Konzept einer ROA erläutert und die darin verwendeten Standards näher betrachtet.

Architekturprinzipien und Konzept

Die gesamte Architektur einer ROA basiert auf dem Verständnis des Begriffs Ressource. Dabei lassen sich vier Konzepte identifizieren (vgl. [RR07]): die **Ressourcen** an sich, die **Namen** von Ressourcen, die **Repräsentation** einer Ressource sowie die **Verknüpfungen** zwischen den einzelnen Ressourcen. Darüber hinaus werden bei REST vier Aspekte beachtet, die den Aufbau einer Systemarchitektur mittels REST stark prägen. Diese sind die **Adressierbarkeit** (*Addressability*), die **Zustandslosigkeit** (*Statelessness*), die **Verbundenheit** (*Connectedness*) und schließlich die Verwendung **einheitlicher Schnittstellen** (*uniform interfaces*). In Abbildung 2.20 ist eine Ontologie dargestellt, die den Zusammenhang der einzelnen Konzepte darstellt.

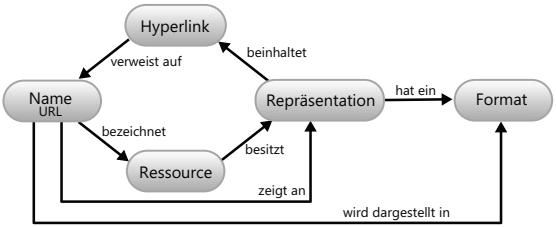


Abbildung 2.20: Ontologie der REST-Konzepte.

Ressourcen

Eine Ressource im Sinne von ROA wird in [RR07, S. 81] folgendermaßen charakterisiert:

> „Usually, a resource is something that can be stored on a computer and represented as a stream of bits: a document, a row in a database, or the result of running an algorithm."

Damit ist gemeint, dass eine Ressource im Grunde für jedes digitale oder sogar physikalische Objekt stehen kann. Da eine ROA strikt auf dem HTTP-Protokoll aufbaut, sind Ressourcen über ein Netzwerk – sei es Inter- oder Intranet – erreichbar. Wenn man beispielsweise das gesamte WWW betrachtet, stellt sich jede Domäne als Baum dar und bietet Zugriff auf die einzelnen Ressourcen über eindeutige URL (vgl. Abbildung 2.21). Ein Hyperlink auf eine Ressource (URL) wird hierbei als *Deep Link* bezeichnet.

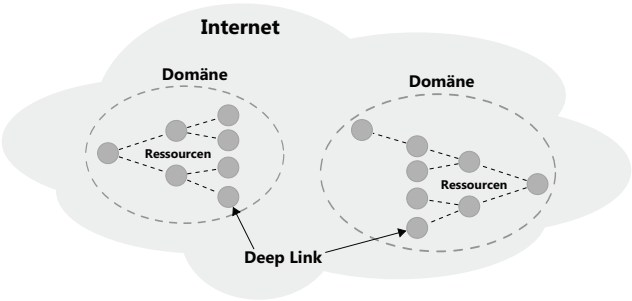


Abbildung 2.21: Vereinfachter Aufbau des WWW.

Namen von Ressourcen

Jede Ressource ist durch mindestens einen Namen ansprechbar. Hierbei wird der Name durch eine URL ausgedrückt. Eine Standard-konforme URL für die Verwendung über HTTP folgt dabei dem schematischen Aufbau des Listings 2.11. Das Schema ist für die Verwendung von REST grundsätzlich *http* und die Autorität der Name einer Domäne oder eine IP-Adresse. Die im Beispiel bezeichnete Ressource soll eine Rechnung des Kunden mit der Identifikationsnummer 12 im XML-Format darstellen.

Ein wichtiger Punkt sollte bei der Namensgebung von Ressourcen auch die Kontinuität spielen. Dabei darf nicht außer Acht gelassen werden, dass eine Systemarchitektur gerade durch Links auf Ressourcen aufgebaut werden kann. Sollten sich diese Links, beispielsweise durch das Verändern eines Ressourcennamens, häufig ändern, wird der Aufbau einer stabilen Architektur unmöglich oder zumindest sehr erschwert. Daher sollte man bei der Namensgebung von Ressourcen sinnvolle Hierarchien unter der Verwendung eindeutiger Bezeichner aufbauen (vgl. [BL98]).

```
URL = Schema ":" Autorität "/" Pfad [ "?" Parameter ]

# Beispiel:
 http://www.domain.de/kunde/12/rechnung?xml
 \___/  \___________/\_________________/ \_/
   |           |              |           |
 Schema     Autorität       Pfad      Parameter
```

Listing 2.11: Schematischer Aufbau einer URL.

Repräsentationen von Ressourcen

Eine Ressource bezeichnet zunächst ein definiertes Objekt, das "Was". Die Darstellung der Ressource ist damit aber noch nicht vorgegeben. Das „Wie", also die Repräsentation der Ressource, kann dabei vielfältige Formen annehmen. Die Literatur versteht unter der Repräsentation einer Ressource auch „ [...] eine nützliche Information über den Zustand einer Ressource." (frei übersetzt aus [RR07, S. 91]). Als Beispiel sei hier wieder die Kundenrechnung angeführt: Diese existiert als Ressource, kann aber in der Repräsentation unterschiedlich ausfallen. Das Format der Ausgabe kann in diesem Fall beispielsweise ein Daten-orientiertes XML-Dokument, eine HTML-Darstellung für den Browser oder ein *Portable Document Format* (PDF)-Dokument sein. Üblich ist hierbei das Verwenden des ersten Parameters nach dem Pfad einer URL. Konkret würden die genannten Ausgabeformate wie folgt aufgerufen werden:

- **XML-Dokument**: `www.domain.de/kunde/12/rechnung?xml`
- **HTML-Dokument**: `www.domain.de/kunde/12/rechnung?html`
- **PDF-Dokument**: `www.domain.de/kunde/12/rechnung?pdf`

Hyperlinks und Verbundenheit

Oft sind Repräsentationen von Ressourcen nicht nur reine Datenwerte, es können damit auch Strukturinformationen wie in HTML- und XML-Dokumenten ausgeliefert werden, die Verknüpfungen zu anderen Ressourcen in sich tragen (so genanntes *Hypermedia*). Verknüpfungen in diesen Datenformaten (*Hyperlinks*) bieten in Verbindung mit URL die Möglichkeit des Auffindens und Verknüpfens von Ressourcen. Dadurch sind keine Verzeichnisse notwendig, die

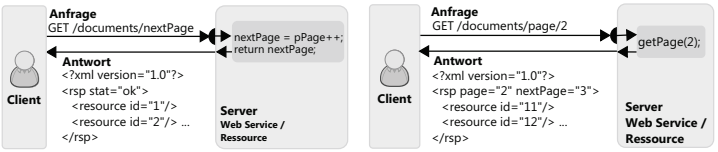
eine Liste von verfügbaren Ressourcen zentral zur Verfügung stellen. Dieses Verlinken von Ressourcen durch die Nutzung von Hyperlinks hat sich seit Jahren im WWW bewährt und ermöglicht das Navigieren von Ressource zu Ressource. In [RR07] wird diese Eigenschaft von REST als Verbundenheit bezeichnet. Dieses Verknüpfen und Bereitstellen von Daten im WWW wird als *Linked Data* bezeichnet und empfiehlt – im Hinblick auf das semantische Web (*Semantic Web*) – zusätzlich die Beschreibung der Ressourcen mittels *Resource Description Framework* (RDF) oder *SPARQL Protocol and RDF Query Language* (SPARQL) (vgl. [BHBL09, BLCC+06]).

Adressierbarkeit

Im Zusammenhang mit einer ROA spricht man von Adressierbarkeit, wenn sich die Ressourcen des Systems direkt aufrufen lassen. Man kann hierbei eine URL angeben und diese beispielsweise als Verknüpfung oder als Dateneingabe für andere Systeme verwenden. Eine Anwendung ist daher adressierbar, wenn Teile der Daten dieser Anwendung als Ressourcen veröffentlicht werden (vgl. [RR07]).

Zustandslosigkeit

Das REST zugrunde liegende HTTP-Protokoll ist zustandslos. Dies bedeutet, dass jeder Aufruf einer Ressource isoliert abläuft und daher alle vom Server für diesen Aufruf benötigten Informationen enthalten muss. Es ist also nicht möglich, mit originären Methoden des HTTP-Protokolls auf einen zuvor vorhandenen Zustand zuzugreifen. Dieses Merkmal kommt aber auch wieder der Adressierbarkeit zugute, da eine eindeutige URL ohne das Wissen eines Zustands des aufgerufenen Systems zu jedem Zeitpunkt direkt ausgeführt werden kann. Als Beispiel sei die Repräsentation einer Listenansicht von 10 Treffern eines Suchergebnisses per Ressource aufrufbar, die per Link seitenweise zu durchblättern ist. Der Zustand wird im Beispiel für das Aufrufen einer spezifischen Seitenzahl benötigt. Wenn der Zustand gespeichert werden kann, führt ein Aufruf des Services mittels `/documents/nextPage` tatsächlich zu dem gewünschten Ergebnis, der Anzeige der nächsten Seite, da innerhalb des Servers die bekannte Variable `pPage` inkrementiert wird (siehe Abbildung 2.22(a)). Im zustandslosen Fall des Aufrufs einer Ressource mittels HTTP muss die Seitenzahl explizit übergeben werden. Der Aufruf `/documents/page/2` führt daher auf dem Server zur Ausgabe der Ressource der 2. Seite der Dokumente. Man beachte hier, dass innerhalb der zurückgegebenen Antwort explizit die aktuel-



(a) Zustandsbehafteter Aufruf (b) Zustandsloser Aufruf

Quelle: angelehnt an [Rod08].

Abbildung 2.22: Beispiel für Zustandslosigkeit eines HTTP-Aufrufs.

le und der Verweis auf die nächste Seite mitgeliefert wird. So kann der nächste Aufruf ebenfalls wieder eine Seitenzahl verwenden (siehe Abbildung 2.22(b)).

Einheitliche Schnittstellen

REST versteht jede URL im Internet als Ressource und sieht die folgenden vier Operationen des HTTP-Standards (siehe Abschnitt 2.2.2) für die Interaktion mit den Ressourcen vor: `POST`, `PUT`, `GET`[24] und `DELETE`. Damit werden die grundlegenden CRUD[25]-Operationen ermöglicht, die auch in jeder Datenbank für das Arbeiten mit Datensätzen definiert sind. REST definiert daher keine Methoden, sondern sieht vor, dass der Name einer URL möglichst selbsterklärend und im Sinne der Zustandslosigkeit vollständig ist. Der Integrationsstil ist hierbei normalerweise als RPC-Aufruf zu verstehen und verwendet grundsätzlich das Nachrichtenaustauschmuster Request-Response, also Anfrage und Antwort, da auf eine HTTP-Anfrage immer eine Antwort erfolgt.

In Abbildung 2.23 ist eine vereinfachte REST-Architektur dargestellt, die zeigt, wie eine Domäne (hier *domain.de*) und deren Ressourcen dargestellt werden können. Zu sehen sind zwei Ressourcenarten: Kunde und Produkt. Beide lassen sich als URL darstellen und können ein oder mehrere Kindelemente besitzen. Im Beispiel ist gezeigt, wie die Ressource „Rechnungen des Kunden mit der Nummer 12" durch die URL `http://www.domain.de/kunde/12/rechnungen` aufgerufen werden kann. Durch eine GET-Anfrage über das HTTP-Protokoll wird dem HTTP-Client eine Liste mit Rechnungen als

[24] GET und HEAD sind idempotente Methodenaufrufe; d. h. es werden keine Veränderungen der Ressource vorgenommen, egal, wie oft die Methode aufgerufen wird.

[25] ***C**reate:* Erstellen, ***R**etrieve*: Anfragen, ***U**pdate*: Ändern und ***D**elete*: Löschen

Ressourcen-Repräsentation zurückgeliefert.[26] Eine weitere Anfrage an `http://www.domain.de/produkt/4` liefert in diesem Beispiel alle Produktdaten des Artikels mit der Nummer 4 zurück.

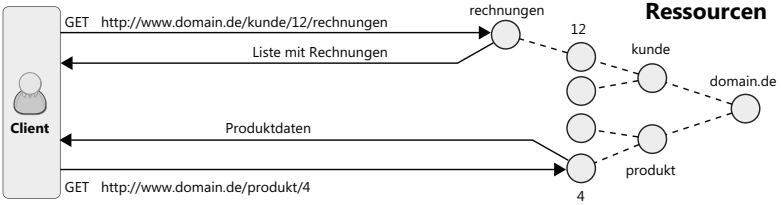


Abbildung 2.23: Das REST-Konzept.

Technologien und Standards

Im Gegensatz zur SOA lässt sich das Konzept einer ROA nicht unabhängig von Standards erläutern, da diese grundsätzlich auf REST und damit HTTP aufbaut. Diese beiden Standards bilden auch bereits den Kern einer ROA. An dieser Stelle wird aber zusätzlich noch JSON als Kernkomponente mit aufgenommen, da damit eine sehr schlanke und verbreitete Datenrepräsentation realisiert werden kann. In einer darüber hinaus gehenden ersten Ausbaustufe lassen sich austauschbare Standards, wie beispielsweise *HTTP-Authentifizierung (HTTPAuth)*, *OpenID* oder *OpenAuth* für die Sicherung von Ressourcen und *Web Application Description Language (WADL)* für die Maschinen-lesbare Beschreibung von REST-Schnittstellen verwenden (vgl. Abbildung 2.24). Im Folgenden wird JSON, WADL und, als einfachste Möglichkeit der Sicherung einer Ressource, HTTPAuth erläutert.

[26] Wie die Repräsentation konkret aussieht, also beispielsweise als HTML, XML, JSON oder Excel-Spreadsheet ist durch die Unabhängigkeit von Ressource und deren Repräsentation unbedeutend.

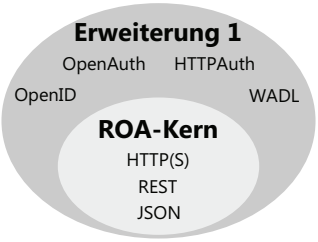


Abbildung 2.24: Standards und Technologien im ROA-Kontext.

JavaScript Object Notation

JSON – das Akronym steht für JavaScript Object Notation – ist ein schlankes Format für die Speicherung semistrukturierter Daten in für Menschen lesbarer Form. Trotz des Namensbestandteils „JavaScript" ist JSON nur indirekt mit JavaScript verwandt, da es eine Untermenge des ECMA Sprachstandards (vgl. [Eur09]) ist, auf dem u. a. auch JavaScript basiert, und daher die gleichen Sprachkonstrukte verwendet. JSON ist ein reines Textformat, welches komplett unabhängig von Programmiersprachen ist, sich aber an gängige Konventionen verbreiteter Programmiersprachen hält. JSON baut auf zwei Eigenschaften auf:

- **Name/Wert-Paare (object)**: Diese stellen eine gängige Form der Speicherung von Werten in diversen Programmiersprachen dar. Sie werden beispielsweise für die Speicherung von Objekten, Strukturen, Hash-Tabellen oder assoziativen Arrays verwendet.
- **Liste (array)**: Diese zeichnet sich durch eine geordnete Menge an Werten aus. Dieses Konzept wird in Programmiersprachen als Array, Vektor, Liste oder Sequenz bezeichnet.

Eine gültige JSON-Beschreibungsdatei wird demnach aus Objekten und Arrays zusammengesetzt, die aus Zeichenketten (*strings*) und Werten (*values*) zusammengebaut werden. Ein Wert kann hierbei durch die folgenden Elemente repräsentiert werden: Zeichenkette (*string*), Nummer (*number*), Objekt (*object*), Array (*array*), das Boolsche Wahr (*true*), das Boolsche Falsch (*false*) und

Null (*null*). Eine formale Beschreibung der JSON-Grammatik ist als erweiterte Backus-Naur Form (EBNF) in Listing 2.12 beschrieben.

```
#Ein Objekt enthält 0 bis n Paare (pair)
object   = '{', [[pair], [{',', pair}], '}';
#Ein Paar ist ein Name/Wert-Paar
pair     = string, ':', value;
#Ein Array besteht aus 0 bis n Werten (value)
array  = '[', [[value], [{',', value}], ']';
#Ein Wert kann eine Zeichenkette, eine Nummer, ein Objekt,
#ein Array, wahr, falsch oder null sein.
value    = string | number | object | array | true | false
         | null;
#Eine Zeichenkette besteht aus 0 bis n Zeichen (char)
string   = '"', [{char}], '"';
#Ein Zeichen ist ein Unicode-Zeichen (mit Ausnahme von
#Kontroll-Zeichen, dem Backslash und Anführungszeichen)
#oder eine andere spezielle Zeichenkombination,
#wie beispielsweise dem Newline-Operator \n
char     = Unicode-character | \" | \\ | \/ | \b | \f | \n |
         \r | \t | \u, four-hex-digits;
#Eine Nummer kann die Formate Integer, Float, etc.
#durch die verschiedenen Elemente annehmen
number   = ['-'], "0"| (digit1-9, [{digit}]),['.', digit,
         [{digit}]],['e'|'E','+'|'-',digit, [{digit}]];
```

Listing 2.12: EBNF des JSON-Formats.

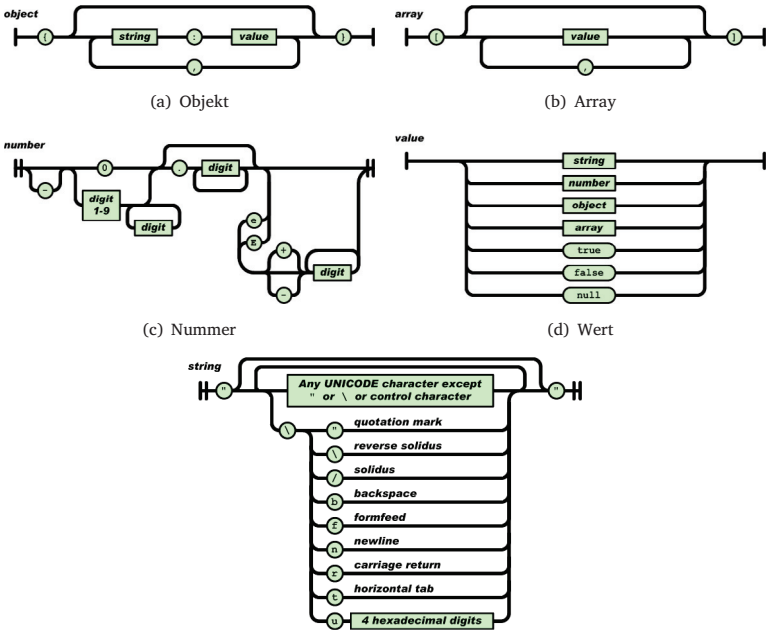
Zum weiteren Verständnis sind die zentralen Grammatik-Elemente der EBNF-Darstellung in den Syntax-Diagrammen der Abbildung 2.25 dargestellt. Im Folgenden wird ein Beispieldatensatz verwendet, um die Elemente des JSON-Formats zu präsentieren. Listing 2.13 zeigt ein Objekt, welches einen Service beschreibt. Durch die Name/Wert-Paare von Zeile 2-6 werden der *Name*, der *Typ* und der *Endpunkt* des Services beschrieben. Darauf folgt ein Objekt mit Namen *Methoden*. Dieses enthält wiederum zwei verschachtelte Objekte *Heute* und *Vorhersage*, die jeweils für einen Methodenaufruf stehen und daher den spezifischen *Endpunkt* und die zu übergebenden *GET-Parameter* darlegen. Die Parameter (Zeile 14 und 20) *Postleitzahl* (Plz) und *Tage* werden als Arrays durch eckige Klammern angegeben.

JSON erfreut sich im Bereich der Web 2.0-Anwendungen – insbesondere in Verbindung mit JavaScript – großer Beliebtheit, da es zum einen sehr leicht

```
{
  #Name/Wert-Paare
  ”Name”:”Wetter-Service”,
  ”Typ”: ”REST”,
  #Sonderzeichen wie der Slash werden maskiert
  ”Endpunkt”:”http:\/\/service.beispiel.de\/wetter”,
  #Objekt mit weiteren Unterobjekten
  ”Methoden”:
  {
    ”Heute”:
    {
      ”Endpunkt”: ”http:\/\/service.beispiel.de\/wetter\/heute
          ”,
      #Array mit einem Wert
      ”GET-Parameter”:[”Plz”]
    },
    ”Vorhersage”:
    {
      ”Endpunkt”: ”http:\/\/service.beispiel.de\/wetter\/
          vorhersage”,
      #Array mit zwei Werten
      ”GET-Parameter”:[”Plz”,”Tage”]
    }
  }
}
```

Listing 2.13: Beispiel eines JSON-Objekts.

lesbar ist, die Struktur weniger Overhead als vergleichbare XML-Strukturen aufweist und es einfach zu parsen ist. Darüber hinaus ist ein JSON-Objekt gleichzeitig ein gültiges JavaScript-Objekt, was die Verwendung innerhalb von JavaScript zusätzlich bestärkt. Viele Programmiersprachen bringen schon De-/Encodierer für JSON mit und erlauben daher die Verwendung des Konzepts. JSON geht auf Douglas Crockford, einem Mitarbeiter von Yahoo!, zurück (vgl. [Cro06b]), der das Konzept von JSON bereits einige Jahre vor dem Web 2.0-Boom vorstellte und in einem RFC 4627 (vgl. [Cro06a]) beschrieb. Darin wird unter anderem ein eigener MIME-Type (*application/json*) vorgeschlagen, um einem Browser das Datenformat für die Übertragung eines JSON-Dokuments zu übermitteln. Aber ohne das starke Interesse an JavaScript-Anwendungen der letzten Jahre hätte JSON, trotz sinnvoller Verwendungsmöglichkeiten, vermutlich nicht eine so große Beachtung erhalten.



(a) Objekt (b) Array

(c) Nummer (d) Wert

(e) Zeichenkette

Quelle: `www.json.org`

Abbildung 2.25: Syntax-Diagramme der JSON-Grammatik.

Web Application Description Language

Durch die zunehmende Anzahl Web-basierter Schnittstellen und HTTP-basierender Anwendungen im Internet, deren Schnittstellenbeschreibungen anfangs zumeist nur durch unstrukturierten Freitext dokumentiert wurden, entwickelte Hadley im Jahr 2006 eine Beschreibungssprache für Services auf XML-Basis (vgl. [Had06]). Zum damaligen Zeitpunkt ließ sich WSDL für diesen Zweck nicht einsetzen, da keinerlei Möglichkeit bestand, HTTP-Methoden innerhalb der Beschreibung zu spezifizieren.[27] Somit wurde eine exakt auf die Bedürfnisse der Web API ausgerichtete Definitionssprache aus der Taufe gehoben, die sich bestens und ohne übermäßig viel Aufwand für die Beschreibung von REST-Services innerhalb einer ROA eignet (vgl. [RR07, TMK+08]). Seit August 2009 ist die WADL auch als *Member Submission* bei dem W3C aufgeführt (vgl. [Hed09]).

Im Folgenden werden einige wichtige Elemente eines WADL-Dokuments erläutert und anschließend an einem Beispiel verdeutlicht. Der konkrete Aufbau lässt sich dabei durch ein *XML Schema Definition* (XSD)-Dokument (siehe Anhang A) ausdrücken. Ein WADL-Dokument besitzt genau einen Wurzelknoten mit der Bezeichnung `<application>`, der die folgenden Kindelemente beinhaltet:

- 0 bis n `<doc>`-Elemente: Jeder Knoten des WADL-Dokuments kann ein Element für dessen Beschreibung enthalten. Sollte dieser Knoten enthalten sein, so muss er die Attribute `xml:lang` zur Definition der Sprache des Elements, sowie das Attribut `title`, welches den Beschreibungstext aufnimmt, enthalten.

- Ein optionales `<grammars>`-Element: Dieses Element kann verwendet werden, um XML-Namensräume zu definieren, die innerhalb des Dokuments verwendet werden. Diese werden durch ein `<include>`-Element eingebunden.

- 0 bis n `<resources>`-Elemente: Diese Elemente bilden den Kern des WADL-Dokuments und dienen als Container für die Beschreibung von

[27] Seit der Einführung von WSDL 2.0 (vgl. [W3C07d]) im Jahr 2007 können auch HTTP-Methoden spezifiziert werden und ermöglichen damit den Einsatz in einer ROA.

konkreten Services (Ressourcen). Hierin werden durch die Verwendung des Elements `<resource>` konkrete Web API beschrieben.

Die Beschreibung einer Ressource beinhaltet zunächst ein `<method>`-Element, welches die HTTP-Methode des Aufrufs definiert. Die Elemente darunter beschreiben das Anfrage-Format (`<request>`) und die möglichen Antworten mit deren HTTP-Statuscode (`<response>`). Die Anfrage wird hierbei weiter unterteilt in einzelne `<param>`-Elemente, die jeweils durch die Angabe von Attributen den Namen (`name`), den Typ (`type`), den Stil (`style`) sowie einen optionalen Standardwert (`default`) eines Parameters beschreiben. Im Antwortteil der Beschreibung wird die Art der Repräsentation definiert: Diese enthält einen Medientyp (`<mediaType>`) und ein Element (`element`), welches beispielsweise durch ein eingebundenes XSD-Dokument im `<grammars>`-Teil eingebunden wurde.

Das Beispiel (siehe Listing 2.14) beschreibt eine Web-Schnittstelle, die eine rudimentäre Suche ermöglicht. Nach der Kurzbeschreibung des Services in Zeile 7 wird die Basis-URL der Ressourcen angegeben (`http://api.domain.de/`; Zeile 8). Die einzige Ressource dieser Schnittstelle ist unter dem Pfad `search` zu erreichen (Zeile 9) und erwartet einen `GET`-Aufruf (Zeile 10) mit den folgenden Parametern:

- **querystring**: Ein obligatorischer Parameter für den Suchbegriff (Zeile 12-14), mit einer Kurzbeschreibung des Parameters (Zeile 13).
- **page**: Ein optionaler Integer-Wert für die Angabe der anzuzeigenden Seite, mit einem Standardwert von 1 (Zeile 15).
- **language**: Ein optionaler Parameter für die Sprache der Suchausgabe (Zeile 16).

Der einzige Rückgabewert ist eine Repräsentation in Form eines XML-Dokuments (Zeile 19) mit dem HTTP-Statuscode 200 (Zeile 20), wobei hier zur Steigerung der Übersichtlichkeit bewusst keine Fehlermeldung definiert wurde.

HTTP Authentifizierung

Eine Methode der Authentifizierung eines Nutzers beim Zugriff auf eine Ressource innerhalb von ROA bietet die HTTP-Authentifizierung (vgl. [FHBH+99]).

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="wadl.xsl"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
 xsi:schemaLocation="http://research.sun.com/wadl wadl.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:d="urn:domain:api" xmlns="http://research.sun.com/wadl
    /2006/10">
 <doc xml:lang="de" title="Beispiel Service">Dies ist ein
    Beispiel-Service.</doc>
 <resources base="http://api.domain.de/">
    <resource path="search">
       <method name="GET" id="search">
          <request>
             <param name="querystring" type="xsd:string" style=
                "query" required="true">
                <doc xml:lang="de" title="Query">Hier wird ein
                   Suchbegriff übergeben.</doc>
             </param>
             <param name="page" type="xsd:int" style="query"
                default="1"/>
             <param name="language" style="query" type="
                xsd:string"/>
          </request>
          <response status="200">
             <representation mediaType="application/xml"
                element="d:ResultSet"/>
          </response>
       </method>
    </resource>
 </resources>
</application>
```

Listing 2.14: Beschreibung eines Services mit WADL.

Diese Methode wurde extra für die Verwendung mit HTTP entwickelt und ist in jedem Browser explizit vorhanden. Sie existiert in zwei Versionen, der *Basic*- und der *Digest*-Variante. Diese beiden unterscheiden sich durch die Art, wie die Zugriffsdaten, eine Kombination aus Nutzername und Passwort, übertragen werden. Während in der Basic-Variante die Nutzerdaten unverschlüsselt im Klartext übertragen werden, kommt bei der Digest-Version eine Hash-

Funktion zum Einsatz, um das Übertragen des Passworts im Klartext zu vermeiden.

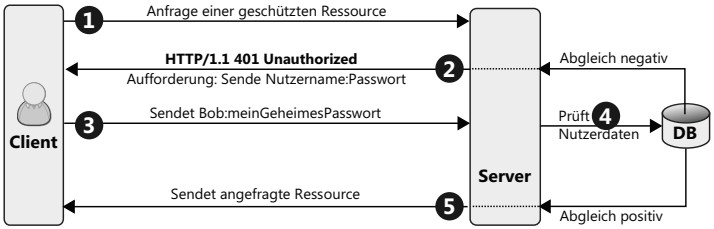


Abbildung 2.26: Der Ablauf einer HTTP Basic Authentifizierung.

Eine Authentifizierung bei einem Aufruf im Browser läuft wie folgt ab (siehe Abbildung 2.26). Im ersten Schritt sendet der Browser eine Anfrage an den Server. Dieser erkennt, dass die angefragte Ressource durch Basic Auth geschützt ist und fordert in einem zweiten Schritt den Browser durch das Senden des HTTP-Statuscodes 401 und der Nachricht „Unauthorized" dazu auf, sich zu authentifizieren. Nun zeigt der Browser im Normalfall einen Dialog an, der den Nutzer auffordert, seine Nutzername-Passwort-Kombination (die sogenannten *Credentials*) für die geschützte Ressource einzugeben.[28] Im dritten Schritt sendet der Browser die eingegebenen Daten an den Server, der diese im nächsten Schritt mit einer Liste der zugelassenen Nutzer, im Beispiel einer Datenbank, abgleicht. Sollte der Abgleich erfolgreich verlaufen, wird die gewünschte Ressource in einem fünften Schritt an den Client gesendet. Andernfalls wird der HTTP-Statuscode 401 gesendet (Schritt 2), um den Browser erneut zur Nutzerdateneingabe aufzufordern. Benötigt man nur Sicherheit auf Basis von Nutzername-Passwort-Kombinationen, ist insbesondere für REST die Basic Authentifizierung von HTTP bestens geeignet. Hierbei kommen die Unterschiede zwischen Browser und Anwendungsprogrammierung zum Tragen. Während der Dialog zur Eingabe der Credentials im Browser oft störend auffällt[29], lassen sich bei der Programmierung eines HTTP-Aufrufs Nutzername und Passwort im Header der HTTP-Anfrage mitschicken. Durch die Verwen-

[28] Nach dem ersten Anmelden merkt sich der Browser die Anmeldedaten so lange, bis er geschlossen wird. Daher wird bei jedem erneuten Zugriff auf die geschützte Ressource keine Authentifizierung verlangt.

[29] Dieser lässt sich nicht in eine HTML-Login-Maske auslagern oder eleganter lösen; es erscheint grundsätzlich eine Browser-spezifische Anmeldemaske.

dung der etwas sichereren Digest-Version oder durch Verschlüsseln der gesamten Verbindung durch SSL lässt sich auch hier das Sicherheitsrisiko, beispielsweise durch dem Datentransfer lauschende Angreifer, minimieren.

2.3.3 Gegenüberstellung von SOA und ROA

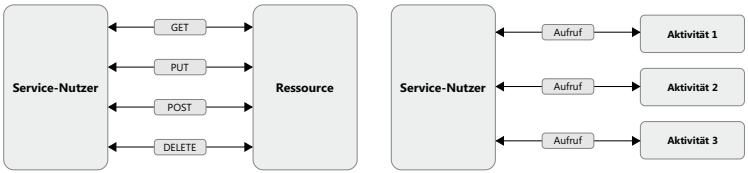
Die Diskussion, welche Architektur für die Realisierung von verteilten Systemen und deren Schnittstellen verwendet werden soll, ist seit dem Aufkommen der ROA neu entbrannt. Im Folgenden wird ein systematischer Vergleich von Service-orientierten und Ressourcen-orientierten Architekturen in drei Kategorien vorgenommen, um die beiden Ansätze gegenüberzustellen. Die Kategorien sind die *Architektur-*, *Konzept-* und *Technologieentscheidungen*, die bei dem Einsatz der jeweiligen Konzepte anwendbar sind. Im Anschluss daran werden potenzielle Stärken und Schwächen der beiden Konzepte zusammengefasst. Bei diesem Vergleich wird der ESB als Architekturkomponente innerhalb einer SOA als gegeben vorausgesetzt, obgleich dieser im SOA-Konzept nicht zwingend notwendig ist.

Architekturentscheidungen

Um vergleichende Aspekte zu definieren, muss man die Softwarearchitekturen und deren Prinzipien zunächst genau betrachten. Auf der Ebene der Architektur lässt sich die Nutzung von Protokollen zum Transport von Nachrichten, die lose Kopplung von Services, die Skalierbarkeit der Architektur sowie der Integrationsstil der Services im verteilten System als Merkmal identifizieren. Im Folgenden werden diese Punkte gegenübergestellt und in Tabelle 2.5 zusammengefasst.

- **Protokollverwendung**: In beiden Konzepten wird HTTP als Protokoll verwendet. Während die Verwendung in einer ROA aber tatsächlich als Anwendungsprotokoll erfolgt, also die Nutzung der vorgegebenen Methoden des Protokolls auch der vorgesehenen Verwendung entspricht, wird in einer SOA HTTP lediglich als Transportprotokoll verwendet (vgl. [PZL08]). Der implizit vorhandene Funktionsumfang wird dabei vernachlässigt und meist nur die `POST`-Methode für die Übertragung von SOAP-

Nachrichten verwendet.[30] Snell beschreibt die Verwendung der vier HTTP-Methoden als Anwendungsprotokoll im Gegensatz zum Aufruf einer Aktivität bei einer SOA, die er als Aktivitäts-orientiert bezeichnet (siehe Abbildung 2.27).



(a) Zugriff auf Ressource (b) Zugriff auf Service/Aktivität

Quelle: [Sne04]

Abbildung 2.27: Vergleich von Ressourcen- und Service-Zugriff.

- **Lose Kopplung**: Die lose Kopplung von beteiligten Softwarekomponenten ist ein Aspekt, der gerade im SOA-Kontext immer wieder hervorgehoben wird. Es lassen sich hierbei drei Dimensionen identifizieren: Verfügbarkeit, Auffindbarkeit sowie Weiterentwicklung eines Services. Die Verfügbarkeit spielt in verteilten Systemen eine sehr große Rolle, da sich beispielsweise Geschäftsprozesse aus vielen kleinen Services zusammensetzen können und bei Ausfall einer Komponente ein gesamter Prozess zum Stehen kommen kann. Diesem Problem begegnet man in einer SOA mit dem Konzept des ESB, der als Middleware vermittelt und im Falle eines nicht erreichbaren Services den Aufruf in eine Warteschlange einreiht, um diesen zu einem späteren Zeitpunkt erneut auszuführen. Die ROA beruht auf HTTP und somit auf direkten, synchronen Aufrufen einer Ressource. Fällt hier ein Service aus, bekommt der Aufrufer keine Antwort und das System kann nicht weiterarbeiten. Im günstigsten Fall lässt sich ein HTTP-Aufruf aus dem Cache eines Servers laden, dies ist aber keine Garantie für Verfügbarkeit oder Aktualität der zwischengespeicherten Ergebnisse. Die Auffindbarkeit von Services ist für beide Konzepte gegeben. In einer SOA wird beispielsweise ein Service-Repository, wie das UDDI oder *WS-Inspection* verwendet, in einer ROA ist durch die Verwendung von Hyperlinks, Hypermedia und

[30] Oft wird dies auch als Vorteil deklariert, da man neben HTTP ebenso auch andere Protokolle, wie beispielsweise FTP oder SMTP zum Senden von SOAP-Nachrichten verwenden kann.

durchsuchbaren Web-Verzeichnissen eine Suche nach Ressourcen kein Problem (siehe 2.3.2). Bei der Frage der Evolution bestehender Services ist dies innerhalb einer ROA implizit festgelegt. Durch die Nutzung von REST gibt es für jede Ressource genau vier Methoden, die vier CRUD-Operatoren. Das führt dazu, dass die Methodensignatur eines Services sich nicht ändern kann. Das Interface bleibt in jedem Fall stabil. Bei der Weiterentwicklung eines Web Services ist dies keinesfalls gegeben. Bereits das Ändern eines Parameters kann dafür sorgen, dass der aufrufende Client nicht mehr funktioniert. Innerhalb einer SOA muss also sehr stark auf Versionierung von Web Services und die damit verbundene Abwärtskompatibilität geachtet werden. (vgl. [PZL08]).

- **Skalierbarkeit**:

  Der Anspruch der Skalierbarkeit ist unmittelbar mit der Architektur eines verteilten Systems verbunden und lässt sich in einer ROA als implizit vorhanden darstellen. Durch den Aufbau des Internets und die bewährte Verwendung von ausgefeilten Caching-Mechanismen und performanten Routing-Algorithmen für HTTP in Verbindung mit der Adressierbarkeit von Ressourcen ist eine ROA automatisch gut skalierbar. Innerhalb einer SOA kann der Zustand der Skalierbarkeit zwar hergestellt werden, ist aber nicht implizit durch die Architektur vorhanden. Der Skalierbarkeit stehen hier die unterschiedlichsten Schnittstellen und das dynamische Auffinden von Services entgegen. Auch die Verwendung eines ESB erzeugt weitere Komplexität bezüglich der Skalierbarkeit (vgl. [BMM09])

  .

- **Integrationsstil**: Als letzter Vergleichspunkt wird hier der Integrationsstil aus Architektursicht herangezogen. Dieses Merkmal ist von Pautasso et. al. in den konzeptionellen Aspekten aufgeführt, wird hier aber als zur Architektur zugehörig erachtet. Eine SOA unterstützt, unter der Prämisse der Nutzung eines ESB, gleich drei Arten verteilter Systeme. Durch die Nutzung des ESB wird der *zentrale Nachrichtenkanal* verwendet, auch direkte Aufrufe von Web Services sind möglich und stellen den *entfernten Funktionsaufruf* dar. Es ist hierbei nicht zwingend vorgeschrieben, dass durch den Einsatz eines ESB auch alle Service-Aufrufe über diesen gesteuert werden müssen. Als *zentraler Dateispeicher* ist der ESB

Tabelle 2.5: Vergleich der architektonischen Aspekte.

Entscheidungen	ROA / REST	SOA / WS-*
Protokollverwendung		
HTTP als Anwendungsprotokoll	✓	-
HTTP als Transportprotokoll	-	✓[a]
Lose Kopplung		
Verfügbarkeit	-	✓
Auffindbarkeit	✓	✓
Weiterentwicklung von Services	✓[b]	-
Skalierbarkeit		
Implizit vorhanden	✓	-
Integrationsstil		
Gemeinsame Datenbank	-	-
Zentraler Nachrichtenkanal	-	✓
Entfernte „Funktionsaufrufe“	✓	✓
Zentraler Dateispeicher	-	(✓)

Quelle: angelehnt an [PZL08]

[a] Verwendet lediglich POST, jedoch ab SOAP 1.2 auch GET möglich.
[b] Durch *Uniform Interface* implizit gegeben.

ggf. ebenfalls einsetzbar und bringt in den meisten Produkten auch eine solche Funktion mit sich. Im Gegensatz dazu ist bei einer ROA nur die Systemarchitektur eines entfernten Funktionsaufrufs enthalten. In diesem Fall ist aber eher von Ressourcenaufruf zu sprechen, da der ROA das Konzept der Funktion fremd ist.

Konzeptionelle Entscheidungen

Die Entscheidungen auf konzeptioneller Ebene umfassen die Beschreibung von Funktionalitäten der Services, das Muster des Nachrichtenaustauschs sowie die Datenrepräsentation. Pautasso et. al. nennt darüber hinaus noch weitere Aspekte, die aber entweder nur auf Seiten einer ROA zu finden sind, wie

beispielsweise Namensgebung von URL oder Interaktion mit Ressourcen, oder auf der anderen Seite SOA-spezifische Aspekte betreffen, wie die Aufzählung von Service-Methoden durch unterschiedliche Kriterien. Diese Aspekte wurden daher bei dem folgenden Vergleich nicht berücksichtigt. Eine Übersicht über die konzeptionellen Entscheidungen ist in Tabelle 2.6 dargestellt.

- **Funktionalitätsbeschreibung**: Die Beschreibung der Funktionalität von Ressourcen innerhalb einer ROA ist implizit durch die Verwendung von HTTP vorgegeben. Die Verwendung der vier CRUD-Operationen ist für jede Ressource identisch und bedarf daher für die Verwendung durch Service-Nutzer keiner weiteren Erläuterung. Web Services in einer SOA sind grundsätzlich durch WSDL zu beschreiben und erlauben somit zwei verschiedene Ansätze: *Contract-first* und *Contract-last*. Die beiden Ansätze unterscheiden sich insofern, als bei dem Ersten die Beschreibung anfangs existiert und daraufhin der Programmcode erstellt wird (*WSDL2Code*). Der zweite Ansatz erzeugt aus dem vorhandenen Programmcode eines Services die Beschreibung (*Code2WSDL*).

- **Muster des Nachrichtenaustauschs**: Der Nachrichtenaustausch ist im Falle der ROA durch die Nutzung von HTTP als *Request-Response* vorgegeben. Daher erfolgt grundsätzlich eine synchrone Kommunikation zwischen den beiden Kommunikationspartnern. Jeder Aufruf einer Ressource erwartet zwingend eine Antwort, auch wenn diese ggf. nur in der Rückgabe eines HTTP-Statuscodes besteht. Die Nutzung von Web Services bei einer SOA-Lösung ermöglicht darüber hinaus auch noch die Verwendung von Einweg-Kommunikation (*one-way*). Sie ist gekennzeichnet durch das Ausbleiben einer Antwort bei einer Anfrage. Dies ist für Services sinnvoll, die keine Antwort auf eine Aktion benötigen.

- **Datenrepräsentation**: Für den Rückgabewert der Daten bei einem Funktionsaufruf, der Datenrepräsentation, ist im SOA-Kontext durch die Verwendung von SOAP als Nachrichtenformat ein XML-Schema vorgegeben. Um damit JSON- oder HTML-Daten zu versenden, lässt sich im body der Nachricht ein Textblock (*CDATA*) einbinden. Dem gegenüber ist die Wahl der Repräsentation bei einer ROA beliebig. Es können hier ebenfalls XML-Schemata, aber eben auch beliebig viele andere Repräsentationsformen hinterlegt werden, wie beispielsweise JSON, HTML oder auch direkt Multimediainhalte.

Tabelle 2.6: Vergleich der konzeptionellen Aspekte.

Entscheidungen	ROA / REST	SOA / WS-*
Funktionalitätsbeschreibung		
Implizit durch Protokoll	✓[a]	-
Contract-first (WDSL to Code)	-	✓
Contract-last (Code to WSDL)	-	✓
Muster des Nachrichtenaustauschs		
Request-Response	✓	✓
One-way	-	✓
Datenrepräsentation		
XML	(✓)	✓
Beliebig	✓	-

Quelle: angelehnt an [PZL08]

[a] Durch die Nutzung von HTTP.

Technologieentscheidung

Wenn Entscheidungen bei der Umsetzung eines verteilten Systems durch SOA oder ROA zu technologischen Aspekten anstehen, hat man gerade im Bereich der WS-Standards relativ viel Auswahl. Die Gremien rund um Service-Orientierung haben für viele Anwendungsfälle Standards und Lösungen entwickelt. Eine genaue Aufzählung der technologischen Aspekte ist [PZL08] entliehen, wobei in der vorliegenden Arbeit eine Unterscheidung zwischen Basisaktivitäten und zusätzlichen Aktivitäten in einem verteilten System getroffen wird. Die Basisaktivitäten werden im Folgenden beschrieben und in Tabelle 2.7 zusammengefasst.

- **Auffinden von Services**: Während das Suchen und Finden von Web Services in einer SOA durch einen Verzeichnisdienst wie UDDI oder mittels WS-Inspection möglich ist, sind innerhalb der ROA keine Mechanismen vorgesehen. Eine Möglichkeit besteht darin, die Nutzung von zentralen Web-Verzeichnisdiensten voranzutreiben. Die Web-Seite `www.programmableweb.com` ist ein solches Verzeichnis mit Tausenden von registrierten Web API und Services.

- **Identifizieren von Services**: Ressourcen in einer ROA werden eindeutig durch ihre URL identifiziert. Innerhalb einer SOA ist dies ebenfalls möglich. Des Weiteren kann hier aber auch der Standard *WS-Addressing* zum Einsatz kommen, der Adressdaten in einer SOAP-Nachricht übermitteln und auswerten kann (vgl. [GHR06]).

- **Service-Beschreibung**: Um Services zu beschreiben, wird in einer SOA die WSDL für die Schnittstellendefinition und zusätzlich ggf. XML-Schemata als Erweiterung des Sprachraums des WSDL-Dokuments verwendet. Dieselben Mittel können in einer ROA verwendet werden[31], lassen aber zusätzlich Freiraum für Alternativen, wie beispielsweise WADL oder reine Textbeschreibungen.

- **Nutzbare Protokolle**: In einer ROA kommt nur HTTP als einziges Protokoll zum Einsatz. Einer SOA stehen hier viele Protokolle zur Verfügung. Die einzige Anforderung ist, dass eine SOAP-Nachricht übermittelt werden kann. Dadurch sind gängige Protokolle wie HTTP, SMTP, FTP oder auch spezifische Protokolle wie *Java Message Service* (JMS) oder *Blocks Extensible Exchange Protocol* (BEEP) verwendbar.

In [PZL08] wird ebenfalls noch das Format der Nutzlast als technologisches Entscheidungsmerkmal genannt, und dabei einer SOA nur das SOAP-Format zugestanden. Wie aber bereits bei dem Aspekt der Datenrepräsentation erläutert, lassen sich hier im SOAP-Body auch binäre Daten jedweder Art in einen Textblock einbetten. So lässt sich als äußeres Format der Nutzlast in einer SOA zwar das XML-Format nicht vermeiden, der tatsächliche Inhalt kann aber in beiden Architekturen beliebige Form annehmen.

Neben alltäglichen Aufgaben in einem verteilten System, wie ungesicherte atomare Aufrufe von Funktionen, müssen oft auch komplexere Anwendungsszenarien umgesetzt werden. Um dabei sichere und nachvollziehbare Kommunikation zu ermöglichen, sind erweiterte Funktionalitäten von Nöten. Diese Bereiche werden im Folgenden erläutert und sind in Tabelle 2.8 zusammengefasst.

- **Service-Komposition**: Die Zusammenstellung von Services zu einem ablauffähigen Prozess wird als Service-Komposition bezeichnet und

[31] WSDL ist ab Version 2.0 auch für REST verwendbar.

Tabelle 2.7: Vergleich der technologischen Aspekte der Basisaktivitäten.

Entscheidungen	ROA / REST	SOA / WS-*
Auffinden von Services		
UDDI / WS-Inspection	-	✓
ProgrammableWeb / eigene Methoden	✓	-
Service-Identifikation		
URL	✓	✓
WS-Addressing	-	✓
Service-Beschreibung		
WSDL	ab 2.0	✓
WADL	✓	-
XML-Schema	✓	✓
Textbeschreibung	✓	-
Nutzbare Protokolle		
HTTP	✓	✓
FTP, SMTP, etc.	-	✓

Quelle: angelehnt an [PZL08]

lässt sich für eine SOA zum einen mittels WS-BPEL realisieren, zum anderen können auch eigene Verfahren entwickelt und genutzt werden. Da WS-BPEL seit Version 1.2 auch mit REST-Services umgehen kann, sind die Mittel innerhalb einer ROA nicht beschränkt. Es lassen sich hier WS-BPEL ebenso wie Mashups[32] oder eigene Verfahren verwenden.

- **Sicherheit**: Um eine möglichst hohe Sicherheit bei dem Aufruf von Services zu erreichen, lässt sich die Transportsicherheit erhöhen. Dafür können SOA als auch ROA auf HTTPS zurückgreifen und somit Punkt-zu-Punkt-Verbindungen absichern. Zum Zwecke der Authentifizierung eines Nutzers lassen sich in der ROA offene Standards wie HTTP Basic

[32] Der Begriff „Mashup“ wurde durch das Web 2.0 geprägt und bezeichnet das Zusammenführen zweier Dienste des Internets zu einem nutzenstiftenden neuen Service (vgl. [VH07, Alb08]).

Authentifizierung, OpenAuth oder OpenId einsetzen. Dagegen wird in der SOA der WS-Security Standard verwendet, der neben der Authentifizierung auch das Signieren und Verschlüsseln von Nachrichten oder Teilen von Nachrichten ermöglicht.

- **Zuverlässigkeit**: Die Zuverlässigkeit des Nachrichtenverkehrs, also die Sicherstellung, dass eine Nachricht auch beim Adressaten ankommt, ist in einer ROA nicht gewährleistet. Durch die Nutzung von HTTP wird keinesfalls sichergestellt, dass eine Nachricht auf jeden Fall den Empfänger erreicht. IBM begann hierfür einen eigenen Standard mit Namen *HTTPR* zu entwickeln, hat dies aber abgebrochen (reliable HTTP, vgl. [TPC05]). Die in einer SOA nutzbaren Standards, wie beispielsweise *WS-Reliability* und dessen Nachfolger *WS-ReliableMessaging* (vgl. [OAS09b]), ermöglichen die Nachrichtensicherheit bei der Verwendung von SOAP-Nachrichten.

- **Transaktionen**: Eine Transaktion ist definiert als das Ausführen einer Folge von Operationen, die entweder komplett oder gar nicht durchgeführt werden. Dies dient meist der Sicherung eines konsistenten Zustands. Im Bereich der Datenbankmanagementsysteme ist dies ein alltägliches Problem. Ist dort eine Transaktion gestartet, lässt sich diese durch ein *Rollback* wieder komplett rückgängig machen. Ein häufiges Beispiel im Service-Bereich ist die Buchung einer Reise: Sollte das Hotel bereits gebucht, aber kein Flug mehr verfügbar sein, muss der gesamte Vorgang rückgängig gemacht werden. Durch Transaktionslogik in Service-Kompositionen lässt sich ein solches Verhalten realisieren. Für eine SOA existieren diverse Standards, wie beispielsweise *WS-AtomicTransaction*, *WS-BusinessActivity* oder auch *WS-Composite Application Framework*. Darüber hinaus lassen sich auch eigene Verfahren implementieren, die eine Rückabwicklung von Service-Aufrufen ermöglicht. In einer ROA lassen sich keine der genannten Standards verwenden, hier ist ein eigener Lösungsansatz notwendig.

Stärken und Schwächen der beiden Technologien

Um die Stärken oder Schwächen einer Software-Architektur zu benennen, muss zunächst ein Ziel deklariert werden, welches durch Einsatz der Archi-

Tabelle 2.8: Vergleich der technologischen Aspekte der erweiterten Aktivitäten.

Entscheidungen	**ROA / REST**	**SOA / WS-***
Service-Komposition		
WS-BPEL	ab BPEL 1.2	✓
Mashups	✓	-
Eigene Verfahren	✓	✓
Sicherheit		
Transport (HTTPS)	✓	✓
Authentifizierung	HTTPAuth, OpenID	WS-Security
Zuverlässigkeit (*Reliability*)	?	Diverse WS-Standards
Transaktionen		
WS-AT, WS-BA	-	✓
WS-CAF	-	✓
Eigene Verfahren	✓	✓

Quelle: angelehnt an [PZL08]

tektur erreicht werden soll. Im vorliegenden Fall wird für SOA sowie ROA das Ziel einer lose gekoppelten, verteilten und Unternehmens-übergreifenden Software-Architektur vorgegeben. Im Anschluss daran lassen sich die für die Umsetzung der Architektur vorgesehenen Methoden und Werkzeuge betrachten, um deren Eignung festzustellen. Obwohl sich die Umsetzung einer SOA nicht zwangsläufig durch die Standards wie SOAP und WSDL definiert, dies aber die häufigste Kombination ist, wird im Kontext dieser Arbeit eine SOA als untrennbar mit diesen Standards verbunden angesehen. Eine weitere Beobachtung ist, dass sich manche Defizite einer Architektur nur langsam während der tatsächlichen Anwendung herauskristallisieren und somit nur explorativ identifizierbar sind. Bezüglich der SOA lässt sich auf viele Jahre der Umsetzungserfahrung zurückblicken, wodurch einige Probleme erkannt wurden. Die folgenden Punkte der Stärken und Schwächen sollen als kurze Zusammenfassung von Meinungen und dem Wiedergeben eines Stimmungsbildes dienen. Hierfür werden auch einige Quellen aus Blogs und Foren herangezogen, da insbesondere kritische Stimmen zu SOA und ROA seltener in wissenschaftlicher Forschung oder Literatur zu finden sind.

Stärken einer SOA

Eine der großen Stärken der SOA ist die Unabhängigkeit vom Transportprotokoll beim Versenden und Empfangen von Nachrichten durch die Verwendung des SOAP-Protokolls (vgl. [Haa05]). Dies ermöglicht die Verwendung verschiedenster Protokolle und erreicht dadurch eine Interoperabilität von Services, die gerade in Bereichen mit hoher Heterogenität einen wichtigen Faktor darstellt (vgl. [AES08]). Dies trifft fast auf jedes Unternehmensnetzwerk zu. Die Beschreibung der Services durch WSDL ermöglicht des Weiteren die Abstraktion von spezifischen fachlichen Beschreibungen und erhöht die Nutzbarkeit über Unternehmens-Grenzen hinaus. Komplexe Operationen der Web Services werden dabei hinter Schnittstellenbeschreibungen versteckt (vgl. [MNS05]). Der Einsatz von SLA in Service-Beschreibungen trägt ebenfalls zur Transparenz der Service-Nutzung bei und ermöglicht das Verwenden von (Fremd-)Services, ohne lange Vertragsverhandlungen eingehen zu müssen (*negotiation transparency*, vgl. [AES08]). Natürlich bringt der Einsatz von WSDL, SLA, dem SOAP-Protokoll und weiteren spezifischen WS-Standards im Zusammenspiel eine erhöhte Komplexität mit sich, diese kann aber häufig durch den Einsatz von geeigneten Werkzeugen (Software-Suiten, etc.) gemildert werden (vgl. [BMM09]).

Schwächen einer SOA

Da SOA kein rein IT-getriebenes Thema, sondern vielmehr eine umfassende Unternehmensstrategie darstellt, ist die Einführung einer SOA trotz der Einfachheit einzelner Standards sehr komplex (vgl. [FH07, DJI05]). Insbesondere die Beherrschung des Web Service-Stacks gestaltet sich aufgrund einer Vielzahl an Standards schwierig. Schroth und Kirchhoff meinen dazu, dass „der inzwischen sehr komplexe Web Service Stack [...] für die Entwicklung einer globalen SOA nicht adäquat [sei]", und „mit WSDL beschriebene Web Services [...] für wenig Technik-affine Nutzer schwer auffindbar und benutzbar [seien]" [SK07, S. 56–57]. Der Verwendbarkeit von WSDL wird auch in [HLV07] eine eher schlechte Note attestiert. Eine weitere Schwäche bezüglich der verwendeten Standards ist die durchgängige Verwendung von XML für jede Schicht einer SOA. Dies äußert sich vor allem im hohen Verbrauch der Ressource CPU-Leistung für das Einlesen, Ändern und Erzeugen von einer Vielzahl an XML-Codes (vgl. [AES08]). Hierzu zählen alle WSDL-Beschreibungen, SOAP-Nachrichten und weiteren XML-basierte Standards. Die Beherrschbarkeit der diversen Standards ist zwar – bereits zuvor als Stärke genannt – durch geeignete Software-Unterstützung gewährleistet, kann sich aber gleichzeitig auch durch den *Vendor-lock-in*-Effekt in eine Schwäche verwandeln. Denn die Interoperabilität von Web Services, vor allem bei der Nutzung von zusätzlichen Standards, die über den Basis-Stack hinausgehen, ist zwar innerhalb einer Produktfamilie meist gewährleistet, kann aber bei der Kopplung mit Systemen anderer Hersteller Inkompatibilitäten aufzeigen. Eine weitere Schwäche ist darin zu sehen, dass das aufrufende Programm immer im Voraus Informationen über Methoden eines Web Services und dessen Semantik benötigt, ohne die keine Service-Verwendung stattfinden kann (vgl. [MNS05]). Dies ist als Gegensatz zum REST-Stil zu sehen, in dem, bei einfach strukturierten Ressourcen, durch die vier feststehenden Methoden von HTTP oft keine, oder nur geringe, semantische Beschreibung mehr notwendig ist.

Die oben genannten Schwächen beziehen sich zumeist auf technische Details einer SOA. Da man SOA aber als Strategie sehen sollte, ergeben sich auch auf Seiten der organisatorischen Bewältigung des SOA-Themas Probleme. Eines ist, dass sie oft als die Lösung für Integrationsprobleme genannt wird. Eine SOA stellt aber keine rein technische Lösung dar, sondern ist vielmehr ein Konzept zur Strukturierung der Unternehmens-IT (vgl. [FH07]). Im Grunde soll die SOA dabei nicht nur die Prozesse eines Unternehmens flexibler gestalten,

sondern als größeres Ziel gesehen, nach außen hin Schnittstellen verfügbar machen und der Vernetzung mit weiteren Unternehmensnetzwerken dienen. Wie die Praxis zeigt, werden SOA aber meist nicht global, sondern nur innerhalb von Unternehmen aufgebaut (vgl. [McA05]).

Eine generelle Aussage zu SOA lässt sich basierend auf der Einschätzung von Gartner aus dem Jahr 2005 treffen: Darin wurde prognostiziert, dass im Jahre 2008 circa 80 % aller Entwicklungsprojekte SOA als Basis verwenden würden (vgl. [CFP05]). Ein klarer Beweis, dass sich diese Prognose nicht bewahrheitet hat, lässt sich nicht vorbringen. Es ist aber stark zu bezweifeln, dass so viele Unternehmen SOA als breite Basis einsetzen, zumal echte Erfolgsgeschichten schwer zu finden sind (vgl. [Vos06]). So basierte eine Untersuchung von Praxisansätzen im SOA-Bereich des Jahres 2006 auf einer Auswahl von nur vier Fallstudien aus der Praxis (vgl. [HLs06]). Darüber hinaus lassen sich positive oder negative Stimmen oft nur in Blogs oder Internetportalen finden. So schrieb Anne Thomas Manes, *Research Director* der *Burton Group VP*, Anfang 2009 in ihrem Blog den Beitrag „SOA is Dead; Long Live Services." und ging darin auf die Problematik der SOA und die mittlerweile die SOA überholenden Konzepte, wie Cloud Computing oder SaaS, ein.[33] Äußerungen dieser Art sind zwar immer mit Vorsicht zu genießen, lassen aber durchaus Rückschlüsse auf ein eher verhaltenes Stimmungsbild im Hinblick auf die Umsetzung von Software-Architekturen mittels SOA zu.

Stärken einer ROA

Der REST-Stil, und damit implizit das Konzept einer ROA, ist im Grunde zwar schon seit der Dissertation von Fielding im Jahre 2000 bekannt, hat aber erst in den letzten Jahren einen hohen Bekanntheitsgrad erlangt. Dass dies zum Teil auch am Aufschwung der Internet-Applikationen und dafür geeigneter Systemarchitekturen liegt, scheint offensichtlich zu sein. Die Stärken, die einer ROA zugeschrieben werden, betreffen zunächst vor allem die Vorteile des WWW und der Verwendung von HTTP. Das Interface eines Services (Ressource) ist durch die Methoden von HTTP festgelegt und ermöglicht so die einfache Interaktion mit einer Ressource (vgl. [MNS05]). Durch das Stützen auf gängige Internet-Standards lassen sich alle Vorteile des WWW nutzen, die über Jahre hinweg optimiert wurden. Dies betrifft zum einen schnelle Webserver sowie intelligente Routing- und Caching-Mechanismen der eingesetzten Kom-

[33] Siehe: `http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html`.

ponenten des Internets und damit auch die optimale Voraussetzung für ein (weltweit) verteiltes System. Zum anderen sind Firewall-Problematiken eher in den Hintergrund gerückt, da der Standard-Port für HTTP-Verbindungen, meist der Port mit der Nummer 80, in vielen Firmennetzwerken freigeschaltet ist.

Neben der Eignung für große, verlinkte Systeme (vgl. [GHN09]) ist auch die Skalierbarkeit, die bei verteilten Systemen immer eine große Rolle spielt, durch die Erfahrungen mit Web-Applikationen in vielerlei Hinsicht untersucht und optimiert worden. Bei der Umsetzung eines verteilten Systems mittels REST müssen durch bereits getroffene Grundsatzentscheidungen, wie beispielsweise die Verwendung von HTTP, sehr wenige architektonische Entscheidungen getroffen werden (vgl. [PZL08]). Zusätzlich werden durch die Verwendung von URL und Hyperlinks Repositorys teilweise überflüssig und der Client benötigt keine speziellen Pfadinformationen, die über den initialen Aufruf der URL hinausgehen (vgl. [MNS05]). Auch die Programmierung von Services innerhalb einer ROA ist durch die Verfügbarkeit von HTTP-Clients in den meisten Programmiersprachen und mittlerweile auch vielen existierenden Frameworks leicht umsetzbar. Als Beispiel sei hier die Implementierung von REST-Services mittels PHP genannt (vgl. [RT10]).

Schwächen einer ROA

Aufgrund der relativ neuen Art der Umsetzung eines verteilten Systems müssen viele grundsätzliche Aufgaben innerhalb einer ROA selbst implementiert werden (vgl. [PZL08]). Darunter fallen beispielsweise sicherheitskritische Funktionen, wie Transaktionen, Verlässlichkeit der Datenübertragung sowie Signierung und Verschlüsselung von Nachrichten. Eine Lösung ist die Verwendung von bekannten Methoden und verfügbaren Technologien des WWW (Zertifikate, HTTPAuth, etc.). Deren Verwendung in einer ROA ist aber nicht fest definiert und damit nicht überall identisch eingesetzt, wie dies im SOA-Kontext durch entsprechende WS-Standards der Fall ist. Der Vorteil des Ressourcen-Konzepts und der damit einhergehenden Vereinfachung der Methodensignatur eines Services wird getrübt durch den Umstand, dass bei komplexen Systemen sehr viele Objekte existieren. Da jede Ressource nur vier Methoden kennt, werden zwangsläufig Objekte, die in einer SOA durch Methoden bearbeitet wurden, in einer ROA durch entsprechend viele Objekte repräsentiert. Dabei kommt zusätzlich die Herausforderung sinnvoller Namensgebung der URL auf den System-Architekten zu (vgl. [MNS05]).

Auch bei der Verwendung des Begriffs REST – im speziellen bei REST-Services – bestehen diverse Ausprägungen: Zum einen werden Begriffe wie HI-REST und LO-REST verwendet und zum anderen existieren Begriffe wie RESTlike. HI-REST zeichnet sich aus durch die korrekte Verwendung aller vier HTTP-Verben (GET, PUT, POST, DELETE) sowie strukturierten URL und meist der Verwendung von XML als Datenformat. LO-REST hingegen verwendet nur POST- und GET-Methode und umgeht somit die Problematik einiger Browser bzw. Server, die weder PUT- noch DELETE-Befehle verarbeiten. LO-REST und RESTlike beschreiben hierbei dieselbe Verwendung von HTTP und sind deckungsgleich mit dem oben definierten Terminus Web API. Beobachtbar sind hierbei Services von Amazon, Google und vielen weiteren Anbietern, die zwar Services über hierarchische Ressourcen-URL adressieren, wie in REST, deren Methodenaufrufe aber über die vier HTTP-Methoden hinausgehen und damit Service-orientierte Ansätze aufweisen. Dabei wird der Begriff REST für das Marketing eigener Dienstleistungen entsprechend angepasst, was zu einem unklaren Bild der ROA führt.

Eine weitere Schwäche, die bei der Verwendung einer idempotenten GET-Abfrage auftreten kann, ist, einen Fehler im Browser durch zu viele Anfrageparameter zu provozieren. Erreicht die Zeichenkette einer GET-Abfrage die Größe von 4 Kilobyte (KB), so reagieren viele Server mit einem Fehler, da die URL zu lang wurde.

Zwischenfazit

Ein treffendes Fazit von Snell, der die Diskussion zwischen ROA und SOA im Jahre 2004 begonnen hat, ist, dass die Entscheidung für eine der beiden Lösungen von der zu entwickelnden Funktionalität eines Systems abhängt. In [Sne04] geht er auf die Eigenheiten beider Ansätze ein, lässt die Frage jedoch offen, welche Kriterien nun zu einer Entscheidung führen können. Sein Credo ist, dass die wichtigste Entscheidung, abgesehen von SOA oder ROA, diejenige sei, sich überhaupt für den Einsatz von Services zu entscheiden. Auch die Ergebnisse von Pautasso et. al. und Haas erläutern die Unterschiede, kommen aber ebenfalls zu dem Schluss, dass diverse Kriterien über das Für und Wider einer Architektur entscheiden (vgl. [PZL08, Haa05]).

2.3.4 Architektonische Klassifizierung von Service-Arten

Bereits in Kapitel 2.3.1 wurde im Rahmen einer SOA der Service-Begriff aus ITIL-Sicht erläutert und darüber hinaus als Ausprägung eines Web Services in technischer Hinsicht dargestellt. Der Vergleich von Konzepten der ROA und SOA zeigt aber, dass es nicht nur **den** Service in Verbindung mit dem Internet gibt. Beispielsweise werden Ressourcen innerhalb einer ROA ebenfalls wie Services behandelt, mit dem Unterschied eines Services, der immer nur vier fest vorgegebene Methoden besitzt. Aufgrund der diversen Möglichkeiten der Ausgestaltung von Services im IKT-Bereich werden im Folgenden zunächst Service-Arten im Hinblick auf deren architektonische Umsetzung klassifiziert. Im Anschluss daran werden fachliche Kriterien identifiziert, um Services unabhängig von ihrer technischen Umsetzung einer der vorgesehenen Nutzung entsprechenden Kategorie zuzuordnen.

Das Aufkommen des Web 2.0 und der anhaltende Boom des Internets im Allgemeinen brachte viele neue Geschäftsmodelle und damit einhergehend Begrifflichkeiten im Service-Bereich mit sich, auch wenn hierbei nicht jedes Konzept wirklich neu ist.[34] Neben autonomen Services und dem Begriff Web API, der das Vorhandensein von Service-Schnittstellen bei einer Web-Anwendung beschreibt, sind wichtige Aspekte der Kategorisierung vor allem XaaS und der oft als Überbegriff verwendete Terminus *Cloud Computing*. Da XaaS auch auf atomaren Diensten und Web API bestehen können, werden diese zuerst erläutert. Anschließend werden die darauf aufbauenden Konzepte des Cloud Computing beschrieben.

Atomare Services

Um entfernte Funktionsaufrufe zu realisieren, die eine überschaubare Aufgabe bewältigen oder nur eine Ressource bearbeiten oder bereitstellen, werden atomare Services eingesetzt. Diese stehen für sich alleine, sind nicht teilbar und benötigen keinen weiteren Kontext, um ihre Aufgabe zu erledigen. Zumeist wird hierbei auch von einem statuslosen Aufruf ausgegangen. Typische Beispiele für einen solchen atomaren Service sind *Really Simple Syndication* (RSS)-Feeds, die von vielen Internetseiten angeboten werden, um Interessenten über

[34] Teilweise werden bereits jahrelang bestehende Konzepte mit nur kleinen Änderungen als Innovationen beworben. So ist beispielsweise der REST-Stil schon seit der Jahrtausendwende bekannt und Konzepte wie Web Services waren in ähnlicher Form (RPC,CORBA) sogar noch eher vorhanden.

Neuigkeiten zu informieren. Dabei werden die übertragenen Nutzdaten meist in Form eines RSS- oder *Atom Syndication Format* (Atom)-Feeds übertragen, welches durch eine spezielle Software oder auch direkt durch den Internetbrowser interpretiert und aufbereitet werden kann. Eine Service-Beschreibung für einen solchen Dienst kann beispielsweise als WADL-Datei vorliegen, wobei dies im Falle von RSS-Feeds typischerweise nicht notwendig ist, da die Internetbrowser aus dem HTML-Seitenkopf etwaige RSS-Angebote direkt auslesen können.[35]

Abbildung 2.28: Schematischer Aufbau eines atomaren Services.

Das Beispiel in Abbildung 2.28 zeigt einen atomaren Service, der seine Funktion unter der Web-Adresse `http://www.beispiel.de/news` anbietet. Dabei wird lediglich eine Funktion aufgerufen, die die neuesten Informationen im XML-Format an den Aufrufer zurückgibt. Wenn man davon ausgeht, dass häufig HTTP als Verbindungsprotokoll verwendet wird, erhält der Aufrufer des Dienstes auch in jedem Fall eine Antwort, unabhängig davon, ob der Dienst Nutzdaten zurückliefert oder nur bestätigt, dass er ausgeführt wurde.

Application Programming Interface

Der Begriff *Application Programming Interface* (API) wird im Softwarebereich für eine definierte Schnittstelle verwendet, die für den Zugriff auf die Funktionalitäten eines IT-Systems bestehen. Dabei ist es unerheblich, wie das dahinter liegende Programm aufgebaut ist. Meist werden mehrere (atomare) Services angeboten, die einen bestimmten Anwendungsfall abdecken und es so ermöglichen, mit dem IT-System, welches durch die API ansprechbar ist, zu arbeiten. Bezogen auf das Internet lässt sich dafür der Begriff Web API verwenden. In Abbildung 2.29 wird der Aufbau einer solchen Web API dargestellt. Hier wird im ersten Schritt eine Suche nach Büchern mit dem Titel „PHP“ an

[35] Dies funktioniert nur, da die Newsfeed-Standards anerkannt sind und daher von den Herstellern in ihre Browser-Software eingebaut werden.

die API gesendet. Eine Funktion innerhalb der Web API bearbeitet die Anfrage und gibt eine Liste mit ISBN-Nummern als XML-Dokument an den Aufrufer zurück. Dieser kann nun eine der Nummern auswählen und übergibt diese im nächsten Aufruf an den Service "kaufen" der Web API.

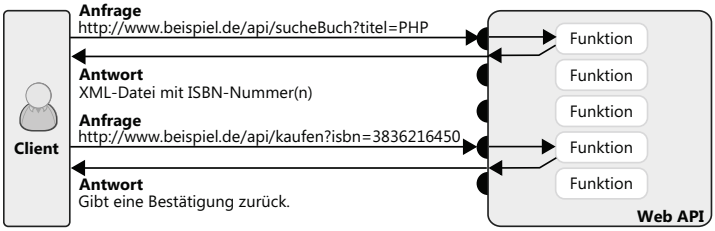


Abbildung 2.29: Schematischer Aufbau einer Web API.

Die gängige Praxis, um die Funktionalitäten einer API im Web 2.0 zu beschreiben, ist die textgebundene Beschreibung und eine Auflistung der möglichen Funktionsaufrufe sowie deren Parameter. Um die Wartbarkeit und die Verwaltung einer API im Zusammenhang mit dem Aufbau eines verteilten Systems strukturierter zu gestalten, sollte die Web API aber besser durch eine geeignete Schnittstellenbeschreibung vorgenommen werden, beispielsweise durch WSDL- oder WADL-Dokumente.

Cloud Computing

Das *National Institute of Standards and Technology* (NIST) definiert Cloud Computing in [MG09] wie folgt:

> „Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential **characteristics**, three **service models**, and four **deployment models**."

So umfasst Cloud Computing laut dieser Definition das Bereitstellen von gemeinsam genutzten IT-Ressourcen, wie zum Beispiel Netzwerke, Server, Speicher, Anwendungen oder IT-Services, die kurzfristig und mit minimalem Aufwand von Seiten des Anbieters bereitgestellt und verwaltet werden können. Dabei beinhaltet eine Cloud fünf Charakteristika, drei Service-Modelle sowie vier unterschiedliche Bereitstellungsarten. Die fünf charakterisierenden Merkmale sind dabei die folgenden:

- **Selbstbedienung nach Bedarf**: Ein Nutzer der Cloud kann ohne Zutun des Anbieters Ressourcen der Cloud nutzen.

- **Umfassender Netzwerkzugriff**: Die Ressourcen der Cloud sind über ein Netzwerk nutzbar, üblicherweise das Internet. Dabei kommen Standardmechanismen zum Einsatz, um durch heterogene Systeme nutzbar zu sein.

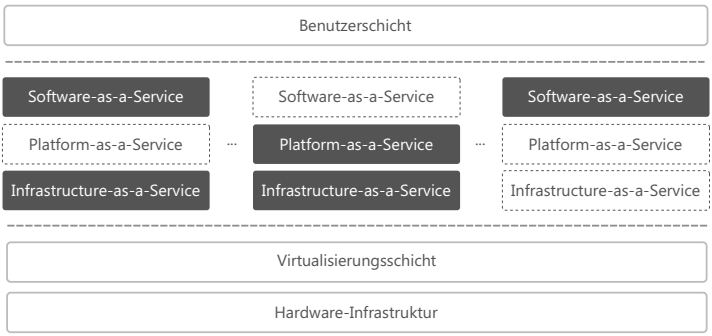
- **Gemeinsame Nutzung physischer Ressourcen**: Der Anbieter bündelt die eigenen IT-Ressourcen derart, dass der Nutzer entsprechend seiner Anforderung Ressourcen zugewiesen bekommt. Dabei werden normalerweise Mandantenfähigkeit, also die Nutzung von Soft- und Hardware durch viele Nutzer gleichzeitig (im Gegensatz zu dezidierten virtuellen oder physikalischen Maschinen im Grid Computing), und transparente Ressourcennutzung vorausgesetzt. Der Nutzer kann so beispielsweise Speicher, Rechenkapazität oder Netzwerkbandbreite nutzen, ohne zu wissen, wo sich die Hardware befindet, die er nutzt, oder welchen Umfang an Ressourcen er benötigt.

- **Unverzügliche Anpassbarkeit an den Ressourcenbedarf**: Ein Cloud-Angebot wird auch durch dessen Elastizität bestimmt, d. h. bei Bedarf kann der Ressourcenverbrauch an die Bedürfnisse angepasst werden. In einigen Fällen kann dies automatisch geschehen, beispielsweise durch das Hinzuschalten von weiteren Servern für eine Nutzeranwendung bei Lastspitzen.

- **Messung der Servicenutzung**: Innerhalb einer Cloud soll der Verbrauch von Ressourcen automatisch überwacht, kontrolliert und aufgezeichnet werden. Dies ist zum einen für die Abrechnung von Kosten

und zum anderen für die Optimierung der Cloud notwendig. Dies ermöglicht auch neue Geschäftsmodelle, wie exakte, nutzungsabhängige Abrechnungen.

Die drei Service-Modelle des Cloud Computing setzen auf unterschiedlichen Ebenen an. Es handelt sich dabei um SaaS, Platform-as-a-Service (PaaS) und Infrastructure-as-a-Service (IaaS). In Abbildung 2.30 ist deren Aufbau gezeigt. Im Gegensatz zum Schichtenmodell einer SOA bauen die drei Schichten nicht unmittelbar aufeinander auf, sondern lassen sich einzeln, wie auch in Kombination miteinander verwenden. Je nach Bedarf kann hier eine von sechs Varianten der Nutzung zum Einsatz kommen. So ist im ersten Szenario eine Nutzung von SaaS in Verbindung mit IaaS dargestellt, im zweiten Szenario die Verbindung von PaaS und IaaS und im letzten Szenario eine ausschließliche Nutzung von SaaS.



Quelle: angelehnt an [HV10]

Abbildung 2.30: Mögliche Kombinationen von XaaS-Diensten.

Über diese drei XaaS-Arten hinaus existieren auch weitere Formen von Angeboten im Cloud Computing, so beispielsweise ein sehr wichtiger Dienst für das Angebot an Datenbanken: Database-as-a-Service (DaaS). Eine Typisierung von DaaS gestaltet sich schwierig, da oft der Blickwinkel des Nutzers ausschlaggebend ist. DaaS kann je nach Blickwinkel sowohl als Infrastruktur, bei *Amazon SimpleDB*, als auch als Software, bei *DabbleDB*, angesehen werden. Ein Leitfaden für die Nutzung von DaaS in kleinen und mittleren Unternehmen ist in [HV10] zu finden.

Bei der Verwendung von Cloud Computing lassen sich zwei grundlegende Betriebsarten unterscheiden: die öffentliche und die private Cloud. Daraus lassen sich wiederum weitere Verfeinerungen ableiten, wie beispielsweise eine gemeinschaftlich genutzte Cloud und hybride Clouds ([HTV10, MG09]):

- **öffentliche Cloud** (*public cloud*): In einer öffentlichen Cloud sind die Dienste von einem Anbieter typischerweise für alle Nutzer des Internets verfügbar. Der Anbieter erhebt dafür im Normalfall ein Nutzungsentgelt nach einem speziellen Geschäftsmodell. Eine öffentliche Cloud stellt beispielsweise die Cloud von Amazon dar.

- **private Cloud** (*private cloud*): Eine private Cloud wird nur für ein einziges Unternehmen oder Organisation betrieben. Die Cloud ist nichtöffentlich und bietet daher ein gewisses Maß an Datensicherheit. Betrieben werden diese meist im eigenen Unternehmen, oder sie sind ausgelagert in ein vertrauenswürdiges Rechenzentrum. Diese Form birgt weniger die typischen Risiken eines Cloud-Angebots, wie Anbieterabhängigkeit und Vetrauensproblematiken, bietet aber auch nicht die vielen Vorteile, wie kostengünstige Skalierbarkeit und Ressourcennutzung. Denn die vorgehaltenen Ressourcen müssen für die private Cloud selbst bereitgestellt werden, und werden nur von dem Besitzer der private Cloud genutzt.

- **gemeinschaftlich genutzte Cloud** (*community cloud*): Bei einer gemeinschaftlich genutzten Cloud verhält es sich, wie bei einer private Cloud. Der einzige Unterschied ist, dass sich eine Gruppe von Unternehmen zusammenschließt, um die private Cloud gemeinsam zu betreiben. Dies ermöglicht dann doch wieder das Erzielen gewisser Kosten- und Skaleneffekte, die bei einer öffentlichen Cloud existieren. Sinnvoll wäre dieser Ansatz beispielsweise bei einer Gruppe von Unternehmen mit gleichen IT-Anforderungen.

- **hybride Cloud** (*hybrid cloud*): Bei der hybriden Cloud schließen sich diverse Clouds zusammen. Dies kann den Zusammenschluss von privaten, öffentlichen und gemeinschaftlich genutzten Clouds bedeuten. Durch eine solche Verknüpfung lassen sich die Ressourcen einer Cloud erweitern, falls diese nicht mehr ausreichen.

Software-as-a-Service

SaaS-Angebote beinhalten eine Web-basierte Anwendung, die in den meisten Fällen einen speziellen Geschäftszweck erfüllt. Eine häufig zu findende Verwendung ist das Nachbilden von Standardsoftware, die als Alternative oder als Zusatz zu einer fest auf Desktop-Rechnern installierten Anwendung dienen kann. Dabei werden interessante Geschäftsmodelle entworfen, die versuchen, gegenüber Desktop-Anwendungen konkurrenzfähig zu sein (in Kapitel 6 wird hierauf verstärkt eingegangen). Neben der normalen Nutzung der Software in einem Browser wird meist auch eine Web API angeboten, die es erlaubt, auf die Daten innerhalb der Anwendung zuzugreifen und diese zu bearbeiten. So lassen sich auch komplexere Integrationsszenarien mit weiteren Daten oder Anwendungen umsetzen. Die verwendeten Daten, die für die Dienstausführung benötigt werden, sind hierbei meist auf Seiten des Dienstanbieters gespeichert. In Abbildung 2.31 ist der schematische Aufbau eines SaaS-Dienstes dargestellt.

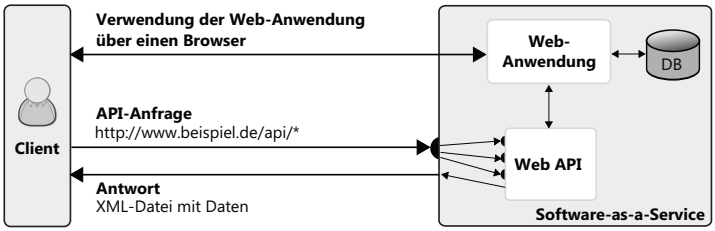


Abbildung 2.31: Schematischer Aufbau eines SaaS.

Dienste wie beispielsweise *Google Analytics*[36] speichern eine große Menge statistischer Daten über die Webseitenaufrufe eines Kunden und bieten darauf Auswertungsfunktionalitäten an. Dabei erfolgen Datenaufbereitung und -speicherung bei Google und nicht beim Kunden. Im Browser bekommt man dann eine Vielzahl an Auswertungsmöglichkeiten geboten. Ein weiteres typisches Beispiel für einen SaaS-Anbieter ist `Salesforce.com`[37], bei dem eine umfassende CRM-Lösung angeboten wird. Die Schnittstellen sind hierbei besonders wichtig, da man Kunden natürlich auch den Zugriff auf relevante Daten durch andere Systeme ermöglichen möchte.

[36] `http://www.google.com/analytics/de-DE/`

[37] `http://www.salesforce.com`

Platform-as-a-Service

Der Vorteil bei der Nutzung von PaaS-Angeboten ist die umfassende Verwendung eigener Programmlogik. Hierbei ist es möglich, entweder durch Programmcode oder auch grafischen Werkzeugen, eigene Abläufe, Prozesse oder auch komplette Anwendungen zu schreiben. Gegenüber herkömmlichen Angeboten von Rechenzentren oder lokalem Hosting in einem Unternehmen bieten die Anbieter meist günstige Konditionen sowie oft auch extra auf die Verwendung von Web-basierten Architekturen ausgerichtete Werkzeuge direkt im Browser an. Zusätzlich sind meist auch Standardfunktionalitäten und der Zugriff auf die eigene Programmlogik per Web API integriert (siehe Abbildung 2.32). Diese Anbieter sind somit als eine Art Plattform innerhalb des Internets zu sehen. Ein Beispiel ist hier der Dienst *bungeeConnect.com*[38] oder auch der Dienst *force.com*[39], der es zum einen ermöglicht, das zuvor genannte CRM-Produkt *Salesforce* mit eigener Programmlogik zu erweitern und zum anderen auch davon losgelöst Unternehmens-Anwendungen in der Cloud zu entwickeln und zu betreiben (vgl. [Oul09]).

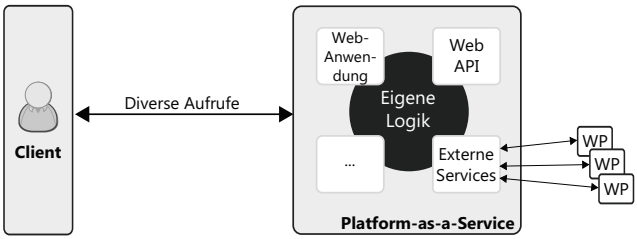


Abbildung 2.32: Schematischer Aufbau eines PaaS.

Infrastructure-as-a-Service

Das Anbieten von Infrastrukturen als Service, das so genannte IaaS, greift das im Serverbetrieb schon gängige Thema Virtualisierung von IT-Ressourcen auf und adaptiert dieses für die Nutzung in der Cloud. Das Angebot erstreckt sich hierbei von virtuellen Servern über Datenbanken und Festplattenspeicherplatz bis hin zu Rechenleistung. Alle diese Dienste lassen sich als virtuelle Res-

[38] http://www.bungeeconnect.com

[39] http://www.salesforce.com/platform/

sourcen im Netz anbieten und können gemäß des Cloud Computing Paradigmas über Web API aufgerufen und an beliebige Dienste oder Anwendungen gekoppelt werden. Eine Nutzung über einen Browser ist hier meist nicht vorgesehen, da Infrastruktur oft durch automatische Prozesse in andere Anwendungen integriert ist. So lässt sich beispielsweise eine generische API für den Datenbankzugriff über REST als Infrastrukturkomponente einsetzen (vgl. [HTV10]) oder dynamisch bei hoher Auslastung Server-Kapazitäten für den eigenen Web-Shop anschalten, beispielsweise durch *Amazons Elastic Compute Cloud (ECC)*[40]. Als Beispiel ist in Abbildung 2.33 ein IaaS-Anbieter mit der Infrastruktur an Datenbanken, Festplattenplatz und Rechenleistung dargestellt.

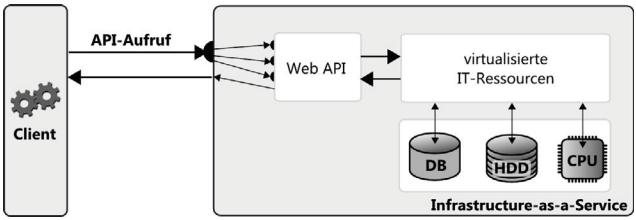


Abbildung 2.33: Schematischer Aufbau eines IaaS.

2.3.5 Funktionale Klassifizierung von Service-Arten

Services – im Sinne von IT-Dienstleistungen – werden im Internet in großer Vielfalt angeboten. Allein die Anzahl an öffentlich registrierten Web API bei *www.programmableweb.com*, einem der größten Verzeichnisse öffentlicher API im Internet, ist mit über 2300[41] gemeldeten Diensten schon beachtlich. Im Folgenden wird eine Klassifizierung der Fachlichkeit von Services vorgenommen. Diese Einstufung wird zum einen als Hilfestellung in der Analysephase des Erstellens einer WOA verwendet und zum anderen als Verzeichnisstruktur für Services in Controllern benutzt. Dadurch wird ein schnelleres Auffinden von potenziell nutzbaren Services ermöglicht. Hierbei verlaufen die Grenzen zwischen der einen oder anderen Service-Art unscharf, da sich ein Service je nach Blickwinkel durchaus unterschiedlich darstellt. Des Weiteren existiert keine

[40] `http://aws.amazon.com/de/ec2/`
[41] Stand: 23.11.2010.

einheitliche Klassifizierung, die eine standardisierte Einordnung ermöglichen würde. Im Folgenden werden gängige Modelle der Service-Klassifizierung vorgestellt und für den weiteren Verlauf der Arbeit eingegrenzt.

Eine sehr verbreitete Sichtweise auf Service-Ebenen ist die Aufteilung in drei Schichten: *Basis-Services*, die Funktionalitäten der darunterliegenden Systeme kapseln, *zusammengesetzte Services* (*composed services*), die fachliche Logik kapseln sowie *Prozess-Services*, die komplexe Geschäftsprozesse darstellen. Diese Ansicht liegt sehr nahe an dem Schichten-Modell einer SOA und wird im Grundsatz in diversen SOA-Fachbüchern vertreten (vgl. [KBS07, FZ09, Erl05, Jos08]). Einige Detailunterschiede sind hierbei die diversen weiteren Zwischenschichten, die darüber hinaus genannt werden. Teilweise werden dabei auf oberster Ebene Anwendungs-Frontends spezifiziert, die nicht als eigentliche Services, sondern eher als *Graphical User Interface* (GUI)-Komponenten gesehen werden können und eine öffentliche Unternehmens-Services-Schicht, die Unternehmens-übergreifende Schnittstellen beinhaltet (vgl. [KBS07]). Des Weiteren werden Zwischenschichten genannt, die zwischen basis- und prozessorientierten Services in Form von Technologie-Gateways und Adaptern existieren (vgl. [KBS07, Erl05]). Die in dieser Klassifizierung angenommene fachliche Ausprägung eines Services wird hierbei mehr an der Platzierung innerhalb der SOA-Schichten vorgenommen als an der tatsächlichen inhaltlichen Logik des Services.

Eine weitere Kategorisierung in drei Service-Ebenen wird in [PVM09] vorgenommen. Darin werden die Ebenen *Data Services*, *Functional Services* und *User Interface Services* voneinander abgegrenzt. Diese Betrachtungsweise ist nahe am klassischen Softwareentwurfsmodell angelehnt, worin drei Ebenen unterschieden werden: grafische Benutzeroberfläche (GUI), Fachlogik und Datenzugriff. Der einzige größere Unterschied ist hierbei, dass auf oberster Ebene keine Geschäftsprozesse, sondern die grafische Ausgabe an den Nutzer als relevant erachtet wird.

Bei genauer Betrachtung der beiden Klassifizierungen fällt auf, dass sich diese nicht sehr voneinander unterscheiden. Die Service-Kategorien der beiden Ansätze lassen sich leicht zueinander in Relation setzen. Der Unterschied besteht hauptsächlich im Grad der Detaillierung und der exakten Definition der Position innerhalb der Schichten. In Abbildung 2.34 ist auf der linken Seite der Ansatz von [Jos08] und [FZ09] dargestellt, mittig in Relation dazu der um Details erweiterte Ansatz, wie er in [Erl05] und [KBS07] verwendet wird und

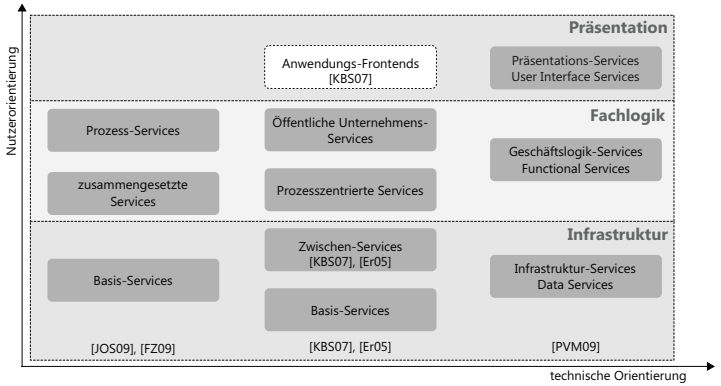


Abbildung 2.34: Verschiedene Service-Klassifizierungen.

schließlich auf der rechten Seite der Ansatz von [PVM09]. Zur Verdeutlichung der Ähnlichkeit wurden hier die drei Schichten: *Präsentation*, *Fachlogik* sowie *Infrastruktur* eingezeichnet.

Welche der Service-Klassifizierungen man nun in einem Projekt verwendet, ist ganz erheblich vom Anwendungszweck abhängig. Die eher technisch motivierte Sichtweise des ersten Klassifizierungsansatzes, vor allem in der erweiterten Form, ist vor allem für Umsetzung einer SOA interessant, da sich die Services gut in eine solche Architektur eingliedern. Der Detailreichtum ist hierbei aber definitiv höher als im zweiten Ansatz und daher für die später folgende Kostenberechnung im Kontext dieser Arbeit weniger geeignet. Es zeigt sich auch, dass sich viele der im Internet angebotenen Services, beispielsweise bei *ProgrammableWeb*, eher in die Kategorien des zweiten Klassifizierungsmodells eingliedern lassen. Die dort verwendeten Kategorien sind auch weniger technisch als vielmehr fachlich motiviert. In einer WOA, die im folgenden Kapitel im Detail erläutert wird, sind vor allem auch extern bereitgestellte Dienste von großem Interesse, die im technischen Modell am ehesten den öffentlichen Unternehmens-Services entsprechen. Daher wird im Folgenden von *Infrastruktur-Services*, *fachlichen Geschäftsprozess-Services* sowie *Präsentations- und Anzeige-Services* gesprochen.

3 Web-orientierte Architekturen

Bei der Umsetzung einer Softwarearchitektur für verteilte Systeme steht am Anfang grundsätzlich die Entscheidung, welchem Paradigma die IT-Architektur folgen soll. Seit den späten 90er Jahren bis heute stand das Thema SOA als Inbegriff für eine flexible, stabile Lösung und hat wenig an Aktualität eingebüßt. Der Nutzen und die Funktionsfähigkeit von SOA oder auch nur einzelner technischer Standards, die für die Umsetzung verwendet werden, sind jedoch häufig Gegenstand kontroverser Diskussionen. Gerade in Presse und Internetblogs liest man oft Meldungen über das Scheitern der SOA, und dass die Versprechen, wie Flexibilität und Einfachheit, nicht erreicht werden. Dies mag natürlich auch daran liegen, dass das Thema SOA viel zu oft als Technologiethema und nicht als Strategie wahrgenommen wurde. Mit dem Aufkommen von REST und der damit verbundenen ROA wurde um das Jahr 2004 (vgl. [Sne04]) ein auf das WWW angelegte Architekturparadigma geprägt, welches oft als Alternative zu einer SOA gesehen wird. Vor allem die Vereinfachung der Schnittstellen und Protokolle sowie die Orientierung an Ressourcen statt an Services soll hierbei den entscheidenden Vorteil gegenüber einer SOA bieten. Eine Entscheidung für oder gegen einen der beiden Architekturstile ist aber nicht unbedingt notwendig. Das Konzept einer Web-orientierten Architektur verbindet beide Welten in einer Softwarearchitektur. Der Grundgedanke ist hierbei die Verknüpfung von Service- und Ressourcen-Orientierung mit dem geringst möglichen Aufwand zur Erreichung der Ziele eines produktiven IT-Systems. Dabei werden auch die aus jahrelangen Erfahrungen mit SOA aufgedeckten Schwächen adressiert.

Den Rahmen für die konkrete Definition einer WOA und deren Prinzipien bildet hierbei der Standard *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (vgl. [IEE01]), der durch den Berufsverband *Institute of Electrical and Electronics Engineers* (IEEE) konzipiert wurde, um das konzeptionelle Grundgerüst einer Softwarearchitektur klar zu definieren. Darin werden Modelle und Methoden beschrieben, die das Erstellen eines konzeptionellen Rahmenwerks (*framework*) mit Bezug auf Softwarearchitek-

turen erleichtern. Im Jahre 2007 wurde dieses Werk dann auch von der ISO mit dem Zusatz „Systems and software engineering" als internationaler Standard (*ISO/IED 42010*) anerkannt. In Abbildung 3.1 ist, abweichend von dem ursprünglichen Modell, eine Dreiteilung in die folgenden Gliederungspunkte vorgenommen worden, um den Aufbau dieses Kapitels zu verdeutlichen.

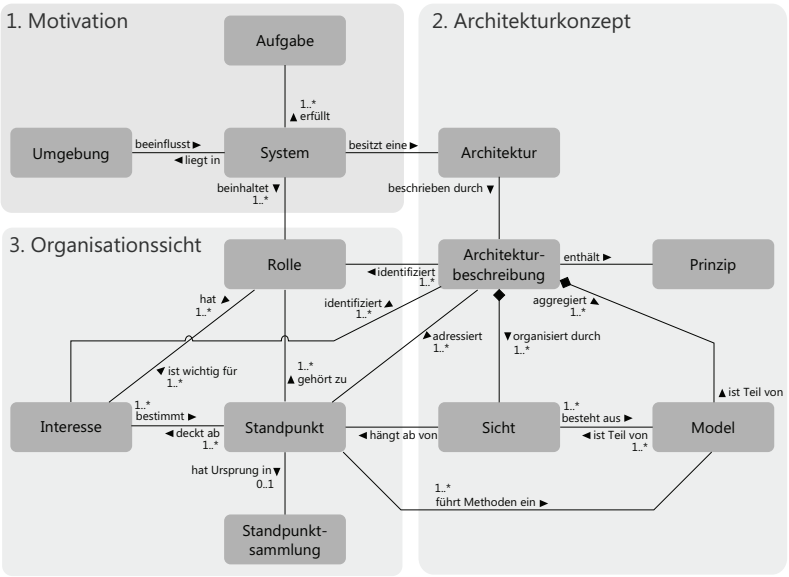
1. **Motivation**: Einer Architektur liegt zumeist eine bestimmte Motivation zugrunde. Die der WOA wird im ersten Abschnitt des Kapitels ausgeführt. Darin werden die Aufgabenstellung und die Umgebung einer WOA skizziert.

2. **Architekturkonzept**: Im zweiten Abschnitt dieses Kapitels werden die Architektur sowie die Grundprinzipien einer WOA beschrieben. Zur Verdeutlichung werden Modelle aus mehreren Sichten zur Beschreibung des WOA-Aufbaus dargelegt. Dazu gehören neben der Netzwerktopologie einer WOA auch verwendete Standards, unterschiedliche WOA-Ausprägungen sowie ein Schichtenmodell zur Einordnung von XaaS-Ausprägungen.

3. **Organisationssicht**: Um neben der technischen Sicht auf die WOA auch die organisatorischen Randbedingungen zu nennen, werden im dritten Abschnitt die Rollen (*Stakeholder*) dargestellt, die sich innerhalb der Planung, Umsetzung und dem Betrieb einer WOA ergeben.

3.1 Motivation und Aufgabe

Die Motivation hinter der konzeptionellen Beschreibung einer Web-basierten Architektur für verteilte Systeme entspringt zu großen Teilen der Beobachtung und Bewertung des Einsatzes von SOA in Unternehmen. Dass hier, neben teilweise auch erfolgreichen Systemen, von sehr vielen Problemen berichtet wird, lässt sich nicht leugnen. So soll eine WOA einfachere Konzepte und Methoden zur Erstellung eines verteilten Systems bieten. Ein anderer Teil der Motivation lässt sich mit dem Aufkommen des *Web 2.0* begründen (vgl. [VH07, Alb08, GHN09]). Darin werden die drei Kernaspekte *Funktionalität*, *Daten* und *Sozialisation* hervorgehoben und das WWW nicht mehr als nur



Quelle: angelehnt an [IEE01]

Abbildung 3.1: Konzeptionelles Modell der IEEE für die Beschreibung von Softwarearchitekturen.

konsumorientiert gesehen, sondern vielmehr als ein von Nutzern mitgestalteter Raum skizziert. Ebenso wie in einer SOA werden im Web 2.0 verteilte Anwendungen zu einem System miteinander verbunden und durchaus auch Prozesse erstellt. Der interessante Punkt ist hierbei, dass fast jede dieser Integrationslösungen ohne große Probleme zu funktionieren scheint. Durch die Verwendung einfacher Web-Standards werden beispielsweise Daten mit Anwendungen verknüpft. Eines der Phänomene ist zwar, dass im Prinzip jede Web-basierte Anwendung zunächst als Beta-Version deklariert wird und somit als noch nicht fertige, stabile Version gekennzeichnet ist, dies aber durch iterative Verbesserung der Anwendung ausgeglichen wird[42]. Die Ähnlichkeit von SOA und dem Web 2.0 sieht [SK07] vor allem bei dem Prinzip der Wiederverwendung, der Komplexitätsreduktion, der Komposition von Services sowie der Agilität. Ein weiterer motivierender Faktor, der die strukturierte Konzeption einer WOA als sinnvoll erscheinen lässt, ist im *Cloud Computing* zu sehen. Darin entsteht durch XaaS-Angebote die Basis für globale, verteilte Systeme, in denen ein Großteil der Funktionalität aus dem Unternehmen und dem eigenen Netzwerk ausgelagert werden kann.

Ausgelöst durch diese motivierenden Faktoren können die Aufgaben einer WOA wie folgt festgestellt werden:

- **Verteiltes IT-System**: Eine WOA soll eine Vernetzung von Systemen, Services und Daten über das Internet zur Unterstützung eines Geschäftsziels ermöglichen. Hierbei soll neben Prozessen für *Maschine-zu-Maschine*- auch *Maschine-zu-Mensch*-Kommunikation unterstützt werden. Die Kontrolle des Systems kann hierbei von verschiedenen Punkten aus erfolgen.

- **Einfache technologische Lösungen**: Das Ziel der Vernetzung soll durch möglichst einfache technologische Lösungen erreicht werden, wobei das Internet bzw. das WWW grundsätzlich als das Rückgrat der Anwendung anzusehen ist.

- **Ausnutzung von vorhandenen IT-Dienstleistungsangeboten**: Die Nutzung von Angeboten aus der Cloud soll zum einen zu einer Reduktion eigener Hardware führen oder sogar ganz zu deren Vermeidung beitragen. Zum anderen sollen kostengünstige XaaS-Angebote von der

[42] Dieser Zustand wird als *perpetual beta* bezeichnet.

Standardsoftware bis hin zu skalierbaren Servern eingesetzt werden, die viele Möglichkeiten im Hinblick auf das Wachstum eines Unternehmens und dessen IT-System bieten.

- **Präzise Kostenkontrolle**: Der Einsatz von vorhandenen XaaS-Angeboten, die in den meisten Fällen günstige Konditionen bieten, soll das präzise Planen und Überwachen von Kosten für ein IT-System erlauben. Durch den Einsatz einer WOA soll dabei der eigene Aufwand für die Wartung und Kontrolle des IT-Systems zusätzlich gesenkt werden.

Nach [IEE01] hat jedes System eine Umgebung und einen Kontext, die das System auf verschiedenen Ebenen beeinflussen können. Dies können beispielsweise politische, betriebliche oder die Entwicklung betreffende Einflüsse sein. Eine WOA dient dem Zweck eines betrieblichen IT-Systems und unterliegt daher grundsätzlich den Einflüssen der Organisationsstruktur und der betrieblichen Politik. Hierbei ist vor allem in größeren Unternehmen darauf zu achten, dass ein verteiltes System zur Unterstützung des Geschäftsziels eines Unternehmens über Abteilungsgrenzen hinweg funktionieren sollte und daher ein hoher Abstimmungsbedarf in allen technischen, wie auch organisatorischen Fragestellungen notwendig ist. Auf der einen Seite steht die technische Verknüpfung von Services, auf der anderen Seite steht das Klären der Zuständigkeiten und Rechte auf Daten (*data ownership*. Letzteres stellt eine nicht zu vernachlässigende Herausforderung dar. Die Frage nach der technologischen Umsetzung wird durch die Beschreibung der Architektur im folgenden Abschnitt erläutert. Die Beantwortung der Frage nach organisatorischen Maßnahmen und Richtlinien zur Erstellung einer WOA wird anschließend im Abschnitt 3.3 begonnen sowie in Kapitel 5 fortgeführt.

3.2 Architekturkonzept und Prinzipien

Der Begriff WOA ist keine neue Erfindung. In vielen Blogs, hauptsächlich aus dem amerikanischen Raum, und bei Marktforschungsinstituten wird er seit 2005 für einen neuen Architekturansatz verwendet. So lautet die Definition von Gartner folgendermaßen (vgl. [Gal08]):

> *„WOA is an architectural substyle of SOA that integrates systems and users via a web of globally linked hypermedia based on the*

> *architecture of the Web. This architecture emphasizes generality of interfaces (UIs and APIs) to achieve global network effects through five fundamental generic interface constraints: Identification of resources, Manipulation of resources through representations, Self-descriptive messages, Hypermedia as the engine of application state, and Application neutrality."*

In dieser Definition wird die Meinung vertreten, dass das Konzept einer WOA eine Untermenge von SOA darstellt und die folgenden fünf Aspekte beinhaltet: Identifikation von Ressourcen, Manipulation von Ressourcen über Repräsentationen, selbst-beschreibende Nachrichten, *Hypermedia* als Anwendungsstatus und Anwendungsneutralität. Vergleicht man diese Definition mit derjenigen von REST (vgl. [Fie00, Sne04]), ist nur die Anwendungsneutralität hinzugekommen. Auch Hinchliffe sieht in WOA lediglich ein Synonym für Ressourcen-Orientierung und damit ROA (vgl. [Hin08]). Eine differenziertere Meinung vertritt Manes, die zwar WOA ebenfalls als Ressourcen-orientiert einstuft, aber davor warnt, einen Vergleich zu SOA zu ziehen. Bei einer SOA gehe es um Services auf konzeptioneller Stufe und den Entwurf eines Architekturstils auf Systemebene, nicht um Technologie. In einer WOA gehe es um einen Architekturstil auf Schnittstellenebene (vgl. [Man08]). Sieht man WOA als synonymen Begriff für ROA, ist dieser Hinweis berechtigt.

An dieser Stelle soll der Begriff WOA mit einer eigenen Definition versehen werden, die von den zuvor beschriebenen Ansichten in einigen Aspekten abweicht und für das Verständnis einer WOA in der vorliegenden Arbeit benötigt wird. Ein sehr wichtiger Punkt ist hierbei das Verständnis einer WOA als umfassende IT-Strategie zum Erreichen einer Web-basierten Systemarchitektur und nicht als reines Technologiekonzept, welches sich lediglich um REST rankt. Das Beschränken auf technologische Aspekte ist ein immer wieder auftretender Fehler bei der SOA-Umsetzung und sollte sich nicht wiederholen. Ein weiterer Unterschied ist in der verwendeten Technologie zu sehen. Während die oben genannten Definitionen grundsätzlich REST als verwendetes Mittel für die Realisierung von Systemaufrufen sehen, wird im Folgenden die Verwendung von Web API in den Vordergrund gestellt. Allerdings wird die Nutzung von REST oder sogar WS-Standards nicht ausgeschlossen, sondern befürwortet, wenn es die Umstände erfordern. So lassen sich ggf. Altsysteme oder bestehende Komponenten eben nur über Web Services anbinden. Web API werden aber aktuell von sehr vielen Anbietern, teilweise auch parallel zu Web Services, im-

plementiert, so dass diese als einfachste Schnittstelle für die Umsetzung von Service-Aufrufen verwendet werden können. Eine genauere Betrachtung von Web Services, REST-Schnittstellen und Web API folgt in Abschnitt 3.2.1.

Um den Unterschied von WOA und SOA zu verdeutlichen, ist vor allem die unbedingte Verwendung von IaaS-Angeboten zur Realisierung von Infrastrukturkomponenten in einer WOA zu nennen. Damit lassen sich Datenbanken, Festplattenspeicher, CPU-Rechenzeit sowie weitere Standardaufgaben auslagern. So wird zum einen eigene Hardwareanschaffung vermieden oder zumindest reduziert, und zum anderen erreicht man leicht skalierbare und vor allem berechenbare IT-Komponenten als Basis für das eigene IT-System. Darüber hinaus sollen auch gerade für Standardaufgaben, wie CRM oder Office-Lösungen, vorhandene SaaS-Angebote verwendet werden, um einen weiteren Grad an Flexibilität und Kostenoptimierung zu erreichen. Eine in Definitionen von SOA oft vergessene und dennoch eingesetzte Technologie ist der ESB, der als Steuerungskomponente unabdingbar ist. Auch innerhalb einer WOA ist eine Governance-Komponente zur Steuerung, Verwaltung und Überwachung notwendig, wobei hier nicht festgelegt wird, in welcher Form diese Komponente vorliegen sollte. Eine eigene WOA-Definition, die alle genannten Punkte entsprechend zusammenfasst, lautet wie folgt:

> *„Eine Web-orientierte Architektur (WOA) ist ein Ansatz zur Umsetzung eines verteilt organisierten Anwendungssystems, das fachliche Dienste und Funktionalitäten sowie den Zugriff auf Ressourcen in Form von* ***dokumentierten***, ***orchestrierbaren Web-Prozeduren*** *(WP) über ein Netzwerk bereitstellt. WP werden, entsprechend ihrem jeweiligen Verwendungszweck, durch* ***gängige Web-Standards*** *realisiert. Ein möglichst hoher Anteil an Infrastrukturkomponenten und WP wird durch* ***Everything-as-a-Service(XaaS)-Angebote*** *umgesetzt und mit mindestens einer* ***Steuerungskomponente*** *zu einem Nutzen stiftenden System kombiniert."*

Aus dieser Definition ergeben sich direkt folgende Eigenschaften und Ansprüche an eine WOA. Diese stellen auch gleichzeitig die Prinzipien dar, die einer Architekturbeschreibung nach [IEE01] zuzuordnen sind:

- **Orchestrierbarkeit**: Durch die Bereitstellung einzelner WP, die sich untereinander verbinden lassen (Orchestrierung), ist es möglich, komplexere Prozesse zu Geschäftsprozessen zusammenzusetzen.

- **Dokumentation von Schnittstellen**: Um die korrekte Verwendung von WP zu gewährleisten und in Programmen sinnvoll zu nutzen, sollte eine Schnittstellendefinition bestehen. Diese muss nicht zwangsläufig mittels WSDL- und WADL-Dokumenten belegt sein. Aber die Verwendung einer Freitextbeschreibung für eine WP führt zu erhöhtem Aufwand bei der Implementierung und der Wartung.

- **Gängige Web-Standards**: Hiermit wird ausgedrückt, dass man immer den einfachsten Weg der Implementierung bzw. Einbindung von Komponenten wählen sollte und jeweils nur den Grad der Komplexität erreicht, der für die Erfüllung des jeweiligen Zwecks unbedingt notwendig ist. Dies gilt vor allem für komplexere Aufgaben, wie beispielsweise der Sicherung und Ausführung von WP sowie der Umsetzung von Verschlüsselung, Authentifizierung, transaktionaler Garantien und Verfügbarkeit.

- **XaaS / Outsourcing**: Die Nutzung bereits vorhandener Dienste des Internets ist ein weiterer wichtiger Punkt, der bei der Umsetzung einer WOA Beachtung findet. Dies fokussiert vor allem auf die Verwendung von IaaS, SaaS sowie PaaS und weiteren Serviceleistungen, um den Bedarf an eigener Hard- und Software möglichst gering zu halten. Die Handlungsprämisse ist hierbei: Es soll so wenig eigene Hardware angeschafft und Unternehmens-interne Softwareprogrammierung vorgenommen werden, wie möglich. Stattdessen soll man fachliche Prozesse weitestgehend auslagern.

- **Steuerungskomponente**: Einzelne WP und Geschäftsprozesse, die aus beliebig vielen WP zusammengesetzt sein können, müssen überwacht, protokolliert und gesteuert werden. Um dies zu gewährleisten, benötigt eine WOA eine Governance-Komponente. Diese Komponente orientiert sich in dem Umfang der grundlegenden Funktionalität am Konzept eines ESB, weist aber dennoch einige Unterschiede auf. Eine genauere Beschreibung der notwendigen Kernfunktionalitäten folgt in Abschnitt 3.2.4.

Aus diesen in der Definition explizit genannten Eigenschaften lassen sich noch weitere implizite Aspekte ableiten, die im Folgenden erläutert werden.

- **Einfachheit**: Durch die Verwendung der jeweils notwendigen Technologien für ein Szenario und die weitverbreiteten Anbieter, die oft subjektiv einfache Web API anbieten, ist die Integration und Kombination von WP zunächst als *einfach* einzustufen. Durch das Hinzufügen weiterer Schichten, wie beispielsweise Sicherheitsaspekte und Transaktionslogik, steigert sich natürlich auch hierbei analog zu einer SOA die Komplexität.

- **Offenheit / Variabilität**: Da sich viele gängige Web-Standards und Technologien verwenden lassen, ist man in einer WOA offen für die Verwendung der meisten WP und bleibt dennoch variabel in der Umsetzung. Man beschränkt sich nicht auf bestimmte Lösungen und kann auch im Bedarfsfall auf komplexere Technologien ausweichen, beispielsweise die Verwendung von WS-Standards für spezielle Anwendungsfälle.

- **Intuitiv**: Nutzt man für WP die Ressourcen-Orientierung, ist deren intuitive Verwendung von Vorteil. Da sich das Vokabular auf vier HTTP-Verben beschränkt, sind die Ressourcen nahezu ohne weitere Beschreibung in einem Prozess verwendbar. Je nach Situation reichen jedoch die hierbei ermöglichten CRUD-Operationen nicht für die Implementierung eines WP aus, so dass man eher zu Web Services oder Web API tendiert. In einem solchen Fall geht der Vorteil des beschränkten Vokabulars von REST verloren, was durch eine geeignete Beschreibung der Schnittstellen durch WADL oder WSDL aufgefangen werden kann.

- **Berechenbare Kosten**: Die Berechnung der anfallenden Kosten einer verteilten Softwarearchitektur, vor allem im Falle einer SOA, ist im Regelfall sehr komplex. Es entstehen Kosten für die Anschaffung von Hard- und Software, Personalkosten für deren Betrieb und weitere Kosten für die Bereitstellung der Systeme während deren Lebensdauer, wie Strom-, Wartungs- und Fehlerbehebungskosten (*Maintenance*). Die möglichst weitgehende Verwendung von XaaS-Angeboten, die meist eindeutige Kostenmodelle vorweisen, hilft der Berechnung der Kosten und deren Zuordnung. Der Berechnung von Kosten, die bei dem Erstellen und dem Betrieb einer WOA entstehen, ist das Kapitel 6 gewidmet.

3.2.1 Topologie

Die Netzwerktopologie einer WOA (siehe Abbildung 3.2) kann zunächst als vermaschtes Netz dargestellt werden, wobei hier die einzelnen WP diverse Komponenten unterschiedlichen Funktionsumfangs darstellen. Dies kann vom rudimentären Währungsumrechnungsservice bis hin zu kompletten XaaS-Diensten reichen. Ein wichtiger Aspekt ist zum einen, dass eine WP in mehreren Kontexten aufrufbar ist. Weitaus wichtiger sind zum anderen die Steuerungskomponenten, im Folgenden als WAC oder kurz *Controller* bezeichnet, die in dem hier gezeigten Modell einer WOA eine Kernfunktion übernehmen: die Steuerung sowie Überwachung der mit ihnen verbundenen Komponenten. Dabei kann ein Controller ebenfalls wieder als WP von weiteren Komponenten angesprochen werden. Durch dieses Verhalten lassen sich beliebige Verschachtelungen realisieren. Die zentrale Funktion des Controllers legt hierbei den Schluss nahe, dass dessen Funktionalität entsprechend flexibel sein muss, um die Aufgabe der Kombination und Steuerung der mit ihm verbundenen WP zu erfüllen.

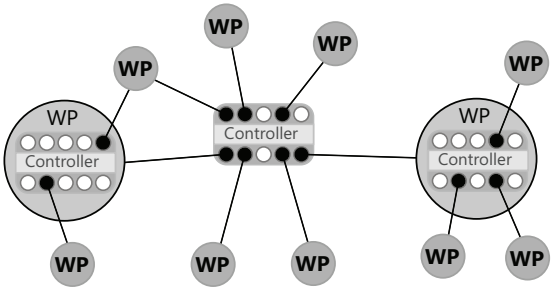


Abbildung 3.2: Vereinfachte WOA-Topologie.

WP werden in der WOA mit als wichtigste Komponente neben den Controllern genannt. Das Verständnis davon, was genau eine WP ist, ergibt sich aus der Aussage, dass alle Komponenten aus dem SOA- als auch ROA-Bereich verwendet werden können. Daher wird der Überbegriff WP für alle Arten von technischen Schnittstellen zu Systemen verwendet, die in diesem Kontext von Belang sind. Eine Einteilung wird hier grob in drei Klassen vorgenommen:

- **Web Services**: Hierunter werden Services verstanden, die zumindest die üblichen SOA-Standards zur Kommunikation (SOAP) und Schnittstellen-Beschreibung (WSDL) verwenden.

- **REST-Services**: Alle Web-Schnittstellen, die dem REST-Paradigma exakt folgen, gehören zur zweiten Kategorie. Sie zeichnen sich durch die korrekte Verwendung der vier HTTP-Verben PUT, POST, GET und DELETE und der durchgängigen Verwendung des Konzepts der Ressourcen-Orientierung aus.

- **Web API**: In die dritte Kategorie fallen alle Services, die über eine HTTP-Verbindung, in den meisten Fällen durch einen GET-Request, einen Service oder eine Ressource aufrufen. Diese Verwendungsart wird oft als „RESTful", „REST-like" oder auch „LO-REST" bezeichnet, da das Konzept von REST oft nur unzureichend beachtet wird.

Für den Aufruf von WP in der hier betrachteten Architektur lassen sich die vier Aspekte *Vokabular*, *Objekte*, *Identifikation* und *Nachrichtenformat* abgrenzen. Die in Tabelle 3.1 dargestellte Übersicht zeigt, dass jede der Technologien einige Punkte aufweist, die variabel sind und mit einem Stern gekennzeichnet werden. Die Identifikation der aufzurufenden Services findet in allen drei Fällen über URI bzw. im Falle der Verwendung von HTTP mittels URL statt. Als Faustregel lässt sich hierbei festhalten: Je weniger variabel die Wahl der Mittel ist, desto einfacher lässt sich ein Service aufrufen. Dies gilt jedoch nicht zwangsläufig auch für das Anbieten eines solchen Services, denn das Programmieren einer Web API ist noch immer ein wenig einfacher, als korrekte REST-konforme Services zur Verfügung zu stellen.[43]

Eine Abgrenzung dieser drei Web-Schnittstellen im Hinblick auf die Komplexität der Umsetzung im Standardfall[44], sowie die Anzahl der Standards, die für die Umsetzung zur Verfügung stehen, ist in Abbildung 3.3 dargestellt. Als einfachste Umsetzung, aber auch mit den wenigsten definierten Standards, ist hier die Web API zu nennen. Etwas komplexer in der Umsetzung, die korrekte Verwendung des REST-Paradigmas vorausgesetzt, sind REST-konforme

[43] Durch die Verwendung von entsprechenden Frameworks wird die Komplexität, wie auch im SOA-Bereich, meist vor den Entwicklern verborgen.

[44] In diesem Kontext ist damit die Implementierung eines einzelnen ungesicherten Services ohne Transaktionslogik oder Verschlüsselung gemeint.

Tabelle 3.1: Vergleich der Aspekte von WP-Arten.

	Web API	**REST**	**Web Service**
Vokabular	*	fix	*
Objekte	*	eindeutige Ressourcen	*
Identifikation	URI/URL	URI/URL	URI/URL
Nachrichtenformat	*	*	SOAP (XML)

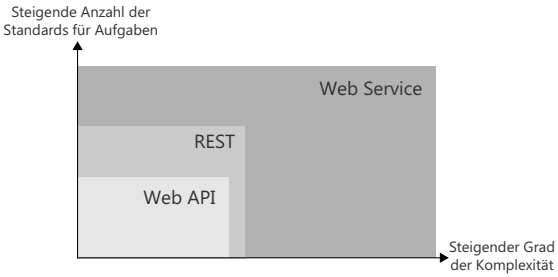


Abbildung 3.3: Komplexitätsmatrix von Web-Prozeduren.

Schnittstellen einzuordnen. Als aufwendigste Lösung sind Web Services anzusehen, die dafür aber auch die meisten definierten Standards zur Verfügung haben. Aus Gründen der Praktikabilität werden hier Schnittstellen, wie CORBA und RPC, nicht berücksichtigt. Für die Verwendung im Kontext einer WOA würden solche Services durch einen Wrapper-Service eingebunden, der einer der drei WP-Kategorien zuzuordnen wäre. Mitunter spricht man bei WP auch von *Web Procedure Call* (WPC), in Anlehnung an RPC, und meint damit alle Arten von Services, die einen entfernten Funktionsaufruf über das Internet ermöglichen (vgl. [VH07]).

3.2.2 Architekturausprägungen

Neben auf den ersten Blick schwer messbaren Größen wie dem Wunsch nach angemessener Technologienutzung für Services oder der Verwendung von klaren Schnittstellenbeschreibungen lässt sich eine WOA auf Strukturebene leichter unterscheiden. Dabei können für die konkrete Ausgestaltung die folgenden vier Punkte als Entscheidungskriterien genannt werden. In Abbildung 3.4 werden zur Vereinfachung nur drei Ausprägungen in Form eines morphologischen Kastens dargestellt, obwohl der Grad der Auslagerung von Daten und Funktionalitäten einen beliebigen prozentualen Wert annehmen kann.

- **Auslagern von Daten**: Dadurch wird ein Maß für die Nutzung von Diensten angegeben, die Daten des Unternehmens speichern. Diese können in der Ausprägung eines DaaS-Angebots oder eines SaaS-Angebots mit eigener Datenhaltung vorkommen.

- **Auslagern von Funktionalität**: Hiermit wird der Grad an verwendeten WP gemessen, die Geschäftslogik enthalten und innerhalb der WOA verwendet werden. Solche Services können auch mit Daten arbeiten, die innerhalb des Unternehmens gespeichert sind.

- **Steuerungspunkte**: Die Lage einzelner Steuerungskomponenten, falls diese existieren, kann als intern oder extern eingestuft werden.

- **Anzahl der Steuerungspunkte**: Diese stellt ein Maß für die Anzahl der WAC-Komponenten dar.

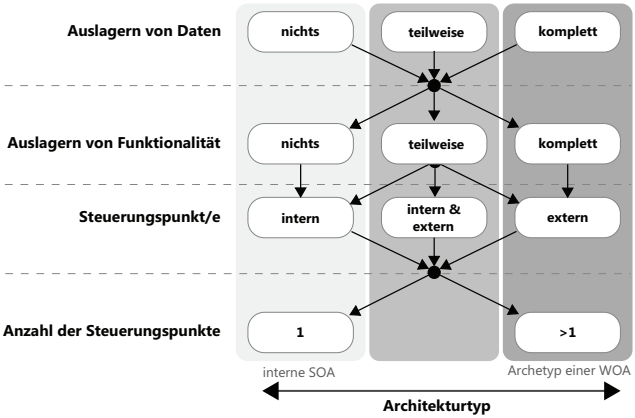


Abbildung 3.4: Morphologischer Kasten der Kernaspekte einer WOA.

Je nach Outsourcing-Affinität des Unternehmens lassen sich daran anschließend vier typische Ausprägungen der Netzwerktopologie einer WOA identifizieren. Die erste Ausprägung ist eine WOA, deren Steuerung innerhalb des Unternehmens liegt und als *Inhouse-WOA* bezeichnet wird. Die zweite Ausprägung beschreibt den Archetypen einer WOA, in dem Angebote des Cloud Computing verwendet werden und eine dezentralisierte, externe Steuerung der WOA realisiert wird. Aus den ersten beiden Ausprägungen lassen sich anschließend zwei Mischformen identifizieren. Zum einen eine WOA mit externer, zentralisierter Steuerung, die ein Einbinden von Altsystemen des Unternehmens ermöglicht. Zum anderen eine WOA, die durch regulative Vorgaben des Unternehmens oder durch Gesetze bestimmte Daten im eigenen Unternehmen speichern muss oder möchte. Abschließend wird noch eine durch das Web 2.0 bekannt gewordene Anwendungsart aus der Sicht einer WOA vorgestellt, das *Mashup*. Die Ausprägungen werden im Folgenden erläutert.

Inhouse-WOA

Die *Unternehmens-interne WOA* liegt, wenn man die Umsetzung betrachtet, am nächsten an einer „klassischen" SOA. Die Kontrolle über alle Prozesse und Services verbleibt komplett innerhalb des Unternehmens. Lediglich einige wenige externe WP werden in die Architektur integriert (vgl. Abbildung 3.5). Diese

Form der Implementierung bedeutet aber auch, dass neben möglichen Altsystemen zusätzlich eigene Hardware betrieben wird, die für die Kontrolle der einzelnen Komponenten notwendig ist. Der Grad der Abhängigkeit zu externen Anbietern sinkt zwar im Vergleich zu Web-lastigeren Lösungen, aber andererseits benötigt man zusätzliche Personalmittel, um beispielsweise Administratoren für die Wartung der Hardware zu beschäftigen. Damit verspielt man die Skalierungs- und Kosteneffekte einer WOA. Darüber hinaus ist eine solche Architekturausprägung nach der hier vertretenen Definition nur dann überhaupt noch eine WOA, wenn die eingebundenen WP zum überwiegenden Teil Infrastrukturkomponenten sind.

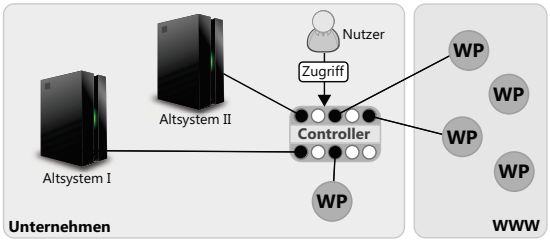


Abbildung 3.5: Unternehmens-interne WOA-Steuerung.

WOA mittels Cloud Computing

Eine WOA-Umsetzung mit mehreren externen Steuerungskomponenten spiegelt die ursprünglich beschriebene WOA-Topologie wider, in der beliebig viele Steuerungselemente zur Verfügung stehen und diese mit WP jeglicher Art verbunden sind. Die Kombination von Prozessen und Orchestrierung von Services geschieht hierbei im gesamten Internet und verteilt sich auf beliebige XaaS-Angebote. Typischerweise wird ein zentraler Zugriffspunkt für Mitarbeiter des Unternehmens implementiert, der hier als ein Pfeil auf einen Controller dargestellt ist. Dies können unter Umständen aber für verschiedene Abteilungen oder Systemteile auch mehrere sein. In Abbildung 3.6 ist dieser Ansatz gezeigt.

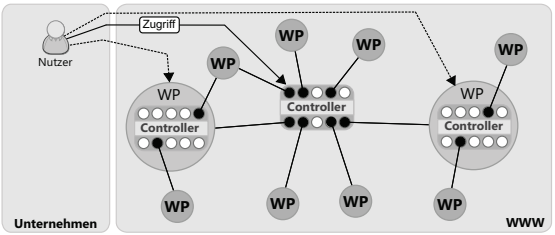


Abbildung 3.6: Externe dezentralisierte WOA-Steuerung.

Mischszenarien

Eine Mischform dieser beiden Ausprägungen manifestiert sich in der Form einer WOA mit Altsystemen und einer zentralisierten Steuerung. Dies bedeutet zunächst, dass die Kontrolle der WOA ins Internet ausgelagert wird und so keine eigene zusätzliche Hardware für das System angeschafft und betrieben werden muss. Darüber hinaus wird ein einziger Controller verwendet, der für alle Prozesse und Abläufe des Systems verantwortlich ist. Dies könnte beispielsweise ein PaaS-Anbieter sein, der es dem Unternehmen ermöglicht, eigene Prozesse auf der bereitgestellten Plattform zu implementieren. Dennoch benötigt das Unternehmen noch Altsysteme, die in die Gesamtarchitektur eingebunden werden müssen und daher von dem im Internet liegenden Controller angesteuert werden (vgl. Abbildung 3.7). In dieser Ausprägung lassen sich einige Vorzüge der WOA nutzen. Darüber hinaus bleiben jedoch Kosten bestehen, wie beispielsweise Personalkosten für Administratoren der Altsysteme.

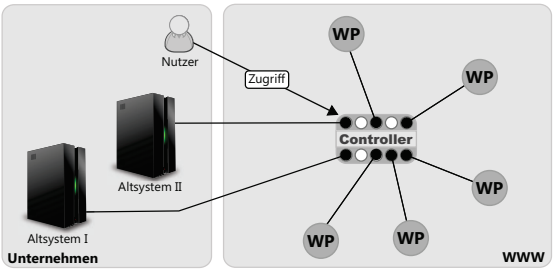


Abbildung 3.7: Externe zentralisierte WOA-Steuerung mit Alt-Systemen.

Hat ein Unternehmen keine Altsysteme oder ersetzt diese durch WP aus dem Internet, so spart es nicht nur Hardware-, sondern zusätzlich auch Personalkosten, da keine Administratoren benötigt werden (vgl. Abbildung 3.8). Der Zugriff der Mitarbeiter geschieht hierbei in beiden Versionen direkt auf einem zentral angelegten Controller.

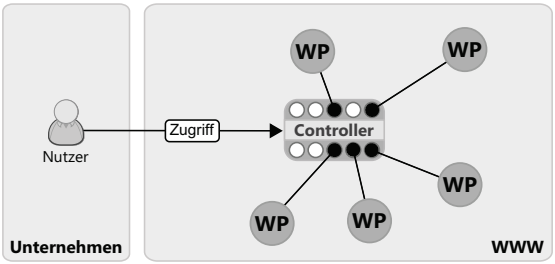


Abbildung 3.8: Externe zentralisierte WOA-Steuerung ohne Alt-Systeme.

Regulatorisch bedingte WOA

Die bisher aufgezeigten WOA-Ausprägungen sind eher theoretischer Natur, und beschreiben, wie die einzelnen Netzwerktopologien bei unterschiedlicher Ausprägung der Architektur aussehen würden. Dabei spielen aber nicht nur technologische und administrative Fragestellungen eine Rolle bei dem Systemdesign einer verteilten Anwendung. Eine realistische Betrachtung der Umsetzung einer WOA muss ebenfalls juristische sowie unternehmerische Richtlinien beachten. Durch Vorgaben zur revisionssicheren Archivierung und Aufbewahrung von elektronischen, geschäftsrelevanten Informationsobjekten, die im Handelsgesetzbuch (HGB) sowie in der Abgabenordnung (AO) geregelt sind, sollten manche Daten innerhalb des Unternehmens gespeichert und verwaltet werden. Selbst wenn ein Unternehmen dafür einen vertraglich gebundenen externen Dienstleister verpflichtet, ist es wahrscheinlich, dass ein gewisses Maß an vertraulichen Dokumenten die Unternehmensgrenzen nicht verlassen soll. Aus diesem Grund wird man in einem realistischen Szenario zusätzlich zu einem dezentralisierten WOA-Ansatz im Internet noch mindestens einen weiteren Controller im Unternehmen vorhalten, der die Verwaltung und

Steuerung von regulatorischen Daten und deren Verarbeitung, ggf. durch weitere interne WP, vornimmt. Dies ist in Abbildung 3.9 dargestellt.

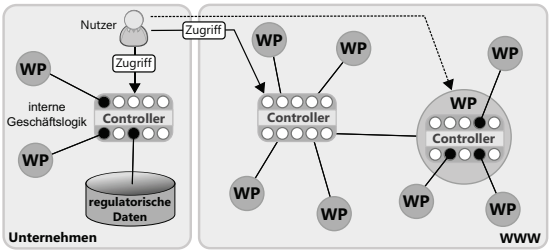


Abbildung 3.9: Regulatorische Daten verbleiben im Unternehmen.

Die aktuelle Forschung im Bereich des Speicherns von verschlüsselten Daten in DBMS, die die Verwendung von DaaS-Angeboten auch für vertrauliche Dokumente ermöglichen würde, ist noch nicht weit genug, um produktiv eingesetzt werden zu können. Das Verschlüsseln an sich ist hierbei weniger ein Problem. Vielmehr ist es schwierig, auf verschlüsselten Daten Indizes zu erstellen, um performante Datenabfragen zu gewährleisten. Es gibt hier aber bereits Ansätze, die sich beispielsweise mit der Abfrage von verschlüsselten XML-Daten als DaaS-Angebot (vgl. [nG08]) oder der generellen Sicherheit von Datenspeicherung in Cloud Computing befassen (vgl. [WWRL09]). Eine Auflistung aktueller Möglichkeiten der Abfrage von verschlüsselten Daten ist in [DFPS07] aufgelistet. Sollten hier in den nächsten Jahren signifikante Verbesserungen auftreten, so wäre auch die Umsetzung einer archetypischen WOA realistisch, die komplett aus einem Unternehmen ausgelagert wird. Eine Zusammenfassung der vorgestellten Architekturausprägungen ist in Tabelle 3.2 dargestellt.

Mashup

Neben den generellen Ausprägungen einer WOA wird nun noch eine im Internet sehr häufige Form der Web-Architektur – das Mashup – vorgestellt, die eine einfache Ausgestaltung der externen, zentralisierten WOA darstellt. Diese spezielle Art der Web-Applikation gründet auf dem Trend des Web 2.0 und versucht, zwei oder mehrere Webseiten zu einem größeren Konstrukt zu

Tabelle 3.2: Überblick über WOA-Ausprägungen.

	Outsourcing		Steuerungspunkte	
	Daten	**Funktionalität**	**Ort**	**Anzahl**
Unternehmens-intern	○	◐	Intern	> 1
Extern, dezentralisiert	●	●	Extern	> 1
Extern, zentralisiert	●	◐/●[a]	Extern	1
Regulatorisch	◐	◐	Extern + Intern	> 2

● = komplett vorhanden; ◐ = teilweise vorhanden; ○ = nicht vorhanden

[a] Mit/ohne Altsysteme

verbinden. Durch die Kombination von verschiedenen Daten oder Funktionalitäten, die zuvor nicht miteinander verbunden waren, lassen sich so neue Funktionen oder Erkenntnisse erschließen (vgl. [Alb08]). Vor Zeiten des Web 2.0 war es oft schwierig, Inhalte von Webseiten für eine Weiterverwendung zu erhalten, so dass nur die Möglichkeit des Extrahierens von Daten aus einer HTML-Seite, das sogenannte *page scraping*, oder das Auslesen von Daten aus Web-Feeds wie RSS und ATOM übrig blieb (vgl. [VH07]). Durch den Trend für die Dienste oder Daten der eigenen Webseite einfache APIs anzubieten, ist die Möglichkeit der Verknüpfung drastisch vereinfacht und damit die Qualität von Mashups gesteigert worden. Mit das bekannteste Beispiel ist hierbei die Web-Applikation *Google Maps*, bei der man zunächst Landkarten betrachten und Routenplanungen vornehmen kann. Durch die Möglichkeit, die Landkartendaten per API zu verwenden und damit weitere Daten zu verknüpfen, entstanden unzählige Mashups, die durch das Einblenden zusätzlicher Daten auf einer Landkarte neuen Nutzen stiften. Eines der ersten Mashups dieser Art ist `HousingMaps.com`, auf dem Immobilien mit Preisen und Fotos auf einer Landkarte angezeigt werden. Dafür werden zum einen der Dienst Google Maps, für die Kartendarstellung, und zum anderen die Webseite craigslist.com, für die Informationen über Immobilien, verwendet.

Bei einem typischen Mashup geht es also darum, zwei APIs miteinander zu verbinden, und die daraus resultierenden Informationen zu nutzen. Hierfür lässt sich jedwede Art von Schnittstelle unabhängig von der technischen Realisierung verwenden. In Abbildung 3.10 ist ein Beispiel gezeigt, welches den Aufbau eines Mashups aus Sicht einer WOA am Beispiel von `HousingMaps.com` zeigt. Dabei werden zwei Applikationen über die zur Verfügung gestellte API

angesprochen und deren Daten oder Funktionalitäten durch einen Controller zu einem Mashup verknüpft. Im speziellen Fall eines Mashups würde hier ein Controller ausreichen, der die APIs aufrufen kann und die Möglichkeit bietet, eigene Logik zu implementieren.

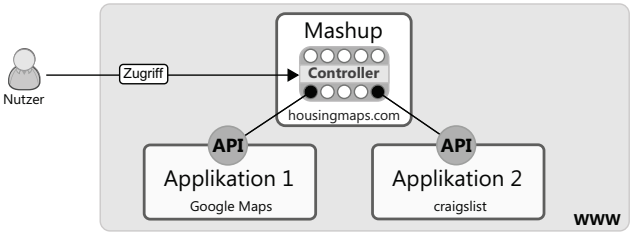


Abbildung 3.10: Architektur eines Mashups aus Sicht einer WOA.

Gerade im Bereich der sozialen Netzwerke und der privaten Nutzung des Internets entstehen immer weitere Mashups, die zeigen, wie einfach eine Vernetzung von Applikationen mit den Standards des Internets funktioniert. Das Service-Verzeichnis `ProgrammableWeb` verzeichnet aktuell über 5.500 Mashups[45]. Auch wenn diese Art der Verwendung von Mashups bisher hauptsächlich auf den privaten Sektor beschränkt blieb, gibt es immer mehr Einflüsse auf die Architekturen und Softwarelandschaften innerhalb von Unternehmen. Ein Stichwort ist hierbei auch *Enterprise Mashups* oder *Enterprise 2.0*, was sich als Thema immer mehr auch in Unternehmen durchsetzt, und im Gegensatz zu klassischen Softwaresystemen, wie einer SOA, durch einfache Programmierung charakterisiert ist (vgl. [HSSJS08]). Eine einfache Variante eines Mashup-Controllers wird dabei durch den Dienst *Yahoo!Pipes* verkörpert[46]. Dieser ermöglicht das Anlegen eines Workflows im Browser, der verschiedene News-Feeds miteinander verknüpfen kann und daraus eine neu kombinierte Informationsquelle erstellt. Diese Anwendung stellt eine rudimentäre Form des Controllers dar, der für eine WOA benötigt wird.

[45] Quelle: `www.programmableweb.com`, Stand: 07.02.2011.

[46] Siehe: `http://pipes.yahoo.com/pipes/`.

3.2.3 Schichten einer WOA

Neben der Sicht auf die Netzwerktopologie und deren Ausprägungen, die dabei eine Rolle spielen können, lassen sich auch Schichten in einer WOA identifizieren. In Abbildung 3.11 ist auf der linken Seite das Unternehmen dargestellt, welches drei optionale Ebenen enthält: eine **Infrastrukturschicht**, in der sich Altsysteme oder Datenbanken für das Speichern regulatorischer Daten befinden, eine **interne Steuerungsschicht**, die IT-Prozesse innerhalb des Unternehmens steuert (beispielsweise ein ESB) und eine **interne Serviceschicht**, die für alle Services steht, die innerhalb des Unternehmens implementiert sind und angeboten werden. Diese drei Ebenen sind optional, da sie in einer archetypischen WOA nicht vorhanden sein sollten. Da jedoch im Falle einer Migration von einem bestehenden IT-System auf eine WOA interne Schichten nicht zu vernachlässigen sind, wurde hierbei die Aufteilung an den Aufbau einer SOA angelehnt. Auf der rechten Seite wird der Teil einer WOA skizziert, der sich im Internet befindet. Hierbei wurden die aus dem Cloud Computing bekannten Schichten verwendet, um eine grobe Schichtenaufteilung zu erreichen. Dies sind die **SaaS-Schicht**, die komplette Software-Angebote enthält, die **PaaS-Schicht**, die für Geschäftslogik und die Steuerung sowie Verwaltung einer WOA vorgesehen ist (hier kommt der in den Topologien gezeigte Controller zum Einsatz), und die **IaaS-Schicht**, die Infrastrukturkomponenten bereitstellt. Eine zusätzliche Schicht, die **externe Serviceschicht**, dient als Sammelbecken für alle weiteren Services, die im Internet über WP angeboten werden, und nicht direkt einer der Kategorien des Cloud Computing zugeordnet werden können. Somit erfolgt die Schichtenaufteilung auch strukturgleich (wenn auch nicht ganz identisch) zur funktionalen Service-Klassifikation, die eine vergleichbare Aufteilung in Infrastruktur, Fachlogik und Präsentation vorsieht (vgl. Kapitel 2.3.5).

Ein wichtiger Punkt ist hierbei, dass sich XaaS-Angebote nicht immer eindeutig einer Schicht zuordnen lassen und die Grenzen zwischen diesen Schichten fließend sind. Das Beispiel DaaS wurde bereits als ein solcher Fall genannt. Ein weiteres Beispiel sind E-Mail-as-a-Service-Angebote, die in Form eines reinen Exchange-Servers einerseits als Infrastruktur angesehen werden können, sich aber andererseits durch das Anbieten einer Web-Oberfläche für das Versenden von Mails auch auf der SaaS-Schicht anordnen lassen. Es kommt daher oft

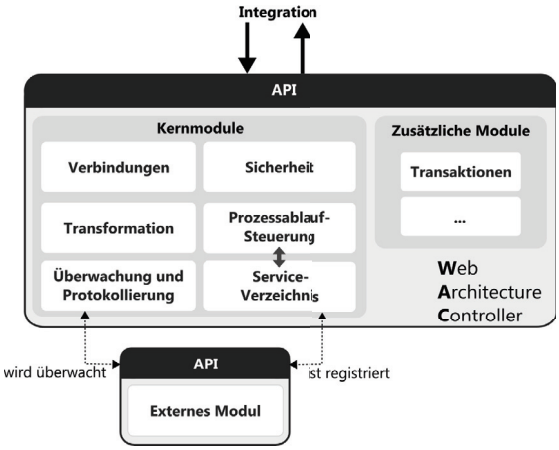
Abbildung 3.11: Schichten einer WOA.

auf den situativen Kontext an, in dem ein XaaS-Angebot oder ein WP verwendet wird, um die Einordnung in die Schichtenarchitektur vorzunehmen.

3.2.4 Steuerungskomponente einer WOA

Die vorgestellten WOA-Ausprägungen bilden eine erste Gliederungsstufe für die umzusetzenden Prozesse und Services. Die Begriffe SaaS und PaaS wurden hierbei bewusst nicht innerhalb der Varianten verwendet, um dem generischen Element des *Controllers* Platz zu lassen. Dies verdeutlicht, dass nicht unbedingt ein vorhandener Service-Anbieter nötig ist, um eine WOA zu verwirklichen. Ebenso ist es möglich, eine eigene Controller-Komponente zu verwenden. Um die Planung der Architektur an dieser Stelle sinnvoll fortzuführen, werden im Folgenden die Anforderungen an einen Controller erläutert, die ebenso für einen PaaS-Anbieter gelten, der Werkzeuge für die Orchestrierung von Services und Prozessen bereitstellt. Die Anforderungen orientieren sich hierbei zum Teil an denen eines ESB, da die zu übernehmende Tätigkeit, nämlich als Vermittler innerhalb der Architektur zu agieren, sehr ähnlich ist. Darüber hinaus sollen im Controller auch Prozesse ablaufen können, die über die Standardfunktionalität eines ESB hinaus gehen (vgl. [Pap07]). Folgende Aspekte sind zu beachten:

- **Dynamisches Verbinden von Services** (*Connectivity*) und **verlässlicher Nachrichtenversand** (*Reliability*): Es muss zum einen jedwede Art von Services aufrufbar sein und zum anderen müssen Nachrichten auch sicher beim Empfänger ankommen. Bedingt durch die Verwendung von



Quelle: angelehnt an [TV09a]

Abbildung 3.12: Schematische Darstellung eines Controllers und seiner Module.

HTTP lassen sich hierbei nur synchrone Transfermuster bei dem Nachrichtenversand realisieren. Dies steigert die Verlässlichkeit insofern, als dass man durch vom Empfänger zurückgesandte Return-Codes eine direkte Rückmeldung über den Erhalt der Nachricht oder aufgetretene Fehler erhält.

- **Nachrichtentransformation** (*Transformation*): Der Controller muss in der Lage sein, die gängigen WOA-Formate ineinander zu transformieren, beispielsweise semistrukturierte Formate wie XML und JSON.

- **Prozessausführung**: Die Ausführung von Transaktionen und lang laufenden Prozessen muss der Controller ebenfalls ermöglichen. Dies ist von sehr großer Bedeutung, da viele Prozesse und deren Logik innerhalb des Controllers implementiert werden sollen. Es lassen sich hierfür beispielsweise Workflow Engines für BPEL oder andere Prozesssprachen einsetzen.

- **Sicherheitsfunktionalitäten**: Neben SSL-Verbindungen muss ein Controller auch die Möglichkeit von Verschlüsselung sowie Erstellung und

Validierung von Signaturen beim Nachrichtenversand ermöglichen. Hierbei sollte der Aufbau einer PKI möglich sein, um ggf. bereits vorhandene X.509-Zertifikate (vgl. [HPFS02]) zu verwenden.

- **Integrationsfähigkeit**: Im Sinne eines ESB bedeutet dies die Möglichkeit zur Einbindung von Altsystemen. Innerhalb einer WOA wird darunter zum einen die Möglichkeit der Integration diverser WP verstanden. Zum anderen ist damit auch die Einbindung eines Controllers selbst als WP gemeint.

- **Steuerung und Überwachung**: Die Steuerung (*Governance*) der Prozesse und Funktionalitäten des Controllers muss über eine Web-Oberfläche und, für die Einhaltung der Integrationsfähigkeit, ebenfalls über eine Web API vorzunehmen sein. Des Weiteren sollen Überwachungsfunktionalitäten angeboten werden, wie beispielsweise ein Zugriffsprotokoll auf Services oder Statistiken der Prozesse.

- **Service-Verzeichnis**: Die in Prozessen verwendeten WP sowie weitere externe Module des Controllers müssen intern mit einer Beschreibung bzw. Schnittstellendefinition gespeichert sein. Auf Basis dieses Verzeichnisses wird auch die Überwachung aller angeschlossenen Komponenten durchgeführt.

- **Skalierbarkeit**: Diese Anforderung beschreibt, dass durch eine Aufstockung der Hardware oder ein Hinzuschalten weiterer Controller, die Leistung des Controllers gesteigert bzw. die Last auf mehrere Controller verteilt werden kann. Damit soll die Arbeit des Controllers beispielsweise bei Lastspitzen oder der geplanten Erweiterung des Systems ohne große Performanzverluste fortgeführt werden. Im Falle von SaaS- und PaaS-Angeboten werden dazu weitere Ressourcen vom Service-Anbieter in Anspruch genommen.

Die Möglichkeiten der Realisierung eines WAC durch aktuelle XaaS-Anbieter im Internet und die darüber hinaus anfallenden Programmierarbeiten werden in Abschnitt 5.8 näher erläutert.

3.2.5 Technologien

Obwohl das Konzept einer WOA als IT-Strategie verstanden wird und nicht nur als reine Technologieumsetzung eines verteilten Systems, ist die Wahl der Mittel zur Umsetzung dennoch eingeschränkt. Als Web-basiertes Konzept bildet HTTP/S in jedem Fall als Transportprotokoll den Kern einer WOA. Gemäß der hier vorgenommenen Definition einer WOA sind zusätzlich Web API als einfachste Umsetzung von Service-Funktionalitäten dem Kern zuzuordnen. Neben der Identifikation einer WOA über die Nutzung von IaaS lässt sich damit bereits eine Abgrenzung gegenüber SOA und ROA auch in informationstechnischer Sicht vornehmen. Zum einen werden in einer ROA nur echte REST-Services verwendet, die nicht mit Web API gleichzusetzen sind, zum anderen verwendet eine typische SOA[47] Web Services auf SOAP-Basis. Als Format der Nutzlast, oder aus ROA-Sicht der Repräsentation, wird häufig das JSON-Format verwendet. Dies wird, obwohl nur als „Best Practice" angesehen, dem Kern einer WOA zugerechnet. Die darüber hinaus anwendbaren Technologien und Standards umfassen, wie bereits angedeutet, alle in einer SOA als auch einer ROA beheimateten Technologien. Je nachdem, welche Mittel nötig sind, um ein softwaretechnisches Ziel zu erreichen, bedient sich eine WOA der im jeweiligen Umfeld vorhandenen Mittel. Diese Auffassung wird durch ein Schalenmodell in Abbildung 3.13 verdeutlicht, welches vom Kern ausgehend mögliche Erweiterungen vorsieht. Innerhalb dieser Erweiterungen liegen die Kerne oder die äußeren Schichten der eingegliederten Bereiche von SOA und ROA. Neben dem Kern werden hier zwei Erweiterungsstufen identifiziert.

Governance-Stufe Die erste Erweiterungsstufe der WOA umfasst zum einen Standards zur Steuerung, Verwaltung und Überwachung einer WOA und zum anderen ggf. weitere Service-Arten, wie Web Services und REST-Services. Zunächst ist der generische Controller in Form eines PaaS-Dienstes zu nennen, der aber nicht als Standard angesehen werden kann, hier aber der Vervollständigung dient. Ein wichtiger Punkt bei der Steuerung von Services ist zunächst deren Beschreibung und die darauf aufbauende Komposition. Dies lässt sich beispielsweise durch WADL, welches sich ausgezeichnet für die Beschreibung von Web API eignet, oder WSDL ab Version 2.0 erledigen. Für die Prozesssteuerung eignet sich BPEL, da sich damit mittlerweile auch HTTP-basierte

[47] In diesem Kontext wird von der gebräuchlichsten Umsetzung einer SOA ausgegangen.

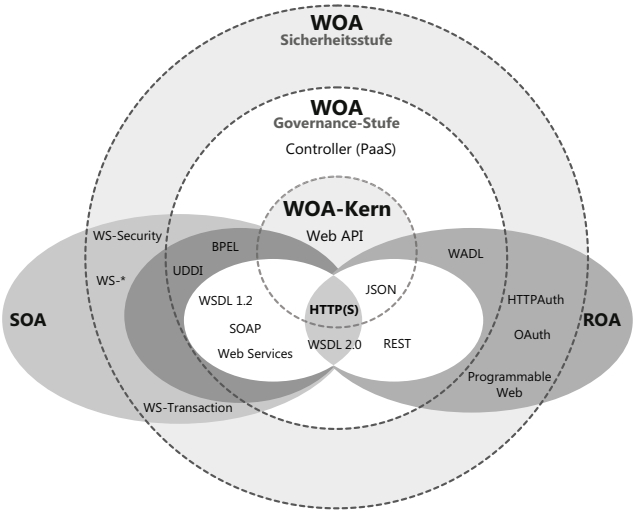


Abbildung 3.13: Technische Einordnung einer WOA.

Services orchestrieren lassen (vgl. [Pau08, Ove07b]). Ein Verzeichnisdienst wie UDDI wird hier zwar genannt, sollte aber zugunsten eines Web-näheren Konzepts, wie dem Verzeichnisdienst *ProgrammableWeb*, nicht verwendet werden.

Sicherheitsstufe Die nächste Erweiterungsstufe einer WOA betrifft vor allem die Sicherheit von Services und Prozessen. Hierbei sind nicht nur Themen wie Authentifizierung und Verschlüsselung, sondern auch Verlässlichkeit und transaktionale Garantien relevant. Ein Weg, um beispielsweise Authentifizierung in einer WOA zu realisieren, ist die Verwendung von HTTPAuth. Ebenso können bei Verwendung von XML als Nutzlast einer Web API auch diverse weitere Standards für Nachrichtensicherheit aus dem Web Service-Bereich eingesetzt werden, wie beispielsweise WS-Security. Darüber hinaus lassen sich hier auch eigene Verfahren entwickeln und einsetzen.

3.3 Stakeholder einer WOA

Der Erfolg eines verteilten IT-Systems für ein Unternehmen hängt, unabhängig von der technologischen Umsetzung, nicht zuletzt von den Personen ab, die das System konzipieren, erstellen, unterstützen, warten und benutzen. Diese Personen mit einem Interesse an dem IT-System (*Stakeholder*) haben unterschiedliche Anforderungen daran und erfüllen unterschiedliche Aufgaben im System-Kontext. In [IEE01] werden vier Stakeholder genannt, die bei der Beschreibung einer Softwarearchitektur die typischen Interessenten darstellen: Entwickler, Nutzer des Systems, technisches Personal (beispielsweise Administratoren für Wartungsaufgaben) und Käufer des Systems. Im Kontext einer WOA lassen sich die folgenden Stakeholder identifizieren und in Relation mit den zuvor genannten setzen:

- **IT-Architekt**: Die Planung und Umsetzung einer WOA wird durch die Rolle eines IT-Architekten ausgeführt. Mehr noch als die des Entwicklers ist diese Rolle für die Konzeption der Struktur einer WOA sowie der Projektdurchführung verantwortlich.

- **Entwickler**: Die Entwickler implementieren fachlogische WP, setzen Geschäftsprozesse aus WP zusammen und sorgen für die technische Umsetzung der WOA.

- **Domänenspezialist**: Diese Rolle erfüllt zum einen eine Beratungsfunktion der IT-Architekten einer WOA und unterstützt zum anderen die Entwickler bei der fachlichen Umsetzung. Darüber hinaus sind Domänenspezialisten auch gleichzeitig Nutzer und Tester des Systems, die fachliche Kompetenz in ihrem Bereich mitbringen und dadurch die erstellten Geschäftsprozesse und Implementierungen aus fachlicher Sicht bewerten können.

- **Dateneigentümer (*data owner*)**: Diese Rolle ist für die erfolgreiche Erstellung eines verteilten Informationssystems, insbesondere in größeren Unternehmen, von Bedeutung. Es existieren meist Daten in einzelnen Fachabteilungen, die angebunden und zur Verfügung gestellt werden müssen. Um hierbei ein Maß an Kontrolle und Überblick über den Zugriff auf Daten und deren Verarbeitung auch in den jeweiligen Fachabteilungen zu belassen, also dezentral zu koordinieren, benötigt man

diverse Verantwortliche. Diese bestimmen über generelle und spezielle Zugriffsrechte auf ihre Daten sowie über Begriffsdefinitionen bei Unklarheiten.

- **Controller**: In einer idealtypischen WOA fallen Wartungsaufgaben fast komplett weg und verringern so den Bedarf an technischem Personal, wie Administratoren. Die Wartung beschränkt sich hierbei hauptsächlich auf das Kontrollieren der Funktionsfähigkeit der Prozesse und externen Services im Betrieb einer WOA. Eine andere Funktion erfüllt diese Rolle noch in Form der Kostenkontrolle einer WOA. Daher wird sie hier nicht als Administrator, sondern als Controller bezeichnet.

- **XaaS-Anbieter**: Dieser Stakeholder befindet sich außerhalb des Unternehmens und hat vor allem Interesse an einer lauffähigen Implementierung, da dies für ihn Umsätze aus laufenden Geschäftsbeziehungen bedeutet.

Ohne auf das Vorgehensmodell zur Umsetzung einer WOA vorzugreifen, lassen sich die generischen Phasen in einem Softwareerstellungsprozess benennen. Diese bestehen aus der **Planung und Analyse**, der **Realisierung und Tests** sowie dem **Betrieb** einer Software (vgl. [Bal01]). Um den Umfang der notwendigen Mitarbeit an der Erstellung und dem Betrieb einer WOA zu verdeutlichen, zeigt Tabelle 3.3 eine Übersicht über die Partizipation der Stakeholder in den einzelnen Phasen. Dabei symbolisieren eingeklammerte Häkchen eine mögliche Mitarbeit des Stakeholders in einer Phase. Anhand dieser Übersicht wird bereits deutlich, dass mitnichten nur „Programmierer" an der Umsetzung eines solchen Systems mitwirken.

Tabelle 3.3: Stakeholder und deren Partizipation im WOA-Lifecycle.

	IT-Architekt	Entwickler	Domänenspezialist	Data owner	Controller	XaaS-Anbieter
Planung und Analyse	✓	(✓)	✓	(✓)		
Realisierung und Tests	✓	✓	✓	✓		✓
Betrieb		(✓)		✓	✓	✓

4 Verwandte Arbeiten

Dieses Kapitel stellt die wichtigsten wissenschaftlichen und praxisorientierten Ansätze vor, die sich mit ähnlichen Fragestellungen befassen wie die vorliegende Arbeit. Dabei werden zur Abgrenzung die jeweiligen identifizierten Unterschiede aufgezeigt. Arbeiten, die Ähnlichkeiten in Planung, Entwicklung und Zielsetzung zu einer WOA aufweisen, lassen sich hauptsächlich im Bereich der Service-orientierten Vorgehensmodelle finden. Einige Aspekte der hier vorgestellten verwandten Methoden finden sich daher auch in der WOA-Methode des nächsten Kapitels in ähnlicher Form wieder. Der in der Einleitung angeführte SDLC lässt sich als grundlegendes Modell eines Softwareerstellungsprozesses ansehen und definiert eine Abfolge sich bedingender Phasen, die aber nicht notwendigerweise in sequenzieller Abfolge durchlaufen werden müssen. Begonnen wird generell bei der Planung und Analyse eines bestehenden oder zu erstellenden Systems, gefolgt von dem Entwurf und der Entwicklung bis zur Implementierung, der Wartung und abschließend der Abschaltung eines Systems. Die vorliegende Arbeit verwendet eine grobe Einteilung des SDLC in die drei Phasen *Analyse und Planung*, *Realisierung* sowie *Betrieb* (siehe Abbildung 1.1). Die hier untersuchten verwandten Arbeiten decken jeweils einzelne Phasen oder sogar den gesamten SDLC ab. Daher werden die im Folgenden vorgestellten Methoden und Modelle aus dem Service-orientierten Bereich zur Übersichtlichkeit in drei Gruppen eingeteilt:

- **Architekturmodelle**: Hier werden Modelle vorgestellt, die die System-Architektur einer SOA spezifizieren. Diese Modelle beziehen sich auf keine spezielle Phase des SDLC und beschreiben auch kein Vorgehen zur Realisierung einer Software. Sie dienen hauptsächlich der Erläuterung von technischen oder fachkonzeptionellen Zusammenhängen.

- **partielle Vorgehensmodelle**: Hier werden einzelne Teile der Planung oder Umsetzung einer SOA beschrieben. Dies können Ansätze für die Durchführung einer kompletten Analyse- und Planungsphase oder auch nur einzelner Phasen sein. Diese Ansätze decken meistens nur einen

Teil des SDLC ab. So lässt sich beispielsweise die Beschreibung der Geschäftsprozessmodellierung für eine SOA in die Analyse- und Planungsphase einordnen und beschreibt damit nur einen kleinen Teilbereich der Softwareerstellung.

- **vollständige Vorgehensmodelle**: Es werden von der Analyse- und Planungsphase bis hin zur Umsetzung weitgehend alle Schritte der SOA-Erstellung erläutert. Einige Modelle gehen darüber hinaus noch auf den Betrieb ein. Hierbei wird oft der komplette SDLC berücksichtigt und alle Schritte nacheinander durchlaufen.

4.1 Architekturmodelle

Die folgenden Ansätze erläutern Architekturmodelle, welche für die Beschreibung einer SOA verwendet werden können. Diese bilden zum Teil die Grundlage für die im Anschluss vorgestellten Vorgehensmodelle.

Eine umfassende Architekturbeschreibung Service-orientierter Systeme stammt vom W3C und wird als *Web Service Architecture (WSA)* bezeichnet. Bei diesem Modell stehen zum einen die Konzepte eines Systems im Vordergrund und definieren dabei die enthaltenen Web Services und ihre Relationen zueinander, zum anderen wird die Architektur einer SOA und die typischerweise verwendeten Technologien beschrieben (vgl. [W3C04c]). Hierin werden vier spezifische Architekturmodelle vorgestellt: das Service-Modell (*service model*), das Nachrichten-orientierte Modell (*message oriented model*), das Regelungs-Modell (*policy model*) sowie das Ressourcen-Modell (*resource model*). Durch diese vier Sichten werden die Web Services innerhalb des Systems zueinander in Relation gesetzt. Dieser Ansatz beschreibt die Systemarchitektur einer SOA, die vollständig aus Web Services zusammengesetzt wird und enthält weder Phasen für Analyse und Planung einer SOA noch konkrete Schritte für die Realisierung. Er ist aus diesem Grund nicht als Vorgehensmodell für die Erstellung einer WOA geeignet.

Das Konsortium *Object Management Group* (OMG), welches unter anderem die Standards BPMN und UML betreut, stellt eine eigene Spezifikation zur Modellierung von SOA bereit: die *Service oriented Architecture Modeling Language* (SoaML). Darin werden UML-Profile zur Beschreibung von Services innerhalb einer SOA erstellt, die durch die Verwendung von UML-Stereotypen

definiert werden (vgl. [Obj09]). Die wichtigsten hierbei definierten Elemente sind die mitwirkenden Komponenten (*participants*), die Schnittstellendefinitionen (*service interface*), die SLAs (*service contract*) und die Orchestrierung (*service architecture*). Eine Umsetzungsmethode, die SoaML verwendet, ist beispielsweise *M³SOA* (siehe `www.mid.de`). Obwohl die Methode des Top-down- und Bottom-up-Ansatzes kurz erwähnt wird, verzichtet diese Architekturbeschreibung ebenfalls auf ein Vorgehensmodell und erläutert vor allem die Beschreibung einer SOA und deren Services durch die Verwendung von UML-Modellen. Der Fokus auf eine festgelegte Technologie sowie das Fehlen von Phasen zur Erstellung eines IT-Systems führen dazu, dass diese Architektur für eine Verwendung im WOA-Kontext nicht geeignet ist. Lediglich die hierin vorgestellte Beschreibung einer Systemarchitektur durch Komponentendiagramme fließt später in die Topologieplanung einer WOA ein.

Das Unternehmen IBM definiert neben einigen Vorgehensmodellen, die später Erwähnung finden, eine SOA-Referenzarchitektur (vgl. [AZE+07a]). Diese ist als eingängiges Schichtenmodell dargestellt und setzt sich aus fünf horizontalen Architekturschichten und vier vertikalen nicht-funktionalen Schichten zusammen. Dieser Aufbau beschreibt ein klassisches SOA-Verständnis, bei dem die unterste Architekturschicht die Anwendungen und Datenbanken beherbergt (*operational layer*). Darauf aufbauend befinden sich die Service-Komponenten (*service components layer*), die Services (*services layer*), die Geschäftsprozesse (*business process layer*) sowie schließlich zuoberst die Nutzer der SOA (*consumers layer*). Schichtenübergreifend werden die vertikalen Schichten verstanden, die für verschiedene nicht-funktionale Aufgaben innerhalb der SOA vorgesehen sind. Diese setzen sich im IBM-Referenzmodell aus der Integrationsschicht (*integration layer*), der Überwachungsschicht (*quality of service layer*), der Metadatenschicht (*information architecture layer*) sowie der Steuerungsschicht (*governance layer*) zusammen. Das Referenzmodell beschreibt die Aufgabe jeder einzelnen Schicht und die Zusammenhänge zwischen diesen. Dieses Modell ist als klassische Schichtenarchitektur für eine SOA-Umsetzung innerhalb eines Unternehmens gut geeignet und wird in den später gezeigten Ansätzen der IBM verwendet. Da hierin aber weder eine Vorgehensmethode beschrieben noch auf XaaS-Dienste eingegangen wird, kann diese Referenzarchitektur nicht für die Realisierung einer WOA eingesetzt werden.

4.2 Partielle Vorgehensmodelle

Die folgenden Ansätze befassen sich mit Teilaspekten der SOA-Systemerstellung. Dabei deckt keines der Modelle alle Phasen ab. Einige behandeln beispielsweise nur planerische Aspekte, während bei anderen weniger die Modellierung als mehr die Umsetzung einer SOA im Vordergrund steht. Um einen besseren Überblick über die einzelnen Ansätze zu bekommen, wurden diese darüber hinaus noch unterteilt in konzeptionelle Modelle und solche, die hauptsächlich in der Praxis verwendet werden.

Konzeptionelle Modelle

Ein Ansatz, der von der WSA inspiriert wurde, resultiert aus dem EU-Projekt *Service Centric Systems Engineering (SeCSE)*. Darin wird ein konzeptionelles Modell zur Beschreibung von Services, von Semantik und von Service-Schnittstellen definiert (vgl. [CDNDP+05]). Zusätzlich wird noch auf das Überwachen von Services (*Monitoring*) eingegangen. Davon abgesehen ist dieser Ansatz nur ein Modell zur Architekturbeschreibung, der keine Vorgehensmethode beinhaltet und aus diesem Grund keine Verwendungsmöglichkeit im Rahmen der Erstellung einer WOA bietet.

Der Beschreibung von Web Services in semantischer Hinsicht sind die Ergebnisse der Arbeitsgruppe *Web Service Modeling Ontology* (WSMO) (vgl. [WSM06]) sowie die Standards OWL-S (vgl. [W3C04b]) und SAWSDL (vgl. [W3C07b]) des W3C gewidmet. Der klare Fokus liegt hierbei auf der Beschreibung einzelner Dienste, anstatt komplette Systemarchitekturen zu berücksichtigen. Alle drei Ansätze bieten umfassende Beschreibungsvorgaben für Services, ohne jedoch eine Vorgehensmethode zu erläutern. Ein interessanter Aspekt im Kontext dieser Arbeiten ist die Zusammenstellung der Service-Eigenschaften, die teilweise in der Beschreibung von WOA-Services berücksichtigt werden.

Der UML verwendende Ansatz in [LSJAnCM08] ist eher auf die Implementierung einer SOA ausgerichtet. Hierin werden UML-Profile erstellt, die SOA-basierende Architekturmodelle darstellen. Im Gegensatz zur WOA-Methode sind hier Prozesse weniger relevant. Die Methode konzentriert sich mehr auf die Definition von Service-Anbietern und konkreten Services durch SOA-Metamodelle. Zusätzlich werden sogenannte *Active Components* festgelegt,

die Service-Anfragen und -Antworten verwalten und mit einem Frontend interagieren. Eine konkrete Umsetzung oder gar der Betrieb einer SOA wird hierbei nicht erläutert.

Das EU-Projekt *SOA4ALL* (vgl. [PD10]) befasst sich mit dem Konzept der *Linked Services* und basiert damit auf der Initiative *Linked Open Data* (LOD). LOD bezeichnet eine Menge an freien Ressourcen (Informationen), die über das Internet per HTTP aufrufbar, untereinander idealerweise verknüpft und semantisch beschrieben sind. Der Grundgedanke ist dabei die Verknüpfung weltweit öffentlicher Informationen zu einem Datennetz, im Gegensatz zum WWW, welches eine lose Verknüpfung von Webseiten darstellt. Vor allem die semantische Verknüpfung zwischen den Ressourcen spielt hierbei eine wichtige Rolle. Der Ansatz von SOA4ALL beschreibt Services eines IT-Systems mittels Eingabe- und Ausgabedaten, wobei funktionale und nicht-funktionale Eigenschaften mittels RDF (vgl. [W3C04a]) erläutert werden, einem W3C-Standard zur semantischen Beschreibung von Ressourcen im Internet. *Linked Services* bilden damit eine Schicht über den aktuell verfügbaren Services des Internets ab. Dieser Ansatz versteht das Internet als Service-Quelle und versucht, diese in eine IT-Architektur einzubinden. Dabei sind aber keine XaaS-Anbieter, wie in einer WOA, als Services vorgesehen, sondern vielmehr die frei verfügbaren, semantisch beschriebenen Informationen des Internets. Darüber hinaus wird hier kein Vorgehensmodell zur Umsetzung eines verteilten Systems beschrieben, sondern nur eine Software-Umgebung zur Erreichung der Zielarchitektur implementiert und vorgestellt. Der Fokus liegt primär auch nicht auf der Realisierung eines IT-Systems für die fachlichen Geschäftsprozesse eines Unternehmens, sondern auf der Nutzung frei verfügbarer Ressourcen und die damit einhergehenden Möglichkeiten des Umgangs mit Informationen.

In der Praxis verwendete Modelle

Zwei Ansätze aus der Praxis befassen sich eher mit den technischen Aspekten einer SOA und lassen in deren Vorgehensmodellen die Geschäftsprozessmodellierung eines Unternehmens außen vor. Fokussierend auf Web Services beschäftigt sich der Ansatz *Service-oriented Transformation of Legacy-Systems* von Nadhan mit der Migration von Altsystemen auf SOA (vgl. [Nad04]), während *Creating Service-oriented Architectures* von Barry & Associates,Inc. eine iterative Einführung von SOA mit einer starken Technik-Zentrierung vorschlägt

(vgl. [Bar03]). Beide Ansätze sind unvollständig in ihrem Vorgehensmodell und stark auf Web Services und deren technische Beschreibung festgelegt, so dass keine WOA-Umsetzung damit möglich ist.

Ebenfalls aus der Praxis kommt der Ansatz *Enterprise-SOA-Roadmap* (ESOA) der SAP AG (vgl. [KP09, HL07]). Im Gegensatz zu den vorherigen Methoden befasst sich diese hauptsächlich mit strategischen und organisatorischen Aspekten einer SOA-Umsetzung. Es erfolgt hierbei daher keine Analyse und Planung des IT-Systems im Sinne einer Softwareerstellung. Des Weiteren ist hierin ein expliziter Bezug auf SAP-Produkte enthalten, womit eine generelle Verwendung für SOA-Projekte erschwert wird. Dies führt dazu, dass sich ESOA nicht für die Planung und Umsetzung einer WOA eignet.

Der Ansatz *A Methodology for service architectures* von Jones befasst sich ausschließlich mit der Top-down-Modellierung der fachlichen Zusammenhänge eines Unternehmens, um daraufhin auf die Verwendung von Services schließen zu können (vgl.[JM05]). Beginnend mit Use-case-Modellen auf oberster fachlicher Ebene werden beliebig viele Iterationen der Verfeinerung durchlaufen. Dabei unterscheidet Jones drei verschiedene Service-Typen: virtuelle (*virtual*), unterstützende (*support*) sowie geteilte (*shared*) Services. Es wird hierbei betont, dass die wichtigsten Fragen einer Modellierung das Was, das Wer, das Warum und das Wie darstellen. Zudem sollte man nicht mit einer Prozessmodellierung starten, um nicht zu früh zu viele Details in der Planungsphase aufzunehmen. Eine Betrachtung der Planung architektonischer Details oder technischer Feinheiten von Services fehlt in dieser Methode vollständig, so dass sich mit diesem Ansatz zwar die ersten Analyseschritte, aber keine SOA-Umsetzung durchführen lässt. Der Ratschlag, mit Use-case-Modellen für die erste Analyse der fachlichen Struktur eines Unternehmens zu beginnen, wird in der später vorgestellten WOA-Methode in der Analyse- und Planungsphase aufgegriffen.

Ein weiteres Vorgehensmodell stellt das *Service-oriented Modelling Framework* (SOMF) dar, welches durch Bell entwickelt wurde (vgl. [Bel08]). Darin werden vier Arbeitspakete definiert, die wiederum verschiedene kleinere Phasen beschreiben. Die Arbeitspakete befassen sich mit Praktiken (*Practices*), Systemumgebung (*Environments*), Disziplinen (*Disciplines*) und Artefakten (*Artifacts*). Innerhalb dieser Pakete werden diverse Analysen als auch Design-Phasen rund um eine SOA durchgeführt. Dabei liegt der Fokus auf der Modellierung einer SOA. Methoden oder Vorgehensweisen für die Umsetzung einer SOA werden

nicht aufgegriffen. Daher eignet sich auch diese Methode nicht für die Planung und Umsetzung einer WOA.

4.3 Vollständige Vorgehensmodelle

Im Folgenden werden Vorgehensmodelle vorgestellt, die als vollständig angesehen werden, da sie von der Analyse über die Planung bis zur Umsetzung einer SOA alle Phasen überdecken.

Konzeptionelle Modelle

Mos et al. beschreiben eine Methode für das Erstellen einer SOA mit dem eindeutigen Ziel der Nutzung eines ESB als Rückgrat des Systems (vgl. [MBQM08]). Hierin wird die Verwendung des Standards *Java Business Integration* (JBI) strikt vorgegeben, was die Umsetzung auf Java-Umgebungen einschränkt. Die Methode teilt sich dabei in drei Phasen ein. Begonnen wird mit der Beschreibung der vorhandenen Prozesse durch Domänenspezialisten mittels BPEL oder BPMN. In der zweiten Phase wird von Software-Architekten die *Service Component Architecture* (SCA) beschrieben. Zuletzt werden die Komponenten (Services) durch ein Technologie-Team in der ESB-Umgebung umgesetzt. Dieser Ansatz ist auf Web Services und JBI ausgerichtet und damit zu speziell für die Umsetzung einer WOA. Die Festlegung der durchführenden Rollen und Aufgaben in einem Entwurfsprozess sind jedoch gut spezifiziert, so dass diese in die WOA-Methode einfließen.

Der Ansatz *Conceptual Service Modelling (COSMO)* beschreibt ebenfalls drei Phasen (vgl. [QSPS07]). Hierbei wird der Fokus auf die grundlegenden Konzepte einer SOA gelegt, um essenzielle, elementare und generische Service-Eigenschaften zu repräsentieren. Prozesse werden hier auf drei Abstraktionsebenen betrachtet: zunächst als einzelne Interaktion, dann als Choreografie und abschließend als Orchestrierung. Für die Beschreibung der Services und ihrer Interaktionen wird neben den Sprachen *Web Ontology Language* (OWL) und SPARQL hauptsächlich die *Interaction System Design Language* (ISDL) verwendet. Da das Grundkonzept einer WOA die Einfachheit der Mittel und Vereinfachung der Architektur anstrebt, ist diese Methode zu technisch festgelegt und zu komplex für die Verwendung. Dennoch ist die hier vorgenommene

Aufteilung von Service-Anbieter und -Nutzer und das Herausstellen der Interaktionen zwischen den beiden Akteuren ein sinnvoller Aspekt, der später in der WOA-Methode aufgegriffen wird.

Als vollständige Planungsmethode für eine SOA versteht sich auch die *Service-orientierte Architektur Methode* (SOAM) von Offermann (vgl. [Off08]). Anhand der Verbindung von Top-down- und Bottom-up-Vorgehen wird eine SOA durch die folgenden sechs Phasen geplant und umgesetzt: Unternehmensanalyse, Identifikation von Serviceoperationen, Bestandssystemanalyse, Konsolidierung, Servicedesign und Prozessaufbereitung. Diese Methode geht grundsätzlich von der Existenz eines IT-Systems aus, welches in die SOA integriert werden soll. Für die Umsetzung einer SOA wird in der Prozessaufbereitungsphase auf Ansätze wie XP oder den *Rational Unified Process* (RUP) verwiesen, diese werden aber nicht weiter ausgeführt. Auch der Betrieb einer SOA wird nicht weiter konkretisiert. Daher ist diese Methode zwar als vollständig im Sinne der Erstellung einer SOA anerkannt, wird jedoch nicht als ausreichend für eine WOA-Umsetzung angesehen.

Das Vorgehensmodell von Lamparter *An Interdisciplinary Methodology for Building Service-oriented Systems on the Web* entstand im Rahmen eines Projekts des Bundesministeriums für Bildung und Forschung (BMBF) namens THESEUS (vgl. [LS08]). Darin werden drei Vorgehensmodelle zu einem Framework zusammengefasst, welches das Erstellen von Service-orientierten Systemen beschreibt und dabei den Fokus auf die Kommunikation zwischen Service-Anbieter und -Nutzer legt. Die Methode umfasst zunächst das *Web Service Engineering* und beschreibt darin die Entwicklung einer SOA. Lamparter verweist für die Durchführung des Vorgehensmodells auf die IBM-Methode *SOAD*, die später skizziert wird. Daneben steht das *Market Engineering*, in dem die Umgebung der SOA als elektronischer Markt aufgefasst wird und Service-Anfragen und -Angebote koordiniert werden müssen. Die dritte Komponente ist das *Ontology Engineering*, worin semantische Modelle und Sprachen für die fachliche Beschreibung der Systemumgebung und der Services bestimmt und definiert werden. Das resultierende Vorgehensmodell setzt sich aus diesen drei Teilbereichen zusammen und durchläuft die Phasen der Anforderungsanalyse, des Planens und Umsetzens sowie der Evaluation. Die Methode wurde bereits verwendet, um einen Online-Marktplatz für Web Services zu realisieren. Der Fokus liegt hierbei deutlich auf der semantischen Beschreibung von Services und Service-Eigenschaften, die vor allem für die Verwendung innerhalb von

dynamischen Marktplätzen geeignet sind. Die Planung und der Entwurf einer SOA bleiben hierbei unberücksichtigt und das gesamte Konzept erinnert eher an die Umsetzung eines Service-Verzeichnisses wie dem UDDI.

Ein Ansatz, der lediglich durch eine einzelne Veröffentlichung gestützt wird, ist *Service Oriented Architecture Framework* (SOAF) (vgl. [EAK06]). Darin werden fünf Phasen durchlaufen, die sich beginnend bei der Analyse eines Systems mit der Identifikation, Beschreibung und Umsetzung von Services befassen und abschließend einen Plan des Systems erstellen. Das Vorgehensmodell bedient sich dabei eines Top-down- sowie eines Bottom-up-Ansatzes, um die Geschäftsprozesse des Unternehmens ebenso wie ein bestehendes IT-System zu berücksichtigen. Dieser Ansatz widmet sich vornehmlich der Analyse und Planung einer SOA, wobei die tatsächliche Umsetzung etwas zu kurz kommt.

Von Papazoglou und Van Heuvel stammt die Methode *Service-oriented Design and Development Methodology*, die auf dem Softwareentwicklungsmodell RUP und Komponenten-basierter Entwicklung aufbaut (vgl. [PH06]). Hierbei werden alle relevanten Phasen einer SOA-Entwicklung im sogenannten *Web Service development life cycle* beschrieben, der sich durch die folgenden fünf Schritte auszeichnet: Analyse und Design, Konstruktion und Testen, Provisionierung, Entwicklung, Ausführung und Überwachung. Jeder Teil wird ausführlich beschrieben und dessen Aufgaben dargelegt. Dabei ist dieser Ansatz ein vollständiges Modell, welches eine SOA von der Analyse bis hin zum Betrieb beschreibt. Das Modell ist aber einerseits stark technisch abhängig von BPEL und der ausschließlichen Verwendung von Web Services und schenkt andererseits externen Services, Web APIs oder XaaS keine Beachtung, so dass es für einen Einsatz im WOA-Kontext zu spezifisch ist. Dennoch fließen Teilaspekte der Methode, beispielsweise die Vorgehensweise zur Spezifikation von Services, im nächsten Kapitel ein.

Ein weiteres Modell eines Standardgremiums stammt von OASIS und nennt sich *Web Service Implementation Methodology* (WSIM). Darin werden alle Phasen der SOA-Realisierung von der Analyse bis hin zur Umsetzung beschrieben (vgl. [OAS05]). Jedoch ist auch dieses Modell starr auf die Verwendung von Web Services ausgerichtet und behandelt keine Identifikation von Services, keine Geschäftsprozesse noch bestehende IT-Systeme. Damit ist auch diese Methode für eine WOA-Realisierung ungeeignet.

In der Praxis verwendete Modelle

IBM als Softwarehersteller und Beratungsunternehmen bietet gleich mehrere Vorgehensmodelle, die sich mit der Umsetzung einer SOA beschäftigen. Dabei kann *Service-oriented Analysis and Design* (SOAD) als grundlegendes Werkzeug für die einzelnen Ansätze angesehen werden (vgl. [ZKG04]). In SOAD werden Techniken empfohlen, die sich für die Umsetzung einer SOA eignen, wie beispielsweise Service-Kategorisierung, Richtlinien und Aspekte, Meet-in-the-Middle-Prozesse, semantisches Verbinden sowie die Suche nach Services. Auf diesen Grundlagen bauen weitere Ansätze von IBM auf:

- *Service-oriented Modeling and Architecture (SOMA)* in [AGA+08] unterteilt die Modellierung einer SOA in drei Phasen: Die Erste behandelt das *Component Business Modeling* (CBM), um alle Prozesse innerhalb eines Unternehmens zu beschreiben. Die Zweite verbindet die vorhandenen Geschäftsprozesse mit den IT-Services des Unternehmens. Innerhalb der dritten Phase werden die einzelnen Komponenten der SOA implementiert bzw. miteinander verbunden. Darüber hinaus wird der Betrieb einer SOA aber nicht behandelt.

- *Business-driven development* in [Mit05] beschreibt ein ähnliches Vorgehen, welches zusätzlich noch Stakeholder innerhalb der Planung und Umsetzung beachtet.

- *Service-Oriented Analysis and Design Activities* in [BBF05] beschreibt ausführlich ein Vorgehen anhand der vier Phasen: Identifikation, Kategorisierung, Spezifikation und Realisierung von Services.

Alle diese Ansätze lassen sich durch den aktuellen *SOA Foundation Life Cycle* von IBM zusammenfassen (vgl. [WAB+07]). Darin wird die aktuelle Sichtweise von IBM auf die SOA-Entwicklung dargestellt, die zuvor von verschiedenen Mitarbeitern teilweise differierend ausgedrückt wurde. Die Grundlage bildet der RUP und es werden grob die Phasen *Assemble*, *Deploy*, *Manage* und *Model* definiert. Alle diese Ansätze besitzen das Defizit, dass der Fokus grundsätzlich auf Web Services und die damit verbundenen Techniken gelegt wird. Außerdem weist die Ausgestaltung der Phasen zumeist eine starke Softwareabhängigkeit zu IBM-Produkten auf. Die IBM-Vorgehensmodelle sind zwar ausführlich beschrieben, jedoch aufgrund der Technologieausrichtung und Produktbindung weniger geeignet, um eine WOA zu realisieren. Neben den bereits

genannten Ausprägungen existieren diverse weitere Ansätze oder Verfeinerungen der IBM-Methode, die in folgenden Quellen ausgeführt werden: *S3* in [AZE+07b], *SOMA-ME* in [ZYCJ06, ZZC+08] sowie eine Betrachtung von REST-Services in diesem Zusammenhang [ZLC08].

Ein aus zehn Schritten bestehender Ansatz wurde von dem Unternehmen *IDS Scheer AG* entwickelt und geht dabei von Geschäftsprozessmodellen aus, die nach und nach durch entsprechende Software-Unterstützung ein funktionierendes IT-System aufbauen (vgl. [Klü07, SI07b]). Die zehn Aufgaben verteilen sich auf die folgenden Aktivitäten: Geschäftsprozessmodellierung, Aufbau der Servicearchitektur, Vervollständigung der Prozessmodelle, automatische Transformation der Prozesse, Vervollständigung der technischen Prozesse, Export in eine virtuelle Plattform, Import und Anpassung an die Ausführungsplattform, Implementierung neuer Services, Anpassung vorhandener Services und abschließend Test, Installation und Nutzung. Die Durchführung der Phasen wird auf die Stakeholder *Business Analyst*, *Process Enigineer* sowie *Software Engineer* aufgeteilt. Im Gegensatz zu der Methode einer WOA-Erstellung sind hierbei keine Betriebs- und Überwachungsschritte beschrieben. Darüber hinaus ist die Methode vollständig auf die ARIS-Produktpalette der IDS Scheer ausgelegt und fokussiert dabei auf die ausschließliche Verwendung von BPEL.

Mathas beschreibt in [Mat07] eine SOA-Methode, die durch eine Erweiterung des klassischen Softwarelebenszyklus entstand. Dieses vollständige Modell beschreibt alle relevanten Phasen von der SOA-Grobspezifikation über die Prozessanalyse und den Systementwurf bis hin zur Systementwicklung und die Begleitung des Operativbetriebs. Auch hier werden die Stakeholder, wie beispielsweise SOA-, Systemarchitekt und Entwickler, den einzelnen Phasen zugeordnet. In diesem Ansatz wird auch über die Möglichkeit unternehmensübergreifender Choreografie nachgedacht, so dass er dem Vorgehensmodell einer WOA-Umsetzung relativ ähnlich ist. Gegen dieses Vorgehen spricht allerdings, dass in diesem Modell stets mit der Betrachtung von Anwendungssystemen begonnen wird. Die in dieser Arbeit bevorzugte Methode für die Erstellung einer WOA ist jedoch grundsätzlich der Einstieg in die Analyse eines Systems über die Geschäftsprozess-Betrachtung. Nur für den Fall eines bestehenden IT-Systems wird zusätzlich der Bottom-up-Weg gewählt und dieses dabei analysiert.

Erl, der als Buchautor im SOA-Bereich sehr anerkannt ist, erläutert in [Erl05] eine Methode für die Realisierung einer SOA, die er *SOA Delivery Strate-*

gies nennt. Darin beschreibt er eine Abfolge von Phasen, die die folgenden sechs Aktivitäten beinhaltet: Service-orientierte Analyse, Service-orientierter Entwurf, Service-Entwicklung, Service-Testen, Service-Veröffentlichung und schließlich Service-Verwaltung. Bei diesem Modell fehlt zum einen eine klare Stakeholder-Zuordnung zu den Phasen und zum anderen betrachtet Erl nicht den Betrieb einer SOA. Obwohl einzelne Phasen, vor allem die Analyse und der Entwurf, sehr ausführlich beschrieben werden, legen diese den Fokus stark auf BPEL, WSDL und SOAP, so dass eine Verwendung dieses Ansatzes für die Realisierung einer WOA nicht möglich ist.

4.4 Zusammenfassung

Durch eine Betrachtung der vorgestellten Ansätze wird klar, dass keines der aufgeführten Vorgehensmodelle im Ganzen für die Planung und Umsetzung einer WOA geeignet ist. Nur wenige Modelle beinhalten eine komplette Betrachtung von der Analyse und Planung, über die Umsetzung bis hin zum Betrieb. Außerdem weist kein Modell eine angemessene Betrachtung existierender XaaS-Angebote auf. Dies ist nicht verwunderlich, da der Kern einer SOA eher innerhalb eines Unternehmens gesehen wird. Dabei werden Schnittstellen zur Außenwelt meist befürwortet, aber nicht im gleichen Maße als notwendig erachtet, wie in einer WOA. In den Ansätzen befinden sich auch nur selten explizite Betrachtungen über organisationsübergreifende Geschäftsszenarien, die dem Grundkonzept einer WOA nahe kommen. Dieses Konzept ist dafür verantwortlich, dass die Ausprägung einer WOA und die Topologieplanung einen sehr hohen Stellenwert im Vorgehensmodell einnehmen.

Ein Entwicklungsmodell für eine WOA, die der hier vorgestellten Definition entspricht, sucht man bisher vergebens. Der Ansatz von *SOA4ALL* befasst sich zwar schon mit der Verlagerung der Funktionalitäten ins Internet und der Nutzung von REST, ist aber nicht für ein fachlich begründetes IT-System eines Unternehmens konstruiert, sondern eher für die Informationsbereitstellung gedacht.

Auch die Betriebsphase, im SOA-Bereich meist als Governance bezeichnet, wird in vielen Modellen eher am Rande erwähnt, ist aber für eine WOA essenziell. Zum einen lassen sich damit Kosten überwachen und ermitteln, zum anderen ist diese Kontrolle der Services aufgrund der hohen Volatilität der XaaS-Anbieterstruktur dringend notwendig. Ein großer Unterschied ist auch

darin zu sehen, dass bei einer SOA in den meisten Fällen, teilweise stillschweigend, davon ausgegangen wird, dass die Services innerhalb des Unternehmens schon bestehen oder dort programmiert werden. In einer WOA sollen vor allem bestehende XaaS-Dienste Verwendung finden, und, wenn möglich, nur in Ausnahmefällen eigener Programmcode geschrieben werden. Bei realistischer Betrachtung lässt sich ein gewisses Maß an Programmierung natürlich weder bei einer SOA noch einer WOA vermeiden.

Ein weiterer oft zu beobachtender Aspekt der SOA-Vorgehensmethoden ist der ausschließliche Einsatz von Web Services und damit einhergehend das Nutzen von SOAP, WSDL und teilweise UDDI. Diese Eigenheit erschwert eine Durchführung solcher Methoden für die WOA-Erstellung. Denn gerade die Verwendung von Web Services und die in diesem Umfeld vorhandenen Standards sollen in einer WOA möglichst vermieden werden, um einfachen Standards, wie HTTP und REST, den Vorzug zu geben. Für die Einbindung von Alt- oder Dritt-Systemen von Lieferanten und Kunden ist der Einsatz natürlich nach wie vor möglich. Aber die Verwendung von REST oder Web APIs ist in keinem der bestehenden Ansätze ähnlich prominent vertreten wie Web Services. Obwohl die technologische Basis einer SOA und einer WOA – ein verteiltes System – identisch ist, wird deutlich, dass sich die Gewichtung einzelner Phasen der Software-Erstellung und deren Ausgestaltung deutlich unterscheiden. Daher wird im nächsten Kapitel eine vollständige Methode zur Konzeption, Planung und Umsetzung einer WOA vorgestellt.

5 Eine WOA-Entwicklungsmethode

In diesem Kapitel der Arbeit wird eine Methode für die Konzeption und Umsetzung einer WOA vorgestellt. Als Grundlage für das Verständnis der im weiteren Verlauf der Arbeit vorkommenden Modelle werden zunächst gängige Modellierungssprachen für Prozesse und Datenmodelle beschrieben. Daraufhin werden die grundlegenden Phasen des WOA-Vorgehensmodells zunächst kurz skizziert. Diese Phasen orientieren sich an der typischen Dreiteilung einer Softwareentwicklung und fallen in die Bereiche: *Analyse und Planung*, *Realisierung* sowie *Betrieb*. Nach einer kurzen Betrachtung der Unternehmensstrategie werden dann die zuvor genannten Phasen des Vorgehensmodells ausführlich im Detail erläutert und dabei deren zentrale Aspekte und Arbeitsschritte beschrieben. Des Weiteren werden die nach jeder Phase vorliegenden Dokumente skizziert, die oftmals die Basis für nachfolgende Schritte bilden. Da bei der Gestaltung und Umsetzung eines verteilten Systems oft auch die bestehende Systemumgebung und vorhandene Altsysteme berücksichtigt werden müssen, folgt ein Abschnitt zur Ist-Systemanalyse. Anschließend wird erläutert, inwieweit die Phasen des Vorgehensmodells bei einer Migrationsstrategie auf eine WOA abgewandelt werden müssen, um bestehende Systeme einzubinden. Eine konkrete Möglichkeit zur WOA-Umsetzung wird daraufhin am Beispiel des agilen Vorgehens FDD erläutert. Zum Abschluss des Kapitels wird an einem weiteren Beispiel gezeigt, inwieweit sich das zum Kernkonzept einer WOA gehörende Element des WAC mittels XaaS-Angeboten umsetzen lässt.

Modellierungssprachen

Das Abbilden von Geschäftsprozessen lässt sich mit diversen Modellierungssprachen umsetzen. Standards wie BPMN und UML sowie Petri Netze und ereignisgesteuerte Prozessketten (EPK) sind weit verbreitet. Darüber hinaus sind für die Darstellung von Datenmodellen auch *Entity Relationship Model* (ERM)-Diagramme geeignet. Im Folgenden wird die jeweilige Syntax von UML, BPMN und ERM beschrieben, da diese für die weiteren Erläuterungen benötigt werden. Die Verwendung von Standard-Modellierungssprachen allein ist

noch kein Garant für das Erstellen von syntaktisch und semantisch korrekten Modellen. Daher soll an dieser Stelle auf die Grundsätze ordnungsmäßiger Modellierung verwiesen werden, die Richtlinien für die Erstellung von Modellen definieren. Darin werden die Grundsätze der Richtigkeit, der Relevanz, der Wirtschaftlichkeit, der Klarheit, der Vergleichbarkeit sowie des systematischen Aufbaus thematisiert, die es für die Erstellung qualitativ hochwertiger Modelle zu beachten gilt (vgl. [BS04]).

Business Process Modelling Notation Für die Darstellung von Geschäftsprozessen in für Menschen lesbarer Form eignet sich die BPMN besonders, da diese eine Fülle verständlicher Symbole enthält, die das Darstellen von Abläufen auch über verschiedene Ebenen bzw. Organisationseinheiten hinweg ermöglicht. Dabei können vor allem die zeitliche Abfolge von Ausführungsschritten und die Choreografie von Komponenten dargestellt werden. Die Modellierungselemente der BPMN lassen sich in vier Klassen unterteilen, deren gängige Elemente durch einen einfachen Beispielprozess in Abbildung 5.1 dargestellt werden (vgl. [Obj10, MW08, Wes07, HKKR05, RQZ07]).

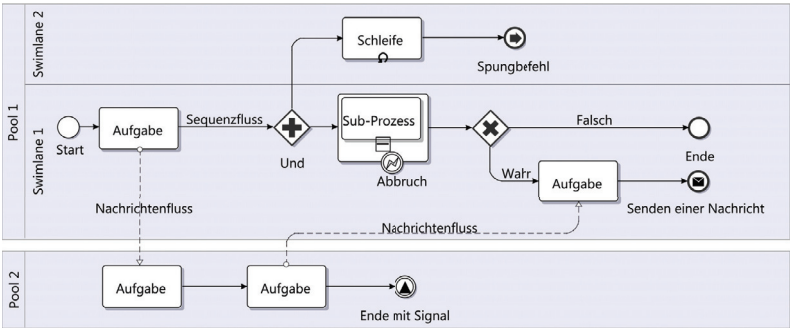


Abbildung 5.1: Beispielprozess mit der BPMN.

- *Pools und Swimlanes*: Ein Pool stellt einen Akteur des Geschäftsprozesses dar. Dieser kann entweder einen Benutzer oder eine Benutzerrolle ebenso wie ein IT-System oder einen XaaS-Anbieter repräsentieren. Ein

Pool kann von Swimlanes weiter unterteilt werden, die beispielsweise Organisationseinheiten oder Subsysteme darstellen. Alle weiteren Elemente eines Geschäftsprozesses liegen innerhalb der Pools und Swimlanes.

- *Flow Objects*: Dies sind die Knoten eines Prozesses, die sich aus Aktivitäten (Tasks oder Sub-Prozesse), Gateways (Entscheidungspunkte wie beispielsweise logisches AND, OR und XOR) sowie Ereignissen (Start-, End- oder Zwischenereignisse) zusammensetzen.

- *Connecting Objects*: Die Kanten, die die verschiedenen Elemente verbinden, sind entweder dem Kontrollfluss (*sequence flow*) oder dem Nachrichtenfluss (*message flow*) zuzuordnen. Der Kontrollfluss verbindet die Flow Objects innerhalb eines Pools miteinander, um die Ausführungsreihenfolge zu definieren. Der Nachrichtenfluss wird für die Poolübergreifende Verbindung verwendet und zeigt damit die Kommunikation zweier Objekte an. Zusätzlich lassen sich noch Assoziationen ausdrücken, die beispielsweise Artefakte zu einem Flow Object zuordnen können.

- *Artifacts*: Artefakte können Kommentare, Datenobjekte oder Gruppierungen sein. Damit lassen sich erläuternde Details oder auch partizipierende Dokumente oder Objekte darstellen. Darüber hinaus erlaubt das Metadatenmodell der BPMN auch die Definition eigener Artefakte.

Unified Modelling Language Eine erste Version der UML wurde von der *Object Management Group* in den 90er Jahren eingeführt. Über mehrere Entwicklungsstadien hinweg wurde 2005 die Version 2.0 veröffentlicht (vgl. [Obj05]), die eine umfassende Anzahl an Modellarten für die Spezifikation und Dokumentation von IT-Systemen enthält. Seitdem umfasst UML jeweils sieben Arten von Struktur- und Verhaltensdiagrammen. Die Möglichkeiten der UML in Bezug auf detaillierte Beschreibungen für das Entwerfen und Dokumentieren von objektorientierten IT-Systemen ist beachtlich. Es gibt aber deutliche Überschneidungen mit der BPMN, die für Programmabläufe auf fachlicher Ebene sehr geeignet ist und hier daher auch Verwendung findet. Die UML eignet sich eher für eine techniknahe und detaillierte Beschreibung von IT-Komponenten und wird später für die Beschreibung von Service-Schnittstellen

verwendet. An dieser Stelle wird daher aus der UML-Familie nur das Komponentendiagramm vorgestellt.

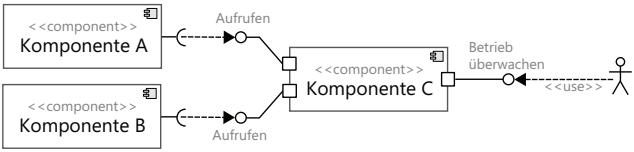


Abbildung 5.2: UML-Komponentendiagramm.

Mit Komponentendiagrammen lassen sich komponentenbasierte Softwaresysteme modellieren. Dabei werden vor allem deren Schnittstellen hervorgehoben, um die einzelnen Verknüpfungen darzustellen. Im Kontext einer WOA lassen sich hiermit verschiedene Komponenten des verteilten Systems, wie XaaS-Anbieter oder interne Systeme, sowie deren Schnittstellen zueinander modellieren. Ein Port kann eine Schnittstelle (*Interface*) anbieten, dargestellt durch einen Kreis, oder aufrufen bzw. benötigen, dargestellt durch einen Halbkreis. Sollten Komponenten weitere Unterelemente besitzen, so können diese durch weitere UML-Sprachkonstrukte eingezeichnet werden. Abbildung 5.2 stellt die Komponente C dar, die durch einen menschlichen Akteur überwacht und von zwei weiteren Komponenten über die zur Verfügung gestellten Schnittstellen aufgerufen wird.

Entity-Relationship-Modelle Eine sehr gebräuchliche Art der Modellierung von Datenmodellen stellen die in [Che76] vorgestellten Entity-Relationship-Modelle dar (vgl. [Vos08]). Darin werden Objekte (Entitäten) und Daten der Objekte (Attribute) sowie deren Beziehungen zueinander (Relationships) dargestellt. Durch die Angabe von Kardinalitäten wird ausgedrückt, wie oft eine Entität eine Beziehung mit einer anderen Entität eingehen kann. Diese werden durch eine Zahlenkombination an der Verbindungskante ausgedrückt. Dabei werden typischerweise die Werte in der Min-Max-Notation angegeben, wobei die erste Zahl das minimale und die zweite Zahl das maximale Vorkommen einer Entität in einer Relation ausdrückt. Darüber hinaus lassen sich, als Erweiterung zur ursprünglichen Variante, Relationship-Typen auch als Entitäten uminterpretieren und können somit weitere Beziehungen eingehen. Eine Entität

wird in der ursprünglichen Notation von Chen durch ein Rechteck, eine Beziehung als Verbindungskante und Relationship-Typen als Rauten dargestellt. Die Kardinalitäten werden nahe des Ausgangs der Kante an der Entität platziert. Attribute einer Entität werden als Ovale dargestellt, die mit der Entität verbunden sind. Gängige Praxis für das Definieren von eindeutig identifizierenden Eigenschaften einer Entität, beispielsweise für Primärschlüssel relationaler Datenbanken, ist die Unterstreichung des Attributs oder der Attributgruppe. In Abbildung 5.3 ist ein ERM gezeigt, das die Beziehungen zwischen Mitarbeiter, Projekten und Kunden der *BBI GmbH* darstellt. Durch die Kardinalitäten in Bezug auf den Relationship-Typ *Projektteam* sind hierbei jedem Projekt keine bis beliebig viele Mitarbeiter $(0, m)$ sowie mindestens ein Kunde $(1, m)$ zugeordnet. Die Zuordnung von Kunden $(1, 1)$ zu Projektteams $(1, m)$ spezifizieren, dass Projektteams zwar mehreren Kunden, aber Kunden jeweils nur einem Projektteam zugeordnet werden können. Die Primärschlüssel für Mitarbeiter, Projekt und Kunde sind jeweils die eindeutigen Nummern der Entitäten.

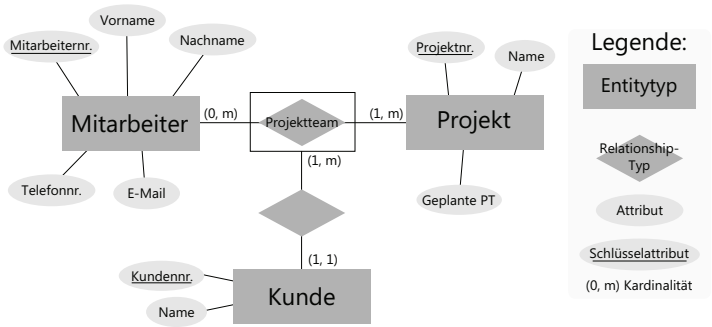


Abbildung 5.3: Projektzuordnung von Mitarbeitern durch ein ERM dargestellt.

Diese ursprüngliche Notation wurde für diverse Modellierungssprachen angepasst. Daher lassen sich ER-Modelle beispielsweise auch durch UML darstellen. Man verwendet dazu ein Komponentendiagramm und nutzt die Angabe des Stereotyps *«entity»* als Bezeichner einer Komponente, um deren Status als persistenter Datenspeicher zu kennzeichnen. Die Beziehungen zwischen den Entitäten werden hierbei durch eine *Relationship*-Verbindung dargestellt, die zusätzliche Annotationen aufweist, um das Datenmodell besser lesbar zu

gestalten. Die Architekturgrafik des IEEE-Standards zu Beginn des Kapitels 3 wurde in einer solchen UML-Notation ausgeführt. Durch die Möglichkeit des expliziten Modellierens von Relationship-Typen, Attributen von Entitäten und der Kennzeichnung von Primärschlüsseln innerhalb des Modells ist die ursprüngliche Notation von Chen aber im Hinblick auf die konkrete Umsetzung als Datenbankmodell sehr geeignet und wird im Folgenden verwendet.

5.1 Generische Phasen einer WOA-Entwicklung

Im Folgenden wird ein WOA-Vorgehensmodell erläutert, welches das Ziel der Planung und Umsetzung einer WOA durch einen Top-down-Ansatz verfolgt. Um eine Systemarchitektur zu gestalten, wird bei diesem Ansatz mit der Betrachtung von Geschäftsprozessen eines Unternehmens begonnen. Diese werden zunächst aus der Vogelperspektive modelliert und dabei wird zunächst völlig von einer ggf. existierenden technischen Umsetzung abstrahiert. Beim Fortschreiten des Vorgehens werden die zu Beginn identifizierten Geschäftsprozesse sukzessive weiter verfeinert. Ohne ein spezifisches Vorgehensmodell vorzuschreiben, werden die grundlegenden Phasen in die drei Bereiche *Analyse und Planung*, *Realisierung* und *Betrieb* unterteilt. Obwohl in dieser Arbeit ein agiles Entwickeln mittels FDD aufgrund einiger Vorteile als Umsetzungsmethode für WOA befürwortet wird (siehe Kapitel 5.7), lässt sich das Konzept durchaus mit unterschiedlichen Softwareentwicklungsmethoden umsetzen. Im Folgenden werden daher die einzelnen Arbeitsschritte in sequenzieller Reihenfolge genauer beschrieben und dabei die notwendigen Vorbedingungen und die in den Schritten erstellten Dokumente auf abstrakter Ebene erläutert. Hierbei sei angemerkt, dass die Arbeitsschritte sowohl für ein Start-up-Unternehmen ohne jegliche vorhandene IT-Infrastruktur als auch für ein Unternehmen mit bereits existierenden Systemen anwendbar sind. Auf Unterschiede in der Durchführung des Vorgehensmodells bei bestehenden IT-Systemen wird in Abschnitt 5.6 explizit eingegangen. Ein Überblick über die konkreten Phasen ist in Abbildung 5.4 dargestellt.

Es ist an dieser Stelle noch anzumerken, dass die ersten drei Phasen der Analyse und Planung eine allgemeine Gültigkeit auch beispielsweise für die Umsetzung einer SOA besitzen. Die Phasen 4-6 hingegen sind WOA-spezifisch und in der hier vorgestellten Form speziell auf die Konzepte einer WOA abgestimmt.

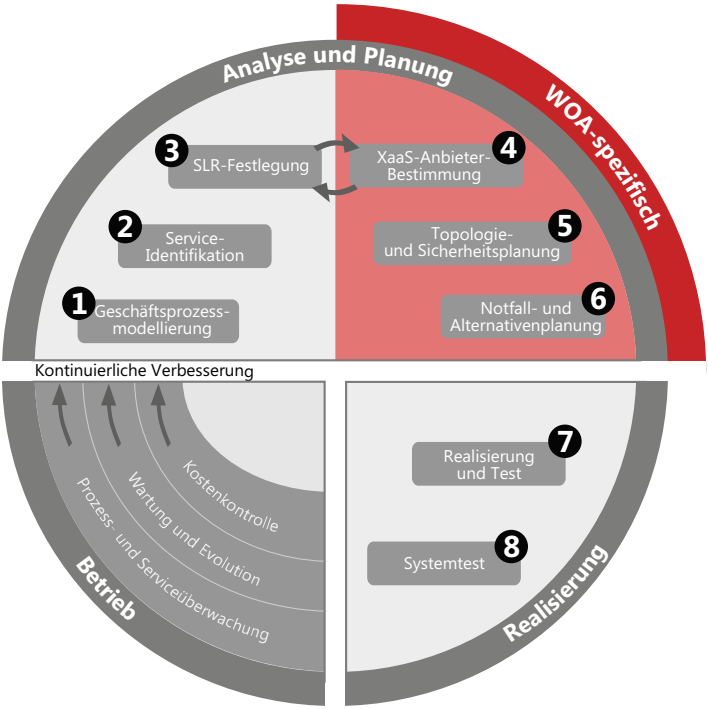


Abbildung 5.4: Die grundlegenden Phasen einer WOA-Umsetzung.

5.1.1 Analyse und Planung

Geschäftsprozessmodellierung In einem Top-down-Ansatz werden zu Beginn der Analyse- und Planungsphase grundsätzlich alle relevanten, das IT-System betreffende, Geschäftsprozesse eines Unternehmens modelliert. Dabei beginnt man mit rudimentären Top-Level-Modellen, die eine fachliche Übersicht ermöglichen und durch Untermodelle beliebig weit verfeinert werden können. Eine vollständige Prozessmodellierung endet, wenn man auf der Ebene von atomaren Aktivitäten angelangt ist. Während der Verfeinerung der Prozesse lassen sich auch bereits rudimentäre Datenobjekte identifizieren. Diese werden für die gleichzeitige Entwicklung von Datenmodellen verwendet. Zusätzlich kann hier nebenläufig ein Organisationsmodell entstehen, welches die einzelnen Abteilungen und Organisationsteile des Unternehmens darstellt. Das Resultat dieses Arbeitsschrittes sind Dokumente mit fachlichen Modellen von Geschäftsprozessen, ein konsolidiertes Datenmodell sowie weitere Dokumente zur Beschreibung von Prozessen. Hierbei sind technische Details noch nicht berücksichtigt, da zunächst eine saubere, fachliche Abbildung der Geschäftsabläufe von Interesse ist.

Service-Identifikation Die Prozessmodelle aus dem vorherigen Arbeitsschritt dienen nun der Identifikation von Services. Hierbei wird jedes Prozessmodell auf automatisierbare Schritte hin untersucht und es wird geklärt, ob man diese mittels eines IT-Systems lösen kann. Dabei werden die vorliegenden Geschäftsprozesse verfeinert. Zusätzlich muss in dieser Phase das Auffinden von redundanten Services stattfinden, so dass identische Aktivitäten über alle Geschäftsprozesse später auch durch denselben Service ausgeführt werden. In dieser Phase entsteht gemäß ITIL ein Business-Service-Katalog, der einen Teil des Service-Level-Managements ausmacht.

Service-Level-Requirements-Festlegung Das Festlegen von *Service Level Requirements* (SLRs) beinhaltet das Niederschreiben von notwendigen Anforderungen an jeden identifizierten IT-Service. Hierbei werden charakteristische, messbare Merkmale von Services aufgenommen, wie beispielsweise Antwortzeit, Verfügbarkeit und Kosten. Darüber hinaus wird in dieser Phase entschieden, ob ein Service innerhalb des Unternehmens verbleiben muss und somit eine Eigenimplementierung notwendig wird, oder ob ein Auslagern möglich ist. Nach Abschluss dieser Phase stehen die SLRs für die zuvor identifizierten

Tabelle 5.1: Vorbedingungen und Dokumentation der allgemeinen Analyse- und Planungsphase.

Vorbedingung	Resultierende Dokumentation
1) Geschäftsprozessmodellierung	
• Keine	• Fachliche Modelle von Geschäftsprozessen • Organisationsmodell • Konsolidiertes Datenmodell • Liste von Prozessverantwortlichen • Geschäftsregeln (*business rules*)
2) Service-Identifikation	
• Modelle von Geschäftsprozessen	• Typisierte Aktivitäten/Services in Geschäftsprozessen • Business Service Katalog (gemäß ITIL)
3) SLR-Festlegung	
• Identifizierte Services	• Service Level Requirements

Services fest und werden durch funktionale und nicht-funktionale Merkmale dokumentiert.

XaaS-Anbieter-Bestimmung Der Business-Service-Katalog und die SLRs dienen nun als Grundlage für die Suche nach potenziellen XaaS-Anbietern. Alle Services, die auch als auslagerbar gekennzeichnet sind, sollen im Idealfall durch ein passendes Angebot ausgeführt werden, sofern am Markt vorhanden. Bei der Suche nach Anbietern können durchaus auch Alternativen für die Service-Umsetzung auftreten, die dann miteinander verglichen werden müssen. Ggf. können diese auch für den Aufbau eines redundanten Systems genutzt werden. Ist ein fachlich passendes Angebot gefunden, müssen die SLRs mit den angebotenen Service-Levels verglichen werden. Dabei gibt es zwei Möglichkeiten: Entweder es lassen sich individuelle SLAs für einen angebote-

nen Service mit dem Anbieter aushandeln und vertraglich absichern oder es bestehen verschiedene, fest vorgegebene Service-Levels, aus denen ein Kunde auswählen kann. Welche Option sich realisieren lässt, hängt mitunter auch vom Unternehmen ab. Die Möglichkeit des Aushandelns individueller SLAs ist bei einem großen Unternehmen mit hoher Marktmacht wahrscheinlicher als bei einem Start-up-Unternehmen.

In dieser Phase springt man ggf. auch zur vorherigen Phase der Service-Level-Festlegung zurück, um Änderungen an den SLRs vorzunehmen. Für die Stammdatenzugriffe, die durch spezifizierte Datenobjekte in der Geschäftsprozessmodellierung identifiziert wurden, werden dabei passende IaaS-Angebote untersucht. Nach Abschluss dieses Arbeitsschrittes liegt der technische Service-Katalog nach dem ITIL-Modell vor, aus dem hervorgeht, welche Services und damit welche Aktivitäten eines Geschäftsprozesses durch welchen konkreten technischen Service oder XaaS-Anbieter abgedeckt werden sollen. Dabei können auch mehrere Alternativen für einen Service existieren, oder Stellen benannt werden, für die kein passender Service gefunden wurde. Dies impliziert, dass die fachliche Logik einer solchen Stelle in späteren Arbeitsschritten selbst implementiert werden muss.

Topologie- und Sicherheitsplanung Da die verschiedenen Komponenten und Services einer WOA auf unterschiedlichen Servern im Intra- und Internet liegen können, soll in diesem Schritt auf Grundlage der Service-annotierten Geschäftsprozessmodelle zunächst eine Netzwerktopologie entwickelt werden. Dabei werden Anzahl und Anordnung von Controllern innerhalb und außerhalb des Unternehmens bestimmt. Außerdem wird festgelegt, welche XaaS-Angebote mit welchem Controller koordiniert oder überwacht werden und wo ggf. Workflows implementiert und ausgeführt werden. Ist diese Planung erfolgt, werden sicherheitsrelevante Aspekte geplant: Diese betreffen beispielsweise verschlüsselte Verbindungen zwischen spezifischen Punkten der Architektur, eine *Public Key Infrastructure* (PKI) für die Authentifizierung, die Festlegung von Datenverantwortlichen (*data ownership*), sowie Zugriffsrechte für Daten und Workflows. Mit Abschluss dieses Planungsschrittes existiert eine technische Dokumentation der Netzwerktopologie, die mit sicherheitsrelevanten Informationen erweitert ist. Außerdem wird die Rechtestruktur für den Zugriff auf Daten und Prozesse (Lese-, Schreib- und Ausführungsrechte) von Personen und Systemen (Rollen und Rechte) festgehalten.

Notfall- und Alternativenplanung In diesem letzten Schritt der WOA-Planung wird die Problematik von ausfallenden Services betrachtet. Da eine WOA auf der Auslagerung möglichst vieler Dienste basiert, muss hier eine Notfallplanung vorgenommen werden, die für das Auftreten von Störungen innerhalb des Systems Lösungsstrategien bereithält. Diese kann zum einen in der bereits zu Beginn der Planung redundanten Auslegung der Systemarchitektur und zum anderen im Vorhalten von Alternativen bestehen, die für ausgefallene Systemteile bei Bedarf einspringen können. Auch der Umgang mit auftretenden (Umwelt-) Katastrophen, dem sogenannten *Disaster Recovery Planning* oder *Disaster Management*, kann je nach architektonischer Ausprägung der WOA ein zu beachtender Aspekt sein. Dazu gehören beispielsweise Störungen des Systems durch Brände, Wasserrohrbrüche, Erdbeben und andere Umwelteinflüsse. Bei einer archetypischen WOA entfallen diese Einflüsse jedoch weitgehend, da alle Komponenten der WOA ausgelagert sind. Da entsprechende Maßnahmen für jeglichen Standort von Hardware anfallen, und nicht spezifisch für eine WOA sind, wird das *Disaster Management* im Kontext dieser Arbeit nicht weiter vertieft.[48] Die resultierende Dokumentation dieser Phase beinhaltet abschließend einen Leitfaden für die zu ergreifenden Maßnahmen bei Störungen oder Teilausfällen von Services.

5.1.2 Realisierung

Realisierung und Test Nach Durchführung der ersten sechs Phasen des Vorgehensmodells existiert eine durchgehende Dokumentation der Systemarchitektur, die nun durch IT-Spezialisten umgesetzt werden kann. Hierbei kann zunächst mit den vorhandenen Service-annotierten Prozessmodellen begonnen werden. Diese müssen zuerst in ausführbare Workflow-Modelle umgesetzt werden. Je nach Ausgangsbasis ist dies mit einigem Aufwand verbunden. Manche Standards, wie beispielsweise BPMN, ermöglichen hierbei durch geeignete Werkzeuge eine direkte Umwandlung in ausführbare BPEL-Prozesse. Beim Einbinden der jeweiligen Services muss dennoch häufig per Hand nachgearbeitet werden. Dies tritt vor allem beim Umsetzen von Service-Berechtigungen, Sicherheitsstrukturen und Kontrollpunkten in WACs etc. auf. Darüber hinaus kann je nach Möglichkeit des eingesetzten WAC auch manuelle Programmierung von Workflows durchgeführt werden.

[48] Weiterführende Lektüre zu diesem Thema ist beispielsweise in [Hia00] oder [Toi02] zu finden.

Tabelle 5.2: Vorbedingungen und Dokumentation der WOA-spezifischen Analyse- und Planungsphase.

Vorbedingung	Resultierende Dokumentation
4) XaaS-Anbieter-Bestimmung	
• Business-Service-Katalog • Geschäftsregeln • SLRs	• Technischer Service Katalog (gemäß ITIL) • Vereinbarte Service Level Agreements
5) Topologie- und Sicherheitsplanung	
• Service-annotierte Prozessmodelle	• technische Dokumentation der Netzwerktopologie • Rechtestruktur für Datenzugriffe (Rollen, Rechte)
6) Notfall- und Alternativenplanung	
• Service-Katalog	• Leitfaden für Notfall

Die Testphase wird hier nicht gesondert aufgeführt, sondern sollte implizit in der Entwicklungsphase enthalten sein. Bei der Realisierung eines Geschäftsprozesses werden alle eingebundenen Services sowie der ablaufende Workflow direkt durch den Entwickler getestet. Eine explizite Teststrategie wird jedoch im WOA-Vorgehensmodell nicht vorgeschrieben. Welche Testart sich hierbei einsetzen lässt, hängt auch maßgeblich vom tatsächlich eingesetzten Softwareentwicklungsmodell ab. Wählt man hierfür beispielsweise das V-Modell XT, ist eine Teststrategie bereits vorgegeben. Der Umfang und die Ausgestaltung der Dokumentation sind ebenfalls von dieser Wahl abhängig. Aber selbst bei Durchführung der Realisierung mittels agiler Methoden sollte eine Dokumentation der Umsetzung zumindest so aussagekräftig sein, dass sich die Topologie, die tatsächlich eingesetzten Services und die Zuständigkeiten klar nachvollziehen lassen. Nach der Realisierungsphase ist die WOA technisch fertiggestellt und dokumentiert.

Systemtest Unabhängig vom ausgeführten Vorgehensmodell soll zum Abschluss der Realisierungsphase in jedem Fall ein Systemtest erfolgen. Dieser

Tabelle 5.3: Vorbedingungen und Dokumentation der Realisierungsphase.

Vorbedingung	Resultierende Dokumentation
7) Realisierung und Test	
• alle vorhergehenden Dokumente	• technische Dokumentation
8) Systemtest	
• technische Dokumentation • Topologieplan (aus Schritt 5)	• Testprotokoll

gewährleistet die Funktionsfähigkeit der gesamten WOA im tatsächlichen Zusammenspiel aller Komponenten. Hierfür sollten verschiedene, sinnvolle Testszenarien untersucht werden. Vorstellbar ist hierfür das Testen mittels einer Simulation von Lastspitzen, künstlichem Ausfall von Teilen des Systems sowie Engpässen der Bandbreite. Nach Abschluss der Testphase existieren ausführliche Testprotokolle, die die Funktionsfähigkeit des Systems attestieren und im Zweifelsfall bei Mängeln auch zu Nachbesserungen, und damit einem Rücksprung in die zuvor ausgeführten Schritte, führen können.

5.1.3 Betrieb

Prozess- und Serviceüberwachung Die während der Analyse- und Planungsphase modellierten Prozesse und identifizierten Services wurden in der Realisierungsphase technisch umgesetzt und müssen während des Betriebs einer WOA ständig überwacht und kontrolliert werden. Die dazu notwendige Infrastruktur sollte, je nach Ausprägung der Topologie, auf verschiedene WACs verteilt sein. Unter Umständen ist es sinnvoll, einen zentralen Kontrollpunkt einzurichten, in dem alle Informationen über die aktuellen Prozessinstanzen und Services zusammenlaufen. Die Werkzeuge zur Überwachung der korrekten Funktionalität von Prozessen und Services sind durch die im Kapitel 3.2.4 geforderten Funktionalitäten eines WAC implizit vorhanden, müssen jedoch auch verwendet werden. Die Kontrolle einiger technischer Aspekte der SLAs, wie die Verfügbarkeit oder die Antwortzeiten, lassen sich hierbei für viele Services vollautomatisch überwachen und durch diverse Kommunikationskanäle an die entsprechenden Stellen melden. Damit können beim Auftreten

eines Fehlers die zuständigen Mitarbeiter durch E-Mails oder SMS direkt benachrichtigt werden. Auch Web-Oberflächen für den schnellen Überblick über alle wichtigen Daten und Kennzahlen, sogenannte Dashboards, lassen sich angepasst an verschiedene Nutzerrollen einrichten, beispielsweise für Manager, Sachbearbeiter oder technisches Personal.

Kostenkontrolle Die zentralen Argumente, welche als großer Vorteil der WOA angesehen werden, sind einerseits das Senken der Kosten für IT-Systeme, womit sich Kapitel 6 beschäftigt, und andererseits die Möglichkeit der ständigen Kostenkontrolle durch die meist klaren Kostenstrukturen von XaaS-Anbietern. In einer WOA sollte dafür die Nutzung der einzelnen Services genau protokolliert werden, so dass sich aufschlüsseln lässt, *wer welchen* Service *wann* und für *wie lange* verwendet hat. Dadurch lassen sich neben den angefallenen Gesamtkosten für XaaS-Angebote auch interne Abrechnungen auf Abteilungsebene ausweisen. Beliebige Kennzahlen der IT-Systeme sind ebenfalls in ein Dashboard integrierbar. Innerhalb dieser Phase entsteht ein ständig wachsendes Protokoll, welches für weitere (Kosten-)Auswertungen herangezogen werden kann.

Wartung und Evolution Die Wartung eines laufenden Systems ist bei verteilten Systemen mit vielen Schnittstellen und heterogenen Komponenten, die durchaus Änderungen unterliegen, eine besondere Herausforderung. Insbesondere in einer WOA, in der unterschiedliche XaaS-Angebote zu einem System zusammen kombiniert wurden, können sich auch kleine Änderungen an Service-Schnittstellen oder Betriebsstörungen sehr schnell negativ auswirken. Es ist daher wichtig, die Weiterentwicklung von Services und deren Versionierung, sowie die potenziellen Alternativangebote ständig im Blick zu behalten. Ein seriöser Anbieter von IT-Services sorgt zwar im Normalfall für Abwärtskompatibilität seiner Services, aber bei größeren Versionssprüngen kann es notwendig werden, die WOA an manchen Stellen anzupassen und weiterzuentwickeln. Hierbei ist auch das rechtzeitige Kommunizieren von Änderungen an Services oder Angeboten sowie Wartungsterminen der XaaS-Angebote durch den Anbieter ein zu berücksichtigender Faktor. Im Gegensatz zu im Unternehmen installierter Standardsoftware, bei der ein Unternehmen normalerweise die Wahl hat, wann und ob man ein Update der Software vornimmt, kann es passieren, dass ein XaaS-Anbieter eine neue Version bereitstellt, ohne

Tabelle 5.4: Vorbedingungen und Dokumentation der Betriebsphase.

Vorbedingung	Resultierende Dokumentation
Prozess- und Serviceüberwachung	
• SLAs • Service-Katalog	• Nutzungsprotokolle
Kostenkontrolle	
• SLAs • Service-Katalog • Kostenmodelle der XaaS-Angebote	• individuelle (innerbetriebliche) Rechnungen
Wartung und Evolution	
• technische Dokumentation	• Fortschreibung der technischen Dokumentation

weiteren Support für alte Versionen zu gewährleisten. Durch die permanente Weiterentwicklung im Kleinen (*perpetual beta*) sind XaaS-Angeboten einem ständigen Verbesserungsprozess unterworfen, so dass Anbieter und Nutzer Sorge tragen müssen, dass sich dies nicht auf ein bestehendes funktionierendes System auswirkt.

5.2 Unternehmensstrategie

Das IT-System eines Unternehmens muss dem Erfüllen des Geschäftszwecks dienlich sein und sollte das Kerngeschäft unterstützen. Um Entscheidungen im Hinblick auf Auswahl, Form und Ausgestaltung von IT-Systemen zu treffen, stehen der Unternehmensführung diverse Analysemethoden zur Verfügung. Hier lassen sich beispielsweise Kontext-, SWOT[49]- und Strategieanalysen für das gesamte Unternehmen oder Teile davon erstellen, um den Entscheidungsraum zu definieren (vgl. [SVOK11]). All diese Methoden dienen der Unternehmensführung in diesem Fall zur Entscheidungsfindung für oder wider die Einführung eines speziellen IT-Systems. Da dieser Prozess stark von der Struktur, dem Betätigungsfeld und auch nicht zuletzt von den Neigungen und Kennt-

[49]Strength, Weakness, Opportunities and Threats: eine Stärken- und Schwächenanalyse.

nissen der Mitarbeiter abhängt, wird auf diese Form von Analysewerkzeugen im Kontext dieser Arbeit nicht näher eingegangen; zu viele Annahmen müssten dafür an dieser Stelle getroffen werden. Dennoch wird im Schlusskapitel eine Handlungsempfehlung formuliert, die eine verkürzte SWOT-Analyse präsentiert. Eine wichtige Annahme soll aber im Folgenden in Bezug auf die Outsourcing-Strategie eines Unternehmens getroffen werden, da sich diese stark auf Teile des Vorgehensmodells auswirkt. Die Prämisse ist hierbei, dass ein Unternehmen generell bereit ist, Teile des IT-Systems, insbesondere Services, auszulagern, und damit einer WOA-Umsetzung positiv gegenübersteht.

Outsourcing-Strategie eines Unternehmens

Bevor auch nur ein einziger XaaS-Anbieter gesucht wird, muss ein Unternehmen eine Entscheidung zum Grad des Outsourcings von Funktionalitäten treffen. Darüber hinaus muss entschieden werden, ob Daten in der Cloud gespeichert werden können und ob Virtualisierung von Systemteilen angestrebt wird. Die Frage ist hierbei, inwiefern die Abhängigkeit von externen Service-Anbietern akzeptabel ist. Anhand der in Kapitel 3.2.2 vorgestellten Ausprägungen einer WOA lässt sich schon erahnen, dass es beliebig viele Zwischenstufen zwischen einer archetypischen WOA und einer Inhouse-WOA gibt. Man muss sich bei der Planung zur Umsetzung einer WOA daher bewusst sein, dass ein vollständig ausgelagertes IT-System bei einem Ausfall von Teilkomponenten ggf. den gesamten IT-Betrieb unterbricht. Dies ist aber dennoch kein Anlass, eine solche Entscheidung aufgrund des zu hohen Risikos abzulehnen. Es gibt genug funktionierende Beispiele im Bereich von Service-Dienstleistern, wie beispielsweise Zulieferern in der Automobilbranche, bei denen durch das Ausbleiben von Lieferungen die gesamte Produktion stillstehen würde. Es ist daher letztendlich immer eine Frage des Vertrauens und der Rechtssicherheit, die bei einer solch kritischen Entscheidung mitspielt. Daneben kann es auch rechtliche Vorgaben geben, die eine Entscheidung beeinflussen. Beim Auslagern von Services sind diverse Aspekte individuell für ein Unternehmen zu bewerten. Dies wird hauptsächlich die Qualität von Services, deren Sicherheit, sowie das Vertrauen in den Anbieter betreffen. Je höher der Anteil an ausgelagerten Services, desto höher ist die Abhängigkeit von den Dienstleistern. Andererseits sinken dadurch aber die zu erwartenden Kosten für eigene Implementierungen und die Wartung der eigenen IT-Systeme, da diese Tätigkeiten nicht mehr selbst, sondern von Service-Anbietern umgesetzt werden.

Um hierbei zu einer Entscheidung über das Outsourcing zu gelangen, muss man sich alle zur Umsetzung vorliegenden Prozesse anschauen und jeweils bewerten, inwiefern man bereit ist, diese Komponente auszulagern. Diese Entscheidung wird u. a. von den oben genannten Faktoren beeinflusst. Je nach Risikobereitschaft des Entscheiders wird hier für oder gegen Outsourcing gestimmt. Um die Entscheidungen bei einzelnen Services für die späteren Phasen zu verdeutlichen, werden die vorliegenden Dokumentationen der fachlichen Service-Beschreibungen um die Outsourcing-Entscheidung erweitert. Diese Kategorisierung muss mit jedem identifizierten IT-Service geschehen, damit die darauffolgenden Arbeitsschritte erfolgen können. Dies gilt ebenfalls für alle innerhalb der ggf. durchgeführten Ist-Analyse eines bestehenden IT-Systems identifizierten Komponenten und Services. Hierbei wird die Dokumentation der Ist-Analyse entsprechend erweitert.

Im Folgenden wird angenommen, dass für die beiden zu Beginn der Arbeit vorgestellten Szenarien die nachfolgenden Entscheidungen getroffen wurden: Die *BBI GmbH* möchte alle Services komplett auslagern und so eine archetypische WOA realisieren. Das Unternehmen *JKHL Enterprises* entschließt sich dazu, Teile der Daten im eigenen Unternehmen zu belassen, jedoch einen Großteil der E-Commerce-Bestandteile des IT-Systems durch XaaS-Anbieter umzusetzen. Damit wird hier eine regulatorische WOA realisiert, in der unternehmenskritische Daten nicht außerhalb des Unternehmens gespeichert oder verarbeitet werden.

5.3 Analyse- und Planungsphase

In der ersten Phase des Vorgehensmodells, welche sich in sechs Teilschritte untergliedert, werden die Anforderungen an das zu erstellende System aufgenommen, die Geschäftsprozesse und ggf. vorhandenen IT-Systeme des Unternehmens analysiert und die konkrete Systemumsetzung geplant. Hierbei wird auf eine explizite Spezifikation der Anforderungen in Form von Lasten- oder Pflichtenheften mit Verweis auf das agile Manifest (siehe Kapitel 2.1) verzichtet, da sich die Anforderungen an einzelne Komponenten und Services während der Planungsphase nach und nach ergeben. Sollte ein Unternehmen durch ein spezifisches Softwareentwicklungsmodell auf diese Dokumente angewiesen sein, so wird einfach eine formale Spezifikationsphase vor der Geschäftsprozessmodellierung durchgeführt (vgl. dazu [Bal01]).

5.3.1 Geschäftsprozessmodellierung

Die Geschäftsprozessmodellierung erlaubt die Darstellung von Abläufen in Unternehmen und Verwaltungen. Dabei soll hier zunächst der Begriff *Prozess* anhand der Definition von Becker und Schütte verdeutlicht werden [BS04, S. 107]:

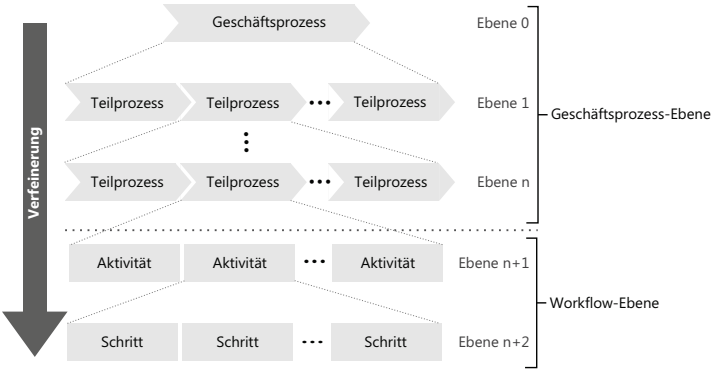
> *„Ein Prozess ist die inhaltlich abgeschlossene, zeitliche und sachlogische Folge von Aktivitäten, die zur Bearbeitung eines betriebswirtschaftlich relevanten Objekts notwendig sind."*

Die allgemeine Prozessdefinition wird für eine Schärfung des Begriffs *Geschäftsprozess* darüber hinaus in [BK08, S. 6f.] weiter eingegrenzt:

> *„Ein Geschäftsprozess ist ein spezieller Prozess, der der Erfüllung der obersten Ziele des Unternehmens dient und das zentrale Geschäftsfeld beschreibt. Wesentliche Merkmale eines Geschäftsprozesses sind die Schnittstellen des Prozesses zu den Marktpartnern des Unternehmens."*

Die in dieser Definition genannten Aspekte, wie die Folge der Aktivitäten sowie die Ausführung zum Zwecke eines Unternehmensziels, werden in ähnlicher Form auch in weiteren Quellen zur Eingrenzung des Begriffs Geschäftsprozess verwendet (vgl. [Wes07, Sta06, Sch01]). An dieser Stelle wird die Annahme getroffen, dass jedes Unternehmen entsprechend zugrunde liegender Geschäftsprozesse oder Geschäftsregeln (*business rules*) handelt und diese daher auch in Modellen abbildbar sind.

In der Literatur werden viele Vorgehensmodelle zur Aufnahme von Geschäftsprozessen diskutiert, die im Kern meist denselben Ansatz verfolgen, sich aber in der Ausgestaltung und den verwendeten Werkzeugen teilweise deutlich unterscheiden. Dieser Kernansatz beginnt mit der Modellierung auf oberster Unternehmensebene, mit wenigen Details, und wird dann durch Untermodelle immer weiter verfeinert, bis ein Detailgrad erreicht ist, der den Anforderungen der jeweiligen Methode genügt (vgl. [Wes07, JM05, SVOK11, Jos08, FH07, Off08]). Im Verlauf dieser Vorgehensweise entstehen unterschiedliche Dokumentationen, Modelle und Artefakte, die für eine spätere Verwendung, beispielsweise der Implementierung eines IT-Systems, von Nutzen sind. Abbildung 5.5 zeigt eine schematische Übersicht über die einzelnen Modellebenen und ermöglicht daran eine Unterscheidung der oft synonym verwendeten

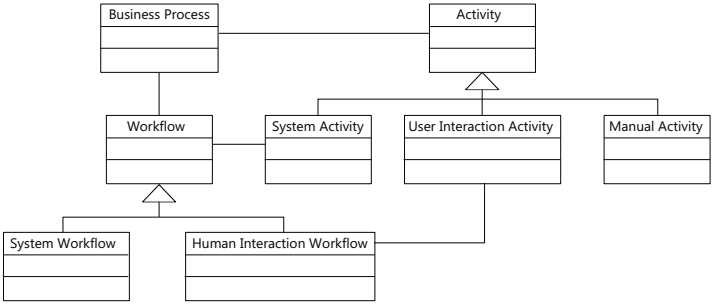


Quelle: angelehnt an [Jos08]

Abbildung 5.5: Geschäftsprozessmodell-Ebenen.

Begriffe Geschäftsprozess und Workflow. Hierbei kann ein Geschäftsprozess durch weitere Teilprozesse auf untergeordneten Ebenen beliebig oft unterteilt werden. Diese Verfeinerung nennt man *funktionale Dekomposition*. Sie kann so lange fortgesetzt werden, bis der Detailgrad des Geschäftsprozesses den Anforderungen genügt. Dies bedeutet im Normalfall, dass man bei einzelnen atomaren Aktivitäten angekommen ist, die nicht weiter teilbar sind. Werden an dieser Stelle, auf Ebene n, nun technische Verfeinerungen vorgenommen, um einen Prozess ablauffähig zu gestalten, erreicht man die Workflow-Ebene (vgl. [Jos08]).

Im Unterschied zum Geschäftsprozess wird der Workflow, wenn er überhaupt abgegrenzt betrachtet wird, meist technisch definiert und beschreibt so den IT-unterstützten Teil eines Geschäftsprozesses. Auch das konzeptionelle Modell aus [Wes07] grenzt den Workflow von dem Geschäftsprozess ab und unterscheidet darüber hinaus drei Aktivitätsarten, die in Geschäftsprozessen definiert werden können: Systemaktivitäten (*system activity*), die durch ein System automatisch ausgeführt werden, Aktivitäten, die durch den Benutzer ausgelöst werden (*user interaction activity*) und manuelle Aktivitäten (*manual activity*), die nicht durch ein IT-System unterstützt werden. In diesem Modell werden Systemaktivitäten mit einem Workflow verbunden, der sich wiederum als automatischer oder durch menschliche Interaktionen gesteuerter Workflow darstellen kann (siehe Abbildung 5.6).



Quelle: [Wes07]

Abbildung 5.6: Konzeptionelles Modell der Struktur von Geschäftsprozessen.

Die Phase der Geschäftsprozessmodellierung läuft in mehreren Schritten ab, die in Abbildung 5.7 dargestellt sind. Hierbei wird mit einer Ebene-0-Modellierung begonnen, um einen Überblick über die zu spezifizierenden Domänen zu erhalten. Anschließend wird der Schritt der funktionalen Dekomposition der Prozesse wiederholt, bis eine atomare Ebene der Aktivitäten erreicht ist. In dieser ersten Phase des Analysierens werden IT-Architekten sowie Domänenspezialisten benötigt, die gemeinsam die Geschäftsprozesse erfassen und verfeinern. Die während dieser Phase entstehenden Dokumentationen sind einerseits die fachlichen Geschäftsprozesse mit identifizierten Datenobjekten, ein Rollenmodell, welches die ausführenden Rollen spezifiziert, sowie eine Liste der Prozessverantwortlichen für jeden einzelnen Geschäftsprozess. Darüber hinaus existiert eine Spezifikation von Geschäftsregeln bei Aktivitäten auf Workflow-Ebene, sofern sich diese beschreiben lassen.

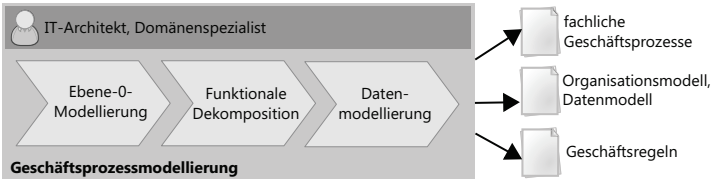
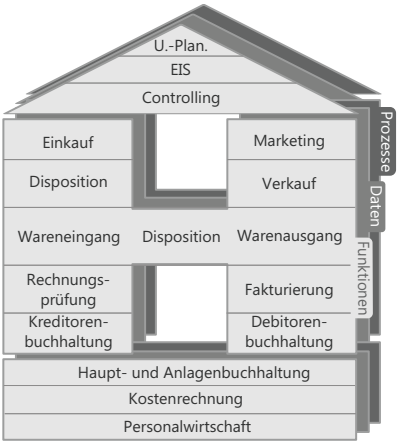


Abbildung 5.7: Die Schritte der Geschäftsprozessmodellierung.

Ebene-0-Modellierung

Die erste zu betrachtende Ebene, die sogenannte *Ebene 0*, soll eine vogelperspektivische Ansicht der zu modellierenden Elemente des betrachteten Unternehmens darstellen. Damit soll ein grober Überblick über den Geschäftsablauf wiedergegeben werden. Die wichtigen Informationen für jedes Modell sind hier die *Akteure*, die *Services*, die hier als eine Art Organisationseinheit betrachtet werden können, und die *Interaktionen* zwischen diesen. In einem Ebene-0-Modell werden dabei nur Services betrachtet, die eine Kernaufgabe des Unternehmens abbilden. Als Kernaufgaben eines produzierenden Unternehmens würden beispielsweise die primären Aktivitäten der unternehmerischen Wertschöpfungskette gelten (vgl. [Por85]). Bei einem Handelsunternehmen, wie der *JKHL Enterprises*, lässt sich das Handels-H-Modell von Becker als Referenz verwenden (siehe Abbildung 5.8). Dieses bildet einen Ordnungsrahmen für Handelsunternehmen und spezifiziert die einzelnen Teilsysteme des Handels, die für ein Warenwirtschaftssystem bzw. ein Handelsinformationssystem relevant sind. Dieser Ordnungsrahmen wird im Folgenden als Startpunkt für die Ebene-0-Modellierung dienen.



Quelle: nach [BS04]

Abbildung 5.8: Das Handels-H-Modell.

Als Werkzeug für das Erstellen eines Geschäftsprozesses auf oberster Ebene (Ebene 0) wird an dieser Stelle die Methode aus [JM05] verwendet. Darin wird zu Beginn der Modellierung ein rudimentäres Interaktionsdiagramm verwendet, welches einem UML-Anwendungsfalldiagramm ähnelt, aber im Gegensatz dazu die Interaktionen zwischen Akteuren und Aufgaben explizit benennt. Die folgenden Schritte werden darin ausgeführt:

1. Identifizierung der Aufgaben
2. Hinzufügen der Akteure
3. Erstellen der Interaktionen zwischen den Aufgaben und Akteuren

Ausgehend von einem Handelsunternehmen werden beispielhaft die folgenden Aufgaben identifiziert: Einkauf, Disposition, Marketing & Verkauf sowie Warenausgang. Diese initialen Aufgaben werden durch das Handels-H-Modell begründet, welches hier als Kontext-Modell für die gesamte Geschäftsprozessmodellierung eines Handelsunternehmens angesehen werden kann. Anschließend werden die an diesen Aufgaben beteiligten Akteure identifiziert, welches hier die Kunden, die Logistikpartner und die Zulieferer sind. Schließlich werden die Interaktionen zwischen Aufgaben und Akteuren herausgestellt. Ein Ebene-0-Modell sollte sich auf wenige Kernaufgaben beschränken und maximal fünf Aufgaben herausstellen (vgl. [JM05]). Die einzelnen Schritte dieses Modellverfahrens sind in Abbildung 5.9 dargestellt.

Ein solches Modell erlaubt es, sich über die Kernaufgaben des Unternehmens und die beteiligten Schlüsselfiguren (Akteure) einen Überblick zu verschaffen. Eine genauere Definition der Akteure oder Rollen wird an dieser Stelle noch nicht vorgenommen, da dies in den nächsten Verfeinerungsschritten erfolgt.

Funktionale Dekomposition

Der Arbeitsschritt der Dekomposition erfordert die Erstellung weiterer Sub-Modelle für jede Aufgabe eines vorhergehenden Modells. Hatte das Ebene-0-Modell vier Aufgaben identifiziert, so gibt es mindestens vier Ebene-1 Modelle. Dieser Schritt wird so lange wiederholt, bis der gewünschte Grad an Granularität erreicht ist. Die Modellierung von Interaktionsmodellen birgt den Vorteil, dass zu diesem Zeitpunkt keine expliziten chronologischen Abläufe modelliert

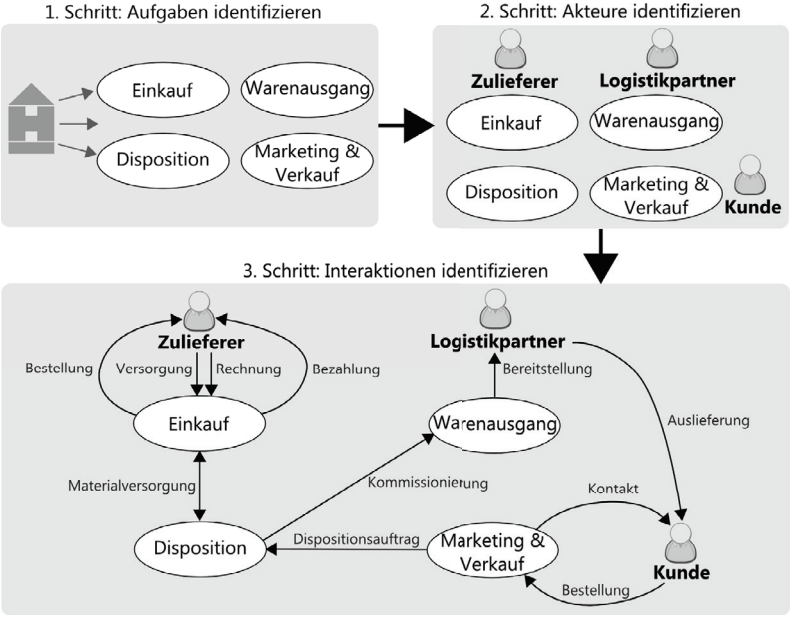


Abbildung 5.9: Modellierung eines Ebene-0-Modells.

werden müssen. Lediglich die Akteure, die vorhandenen Aufgaben (auf einem hohen Abstraktionsniveau) und deren Interaktionen werden verdeutlicht. Diesem Umstand ist es zu verdanken, dass man die benötigten Modelle relativ zügig erstellen kann. Je näher man dem nicht festgelegten „finalen" Abstraktionsgrad kommt, desto mehr wächst der Wunsch nach einer detaillierten Prozessansicht, um die Abläufe später tatsächlich durch IT-Prozesse zu realisieren. So wird in der hier vorgestellten Methode ab einem bestimmten Grad der Abstraktion auf die Modellierung mittels BPMN umgestiegen. Wichtig ist, dass dieser Umstieg erst erfolgt, wenn der exakte Prozessablauf für das Verständnis des Prozesses notwendig ist. Je nach Komplexität des betrachteten Unternehmens kann wiederum eine mehrstufige Verfeinerung der BPMN-Modelle notwendig und sinnvoll sein. Als Beispiel wird hier die Aufgabe *Marketing & Verkauf* des zuvor modellierten Ebene-0-Modells herangezogen.

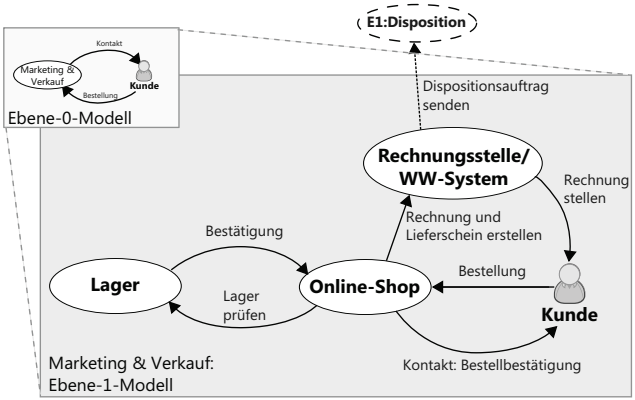


Abbildung 5.10: Marketing & Verkauf als Ebene-1-Modell (nach der Dekomposition).

Im ersten Schritt der Dekomposition wird dafür die zugrunde liegende Aufgabe verfeinert. Dabei wird die aus dem Elternmodell stammende Aufgabe betrachtet und in Teilprozesse gegliedert. Es ergeben sich daraus weitere Aufgaben, die zusätzliche Interaktionen erfordern und ggf. sogar weitere Akteure identifizieren. In Abbildung 5.10 ist das resultierende Ebene-1-Modell für *Marketing & Verkauf* dargestellt. Darin wurden „verfeinerte" Aufgaben, wie

Online-Shop, Lager und Rechnungsstelle, als Teil eines Warenwirtschaftssystems identifiziert. Durch den gestrichelten Interaktionspfeil auf die Aufgabe *E1:Disposition* wird die Interaktion mit einem weiteren Ebene-1-Modell dargestellt, welches durch den Namen, hier *Disposition*, identifiziert wird.

Während der Dekomposition wird kontinuierlich eine Liste mit Akteuren gepflegt, in der diese tabellarisch aufgeführt sind. Dies erlaubt in späteren Schritten das Aufstellen eines konkreten Rollen- und Rechtemodells für die einzelnen Geschäftsprozesse auf Basis der identifizierten Akteure. Dadurch entsteht eine Übersicht über die in Prozessen benötigten Rollen. Dabei muss ein Akteur nicht zwangsläufig ein menschliches Wesen sein: Je detaillierter ein Prozess ist, desto eher treten Systeme und Applikationen als Akteure auf, was vor allem dem Automatisierungsgrad der Prozesse zugute kommt.

Die Frage, wann man zur Geschäftsprozessmodellierung mit BPMN übergeht, ist nicht anhand einer genauen Anweisung, beispielsweise „ab der fünften Ebene werden BPMN-Modelle verwendet", zu beantworten. Ein BPMN-Modell der ersten Ebene für das Modell aus Abbildung 5.10 ist in Abbildung 5.11 gezeigt. Hierbei kann man auf den ersten Blick erkennen, dass mehr Informationen im Modell zu finden sind. Dies ist zum einen der korrekte chronologische Ablauf des Prozesses, die einzelnen Aufgaben (und wer diese tatsächlich ausführt) und diverse Angaben zu Datenobjekten. Letztere werden zusätzlich durch Hinweise der Verwendung verfeinert, wie beispielsweise schreibende oder lesende Zugriffe, und dienen später der Erstellung von Datenmodellen.

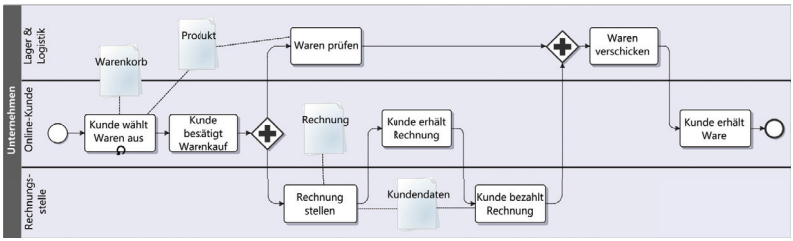


Abbildung 5.11: BPMN-Modell der Ebene 1 für das Modell *E1:M&K*.

In [JM05] wird vorgeschlagen, eine Liste mit Prozessverantwortlichen sowie Akteuren und Beschreibungen der einzelnen Aktivitäten zu pflegen. Daher

wird spätestens bei dem Wechsel auf die Modellierung mit BPMN-Modellen für jeden Prozess eine verantwortliche Rolle benannt. Es ist hierbei besser, einen Prozessverantwortlichen mit der qualifizierenden Rolle, beispielsweise „Abteilungsleiter Marketing", zu bezeichnen, als reale Personen zu verwenden, da sich solche Zuordnungen im Lauf der Zeit schnell ändern können. Zusätzlich werden für einzelne Aktivitäten an dieser Stelle, sofern dies möglich ist, fachliche Beschreibungen bzw. Geschäftsregeln hinterlegt. Diese können für die später folgende Realisierungsphase als Richtlinie für die Entwickler dienen. Mit Geschäftsregeln und der damit möglichen dynamischen Steuerung von Services beschäftigt sich das Feld des *Business-Rules-Managements*. Dabei besteht die Möglichkeit, die spezifizierten Regeln in einem so genannten *Business-Rule-Repository* zu hinterlegen und so dynamische Änderungen an Abläufen einfacher zu gestalten, statt die Regeln fest in Prozesse zu programmieren (vgl. [Ros03, GS06]). Ob sich eine Nutzung von Geschäftsregeln innerhalb von Workflows realisieren lässt, hängt hierbei von der Funktionalität eines zu nutzenden XaaS-Dienstes ab. Auf der Ebene der Geschäftsprozessmodellierung werden an dieser Stelle zunächst informelle Wenn-Dann-Regeln spezifiziert. Eine solche Regel für die Aktivität „Rechnung stellen" in Bezug auf die Verwendung eines Einkaufsgutscheins des Kunden könnte beschrieben werden, wie in Listing 5.1 gezeigt.

```
1 WENN
2  Einkaufsgutschein vorliegt
3  UND
4  Letzter Einkaufsgutschein vor mehr als einem Monat eingelöst
      wurde
5 DANN Einkaufsgutschein auf Rechnungsbetrag anrechnen
6 SONST Vollen Rechnungsbetrag verwenden
```

Listing 5.1: Geschäftsregel für die Aktivität „Rechnung stellen".

Eine mögliche Ausgestaltung der gesammelten Dokumentation zum oben ausgeführten Prozess ist in Abbildung 5.12 ausgeführt.

Der Schritt der Dekomposition wird erst abgeschlossen, wenn alle Prozesse des betrachteten Projekts (des Unternehmens) in ausreichender Granularität erfasst sind. Im Grunde ist dies dann der Fall, wenn man bei atomaren Aktivitäten innerhalb eines Prozesses angekommen ist. Die Aktivität „Kunde erhält Ware" ist beispielsweise ein atomarer Service, der nicht weiter untergliedert

Geschäftsprozess

Prozess-ID:	E1:M&K	**Prozessverantwortung:**	Abteilungsleiter M&K
Prozessname:	Marketing & Verkauf		
Elternprozess:	E0	**Dokumentiert am:**	10.12.2010

Beteiligte Rollen

Rolle	Beschreibung
Online-Kunde	Kunde des Unternehmens, der sich am Online-Shop-Portal angemeldet hat.
Rechnungsstelle	Internes Warenwirtschaftssystem und die Mitarbeiter der Abteilung M.
Lager & Logistik	-

Primäraktivitäten

Aktivität	Beschreibung	Geschäftsregel
Rechnung stellen	Die Rechnung für den Kunden wird erstellt und versendet	WENN .... DANN ...
Kunde wählt Waren aus ...	...	...

Beteiligte Datenobjekte

Objekt	Beschreibung
Warenkorb	Session-Daten während des Einkaufs (nicht persistent)
Kunde	Lesend: Kundendaten für das Ausstellen der Rechnung Schreibend: Fortschreiben des Kundenkontos
...	

Abbildung 5.12: Dokumentation des Prozessmodells *E1:M&K*.

werden kann. Nach Abschluss der Dekomposition sind die wichtigsten Informationen zum Prozessablauf bekannt und liegen in Form von fachlichen Prozessmodellen vor. Ebenso existiert eine Aufstellung von Rollen im Hinblick auf Prozessverantwortung sowie Beschreibung der einzelnen Akteure innerhalb der Prozesse.

Datenmodellierung

Durch die Identifikation der Datenobjekte aus der Prozessmodellierung können nun die dem IT-System zugrunde liegenden Datenstrukturen betrachtet werden. Dazu werden die beschriebenen Datenobjekte in Relation zueinander gesetzt und durch Domänenspezialisten ergänzt. Die Datenmodellierung durch Nutzung von ER-Diagrammen lässt sich hierbei gut einsetzen, da man analog zur Geschäftsprozessmodellierung mit den einzelnen Komponenten beginnt und diese weiter verfeinern kann. Die im Beispielprozess *E1:M&K* identifizierten Datenobjekte *Warenkorb*, *Produkt*, *Kunde* und *Rechnung* bilden den Ausgangspunkt der Datenmodellierung (siehe Abbildung 5.13).

Abbildung 5.13: 1. Schritt zum ER-Modell des Prozesses *E1:M&K*.

Die in den Prozessbeschreibungen hinterlegten Informationen, wie Lese- und Schreibzugriffe auf Daten sowie Geschäftsregeln, liefern weitere Kontextinformationen für den Entwurf. Im vorliegenden Beispiel wurden die Datenobjekte Kunde und Produkt eindeutig als Entitätstypen identifiziert, während der Warenkorb und die Rechnung nur in Verbindung mit weiteren Entitäten Sinn ergeben, und damit als Relationship-Typen verwendet werden. Nach der Verfeinerung der Beziehungen und dem Spezifizieren von Kardinalitäten resultiert hieraus ein aus dem Handel bekanntes ER-Modell mit zusätzlichen Relationship-Typen, wie Bestellkopf sowie Bestellposition, und weiteren Hilfs-Entitäten, wie in diesem Fall dem Kaufvertrag. In Abbildung 5.14 ist das resultierende ER-Modell dargestellt.

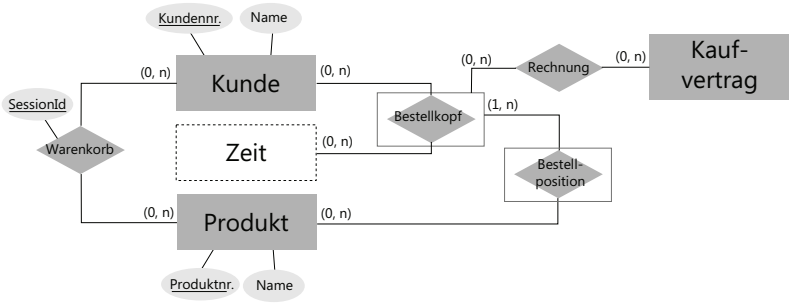


Abbildung 5.14: 2. Schritt zum ER-Modell des Prozesses *E1:M&K*.

Natürlich stellt dieses ER-Modell nur einen Ausschnitt des Datenmodells dar, welches die Datenstrukturen des Unternehmens abbildet. Daher muss während der Modellierung der einzelnen Teilmodelle bereits darauf geachtet werden, dass identische Datenobjekte oder Strukturen in Einklang gebracht werden und so ein Datenmodell entsteht, das alle benötigten Entitäten, Attribute und Relationship-Typen als fachliches Konzept in einem Modell vereint. Nach Abschluss der Datenmodellierung steht das Fachkonzept der Datenhaltung fest und kann für nachfolgende Phasen als Richtlinie verwendet werden. Geeignete Verfahren zur Erstellung von Datenmodellen sind beispielsweise in [Vos08] oder [Sta05] zu finden.

In einem letzten Schritt der Datenmodellierung sollten die Dateneigentümer, sofern vorhanden und zu identifizieren, in einer Liste festgehalten werden. Dieser Schritt ist vor allem für große Unternehmen relevant, in denen eine Abteilung oder eine spezielle Rolle als Dateneigentümer auftritt und ggf. den Zugriff auf die eigenen Daten reglementieren und steuern muss. Dies kann sich später auch durchaus auf die Ausgestaltung einer WOA auswirken, indem eine Rechtestruktur für den Zugriff auf im Internet liegende Daten eingerichtet werden muss, um bestehende Datenverantwortlichkeiten nach- bzw. abzubilden.

An dieser Stelle des Vorgehensmodells wird noch keine Betrachtung der Datenqualität oder -lokalität vorgenommen. Wo die hier modellierten und spezifizierten Daten in einer WOA später tatsächlich gespeichert werden, lässt sich zu diesem Zeitpunkt nicht festlegen, da noch keinerlei Auswahl von XaaS-

Diensten erfolgte. Dieser Aspekt einer WOA wird bei der Definition von Datenflüssen in Kapitel 5.3.5 erläutert.

5.3.2 Service-Identifikation

Durch die Phase der Geschäftsprozessmodellierung sind alle Prozesse des betrachteten Unternehmens bis zu einem bestimmten Detailgrad bekannt. Da diese vornehmlich die Abläufe bestimmen, soll nun in der Phase der Service-Identifikation herausgestellt werden, welche Aktivitäten in den Prozessen als IT-Services zu betrachten sind und welche nicht. Dies bildet die Grundlage für eine später folgende IT-Umsetzung der Prozesse.

Die in dieser Phase des Vorgehensmodells notwendigen Schritte umfassen zunächst die Typisierung der Aktivitäten innerhalb der Geschäftsprozesse sowie im Anschluss den Aufbau eines Service-Katalogs mit potenziellen XaaS-Anbietern für die auszulagernden Services. Diese Phase wird von IT-Architekten und Domänenspezialisten durchgeführt. Zum Aufbau des Service-Katalogs können zusätzlich noch Entwickler hinzugezogen werden, wenn diese breites Wissen über XaaS-Angebote beisteuern können. Das Resultat dieses Arbeitsschrittes sind Geschäftsprozesse mit typisierten Services und ein Service-Katalog. Darin lassen sich auch Services verzeichnen, die nicht ausgelagert werden dürfen oder können (z. B. aus Mangel an Anbietern), und daher eine eigene Implementierung erfordern. An dieser Stelle sei angemerkt, dass im Fall einer Migration eines bestehenden IT-Systems in dieser Phase noch ein Abgleich mit bereits existierenden Services bzw. einer IT-Infrastruktur vorgenommen werden muss, was in Kapitel 5.6 erläutert wird. Die einzelnen Schritte der Service-Identifikation sind in Abbildung 5.15 dargelegt.

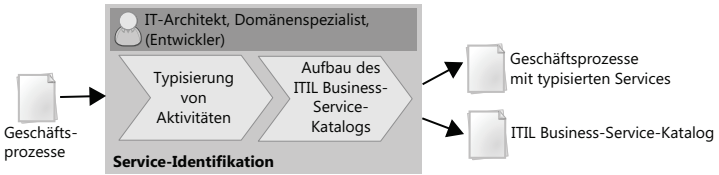


Abbildung 5.15: Die Schritte der Service-Identifikation.

Typisierung von Aktivitäten

Eine wichtige Angabe innerhalb der Prozesse ist für die spätere Umsetzung der WOA die Unterscheidung von IT-unterstützten Aktivitäten auf der einen Seite und manuellen Aktivitäten auf der anderen Seite. Manuelle Aktivitäten benötigen grundsätzlich eine Möglichkeit zur Interaktion mit dem IT-System. Dies kann in Form von Eingabe- oder Ausgabemöglichkeiten umgesetzt sein, also einer Schnittstelle zu einem menschlichen Akteur. IT-unterstützte Aktivitäten können diverse Formen annehmen und werden in der Realisierungsphase durch WP ersetzt. Aus diesem Grund werden die fachlichen Prozessmodelle um eine weitere Angabe, der Art der Aktivität, erweitert. Bei der Verwendung von BPMN zur Prozessmodellierung können seit Version 2.0 Aktivitäten durch zusätzliche grafische Annotationen einen Aufgabentyp zugeordnet bekommen, der die Aktivität charakterisiert. Dabei lassen sich die folgende Attribute zuordnen: *Senden*, *Empfangen*, *Benutzer*, *Manuell*, *Geschäftsregel*, *Service*, *Skript* (siehe Abbildung 5.16).

Abbildung 5.16: Charakterisierungen von Aktivitätstypen der BPMN.

Das tatsächliche Identifizieren von Services wird in vielen Vorgehensmodellen zur Umsetzung von SOA beschrieben. Es werden darin entweder fachliche oder technische Vorgehensweisen erläutert. Ein technischer Ansatz geht dabei immer von einem existierenden IT-System aus, bei dem die bestehenden Komponenten untersucht und klassifiziert werden können. Der fachliche Ansatz unterscheidet sich stark in der Ausgestaltung. Um Services zu identifizieren, wird in [Zac05] das Untersuchen von Aktivitäten in Prozessmodellen auf lose Kopplung der Aktivitäten und deren Kohäsion vorgeschlagen. Andere Ansätze gehen Schritt für Schritt vor und schließen zuerst manuelle Tätigkeiten und Aktivitäten von Altsysteme aus, um nach und nach Service-Kandidaten herauszustellen (vgl. [Erl07]) oder versuchen Aktivitäten zu erschließen, die in irgendeiner Form wiederverwendet werden (vgl. [Nad04]). Auch das Betrachten von Stakeholdern eines Geschäftsprozesses ist eine Möglichkeit, um auf Services zu schließen (vgl. [KKB07]). So verschieden die Ansätze auch sein mögen, das Ziel ist hierbei immer das Identifizieren von Aktivitäten, die aufgrund der Wiederverwendbarkeit oder Kapselung von Systemfunktionalitäten

als Services implementiert werden können und dadurch einen gewissen Mehrwert bieten. Die Prämisse bei der Umsetzung einer WOA ist jedoch das großflächige Auslagern von Funktionalitäten, so dass sich die Frage nicht stellt, ob das Erstellen eines Services angebracht ist oder nicht. Das Identifizieren von IT-gestützten Aktivitäten ist aufgrund des Wissens von Domänenspezialisten und IT-Architekten in dieser Phase trivial, so dass sich bei einer WOA-Umsetzung das Finden von XaaS-Angeboten für die erkannten IT-Services in den Vordergrund schiebt.

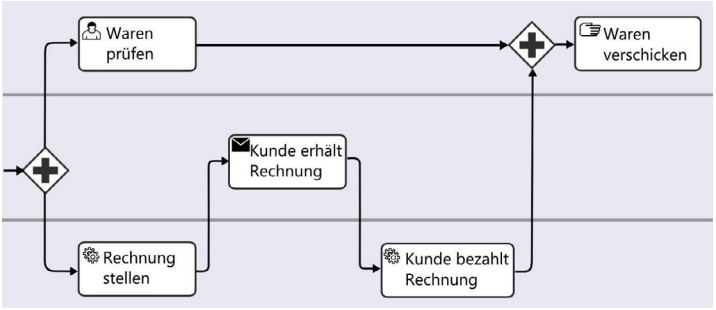


Abbildung 5.17: BPMN-Modell mit differenzierten Services.

Aus diesem Grund wird in dieser Phase zunächst die Typisierung der Aktivitäten eines Geschäftsprozesses vorgenommen. Für einen Ausschnitt aus dem Beispiel-Prozess aus Abbildung 5.11 lassen sich verschiedene Typen identifizieren und in das Prozessmodell eintragen (siehe Abbildung 5.17). Hierbei wurden neben rein manuellen Aktivitäten (Ware verschicken), auch durch den Benutzer durchzuführende Aktivitäten (Waren prüfen) sowie Aktivitäten mit Service-Aufrufen (Rechnung stellen) oder auch dem Mailversand (Kunde erhält Rechnung) identifiziert. Dieser Schritt wird nun kontinuierlich für jeden Prozess der untersten Ebene durchgeführt, um alle Aktivitäten zu typisieren.

Aufbau des Business-Service-Katalogs

ITIL schlägt im Rahmen des Service-Managements das Erstellen eines zweigeteilten Service-Katalogs vor. Im ersten Teil, dem *Business-Service-Katalog*, werden die Verknüpfungen zwischen Geschäftsprozessen und fachlichen Services

festgelegt, im zweiten Teil, dem *Technical-Services-Katalog*, sind die Verknüpfungen von fachlichen Services zu der tatsächlichen technischen Umsetzung beschrieben. Nach dem Verständnis von ITIL werden auf der technischen Ebene Hardware, Software, Daten und Anwendungen dargestellt. Dies wird für den WOA-Kontext in konkrete XaaS-Angebote und Altsysteme abgeändert. In Abbildung 5.18 ist diese Ansicht abgebildet.

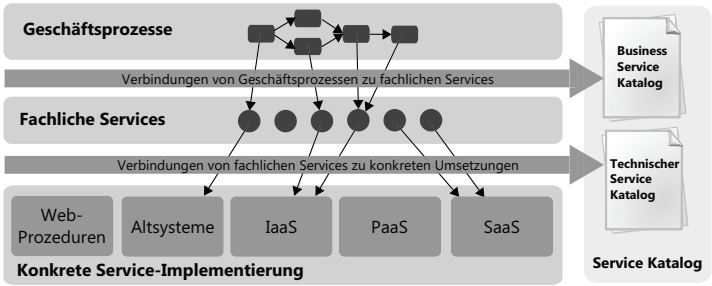


Abbildung 5.18: An WOA angepasster ITIL-Service-Katalog.

Bei der Aufnahme der typisierten Services in den Katalog, die später durch XaaS-Angebote umzusetzen sind, muss darauf geachtet werden, dass man versucht, identische Services nicht mehrmals aufzuführen, sondern zusammenfasst und somit die Möglichkeit der Wiederverwendung von Komponenten steigert. Eine Maßnahme zur Identifikation von wiederverwendbaren Services mit Fokus auf Web Services wird in [Let07] beschrieben: Darin wird ein Ansatz verfolgt, der es ermöglicht, auf Basis von semantischen Beschreibungen von Services und einem Schema-Vergleich identische Services während der Analyse und Planung zu identifizieren. Darüber hinaus sind auch weitere Ansätze zur Service-Beschreibung und dem Aufdecken von Ähnlichkeiten durch die Nutzung semantischer Annotationen bekannt. Dazu gehört beispielsweise die *Semantic Annotations for WSDL* (SAWSDL), worin WSDL-Beschreibungen von Services durch Annotationen um Ontologie-Verknüpfungen erweitert werden (vgl. [W3C07a]), ebenso wie die *Web Ontology Language for Services* (OWL-S), die sich mit der Spezifizierung von Service Profilen zur Unterstützung der Suche nach Web Services beschäftigt (vgl. [MBH+06]). Solche Methoden sollen dem Suchen und Finden von Services zuträglich sein, fokussieren sich aber ausschließlich auf Web Services und weitgehend auf WSDL als Beschrei-

bung von Services. Dadurch sind diese Ansätze im Kontext einer WOA nicht geeignet. Außerdem existieren Schnittstellen-Beschreibungen von Services zu diesem Zeitpunkt noch nicht, da zuvor gerade erst die Geschäftsprozessmodellierung abgeschlossen wurde. Im Gegensatz zu einer SOA und den darin implementierten Services werden innerhalb einer WOA möglichst viele Services an XaaS-Anbieter ausgelagert. Aus diesem Grund wird im Folgenden davon ausgegangen, dass man spätestens durch das Finden von passenden XaaS-Anbietern ähnliche Services identifizieren kann. Somit stellt sich das Problem der doppelten Entwicklung nicht als gravierend dar. Bei WOA-Ausprägungen, die viele Services im eigenen Unternehmen belassen und diese selbst implementieren, besteht es aber nach wie vor, wird hier aber als Randthema begriffen und daher nicht weiter thematisiert.

Als hilfreiches Detail für die spätere Verfeinerung der Services und die Suche nach XaaS-Anbietern wird zusätzlich eine erste Klassifizierung der soeben typisierten Aktivitäten vorgenommen. Zur ersten Einordnung dienen hierfür die im Grundlagenkapitel vorgestellten drei Kategorien: Infrastruktur-Services, fachliche Geschäftsprozess-Services sowie Präsentations- und Anzeige-Services. Für jede Aktivität wird an dieser Stelle eine Zuordnung notiert. Darüber hinaus sollte auch versucht werden, eine zusätzliche Kategorisierung aufgrund der Aktivitätsbeschreibung aus der Prozessmodellierung vorzunehmen. Dazu genügt das Hinzufügen von Attributen wie Datenbank, Bezahlung, grafische Auswertung etc. Der als Service typisierten Aktivität „Kunde bezahlt Rechnung" aus Abbildung 5.17 lässt sich beispielsweise die Kategorie Geschäftsprozess-Service und das Attribut *Bezahlung* zuordnen. Dies vereinfacht die Suche nach einem passenden XaaS-Anbieter für die Realisierung eines Services.

5.3.3 Service-Level-Requirements-Festlegung

Ein wichtiger Planungsschritt in jeder Softwareerstellung ist das Aufnehmen von Anforderungen an das zu erstellende System. Dies wird oft auch als Anforderungsanalyse (*Requirements Analysis*) bezeichnet. Dabei werden diverse Vorgehen für das Ermitteln der Anforderungen in der Literatur vorgeschlagen. Diese reichen von der Betrachtung bestehender Systeme über Interviews mit Anwendern bis zu kreativen Methoden, um zu ermitteln, was das System können muss (vgl. [PR10]). Eine konkrete Methode zur Erhebung der Anforderungen an die Services wird im Kontext dieser Arbeit jedoch nicht weiter vertieft. Es wird allerdings erläutert, welche Aspekte von Services beschrieben

werden müssen, um diese in einer WOA verwenden und verwalten zu können. In dieser Phase der Planung werden die zuvor identifizierten Services betrachtet, so dass man spezifische Anforderungen an diese aufnehmen kann. In der „Best Practices"-Sammlung ITIL wird hierbei von SLRs gesprochen, die als Vorläufer der konkreten Vereinbarungen mit den Anbietern, den SLAs, gesehen werden. Das von der *OASIS* festgelegte Beschreibungsmodell für Services einer SOA schlägt diverse Informationsbereiche vor, die für einen Service spezifiziert werden können (vgl. [MELT08]). Die fünf hierin genannten Kategorien von Informationen sind: Erreichbarkeit (*Service Reachability*), Schnittstellen (*Service Interface*), Funktionalität (*Service Functionality*), vertragliche Regelungen (*Policies & Contracts*) und Metriken (*Metrics*). Diese Kategorien bilden einen ersten Anhaltspunkt, sind aber durch die nicht vorhandene Trennung von funktionalen und nicht-funktionalen Beschreibungen nicht zufriedenstellend. In dieser Phase wird daher zunächst jeder Service auf seine nicht-funktionalen Eigenschaften hin skizziert, um anschließend durch technische Anforderungen erweitert zu werden. Der Vorgang wird durch Domänenspezialisten und IT-Architekten begleitet, die sich auf Anforderungen einigen müssen (siehe Abbildung 5.19).

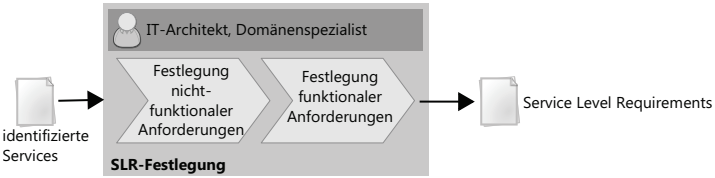


Abbildung 5.19: Die Schritte der SLR-Festlegung.

Nicht-funktionale Anforderungen

Die Anforderungen an einen Service, die nicht primär technischer Natur sind, befassen sich beispielsweise mit organisatorischen, rechtlichen und wirtschaftlichen Aspekten eines Services. Eine exakte Vorgabe von Kategorien für die Beschreibung solcher Anforderungen lässt sich nicht abschließend bestimmen, da viele Quellen auf unterschiedliche Ergebnisse kommen. So werden in [O'S06] durch Studien an vorhandenen Services im Internet die folgenden Merkmale einer nicht-funktionalen Service-Beschreibung hervorgehoben:

Verfügbarkeit, Preis, Bezahlung, Nachlass und Strafen, Verpflichtungen, Rechte, Qualität, Sicherheit und Vertrauen. Darin werden 80 Modelle aufgestellt, die die Verwendung dieser Aspekte erläutern. Als organisatorische und wirtschaftliche Rahmenpunkte der Service-Beschreibung werden in [DJI05] dagegen Pflichten des Dienstanbieters und Kunden, Konditionen, Prozesse zur Problemeskalation, Reporting und externe Kontrolle angeführt. *ITIL* wiederum nennt als Aspekte der SLRs u. a. die Kategorisierung des Services, die Erreichbarkeit, die Verfügbarkeit, die Performance und die Anforderungen bei Störungen (vgl. [Els06, SZ08]). Da sich die Anforderungen an eine Service-Beschreibung derart inhomogen zeigen, werden im Folgenden aus den vorgeschlagenen Aspekten die für eine WOA als wichtig erachteten erläutert und beispielhaft dargestellt.

Verfügbarkeit Je wichtiger ein Service für die täglichen Abläufe innerhalb des Unternehmens ist, desto höher muss dessen Verfügbarkeit sein. Ein wichtiger Faktor ist die Ausfallsicherheit des Services. Diese wird normalerweise als Prozentwert durch einen XaaS-Anbieter angegeben und kann dann durch den Service-Nutzer überwacht werden. Es gilt hierbei: je höher die Ausfallsicherheit, desto höher die Verfügbarkeit des Services. Aber auch der Preis eines Angebots steigt mit dem Anspruch an die Verfügbarkeit. Darüber hinaus reicht es auch nicht, nur einen Wert von beispielsweise 99,9 % als Verfügbarkeit des Services zu vereinbaren, sondern gleichzeitig auch die erlaubten Ausfallzeiten am Stück zu spezifizieren. Denn in diesem Beispiel würden im schlimmsten Fall die 0,1 % Ausfallzeit auf ein Jahr gerechnet und am Stück auftretend gute 8 Stunden betragen. Dies könnte den Ausfall eines möglicherweise geschäftskritischen Services für einen kompletten Arbeitstag bedeuten, ohne die vereinbarten Service-Levels des XaaS-Anbieters zu verletzen. Daher sollte zusätzlich zu einer Festlegung der prozentualen Verfügbarkeit auch die mittlere Reparaturzeit (*Mean Time To Repair*) und die mittlere Betriebsdauer zwischen Ausfällen (*Mean Time Between Failures*) als Anforderungen aufgenommen werden. Ebenfalls möglich ist die Festlegung von einer Anzahl erlaubter Störungen sowie Ankündigungsfristen von Wartungen an Services. Um auf sinnvolle Werte zu kommen, muss abgewägt werden, ob der Service rund um die Uhr verfügbar sein muss, oder es genügt, dass dieser während der normalen Geschäftszeiten ansprechbar ist.

Performanz Die Leistung eines Services sollte ebenfalls als qualitativer Aspekt festgelegt werden. Dies bedeutet einerseits die bis zu einer Antwort des Services verstreichende Zeit, andererseits die Zeit, die der Service für die tatsächliche Verarbeitung benötigt. Beispielsweise für den Einsatz eines Services für die Suche nach Produkten soll die Antwortzeit des Services auf eine Suchanfrage maximal 0,5 Sekunden betragen. Ein anderer Service wird benötigt, der die Verkaufsprotokolle der Kassensysteme der *JKHL Enterprises* jede Stunde aus allen Filialen zu einem Data-Warehouse-Dienst hochlädt, um diese analysieren zu lassen. Hierbei ist nun die Antwortzeit des Services weniger relevant, als die Verarbeitung der Datenanalyse im Hintergrund. Daher möchte man hier ggf. ein qualitatives Maß der fachlichen Funktion angeben können, wie beispielsweise: „Es werden 5 Gigabyte an Verkaufsdaten innerhalb von 5 Minuten analysiert". Demnach muss die fachliche Nutzung eines Services geklärt sein, um hier sinnvolle Anforderungen an die Performanz des Services stellen zu können.

Belastbarkeit/Nutzung Sehr eng mit der Performanz ist die Belastbarkeit bzw. die vorgesehene Nutzung des Dienstes verbunden. Hierbei geht es vor allem um die Anzahl und das geschätzte Datenvolumen von Aufrufen, womit die benötigte Netzwerkbandbreite und Leistungsfähigkeit eines Services beziffert werden kann. Denn es macht einen großen Unterschied, ob 20 oder 10.000 Anfragen pro Minute an einen Service geschickt werden. So können hier bereits Services mit einer absehbar hohen Datenintensität herausgestellt werden, die im Verlauf der Planungsphase ggf. einer besonderen Betrachtung unterzogen werden müssen. Dies ist nicht nur für das folgende Aushandeln von SLAs mit einem XaaS-Anbieter von Nutzen, sondern ebenso ein wichtiger Planungsaspekt für den Aufbau der Steuerungs- und Überwachungsschicht einer WOA.

Sicherheit Ein vor allem organisatorisch relevanter Aspekt betrifft die Sicherheit eines Services. Die hier zu spezifizierenden Anforderungen sind aus nichtfunktionaler Sicht festzulegen. Somit wird hier zum einen bestimmt, welche Person bzw. welche Rolle einen Service ausführen darf. Dabei kann dies eine natürliche Person ebenso wie ein System sein. Zum anderen wird hier festgelegt, in welcher Form die verwendeten oder zu verarbeitenden Daten vertraulich zu behandeln sind. Sollte hier beispielsweise angegeben werden, dass

Tabelle 5.5: Nicht-funktionale Anforderungen an einen Bezahlungs-Service.

Verfügbarkeit	24/7 (ständige Verfügbarkeit) *Ausfallsicherheit*: 99,9 % *MTTR*: 15 Minuten *Maximale Anzahl Ausfälle (inkl. angekündigte)*: 12 pro Jahr *MTBF*: 11 Tage *Wartungsankündigung*: 7 Tage vor Wartung
Performanz	Antwortzeit: 0,5 Sekunden Abwickeln des Zahlungsvorgangs: max. 2 Tage
Belastbarkeit / Nutzung	Bis zu 1.000 Transaktionen / Minute *Datenvolumen*: wenige Kilobyte pro Aufruf
Sicherheit (Ausführung) *Sicherheit (Daten)*	Nur das Online-Shop-System darf den Vorgang anstoßen. Transport der Zahlungsdaten verschlüsseln.
Preis	0,10 € pro Aufruf bzw. maximal 2 % des Transaktionsvolumens
Platzierung	Internet

die Nutzlast streng vertraulich ist, so hat dies Auswirkungen auf die folgenden funktionalen Anforderungen im Sinne von Verschlüsselung des Transportmediums, wie auch der Daten selbst, sowie die Planung der Sicherheitsstrukturen im Verlauf der Analyse- und Planungsphase.

Preis und Kosten Eine weitere wichtige Anforderung stellt der Preis eines Services dar. Dabei soll, je nach Natur eines Services, der maximale Preis bestimmt werden, den ein Service pro Abrechnungsperiode kosten darf. Hierfür lässt sich beispielsweise der Preis pro Aufruf oder Zeiteinheit angeben, wobei diese Berechnungsmöglichkeit stark von den Geschäftsmodellen der XaaS-Anbieter abhängt und somit lediglich eine grobe Richtlinie darstellt.

Platzierung Die Platzierung im Netzwerk wird hier als zusätzliches, WOA-spezifisches Merkmal aufgenommen und beschreibt, ob ein Service ausgelagert werden darf und soll. Dieses Merkmal ist unmittelbar von der strategischen Zielsetzung des Unternehmens beeinflusst. Hat sich das Unternehmen beispielsweise für ein komplettes Outsourcing entschieden, so wird jedem Service hier das Attribut *Internet* zugeordnet. Sollte ein Service nicht ausgelagert werden, gibt es unterschiedliche Abstufungen: So lässt sich ein Service beispielsweise auf einem eigenen virtuellen Server, in der privaten Cloud oder innerhalb des eigenen Netzwerks platzieren. Die Aussage über die Platzierung im Netzwerk ist in der folgende Phase ein Kriterium, ob für den entsprechenden Service nach einem XaaS-Anbieter gesucht werden muss.

Ein wichtiger Teil der Kontrolle von Services während des Betriebs stellen die sogenannten *Key Performance Indicators (KPIs)* dar, die einzelne, überprüfbare Merkmale eines Services überwachen. Diese können aus den SLRs abgeleitet und in den SLAs mit den jeweiligen Anbietern festgelegt werden. Vor allem Metriken für die Überwachung der Verfügbarkeit und der Performanz sind hierbei erstrebenswert. Auf diesen Punkt wird in Kapitel 5.5.1 noch näher eingegangen. In Tabelle 5.5 ist eine beispielhafte und sehr verkürzte Darstellung der nicht-funktionalen Anforderungen für die Nutzung eines Kreditkarten-Bezahlungs-Services dargestellt. Dieser wird beispielsweise im Handelsunternehmen *JKHL Enterprises* für den Online-Shop eingesetzt.

Funktionale Anforderungen

Funktionale Anforderungen an Services betreffen eher die technische Ebene und werden vor allem später für die Implementierung einer WOA benötigt. In [JM05] sowie [WSM06] werden die relevanten funktionalen Anforderungen benannt, die für eine Service-Beschreibung vorhanden sein müssen. Einige dieser Aspekte finden sich auch in einer WSDL-Datei wieder, die im SOA-Bereich der Standard für die Beschreibung von funktionalen Eigenschaften eines Web Services darstellt (vgl. [W3C07d]). Folgende funktionale Eigenschaften sind hierbei relevant: Eingabedaten, Ausgabedaten, Vor- und Nachbedingungen. Darüber hinaus werden innerhalb des WOA-Kontextes noch zwei weitere Kategorien hinzugefügt, die als wichtig erachtet werden: das gewünschte Antwort- sowie Fehlerverhalten eines Services sowie ggf. eine spezielle Technologieverwendung. Die Anforderungen an identifizierte Services werden daher in den folgenden Ausprägungen vorgenommen.

Ein- und Ausgaben Hierbei wird spezifiziert, welche Daten an den Service übergeben werden bzw. vom Service zurückgegeben werden, und welches Format diese haben. Darüber hinaus wird auch noch das Protokoll bestimmt, mit dem die Daten übertragen werden sollen. Auch muss hier analog zu den nicht-funktionalen Anforderungen die Sicherheit der Daten beschrieben werden. Dies kann Authentifizierung, Zertifizierung bzw. Verschlüsselung sowie weitere gewünschte sicherheitsrelevante Anforderungen betreffen.

Vor- und Nachbedingungen Es kann vorkommen, dass gewisse Vorbedingungen für den Aufruf eines Services gegeben sein müssen. Diese können an dieser Stelle spezifiziert werden. Eine Voraussetzung könnte sein, dass ein Nutzer am System eingeloggt sein muss, um einen Service auszuführen, oder dass innerhalb der Softwareumgebung eine Variable einen speziellen Wert enthält. In den Nachbedingungen kann definiert werden, welche Änderung sich durch die Ausführung des Services an der IT-Umgebung ergibt. Diese Angaben sind vor allem für die Möglichkeit des Systemtests und der Systemüberwachung von Relevanz.

Antwortverhalten Hiermit lässt sich spezifizieren, nach welchem Muster Daten gesendet und empfangen werden. Es können diverse Ausprägungen für

Tabelle 5.6: Funktionale Anforderungen an einen Bezahlungs-Service.

Eingabedaten	*Inhalt*: Kreditkartenunternehmen, Kreditkartennummer, Name des Kunden, Betrag, Währung *Format*: JSON-Daten *Transportprotokoll*: HTTPS (verschlüsselt)
Ausgabedaten	*Inhalt I*: Bestätigung der angewiesenen Bezahlung *Inhalt II*: Fehlermeldung (siehe unten) *Format*: JSON/XML/serialisiertes PHP *Transportprotokoll*: HTTPS (verschlüsselt)
Vorbedingungen	*Authentifizierung*: Nutzer muss mit eigener Kennung am Shop angemeldet sein.
Nachbedingungen	–
Antwortverhalten	Synchroner Serviceaufruf
Fehlerverhalten	*Keine Zahlung möglich*: Kunde um andere Zahlungsart bitten. *Karte abgelaufen*: Kunde benachrichtigen. *Service-Timeout*: Kurze Timeout und bis zu 3 weitere Versuche vornehmen. Dann zu „Unbekannter Fehler“ weiterleiten. *Unbekannter Fehler*: Warenkorb des Kunden für späteres Bezahlen speichern und entsprechend benachrichtigen.
Spezielle Technologieverwendung	–

Service-Aufrufe sinnvoll sein. Nicht jeder Aufruf eines Services muss dabei synchron verlaufen und Daten versenden und eine Antwort empfangen. Es gibt hierbei z. B. auch die Möglichkeit, Daten nur zu senden ohne auf eine Antwort zu warten, oder als asynchronen Aufruf zu kennzeichnen, der von einer anderen Funktion bzw. einem Service zu einem späteren Zeitpunkt die Weiterverarbeitung übernimmt.

Fehlerverhalten Ein nicht zu vernachlässigender Aspekt ist das Auftreten von Fehlern. Dazu muss zunächst geklärt werden, welche Fehler auftreten können und an welcher Stelle diese auftreten. Hier lassen sich beispielsweise Netzwerkfehler und Serverfehler nennen, die dazu führen, dass die Kommunikation mit einem Service nicht möglich ist. Es können auf Serverseite Validierungsfehler auftreten, wenn falsche Werte an einen Service übermittelt werden. Darüber hinaus können Verarbeitungsfehler, Laufzeitfehler und dergleichen mehr auftreten, die nicht vom Service-Nutzer beeinflussbar sind. Wichtig ist hierbei, dass die vom Service gesendeten Fehlermeldungen verstanden werden können und die Geschäftsprozessinstanz bzw. die implementierte Logik entsprechend reagiert. Ebenfalls müssen Fehler, die auftreten, ohne dass der Service eine Fehlermeldung übertragen konnte, abgefangen und berücksichtigt werden. Ein wichtiger Aspekt ist hierbei auch das Erkennen eines Fehlers. Dies lässt sich in vielen Fällen durch die Überprüfung der zurückgegebenen Werte eines Service-Aufrufs vornehmen. Hierzu können beispielsweise Fehlercodes zum Einsatz kommen, die bei Web Services im Payload einer Antwort gesendet werden. Bei REST-Services ist dies durch die Verwendung von HTTP noch einfacher, da sich zum Identifizieren von Fehlerverhalten die HTTP-Statuscodes einer Nachricht betrachten lassen. Alle Codes größer als 400 signalisieren hierbei einen Fehler (siehe Kapitel 2.2.2). Eine ganz spezielle Herausforderung stellen in diesem Kontext Transaktionen dar, in denen mehrere Serviceaufrufe als eine atomare Aktion angesehen werden und nur zusammen oder gar nicht ausgeführt werden sollen. Hier müssen spezielle Maßnahmen getroffen werden.

Technologieverwendung Durch die Verwendung eines Altsystems ist ggf. eine spezielle Programmiersprache oder Technologie zu verwenden. Diese Anforderung kann an dieser Stelle hinterlegt werden, um die Auswahl von

Services zu erleichtern.

Auch bei den funktionalen Anforderungen ist wichtig, dass (bis auf die Spezifikation ggf. notwendiger Systeme bei der *Technologieverwendung*) keine technischen Details eines Services spezifiziert werden, wie beispielsweise Schnittstellen oder konkrete Endpunkte. Die hier festgelegten Anforderungen sollten rein fachlicher Natur sein, um eine später zu erfolgende Suche nach passenden Services zu vereinfachen. Ein Beispiel für den Service „Kunde bezahlt Rechnung" ist in Tabelle 5.6 gezeigt.

5.3.4 XaaS-Anbieter-Bestimmung

In der fünften Phase des Vorgehensmodells zur Erstellung einer WOA werden konkrete Service-Anbieter gesucht, die für die zuvor identifizierten und analysierten fachlichen Services verwendet werden können. Hierbei werden alle fachlichen Services des Business-Service-Katalogs herangezogen und ausgewertet. Das erste zu untersuchende Merkmal ist der nicht-funktionale Parameter *Platzierung* eines Services. Je nach Ausprägung des Merkmals entscheidet sich, ob eine Suche erfolgen muss. Die Nutzung eines XaaS-Angebots ist nur dann möglich, wenn der Service ausgelagert werden darf und daher hier der Wert „Internet" hinterlegt ist. Für alle anderen Werte, beispielsweise „Intranet", „virtueller Server" oder „private Cloud" wird entweder eine eigene Implementierung des Services notwendig oder es müssen Service-Angebote gesucht werden, die sich auf eigenen Servern installieren lassen. Wurden Service-Anbieter ausfindig gemacht, müssen die SLAs festgelegt werden. Oft ist dieser Schritt schnell erledigt, da bei vielen XaaS-Anbietern die angebotenen Services und die dazugehörigen Service Levels bereits durch unterschiedliche Pakete vorgegeben sind. Möglicherweise lassen sich aber auch individuelle SLAs festlegen. Nach Abschluss der XaaS-Anbieter-Bestimmung sind die SLAs festgelegt und damit liegt ein technischer Service-Katalog (nach ITIL) vor, der die Verbindung von fachlichen Services zu konkreten Umsetzungen von Services beinhaltet. Dieses Vorgehen ist in Abbildung 5.20 skizziert.

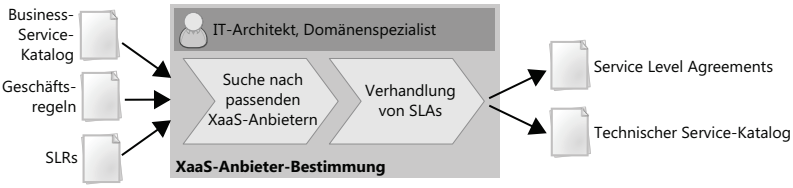


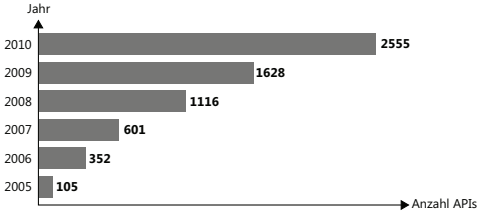
Abbildung 5.20: Die Schritte der XaaS-Anbieter-Bestimmung.

Suche nach XaaS-Anbietern

Zunächst muss für jeden auszulagernden Service eine Möglichkeit gesucht werden, diesen durch einen XaaS-Anbieter zu realisieren. Dieser Schritt ist nicht ganz trivial, da man entsprechend der spezifizierten SLRs, den Service-Beschreibungen sowie den Geschäftsregeln für einen Service einen passenden XaaS-Anbieter suchen muss. Die Möglichkeit der Service-Suche im Kontext einer SOA durch ein UDDI-Verzeichnis wurde bereits im Grundlagenkapitel erörtert und ist für eine globale Suche nicht geeignet, ganz abgesehen davon, dass kein umfassendes UDDI-Verzeichnis existiert, welches einen Großteil aller Web Services im Web-Umfeld umfasst. Da sich an dieser Stelle die automatische Suche nach passenden Services aus Mangel an Optionen ausschließt, müssen existierende Service-Verzeichnisse von Hand durchsucht werden. Hierfür eignet sich in Bezug auf WP die Webseite `www.programmableweb.com`, die sich als umfassendes Verzeichnis von Web APIs durchgesetzt hat[50]. Ende des Jahres 2010 waren dort über 2500 Web APIs und über 5000 Mashups registriert. Natürlich sind hierbei nicht alle Web APIs professionell nutzbar, aber allein der stetige Zuwachs an registrierten APIs in den letzten 5 Jahren ist bemerkenswert und dokumentiert das steigende Interesse an Web-basierten Schnittstellen (vgl. Abbildung 5.21).

Die meisten Verzeichnisse verfügen über Services, die nach Kategorien geordnet sind, und deren APIs textuell und technisch beschrieben werden. Hierbei reicht die Auswahl von einfachen Informations-Services, beispielsweise Börsenkurse oder Wetter-Services, bis hin zu komplexen SaaS-Diensten. Im

[50] Weitere Service-Kataloge oder -Sammlungen sind beispielsweise `http://techmagazine.ws/full-web-20-api-list/`, `http://www.webmashup.com` oder `http://www.service-repository.com/`.



Quelle: aus [Mus10]

Abbildung 5.21: Registrierte APIs bei `www.programmableweb.com`.

Fall von `ProgrammableWeb` wird jede einzelne API über eine Detailseite erläutert, die u. a. eine Kurzbeschreibung des Dienstes, mit diesem Service realisierte Mashups, ähnliche Services aus der gleichen Kategorie sowie Anleitungen und Referenzen zur API-Nutzung bereitstellt. Der aufwendige Teil ist nun, einen Service-Anbieter zu finden, der den zuvor beschriebenen SLRs und den fachlichen Service-Beschreibungen genügt. Eine Hilfestellung ist hierbei die in der Phase der Service-Identifikation vorgenommene erste Kategorisierung eines Services als Infrastruktur-, Fachlogik- oder Präsentations-Service sowie die Vergabe von Attributen als Beschreibung der Funktionalität, wie beispielsweise *Bezahlung* für einen Online-Bezahlungs-Service.

Als Beispiel für die Service-Suche dient hier wiederum der Online-Shop der *JKHL Enterprises*, worin ein Bezahlungsvorgang benötigt wird, der mit Kreditkartendaten umgehen kann und sich mittels JSON-Nachrichten in den Shop (der an dieser Stelle nicht weiter definiert wird) anbinden lässt. Nach der Einschränkung der Kategorie auf Bezahlungsdienste (*Payment*) listet `ProgrammableWeb` 30 unterschiedliche APIs, die durch das Vorgeben des Protokolls REST auf 17 Dienste reduziert werden. Die hierfür verwendete Suchmaske, die noch weitere Filtermöglichkeiten zeigt, ist in Abbildung 5.22 dargestellt.

Abbildung 5.22: Suchmaske von `ProgrammableWeb`.

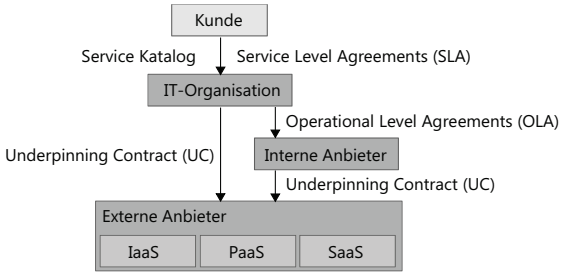
Ein passender Service-Anbieter für den Bezahlungsprozess scheint hier beispielsweise „Amazon Flexible Payments Service" zu sein. Dieser ermöglicht Kunden, Waren über einen eigenen Amazon-Account zu bezahlen. Die anfallenden Gebühren berechnen sich je nach Höhe der Transaktion aus einem fixen Anteil, der bis zu 0,30 $ kosten kann, und einem variablen Anteil, der maximal 5 % des Transaktionsvolumens ausmacht.

Wurde ein passender Service identifiziert, wird dieser zunächst als Alternative vorgemerkt, bevor im nächsten Schritt die konkreten SLAs ausgehandelt werden. Sollte man nun über den gesuchten Service hinausgehend einen SaaS- oder PaaS-Anbieter finden, der auf mehrere der fachlichen Beschreibungen passt, so kann man ggf. komplette Prozesse durch einen solchen Anbieter realisieren. Beispielsweise im Bereich von CRM- und *Enterprise Resource Planing* (ERP)-Systemen gibt es diverse Angebote, die günstige Web-Dienste anbieten. Sollte kein passender Service zu finden sein, so wird dies innerhalb des Service-Katalogs entsprechend verzeichnet. Diese Komponenten müssen später in der Realisierungsphase im Intranet, einer privaten Cloud oder innerhalb eines PaaS-Anbieters selbst programmiert werden.

Verhandlung von Service Level Agreements

Nachdem die Alternativen für die Umsetzung von Services ausgewählt wurden, müssen die SLAs mit dem Anbieter verhandelt werden. Viele Dienste bieten bereits fertige SLAs an, wobei dann meist höhere Verfügbarkeit, mehr Speicher oder Funktionalität durch höhere Kosten ermöglicht wird. In dieser Phase müssen die spezifizierten SLRs der Services mit den gebotenen SLAs verglichen und entweder eine Verhandlung aufgenommen oder ein anderer Dienst ausgewählt werden. Das ITIL-Modell der Service-Leistungs-Beschreibung unterscheidet hierbei den Kunden eines Services, die IT-Organisation, die den Service unternehmensintern anbietet, sowie externe Anbieter (siehe Abbildung 5.23). Die zwischen Kunde und IT-Organisation getroffenen Vereinbarungen nennt man SLAs, die Regelungen zwischen der IT-Organisation und tatsächlichen externen IT-Anbietern werden dagegen in ITIL als *Underpinning Contract* (UC) bezeichnet. Für die unternehmensinternen Vereinbarungen zwischen verschiedenen IT-Service-Anbietern wird darüber hinaus noch der Begriff *Operational Level Agreements* verwendet. Streng genommen handelt es sich also bei den hier zu treffenden Vereinbarungen zwischen Unternehmen und XaaS-Anbietern um UCs. Da aber im Falle einer WOA das Unternehmen als Kunde

gesehen wird, und selbst die XaaS-Anbieter in den meisten Fällen den Begriff SLAs verwenden, soll im Folgenden als vereinfachte Sichtweise die Vereinbarung über den Umfang von Leistungen zwischen Unternehmen und externen Anbietern als SLAs bezeichnet werden. Gerade für kleine Unternehmen ist das ITIL-Modell zu komplex, da hierbei die Unterscheidung von Kunden und internen Service-Anbietern meist weniger relevant ist.



Quelle: www.itil.org

Abbildung 5.23: Konzept der SLAs im ITIL Service Management.

Im Falle des Dienstes *Amazon FPS* sind beispielsweise Regelungen vorhanden, die neben organisatorischen auch rechtliche Aspekte abdecken. Echte SLAs, die beispielsweise die Verfügbarkeit regeln, sucht man aber vergebens. An dieser Stelle sollte man dann mit dem Anbieter in Kontakt treten und entsprechend seiner Vorstellung SLAs aushandeln und beim Scheitern von SLA-Verhandlungen entweder die eigenen Anforderungen entsprechend anpassen oder einen anderen Anbieter auswählen.

Nach diesem Schritt liegen für jeden zu realisierenden Service SLAs vor, die den genauen Umfang der zu erbringenden Leistung des XaaS-Anbieters festschreiben. Im Idealfall lassen sich diese durch Metriken beschreiben, so dass die zugesicherten Leistungen im Betrieb einer WOA auch überprüfbar sind.

5.3.5 Topologie- und Sicherheitsplanung

Die Planung der Topologie sowie der Aspekte zur Sicherheit umfasst mehrere aufeinanderfolgende Schritte. Im ersten Schritt werden die vorliegenden Geschäftsprozesse weiter detailliert und die typisierten Aktivitäten um Details der

XaaS-Anbieter verfeinert. Dabei entstehen Strukturen, die darüber Auskunft geben, welche Akteure an einem Prozess beteiligt sind und welche Schritte durch diese auszuführen sind. Dabei werden die Services auch in technischer Hinsicht näher beschrieben und um Service-Beschreibungen wie WSDL oder WADL erweitert. In einem nächsten Schritt werden die Datenflüsse innerhalb der Prozesse detailliert beschrieben. Dies ist notwendig, damit in der Realisierungsphase die Services mit den notwendigen Daten angesprochen werden können. Im Anschluss daran kann nun die Planung der Netzwerk-Topologie folgen, in der die fertig spezifizierten Prozesse aus einer Vogelperspektive heraus betrachtet werden. Hierbei werden der Aufbau und die Vernetzung der konkret zu implementierenden IT-Komponenten geplant und entschieden, mit welchen Komponenten, auf welchen Servern die Prozesse tatsächlich umzusetzen sind. Der letzte Schritt dieser Phase besteht in der Planung der Sicherheitsstruktur einer WOA. Darin werden Regelungen zur Absicherung von Services getroffen und ggf. Richtlinien zu Zugriffsrechten auf Daten oder Prozesse spezifiziert. So lassen sich hierbei verschiedene Stufen der Sicherheit oder gar Verschlüsselung von Daten festlegen. In Abbildung 5.24 sind die Schritte dargestellt. Auch diese Planung wird von IT-Architekten und Domänenspezialisten durchgeführt.

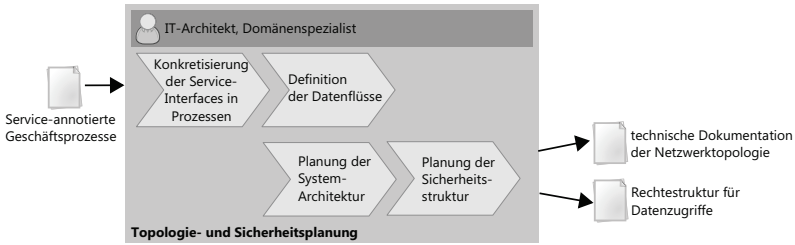


Abbildung 5.24: Die Schritte der Topologie- und Sicherheitsplanung.

Konkretisierung von Service-Interfaces in Prozessen

Dieser Schritt widmet sich der Spezifizierung von Aufrufen konkreter Services in den Geschäftsprozessen. Es werden also die Prozessmodelle um Informationen erweitert, die genau beschreiben, *welcher* XaaS-Dienst *wie* aufgerufen

wird. Bisher waren die Prozessmodelle, unabhängig von der verwendeten Prozessmodellierungssprache, frei von technischen Details, was sich in diesem Schritt ändert. Die Grundlagen hierfür sind durch vorherige Arbeitsschritte, wie die Suche nach XaaS-Anbietern und die Beschreibung von Services und Datenmodellen, bereits gelegt worden. Nun wird das Ziel verfolgt, einen Ausführungsschritt des Prozessmodells fest mit einer Service-Beschreibung zu verbinden und damit den aufgerufenen XaaS-Dienst kenntlich zu machen (vgl. [TV09b]). Generell sollte sich dieses Verfahren mit jeder Prozessmodellierungssprache durchführen lassen, wird hier aber anhand der BPMN auf Basis der bereits erstellten Modelle erläutert. Die folgenden Aktivitäten werden nun für jedes einzelne Geschäftsprozessmodell ausgeführt:

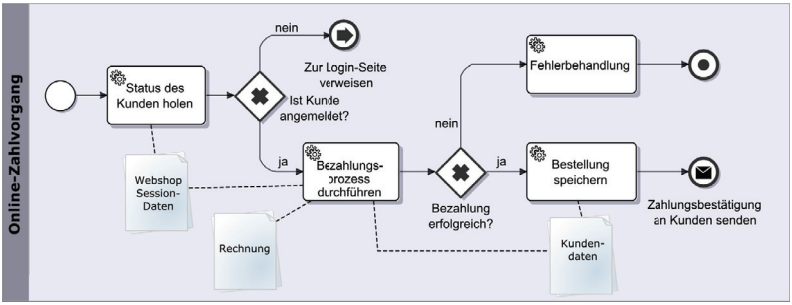
- Jede Aktivität, die durch einen XaaS-Anbieter aus dem Internet ausgeführt werden soll, wird in einen eigenen Pool verlagert. Dies betrifft Aufrufe der API des Anbieters über jedwede WP. Sollten zusätzlich eigene WP implementiert werden, die beispielsweise auf einem virtuellen Server liegen sollen, so werden diese ebenfalls in einen eigenen Pool verschoben.

- Alle Aktivitäten, die durch einen Inhouse-Service ausgeführt werden, werden ebenfalls in einen eigenen Pool verschoben. Dies betrifft beispielsweise ein Altsystem, für das eigene Services für die Anbindung an eine WOA zu implementieren sind oder WP, die im eigenen Unternehmen verbleiben. Solche Services sollen durch die Bezeichnung des Pools kenntlich machen, welches System betroffen ist.

- Ein Pool mit Namen „Logik" wird für die generelle Ablaufsteuerung des Prozesses angelegt. Wo und wie sich dieser realisieren lässt, wird später definiert. Hierin finden sich auch alle manuellen Services wieder, die durch einen menschlichen Akteur ausgeführt werden müssen.

Nach der Anwendung dieser Regeln wird es in jedem Prozessmodell mindestens den Pool „Logik" geben. Die Anzahl der weiteren Pools ist nicht beschränkt, sollte sich aber im Rahmen einer 1-stelligen Anzahl bewegen, da sonst die Übersichtlichkeit verloren geht. Wird diese Grenze überschritten, ist der Prozess trotz durchgeführter Geschäftsprozessmodellierung vermutlich noch zu groß und sollte daher in weitere Sub-Prozesse aufgeteilt werden. Um das oben genannte Vorgehen an einem Beispiel praktisch zu veranschaulichen,

wird hier ein Beispielprozess „Online-Bezahlung“ verfeinert, der den Zahlungsvorgang in einem E-Commerce-Shop des Unternehmens *JKHL Enterprises* darstellt. In Abbildung 5.25 ist dieser als reine Workflowansicht dargestellt, bevor eine Spezifizierung der Services vorgenommen wird.



Quelle: [TV09b]

Abbildung 5.25: Beispielprozess „Online-Bezahlung“.

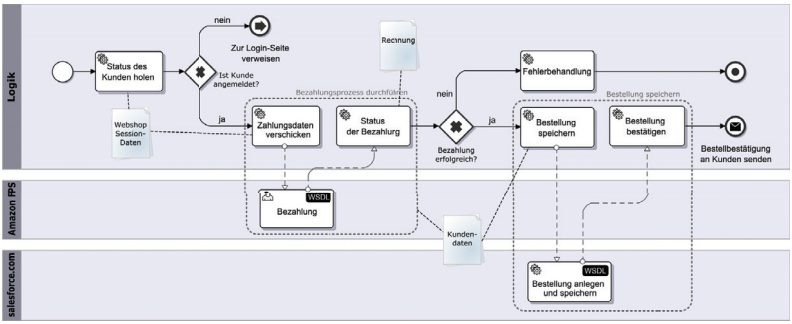
Die beiden Aktivitäten „Bezahlungsprozess durchführen“ und „Bestellung speichern“, die in vorangegangenen Phasen als *Service* typisiert wurden, sollen nun ausgelagert werden. Hier werden beispielhaft die folgenden XaaS-Dienste ausgewählt:

- Amazon *Flexible Payment Service (FPS)* für die Bezahlung der Bestellung, und
- der SaaS-Anbieter *salesforce.com* für die Speicherung der Bestellung.

Gemäß der oben genannten Regeln werden nun konsequent die beiden Pools „Amazon FPS“ und „salesforce.com“ angelegt und der jeweils darin aufzurufende Service mittels WSDL-Dokument der Anbieter[51] beschrieben. Der so verfeinerte Prozess ist in Abbildung 5.26 dargestellt.

Nach diesem Schritt besitzt dieser Geschäftsprozess nun drei Pools, die den Prozessablauf konkretisieren. Hierbei wurde angenommen, dass die aufzurufenden Services eine Service-Spezifikation durch WSDL- oder WADL-Dokumente besitzen, so dass eine grundlegende Dokumentation vorliegt, die

[51] Zur Vereinfachung wird unterstellt, dass jedes WSDL-Dokument genau einen einzigen Service beschreibt.



Quelle: [TV09b]

Abbildung 5.26: Prozess „Online-Bezahlung“ nach der 1. Verfeinerungsstufe.

es ermöglicht, die Services aufzurufen. Bei dem großen Angebot von XaaS-Anbietern gibt es aber durchaus auch rein informell beschriebene Services, so dass diese an der jeweiligen Stelle auch durch eine textuelle Spezifikation näher erläutert werden müssten. Eine technische Spezifikation sollte bei allen Services die folgenden Aspekte näher beschreiben, damit diese in der Realisierungsphase leichter umzusetzen sind[52]:

- **Schnittstellenbeschreibung des Services**: Die Wahl des Transportprotokolls muss festgelegt werden. Hierbei lässt sich beispielsweise HTTP, SMTP, CORBA, etc. spezifizieren. Darüber hinaus muss natürlich auch der aufzurufende Endpunkt definiert werden. Im gängigen Fall eines Endpunktes, der mittels HTTP aufgerufen wird, kann auch der Port noch eine relevante Angabe darstellen. In den meisten Fällen ist dies der Port 80. Neben dem Protokoll ist das Format der Nachricht anzugeben. Hier lassen sich als Nachrichtenprotokoll beispielsweise SOAP oder auch eigene Formate verwenden. Ebenso ist die Nutzlast zu beschreiben (z. B. JSON, XML-Format, etc.). Bei Services, die nicht durch eine Standard-Schnittstellenbeschreibung definiert sind, müssen noch Ein- und Ausgabeparameter eines Services spezifiziert werden. Bei einer Web API

[52] Diese technischen Details der Service-Beschreibung wurden während der SLR-Festlegung bewusst nicht mit aufgenommen, da dort nur fachlich getriebene Informationen zulässig waren.

genügt dafür eine Liste mit Parameternamen, die dann für die Festlegung des Datenflusses verwendet werden kann.

- **Sicherheitstechnische Beschreibung**: Hier muss angegeben werden, inwieweit eine sichere Übertragung für den Service-Aufruf notwendig ist. Dafür lässt sich beispielsweise eine SSL-Verschlüsselung vorschreiben. Darüber hinaus sind auch weitere Stufen denkbar: beispielsweise eine Verschlüsselung von Nachrichtenteilen, eine Authentifizierung des Nutzers beim Service-Anbieter oder andere sicherheitsrelevante Angaben. Eine vollständige Aufzählung der hierbei verwendbaren Aspekte ist kaum möglich, da sich neben Standard-Mechanismen wie SSL auch diverse WS-Standards wie WS-Policy oder WS-Security verwenden lassen.

- **Muster des Service-Aufrufs**: Eine weitere technische Beschreibung betrifft die Art der Nachrichtenübermittlung beim Service-Aufruf. Hierbei lassen sich verschiedene Muster spezifizieren: Die gängigen Möglichkeiten der Kommunikation sind synchrone oder asynchrone Aufrufe. Darüber hinaus existieren diverse Spielarten, wie beispielsweise *Multiple-In-Rules* und unterschiedliche Warteschlangen-Verhalten.

Definition der Datenflüsse in Prozessen

Der zweite Schritt der Verfeinerung von Prozessen ist das Spezifizieren des Datenflusses. Hierbei soll vor allem geklärt werden, welche Daten innerhalb eines Prozesses vorhanden sein müssen und welche dieser Daten für einen Service-Aufruf verwendet werden. Ebenso werden die zurückgelieferten Daten spezifiziert und ggf. in ein lokales Objekt zurückgespeichert. Diese Angaben werden benötigt, damit in der Realisierungsphase die zu bearbeitenden Daten feststehen und die Services konfiguriert, programmiert bzw. korrekt mit Daten beschickt werden können. An diesem Punkt angelangt, muss man sich bewusst sein, dass die Datenflüsse zwischen Komponenten, hier in erster Linie XaaS-Dienste, einen wichtigen Teil eines verteilten Systems ausmachen. Werden bei den hier getroffenen Entwurfsentscheidungen einer WOA Fehler gemacht, kann dies leicht zu einem nicht zufriedenstellend arbeitenden System führen. Die zu betrachtenden Aspekte sind auf der einen Seite die Datenqualität und Datenintegrität, die ein gängiges Problem von Daten verarbeitenden

Systemen darstellen, und auf der anderen Seite die Datenlokalität, die vor allem bei einem verteilten System wie einer WOA von großer Bedeutung ist.

Datenqualität und -integrität Datenqualität ist eine grundlegende Problematik, die in Systemen auftritt, die sich mit der Speicherung, der Verarbeitung sowie dem Austausch von Daten befassen. Hohe Datenqualität liegt im Allgemeinen dann vor, wenn sich die vorliegenden Daten für den Verarbeitungszweck in einem System eignen (vgl. [Nau07]). Fehler, die zu einer niedrigen Datenqualität führen, können auf Schema- oder Datenebene auftreten (vgl. [RD00]). Auf Schemaebene lassen sich hier u. a. unzulässige Werte sowie eine Verletzung der Eindeutigkeit, der Attributabhängigkeit oder der referentiellen Integrität nennen. Auf Datenebene treten Fehler durch fehlende Werte, Schreibfehler, widersprüchliche Werte, Duplikate und diverse weitere Ursachen auf. Für die Sicherstellung von Datenqualität lassen sich beispielsweise *Extraktion, Transformation, Laden* (ETL)-Prozesse oder die Data-Fusion-Methode auf die Datenquellen anwenden (vgl. [Vos08, Tor03]).

Die Datenintegrität ist dann erfüllt, wenn vorgegebene Integritätsbedingungen eingehalten werden. Diese werden in einer relationalen Datenbank beispielsweise durch Primär- und Fremdschlüsseldefinitionen, Wertebereichsangaben oder einzelne Attributbedingungen repräsentiert (vgl. [Vos08]). Diese Bedingungen können entweder auf der Datenbankebene über das Transaktionskonzept oder auf der Applikationsebene durch geeignete programmtechnische Maßnahmen sichergestellt werden.

In Bezug auf die WOA wird hier von der Prämisse ausgegangen, dass die verwendeten XaaS-Dienste entweder eigene Datenmodelle zur Verfügung stellen und dann auch die für die Nutzung des Dienstes notwendigen Daten selbst speichern oder Schnittstellen anbieten, die Daten entgegennehmen, damit arbeiten und wieder zurückgeben. Die Datenintegrität ist daher von dem jeweiligen Anbieter selbst sicherzustellen, dessen API auch geeignete Fehlermeldungen liefern sollte, wenn die übergebenen Daten nicht den geforderten Integritätsbedingungen entsprechen. Eine Ausnahme bilden DaaS-Dienste, die innerhalb einer WOA für den Aufbau eigener Datenstrukturen verwendet werden. Hierbei obliegt es dem Entwickler, geeignete Prüfungen der Integrität und Qualität der Daten einzubauen. Bei der Verknüpfung mehrerer XaaS-Dienste in einem Geschäftsprozess, die beispielsweise alle eine konsistente Datenänderung vornehmen sollen, muss eine Transaktions-Logik auf Workflowebene

modelliert und in der Realisierungsphase umgesetzt werden. Die Datenqualität ist hierbei eine Herausforderung, die durch geeignete Eingabeprüfungen und Programmlogik behandelt werden sollte. Dies muss entweder von den XaaS-Anbietern vorgegeben sein oder in der Realisierungsphase der WOA durch die Entwickler umgesetzt werden. An dieser Stelle wird auf die Datenqualität sowie -integrität nicht weiter eingegangen, da diese nicht als WOA-spezifisches Problem angesehen werden, sondern sich generell in allen Daten verarbeitenden Systemen wiederfinden und gängige Lösungsstrategien, wie zuvor genannt, angewendet werden können.

Datenlokalität Der physikalische Ort, an dem Daten gespeichert und verarbeitet werden, ist ein spezifischer Aspekt einer WOA bzw. eines verteilten Systems. Ist die zuvor genannte Prämisse zutreffend, dass Daten dort gespeichert sind, wo sie verarbeitet werden sollen, besteht an dieser Stelle keine Veranlassung zur genaueren Betrachtung von Datenlokalität. Liegen aber Daten auf einem Server A vor und müssen zur Verarbeitung auf den Server B kopiert werden und ggf. anschließend wieder zurück, tritt je nach Art und Umfang der Daten ein sehr hohes Transfervolumen auf. Betrachtet man die WOA aus rein funktionaler Sicht, so wird innerhalb eines hier definierten Prozesses lediglich ein entfernter Dienst aufgerufen und mit den benötigten Daten beliefert. Die Betrachtung der benötigten Daten für eine Funktion sieht aber nicht so nüchtern aus: Geht man beispielsweise von einem Szenario zur Umsetzung eines Data Warehouse aus, in dem ein XaaS-Dienst alle Daten speichert und ein anderer die fachliche Logik zur Auswertung und Anzeige der Ergebnisse bietet, die Daten aber nicht selbst speichert, müssen viele Daten transferiert werden. Je nach Anwendungsfall können hierbei leicht Daten im Gigabyte-Bereich auftreten, die nicht mehr einfach per Funktionsaufruf über HTTP übertragen werden können. Hierbei sind vor allem die Transportkosten für die verwendete Bandbreite mit den möglichen Einsparungen durch die Verwendung eines XaaS-Angebots in Relation zu setzen. Darüber hinaus können auch technische Probleme auftreten, wie beispielsweise Verbindungsunterbrechungen oder einfach Mengenbeschränkungen der HTTP-Methoden, wenn die Übertragung von größeren Datenmengen über das Internet vollzogen werden soll. Zwar bieten hier Unternehmen wie Amazon mittlerweile Lösungen zur initialen Befüllung von Amazons Datenspeicher S3[53] an, indem man physikalische Fest-

[53] Siehe: `http://aws.amazon.com/de/importexport/`

platten an Amazon schickt. Eine solche Verfahrensweise ist aber mit Sicherheit nur eine Lösung für das erste Befüllen mit Daten.

Eine notwendige Maßnahme ist daher, dass man für jeden Serviceaufruf innerhalb eines hier verfeinerten Prozesses die nicht-funktionalen Anforderungen überprüft, in denen die Belastbarkeit bzw. Nutzung eines Services beschrieben wurde. Sollte sich hier ergeben, dass große Datenmengen zu transferieren sind, muss zunächst überprüft werden, ob diese Datenversendung zwischen verschiedenen Anbietern stattfindet. Sollte es sich hierbei um den Aufruf zwischen zwei Services desselben XaaS-Anbieters handeln, müssen die Daten ggf. nicht kopiert werden, so dass eine Verwendung zu vertreten ist. Sind unterschiedliche Anbieter beteiligt und sind hohe Datenvolumen von einem Server zu einem anderen zu kopieren, so muss eine Alternative gefunden werden, bei der die Daten dort verarbeitet werden, wo diese auch gespeichert sind. Da in der zuvor durchgeführten Phase der XaaS-Anbieterbestimmung die Zusammenhänge von Service-Aufrufen durch die noch nicht verfeinerten Geschäftsprozesse noch unklar waren, lässt sich erst zu diesem Zeitpunkt eine konkrete Problematik erkennen. Daher ist hier ggf. der Rücksprung in die vorherige Phase notwendig, um einen alternativen Dienstleister zu finden.

Es lässt sich hierbei die ganz klare Handlungsempfehlung geben, dass Services, bei denen große Datenmengen zwischen verschiedenen physikalischen Servern zu übertragen sind, generell keine geeigneten Kandidaten für die Verwendung in einer WOA sind. Dennoch lassen sich diese manchmal technisch umsetzen.

Definition des Datenflusses Durch die funktionale Beschreibung mittels WSDL- oder WADL-Dokument stehen die für einen Service benötigten Eingabe- sowie Ausgabeparameter eindeutig fest und zur Verfügung stehende Daten können für jedes Datenobjekt dargestellt werden. Darüber hinaus stehen zusätzlich Daten zu nicht-funktionalen Aspekten ebenfalls fest, die die Beurteilung des Datenaufkommens für die Verwendung eines Services erlauben. Ein Datenfluss innerhalb eines BPMN-Prozesses wird nun, den folgenden Regeln entsprechend, angegeben:

- Jeder initial ausgehende Nachrichtenversand (*Message Flow*) zu einer Aktivität in einem anderen Pool stellt eine Nachricht dar, die durch eine Datenflussangabe spezifiziert werden muss. Dies gilt nur für ausgehen-

de Nachrichten aus einem Pool, der unter eigener Kontrolle steht, also beispielsweise im Pool „Logik".

- Die Angaben zur Belastung und Nutzung des Services, der bei einem Message Flow den Ausgangspunkt des Datenversands darstellt, werden an dieser Stelle überprüft. Diese Daten lassen sich aus den nichtfunktionalen Eigenschaften des jeweiligen Services ablesen. Sind hierbei die Datenmengen so groß, dass sich ein Datenversand als zu teuer oder fehleranfällig herausstellen sollte, muss dieser Prozess überarbeitet werden, so dass ein Modell entsteht, bei dem die Daten dort gespeichert sind, wo sie verarbeitet werden sollen. Dann wird hier die Datenflussspezifizierung beendet, der Prozess für die Überarbeitung gekennzeichnet und der Nächste bearbeitet.

- Ein Datenflusselement wird innerhalb des Pools definiert, aus dem die Nachricht hervorgeht. Es hat hierbei ausschließlich Zugriff auf die Datenobjekte **innerhalb** des eigenen Pools.

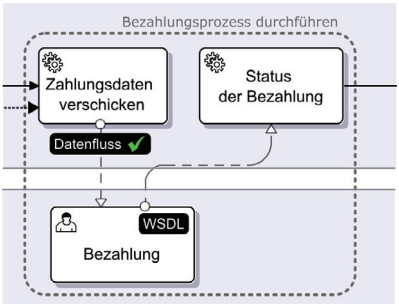
- Die Antwort eines Service-Aufrufs steht der Aktivität zur Verfügung, die den Nachrichtenfluss entgegen nimmt.

Um diese Anforderungen in den BPMN-Prozessen umzusetzen, muss das Nachrichtenfluss-Element um zusätzliche Informationen erweitert werden. Obwohl die Methode soweit wie möglich technologieunabhängig bleiben soll, wird an dieser Stelle ein spezifisches Konstrukt verwendet, welches sich aus der Workflowsprache BPEL ableitet. Durch die Verwendung von Standard-BPEL kann aber eine eigene Definition der Regel vermieden werden. Es wird hierbei für die Zuweisung von Daten zu einer Aktivität das XML-Konstrukt `<assign>` verwendet, welches Quelle und Ziel von Daten spezifiziert, die innerhalb eines Workflows verwendet werden (vgl. [OAS07]). Im Gegensatz zu anderen Sprachen, wie bspw. *Web Service Flow Language*, die auf WSDL aufbaut (vgl. [Ley01]), weist BPEL einen relativ geringen Umfang auf. In Listing 5.2 wird für einen Datenfluss spezifiziert, dass die „customerID" und der zu zahlende Betrag „amount" aus dem Datenobjekt „Session Data" an den Service „Bezahlung" und dort den Eingabeparameter „id" und „amount" gesendet wird. Das *Tag* `from` verweist auf ein Datenobjekt des Pools, in dem der Nachrichtenfluss beginnt. Das *Tag* `to` verweist auf einen Service in einem anderen Pool.

```
<assign>
   <copy>
      <from container="Session Data" part="customerId"/>
      <to container="Bezahlung" part="id"/>
   </copy>
   <copy>
      <from container="Session Data" part="amount"/>
      <to container="Bezahlung" part="amount"/>
   </copy>
   <copy>
      ...
   </copy>
</assign>
```

Listing 5.2: Beschreibung des Datenflusses.

Die grafische Darstellung von funktionalen Beschreibungen sowie der Datenflüsse innerhalb eines Prozesses ist in Abbildung 5.27 beispielhaft als Ausschnitt aus dem zuvor gezeigten Prozess durch die Erweiterung mit eigenen Symbolen gezeigt.

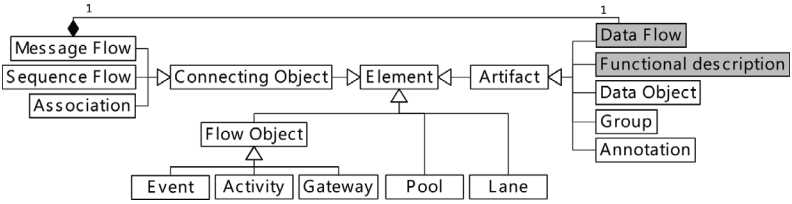


Quelle: [TV09b]

Abbildung 5.27: Vollständig verfeinerter Prozess.

Der aktuelle Stand der BPMN sieht weder eine Erweiterung der Aufgaben um eine funktionale Beschreibung noch eine Spezifizierung des Nachrichtenflusses durch eine grafische Notation vor. Das Modell an sich ist aber durch zusätzliche Artefakte erweiterbar, so dass prototypisch die Elemente Datenfluss (*Data Flow*) und funktionale Beschreibung (*Functional Description*) zum

Objekt-Modell hinzugefügt werden können. Als zusätzliche Spezifikation darf ein Datenflussartefakt nur an einen Nachrichtenfluss angehängt werden. In Abbildung 5.28 sind die hierfür notwendigen Erweiterungen am Metadatenmodell der BPMN dunkelgrau hervorgehoben.



Quelle: [TV09b]

Abbildung 5.28: Erweiterung von BPMN durch eigene Artefakte.

Planung der System-Architektur

Um nach der Verfeinerung der Geschäftsprozessmodelle nun eine konkrete Netzwerktopologie zu erstellen, müssen die folgenden Fragen beantwortet werden:

- **Welche** Komponenten (Services) der Prozesse werden
- **wo** (Ort) implementiert?

Anhand der Prozessdokumentation und der im vorherigen Schritt verfeinerten Prozessmodelle sind viele Entscheidungen bereits vorweggenommen. Je nachdem, wie die Entscheidung über das Outsourcing des Unternehmens ausfällt, müssen einige Prozesse innerhalb und andere außerhalb des Unternehmens umgesetzt werden. Diverse identifizierte Services, wie bspw. SaaS- und PaaS-Anbieter sind per Definition bereits im Internet zu realisieren. In der jetzt anstehenden Planung werden daher die verschiedenen Pools aus den Prozessmodellen betrachtet und entschieden, welche konkrete Umsetzung erforderlich ist. Um die Realisierung einer WOA hierbei optimal vorzubereiten, wird nun für jeden BPMN-Prozess der untersten Ebene eine Architekturskizze angefertigt, die zwei Aspekte zum Ausdruck bringen soll:

1. **Umsetzung der fachlichen Inhalte des Prozesses**: Konkrete Aussagen über einzusetzende Services und Dienstleister sowie die Topologie der Lösung.

2. **Einbau von Steuerungs- und Überwachungselementen**: Hierbei soll geklärt werden, welche Überwachungswerkzeuge zum Einsatz kommen.

Am Beispiel des in Abbildung 5.25 dargestellten Prozesses „Online-Bezahlung" wird nun das Verfahren zum Erstellen von Architekturskizzen erläutert. Durch die bereits in der Verfeinerung erstellten Pools innerhalb eines Prozesses, und der dazugehörigen Prozessdokumentation, sind die einzubindenden Services vollständig beschrieben. Jeder Service (aus einem Pool) wird daher als eigenständige Komponente in die Architektur übernommen. Der Logik-Pool wird in Form einer Workflow-Engine in eine Server-Komponente im Internet oder im Unternehmen (je nach Spezifizierung des Outsourcings) eingeordnet. Anschließend werden die einzelnen Verbindungen gekennzeichnet. Die Architekturskizzen beinhalten keine Logik, sondern zeigen nur den Aufbau der einzelnen IT-Komponenten für einen Prozess. In diesem Beispiel wurde ein UML-Komponentendiagramm für die Topologie-Darstellung verwendet.

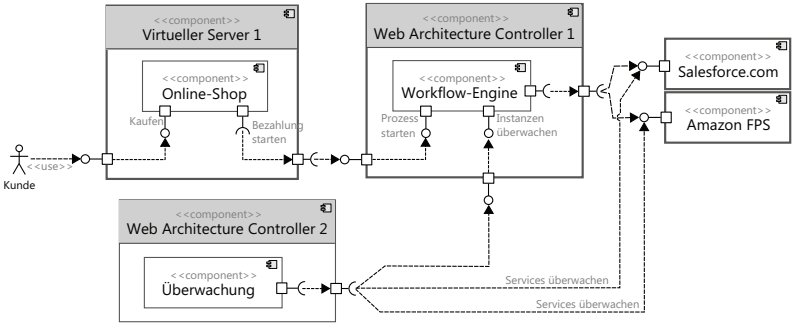


Abbildung 5.29: Architekturskizze für den „Online-Bestell"-Prozess.

Das für diesen Geschäftsprozess erstellte Modell besteht aus einem virtuellen Server, der den Online-Shop enthalten soll. Darüber hinaus werden zwei WAC geplant. Der eine dient hierbei als Workflow-Engine (also die Logik-

Komponente) und bindet die beiden externen Services (Amazon FPS und salesforce.com) ein. Der andere dient der Kontrolle des Logik-WAC sowie der angeschlossenen XaaS-Anbieter (siehe Abbildung 5.29). Dabei wird die Verfügbarkeit der Services und der Workflow-Engine überwacht und protokolliert sowie beim Auftreten von Fehlern eine entsprechende Benachrichtigung an die WOA-Betreiber veranlasst. Zum Abschluss dieses Arbeitsschrittes existieren zum einen vollständig verfeinerte Geschäftsprozessmodelle mit technisch beschriebenen Services, zum anderen Topologie-Modelle, die beschreiben, welche Services auf welchem physikalischen bzw. virtuellen Server oder XaaS-Anbieter realisiert werden sollen.

Die letzte Aktivität in der Topologieplanung besteht aus der konkreten Auswahl von XaaS-Anbietern für die Realisierung von virtuellen Servern, Controllern, etc., die gerade zuvor identifiziert wurden. Es wird dabei ein ähnlicher Vorgang durchlaufen, wie bei der Auswahl von XaaS-Anbietern in Kapitel 5.3.4: Zuerst werden passende XaaS-Anbieter gesucht, um mit diesen anschließend SLAs auszuhandeln. Je nach Größe des Unternehmens und des umzusetzenden IT-Systems ist dieser Schritt aber relativ schnell durchführbar, da nur wenige Anbieter ausgewählt werden müssen. Möglicherweise genügt hierbei bereits ein einzelner XaaS-Anbieter, der von Workflow-Realisierung bis hin zu Programmierung eigener Logik eine ausreichende Funktionalität anbietet. Die Wahl sollte allerdings auch nicht leichtfertig getroffen werden, da die eingesetzten Komponenten für den oder die Controller innerhalb der WOA eine zentrale Rolle spielen und mit für die Stabilität und Wartbarkeit der gesamten WOA verantwortlich sind. PaaS-Anbieter, wie beispielsweise *Force.com*, *Intalio*, *LongJump* oder *bungeeConnect* bieten sehr umfassende XaaS-Lösungen für das Erstellen Cloud-basierter Systeme. Bei der Realisierung und Umsetzung von Workflows ermöglichen einige das Modellieren von BPMN-Prozessen, andere wiederum bieten eine eigene (meist vereinfachte) Workflows-Sprache an. Interessant ist, dass sich kaum ein Anbieter findet, der es ermöglicht, BPEL als Prozessbeschreibung zu verwenden. Dies ist vermutlich zum einen dem Fehlen einer festgelegten grafischen Darstellung von BPEL und zum anderen dem Streben nach möglichst einfacher Erstellung von Prozessen – ganz im Sinne des Web 2.0 – zu verdanken. Die Entscheidung für oder gegen einen entsprechenden XaaS-Anbieter lässt sich generell schwer systematisieren, dafür sind die Angebote zu heterogen. Es lässt sich aber grundlegend feststellen, dass Aspekte wie Verfügbarkeit der Dienste, ein guter Kundensupport und nicht zuletzt vertraglich festgelegte SLAs im Fokus stehen.

Planung der Sicherheitsstruktur

Je größer ein Unternehmen ist, desto mehr Personen und organisatorische sowie systembedingte Rollen beeinflussen ein umzusetzendes IT-System. Nicht jede Person besitzt hierbei dieselben Rechte. Die für die Planung einer Sicherheitsstruktur notwendigen Informationen umfassen die identifizierten Rollen aus vorherigen Phasen, die Beschreibungen der Datenobjekte in den Services, die Datenverantwortlichen sowie die sicherheitsrelevanten, technischen Beschreibungen der konkreten Services. Die identifizierten Rollen lassen sich in menschliche und technische Akteure einteilen. Beispiele für menschliche Rollen sind: „Mitarbeiter Einkauf", „Mitarbeiter Marketing" oder „Abteilungsleiter Einkauf". Daneben sind technische Rollen als Platzhalter für existierende Systeme oder weitere Service-Komponenten zu sehen: „Server XY", „WAC Nr. 2" oder „ERP-System". Für jede Rolle lassen sich darüber hinaus noch verschiedene Arten des Zugriffsrechts unterscheiden. Dies können zum einen lesende oder schreibende Informationsrechte auf Daten sein, sowie Ausführungsrechte für Prozesse, um beispielsweise Abläufe starten oder stoppen zu können.

Die folgenden Schritte werden nun zunächst für die Spezifikation der notwendigen Rechtestruktur in einer WOA ausgeführt:

- **Identifikation eines Objekts**: Für jeden Geschäftsprozess auf unterster Ebene wird überprüft, ob sicherheitsrelevante Angaben zu spezifizieren sind. Zunächst lassen sich hierbei Datenobjekte betrachten, die in den Geschäftsprozessen durch die Datenflussspezifikation explizit dargestellt sind. Darüber hinaus können alle Services, die in einem WAC oder auf internen Systemen ausgeführt werden, überprüft werden. Eine weitere Quelle für potenzielle Sicherheitsspezifikationen sind die Schnittstellenaufrufe, die durch die Architekturskizzen der Prozesse ersichtlich werden. Bei jedem Aufruf einer Komponente sind ggf. Rechte zu definieren.

- **Überprüfen der Notwendigkeit zur Reglementierung**: Dazu werden nun im Falle von Datenobjekten die Datenverantwortlichen befragt, ob für den Zugriff auf das Datenobjekt Zugriffsrechte zu spezifizieren sind. Handelt es sich um Services, werden dagegen die Ausführungsrechte definiert. Diese können entweder technisch oder organisatorisch bedingt sein. Des Weiteren können auch ganze Workflows mit Ausführungsrechten versehen werden.

Tabelle 5.7: Rollen-Rechte-Zuordnung für den Beispielprozess.

Rolle	Objekt	Recht
P:Kunde	COMP:Online-Shop	Vollständig
T:Online-Shop	DATA:Session-Daten API:WAC1:Workflow-Engine:payment	Lesen/Schreiben Start/Stop
T:WAC1	DATA:Kundendaten API:Salesforce.com API:amazonFPS	Lesen Verwenden Aufrufen
T:WAC2	API:Salesforce.com API:amazonFPS API:WAC1:Workflow-Engine:monitoring	Ping Ping Vollständig

- **Festlegung der Rechte**: Für alle Datenobjekte, die nicht allgemein zugreifbar sind, werden nun Lese- und Schreibrechte festgelegt. Bei Services oder ganzen Workflows werden die Ausführungsrechte (starten, stoppen, überwachen, etc.) näher definiert. Die Schnittstellenaufrufe werden benannt und ebenfalls spezifiziert. In diesem Schritt entsteht eine Liste, die eine Rolle mit einem Objekt (Datenobjekt, Service, Workflow, Schnittstellenaufruf) verknüpft und die dafür zulässigen Rechte benennt.

Für die Rechte des Beispielprozesses aus Abbildung 5.26 unter Berücksichtigung der Architekturskizze aus Abbildung 5.29 lässt sich eine beispielhafte Rechtestruktur definieren, die in Tabelle 5.7 zusammengefasst ist. Darin werden die Abkürzungen *P* für menschliche und *T* für technische Rollen, *COMP* für Komponenten, *DATA* für Datenobjekte sowie *API* für Schnittstellenaufrufe verwendet. So wird beispielsweise dem menschlichen Akteur „Kunde" auf die Komponente „Online-Shop" ein vollständiger Zugriff zugestanden, der technischen Rolle „WAC1" dagegen auf das Datenobjekt „Kundendaten" nur ein lesender Zugriff.

Die aus der Festlegung der Rechte resultierenden technischen Anforderungen werden während der Realisierungsphase einer WOA umgesetzt. Falls dort

beispielsweise diverse Rollen auf ein System zugreifen dürfen, und die Verbindung durch ein Zertifikat verschlüsselt werden soll, so muss eine PKI aufgebaut werden. Die Struktur lässt sich dann anhand der Service-Beschreibungen sowie der Rollen-und-Rechte-Spezifikation aufbauen.

5.3.6 Notfall- und Alternativenplanung

Dass sich im Grunde kein IT-System fehlerfrei nennen kann, ist nichts Ungewöhnliches. Je komplexer ein Programm oder ein System ist, desto höher ist auch die Wahrscheinlichkeit für das Auftreten eines Fehlers. Softwarefehler treten dabei in diversen Ausprägungen auf, die von der Art der Software oder des Systems und dem jeweiligen Einsatzzweck abhängen. Zum einen sind während der Planung und Erstellung eines Systems auftretende Fehler zu beachten, wie beispielsweise logische Fehler, Syntax- und Designfehler. Durch geeignete Testverfahren während der Realisierungsphase sollten diese aber weitestgehend reduziert werden. Dies wird in Kapitel 5.4 näher erläutert. Dabei lässt sich mit geeigneten Testmethoden ein System verbessern und auf Fehler prüfen. Aber auch damit kann bei produktiven IT-Systemen das Auftreten von Fehlern nicht zu 100 % ausgeschlossen werden. Zum anderen sind vor allem Laufzeitfehler zu beachten, die erst während des Betriebs eines Systems auftreten. Gerade im Kontext eines verteilten Systems wie einer WOA hängt der reibungslose Betrieb von der ordnungsgemäßen Funktion der eingebundenen XaaS-Dienste ab.

Für die Notfall- und Alternativenplanung geht man zunächst von der Annahme eines fehlerfrei programmierten Systems aus. Die relevanten Fehlerquellen im Betrieb einer WOA lassen sich so auf die eingebundenen Anbieter und die zwischen diesen eingerichteten bidirektionalen Verbindungen eingrenzen. Das Ziel dieser Phase ist das Erarbeiten eines Notfallplans, der genau beschreibt, was im Falle von fehlerhaft arbeitenden oder vollständig ausgefallenen Diensten geschehen soll. Damit sollen die Ausfallzeiten des IT-Systems möglichst gering gehalten werden. Im Idealfall werden Fehler des Betriebs durch geeignete Werkzeuge automatisch erkannt und Maßnahmen sofort automatisch oder manuell eingeleitet. Wie sich die im Betrieb einer WOA verwendeten XaaS-Services überwachen lassen, wird in Kapitel 5.5.1 erläutert. Durch den Namen der Phase wird bereits angedeutet, dass zum einen ein Notfall skizziert wird und zum anderen Alternativen für möglicherweise ausfallende Systemteile geplant werden. Als Grundlage der Planung dient hier der zuvor

erstellte Service-Katalog, aus dem alle verwendeten XaaS-Anbieter hervorgehen. Mit Abschluss der Phase steht ein Leitfaden für den Notfall bereit, der Auskunft über die auszuführenden Maßnahmen im Falle eines auftretenden Fehlers gibt (siehe Abbildung 5.30). In dieser Phase werden Domänenspezialisten nur anfänglich für die Relevanzbestimmung von einzelnen Workflows und Systemteilen benötigt, da diese das fachliche Know-How besitzen, um eine Einschätzung der Wichtigkeit eines Geschäftsprozesses abgeben zu können. Die konkreten Planungen für den Notfall oder das Suchen nach Alternativen führen dann IT-Architekten und ggf. Entwickler durch.

Abbildung 5.30: Die Schritte der Notfall- und Alternativenplanung.

Bewertung von Relevanz und Ausfallrisiko eines XaaS-Dienstes

Im ersten Schritt dieser Phase werden die bereits zuvor ausgewählten und beschriebenen XaaS-Dienste auf ihre Relevanz in Bezug auf den Geschäftszweck des Unternehmens und deren Ausfallrisiko hin bewertet. Dabei fallen verschiedene Merkmale des Anbieters sowie die Art der Verwendung innerhalb der WOA ins Gewicht. Zunächst muss bewertet werden, wie relevant sich die Nutzung des Dienstes für den Geschäftszweck des Unternehmens darstellt. So ist beispielsweise ein Online-Bezahlungs-Dienst für Anbieter einer E-Commerce-Plattform von zentraler Bedeutung und muss rund um die Uhr funktionieren, während ein Dokumentenmanagement-Werkzeug auch bei einem Ausfall keinen sofortigen Effekt auf den Umsatz des Unternehmens hat. Dieselbe SaaS-Lösung kann allerdings bei einem Callcenter, das relevante Beratungsdokumentation speichert, sehr wohl ein zentrales Element des IT-Systems sein. Die Bewertung der Relevanz hängt somit stark vom individuellen Einsatzzweck eines Dienstes im IT-System des Unternehmens ab, so dass sich an dieser Stelle

keine allgemeingültigen Aussagen treffen lassen. Um die Relevanz zu bestimmen, lassen sich neben dem Service-Katalog, der die Verwendung von XaaS-Diensten in Prozessen darlegt, auch weitere Dokumentationen heranziehen, wie beispielsweise die Kurzbeschreibungen der Service. Auch die Meinung von Domänenspezialisten kann zur Bewertung eingeholt werden. Jeder verwendete XaaS-Dienst wird somit einzeln betrachtet und in eine Rangfolge gebracht, in der die wichtigsten Dienste an oberster Stelle eingeordnet werden.

Nach der Betrachtung der Relevanz eines Dienstes für das Unternehmen lässt sich der Anbieter auf sein potenzielles Ausfallrisiko hin untersuchen. Oft wird bereits durch diesen eine hohe Verfügbarkeit gewährleistet, die durch die vereinbarten SLAs vertraglich vereinbart wurde. Je nach Größe und Qualität des Anbieters und vor allem Relevanz des Dienstes für das Unternehmen muss aber eine dauerhafte Verfügbarkeit ohne jegliche Ausfälle angestrebt werden. In diesem Schritt wird daher bewertet, inwiefern der gewählte Anbieter für einen umzusetzenden Service als ausreichend angesehen wird, oder ob hierbei ebenfalls eine Notfallmaßnahme oder Alternative konzipiert werden muss.

Zum Abschluss der Bewertung existiert eine Liste mit XaaS-Angeboten, die nach Relevanz für das Erreichen des Geschäftsziels geordnet sind. Für jedes Angebot wird dabei vermerkt, ob eine Notfall- oder Alternativenplanung sinnvoll und gewünscht ist. Mit dieser Liste wird im nächsten Schritt fortgefahren.

Planung einer Alternative oder Notfallmaßnahme

Die Liste aus dem vorangegangenen Schritt muss nun abgearbeitet und für jeden als relevant erachteten XaaS-Dienst eine Möglichkeit gefunden werden, einen Ausfall des Dienstes zu kompensieren. Zunächst wird geprüft, ob es Alternativen zu den bereits ausgewählten XaaS-Anbietern gibt, die bei einem Ausfall verwendet werden können. Im Idealfall lassen sich hier Angebote finden, die sich mit wenig Aufwand oder Anpassung während des Betriebs einer WOA einfach umschalten lassen. Als Beispiel sei hier die Zahlungskomponente des E-Shops von *JKHL Enterprises* angeführt, der *Amazon FPS* verwendet. Sollte Amazons Dienst ausfallen, so ließe sich beispielsweise *PayPal* als Alternative verwenden. Ebenso ist die Speicherung von Daten unter *SimpleDB* durch eine Alternative wie *Zoho Creator* ersetzbar. Es muss hierbei im Vorfeld geklärt werden, wie die Nutzungsmodelle der Anbieter aussehen, und welche Folgen die Nutzung eines Alternativanbieters im Einzelnen hat: beispielsweise den Auf-

wand der Datensynchronisation im Falle des Wechsels eines DaaS-Anbieters. Sollte man lediglich für den genutzten Speicher und das Transfervolumen zahlen, ist der kurzfristige Einsatz eines anderen Speicherdienstes meist akzeptabel. Bei monatlichen Kosten, die nur für das Bereitstellen einer Notfallstrategie anfallen, müssen diese gegen das Risiko eines Ausfalls abgewägt werden. Wurde ein alternativer XaaS-Anbieter gefunden, so lassen sich für den potenziellen Fehlerfall bereits Redundanzen auf der Ebene der Workflows einarbeiten. Ein Workflow sollte daher die Verfügbarkeit des primär eingesetzten Services abfragen und bei Auftreten eines Fehlers direkt den sekundären Service verwenden. Zusätzlich muss natürlich dann eine Benachrichtigung an die Systemverantwortlichen geschickt werden. Dies kommt einer vollautomatischen Notfallmaßnahme gleich, die ohne manuellen Eingriff funktioniert. Das Konzept von redundanten Systemteilen in Systemarchitekturen, egal ob Datenbanken, Servern oder weiteren operativen Komponenten, ist für komplexe, ausfallsichere Systeme eine bekannte Strategie, die sich hier auf Workflow-Ebene realisieren lässt.

Da in einer archetypischen WOA alle Systemkomponenten ausgelagert werden, lassen sich auch virtuelle Server oder PaaS-Anbieter in diese Überlegung einbeziehen. Diese Komponenten werden im Service-Katalog nicht aufgeführt, da sie nicht für fachliche Abläufe in Prozessen spezifiziert wurden, sondern später innerhalb der Topologieplanung identifiziert worden sind. Hierbei lassen sich, je nach gewünschter Belastbarkeit bzw. Ausfallsicherheit des Systems, ebenfalls mehrere Alternativen auswählen, um das Risiko eines Ausfalls zu minimieren. Dies gilt insbesondere für die Platzierung von WACs, die auf einem PaaS oder einem eigenen virtuellen Server ausgelagert sind. Wenn eine solch zentrale Komponente ausfällt, werden alle anderen Maßnahmen der Alternativenplanung schnell zunichtegemacht. Daher sollte man sich auch hier entsprechend durch ein redundantes System absichern. In Abbildung 5.31 ist die Topologieskizze aus Abbildung 5.29 durch einen redundanten Zahlungsdienst (hier: *PayPal*) erweitert worden.

Für den Fall, dass keine alternativen Anbieter für einen speziellen Service gefunden werden können, oder das Ausweichen auf einen anderen Anbieter mit zu viel Aufwand verbunden ist, müssen Notfallmaßnahmen festgelegt werden. Diese sollen dann den Ablauf von ggf. manuellen Workflows regeln und den Schaden für das Unternehmen möglichst gering halten. Hierbei lassen sich Störungen für die Mitarbeiter und für die Kunden eines Unternehmens un-

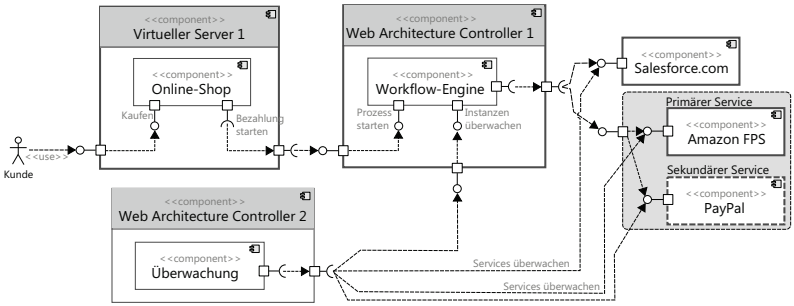


Abbildung 5.31: Architekturskizze mit redundant ausgelegtem Zahlungsdienst.

terscheiden, die differenziert zu behandeln sind. Während zeitlich begrenzte Service-Ausfälle auf Mitarbeiterseite in erster Linie störend sind und ggf. die Produktivität der Mitarbeiter einschränkt, können sich Störungen auf Kundenseite sehr geschäftsschädigend auswirken.

Der Ausfall eines CRM-Systems wie *Salesforce.com* bei der *BBI GmbH* ist ein Beispiel für Störungen des Systems für den Mitarbeiter. Durch die Menge an Stammdaten und zusätzlichen Informationen über Kunden, Produkte und Kontakte ist ein kurzfristiges Umschalten auf ein anderes System zwar möglich, jedoch sehr aufwendig.[54] Zudem hat der Ausfall des CRM-Systems in diesem Szenario keine unmittelbare Schädigung des Unternehmens zur Folge, da man für kurze Zeit auf dieses operative System verzichten kann. Es ist hierbei also festzuhalten, welche Maßnahmen im Fehlerfall auszuführen sind: beispielsweise das Speichern von vorzunehmenden Änderungen an den Stammdaten in einem im Internet angelegten Wiki, welches normalerweise dem Know-How-Transfer der Mitarbeiter dient, um später die Daten in das CRM-System nachzutragen. Bei einem kleinen Unternehmen wie der *BBI GmbH* wäre dies ein vertretbarer Aufwand. Zusätzlich lassen sich bei Web-basierten Frontends durch technische Lösungen kurzzeitige Workarounds realisieren. So können bei JavaScript-basierten Web-Applikationen Daten zunächst im lokalen Speicher abgelegt werden (durch Lösungen moderner Browser-Technologie wie

[54] Dazu müssten alle Daten von Salesforce.com in regelmäßigen Abständen zu einem Alternativanbieter transferiert werden.

HTML5 oder Google Gears), um diese bei erneuter Verfügbarkeit des XaaS-Anbieters zu synchronisieren. Dies ist auch eine mögliche Vorgehensweise bei einer Störung der Netzwerkverbindung des Unternehmens zum Internet. Doch müssen diese Maßnahmen in der Software des Anbieters bereits vorgesehen sein. Eine eigene Implementierung lässt sich nur schwer nachträglich hinzufügen.

Mögliche Störungen für den Kunden eines Anbieters, wie der Ausfall von Zahlungsdiensten eines E-Shops oder eines virtuellen Servers, müssen sehr sorgfältig betrachtet werden. Lassen sich hierbei keine automatischen Fallback-Mechanismen realisieren, wie oben beschrieben, so müssen Maßnahmen konzipiert werden, die auf Kundenseite möglichst wenig auffallen. Bietet man beispielsweise einen E-Shop mit Produkten eines hart umkämpften Marktes an, in dem sich viele Mitbewerber befinden, so kann die Störung einer zentralen Komponente schnell zu einer Kundenabwanderung führen. Es gilt, hierfür entsprechende Maßnahmen zu konzipieren. Möglichkeiten der Kompensation gibt es viele, ein Beispiel soll dies verdeutlichen: Sollte ein Kunde durch den Ausfall einer Systemkomponente im Online-Shop nicht zur Kasse gelangen, so könnte man sich dafür förmlich entschuldigen, den aktuellen Warenkorb abspeichern und den Kunden bitten, etwas später den Kauf abzuschließen. Darüber hinaus ließe sich noch ein Einkaufsgutschein als Wiedergutmachung hinzubuchen, um den Kunden milde zu stimmen und nicht zu verlieren.

Das Einleiten von Notfallmaßnahmen oder die Verwendung alternativer XaaS-Anbieter setzt allerdings das Erkennen von Fehlern voraus. Dies klingt zwar trivial, ist es unter Umständen aber nicht. Im Falle der Nutzung von SaaS-Angeboten durch Mitarbeiter ist ein auftretender Fehler schnell erkannt, und der Mitarbeiter kann die entsprechenden Maßnahmen einleiten. Bei automatisch ablaufenden Workflows, vor allem bei solchen, die durch den Kunden oder sogar durch IT-Systeme ausgelöst werden, ist das Bemerken von Fehlern durch das Unternehmen etwas schwieriger. Hierbei helfen nur das Überwachen von Services im Betrieb und das Erstellen von Metriken, die durch die SLAs vorgegeben sind. Mit der Überwachung von Services einer WOA im Betrieb befasst sich Kapitel 5.5.1.

5.4 Realisierungsphase

Mit dem Abschluss aller Schritte der Analyse- und Planungsphase beginnt die zweite Phase in der WOA-Methode. Die zuvor erstellten Konzepte und Geschäftsprozesse werden in diesem Schritt realisiert und getestet. Hierbei werden bestehende XaaS-Anbieter miteinander verknüpft, Services orchestriert, ggf. eigene Services programmiert und Controller eingesetzt, um ein funktionierendes IT-System zu gestalten.

5.4.1 Realisierung und Test

Um eine WOA umzusetzen, müssen zunächst die modellierten Geschäftsprozesse in automatisch ablauffähige Workflows umgewandelt werden. In welcher Form und mit welchen Werkzeugen dies geschieht, hängt stark davon ab, welchen PaaS-Anbieter man für die Umsetzung der Controller in einer WOA einsetzen möchte. Die Wahl eines Anbieters fand in der Topologieplanung statt und wirkt sich auf diesen Schritt aus, da die technische Realisierung darauf abgestimmt erfolgen muss. Anschließend wird damit begonnen, eigene Logik zu implementieren und ggf. auch eigene Service-Implementierungen vorzunehmen. Im Idealfall bietet der XaaS-Anbieter hierbei eine Möglichkeit, solche Komponenten direkt auf der Plattform zu entwickeln und auszuführen und ggf. mit den einzelnen Aktivitäten des Workflows zu verbinden. Sollten Services im eigenen Unternehmen zu erstellen sein, beispielsweise für die Anbindung von Altsystemen, so werden diese implementiert und ebenfalls an entsprechenden Stellen in den Workflow eingebunden. Auf diese Implementierungsdetails wird nicht weiter eingegangen. Es wird an dieser Stelle aber die Annahme getroffen, dass ein selbst implementierter Service den Vorstellungen einer WOA genügt und daher durch eine WP aufrufbar ist. Der dritte Schritt besteht im Testen der Workflows. Dabei werden diese einzeln betrachtet und geprüft, ob der erwartete Effekt durch das Ausführen eintritt und keine Fehler zu entdecken sind bzw. der Service auf Fehlersituation wie vereinbart reagiert. Diese drei Schritte werden nun für jeden modellierten Geschäftsprozess wiederholt durchgeführt. Nach Abschluss dieser Phase ist das IT-System im Kern fertig realisiert und bereit für den Systemtest (siehe Abbildung 5.32). Die Erläuterung zu den einzelnen Schritten dieser Phase und deren Ausgestaltung werden an dieser Stelle aus zwei Gründen sehr kurz gehalten. Zum einen ist die Vielfalt an möglichen PaaS-Anbietern groß und dadurch die Umsetzungs-

details zu heterogen, um diese an dieser Stelle sinnvoll zu beschreiben. Zum anderen wird der Ablauf einer WOA-Realisierung in Kapitel 5.7 noch etwas detaillierter durch die Verwendung des agilen Vorgehensmodells FDD erläutert.

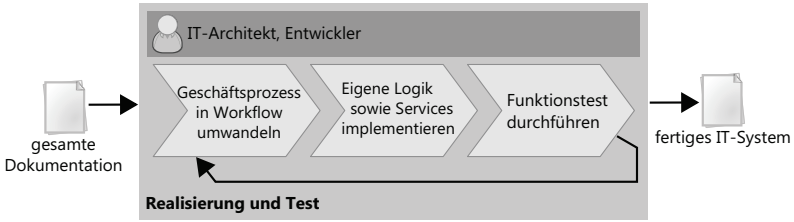


Abbildung 5.32: Die Schritte der Realisierungs- und Testphase.

Die hier vorgeschlagenen Schritte der Umsetzung eines Prozesses, der Implementierung der notwendigen Logik sowie des Testens des Workflows für jeden einzelnen Geschäftsprozess ähnelt dem Vorgehen von IBM bei der Einbettung von IT-Komponenten in eine SOA-Umgebung (vgl. [MBQM08]).

Geschäftsprozess in Workflow umwandeln

Die vorliegenden Geschäftsprozesse müssen nun in das jeweils notwendige Format für die Verwendung in der Workflow-Engine des Controllers umgewandelt werden. Hierbei gibt es diverse Möglichkeiten, von denen einige im Folgenden angesprochen werden. Die einfachste Möglichkeit besteht darin, dass ein PaaS-Anbieter gewählt wurde, dessen Ausführungsschicht BPMN-Prozesse direkt ausführen kann (beispielsweise *Intellio PaaS*). In diesem Fall werden die Geschäftsprozessmodelle im Grunde nur zum Anbieter übertragen und spezifische Angaben, wie Service-Aufrufe und Datenflussangaben, in der jeweils vom Anbieter vorgesehenen Form nachgetragen bzw. spezifiziert. Diese Variante erlaubt die einfachste Form der Prozessübernahme in eine Workflow-Engine. Eine zweite Möglichkeit besteht in der Verwendung von BPEL zur Prozessbeschreibung. Hierbei müssen die in BPMN definierten Prozesse in BPEL umgewandelt werden, um in der Workflow-Engine ablauffähig zu sein. Eine Vorgehensweise für diese Umwandlung ist beschrieben in [OADH06]. Da BPEL ursprünglich für die reine Verwendung von Web Services

konzipiert wurde, befassen sich [Pau08, Ove07b] speziell mit der Verwendung von REST-Services innerhalb eines BPEL-Prozesses. Sollten Prozesse in EPK-Notation vorliegen, so lassen sich diese ebenfalls mit geeigneten Verfahren in BPEL umwandeln (vgl. [SI07a]). Eine dritte Möglichkeit, die weit verbreitet ist, besteht in der Verwendung einer eigenen, einfach gehaltenen Workflow-Beschreibungssprache des PaaS-Anbieters, bei der häufig direkt im Browser ein Ablauf „zusammengeklickt" wird. Dann müssen jedoch alle zuvor modellierten Prozesse in der Plattform des Anbieters neu erfasst werden. Dies stellt einen nicht zu vernachlässigenden Aufwand dar. Ebenso muss bedacht werden, dass sich bei einer solchen Lösung der Wechsel zu einem anderen Anbieter als schwierig herausstellen kann, da die verwendete Syntax ggf. nicht mit einem anderen PaaS-Anbieter kompatibel ist. Dies ist insbesondere bei der Realisierung von redundanten Systemen, also dem Aufbau zweier identischer Controller, ein wichtiger Faktor, da hier sehr viel doppelte Arbeit anstehen könnte. Ein weiteres Problem bei proprietären Workflow-Sprachen ist im *Change Management* zu sehen. Wenn ein Workflow angepasst wird, so muss diese Änderung ebenfalls in der Geschäftsprozessdokumentation des Unternehmens nachgetragen werden und umgekehrt. Unabhängig von der Wahl des PaaS-Anbieters zur Umsetzung eines WAC bieten im Grunde alle Anbieter eine Vereinfachung der grafischen Workflow-Definition, so dass händisches Programmieren von Abläufen reduziert wird.

Für jeden vorliegenden Geschäftsprozess müssen anschließend noch die folgenden manuellen Schritte durchgeführt werden. Diese sind an dieser Stelle für Prozesse ausgeführt, die als BPMN-Modelle dargestellt sind, lassen sich aber ähnlich auf andere Prozessmodellierungssprachen anwenden.

- Für jeden Pool in einem BPMN-Prozess, der kein Logik-Pool ist, werden die zugeordneten Services auf Korrektheit der Beschreibung sowie deren Erreichbarkeit überprüft.

- Die mit den XaaS-Anbietern vereinbarten SLAs müssen für jeden Service, wenn möglich, stichprobenartig geprüft und ggf. für eine Überwachung vorgemerkt werden. So lassen sich an dieser Stelle bereits wichtige Informationen für die später zu realisierenden Überwachungskomponenten festlegen.

Eigene Logik sowie Services implementieren

Abhängig vom Umfang der Werkzeuge des PaaS-Anbieters wird es in vielen Fällen bei der Umsetzung von Geschäftsprozessen dennoch notwendig sein, auch eigene Logik zu implementieren. Beispielsweise lassen sich so Geschäftsregeln festlegen, die eine bestimmte Logik zu einem Workflow hinzufügen sollen. Teilweise ist dies auch mit vorhandenen *Business-Rules-Engines* des Anbieters möglich. Auch in der Wahl der eingesetzten Programmiersprache ist man durch die Festlegung auf einen PaaS-Anbieter eingeschränkt. Oft werden hier aber die gängigen Sprachen, wie Java, .NET, PHP, Ruby, Perl und weitere angeboten. Sollte die Auswahl des PaaS-Anbieters nicht genügen, besteht auch die Möglichkeit, eigene Logik in eine WP auszulagern und auf einem eigenen physischen oder virtuellen Server mit der gewünschten Programmiersprache zu implementieren. Dies erhöht zwar den Kommunikationsaufwand innerhalb einer WOA, ermöglicht aber ggf. eine bessere Kontrolle über die einzelnen selbst erstellten Programmteile. Der Traum eines rein durch Konfiguration und Orchestrierung geschaffenen IT-Systems ist auch im Falle der Erstellung einer WOA nicht zu erfüllen. Aber je nach Komplexität der Workflows und Anzahl der zu vernetzenden Systeme ist ein Großteil der Arbeit durch grafische Modellierung zu erledigen und wenig eigene Implementierung notwendig.

Funktionstest durchführen

Je nachdem mit welchem Softwareentwicklungsmodell die Realisierung ausgeführt wird, ist das Erstellen von Testfällen für die Workflows ggf. vorgeschrieben. Selbst wenn dies nicht der Fall sein sollte, sind Tests für einzelne Workflows sinnvoll. Hierbei sollten möglichst viele Varianten der Ausführung abgedeckt werden, beispielsweise für verschiedene Eingabevariablen oder auch simulierte Ausfälle von einzelnen Services. Dabei sollten die Ausgabewerte oder die in der Systemumgebung getätigten Änderungen mit Soll-Vorgaben überprüft werden. Im Idealfall sind die hierbei erstellten Tests automatisch ausführbar, so dass jede Änderung an einem Workflow oder der programmierten Geschäftslogik sofort überprüft wird und auftretende Fehler direkt an die verantwortlichen Stellen gemeldet werden können. Im Bereich des Software-Testens auf Implementierungsseite lassen sich hierbei drei klassische Testverfahren unterscheiden (vgl. [Bal01]):

- Komponenten- oder Modultests (Unit-Tests): Es werden einzelne Module, Programmteile oder Klassen auf deren ordnungsgemäße Funktionalität hin überprüft.
- Integrationstests: Hierbei wird die Zusammenarbeit der einzelnen voneinander unabhängigen Komponenten im Zusammenspiel getestet. Dabei stehen die Schnittstellen der Komponenten im Vordergrund.
- Systemtest: Es wird das gesamte IT-System getestet und die funktionalen und nicht-funktionalen Anforderungen verifiziert.

Die in diesem Schritt durchgeführten Funktionstests einzelner Workflows lassen sich am ehesten mit Integrationstests vergleichen, da in einem Workflow oft mehrere Services miteinander kombiniert getestet werden. Der Systemtest wird im nächsten Abschnitt erläutert.

5.4.2 Systemtest

Der Systemtest einer WOA befasst sich mit dem kompletten Zusammenspiel des IT-Systems und der Überprüfung, ob das System den Zweck erfüllt, für den es realisiert wurde. Für das ordungsgemäße Testen wurde in [PKS02] ein Phasenmodell vorgeschlagen, welches sich aus Planung, Vorbereitung, Spezifikation, Durchführung, Auswertung und Abschluss des Testens zusammensetzt. Die Durchführung eines kompletten Systemtests wird an dieser Stelle nicht weiter vertieft, da sich die vorliegende Arbeit mit einer Methode der Erstellung einer WOA und nicht mit der kompletten Testsystematik einer Anwendung befasst. Weitere Quellen, die sich mit dem Testen von Systemen befassen, sind beispielsweise [SBS08, Bal01], oder mit Fokus auf verteilten Systemen auch [Gra03, CDOML08]. Unabhängig von der gewählten Teststrategie sind für einen Systemtest einer WOA die folgenden Gegebenheiten zu beachten. Eine „Best Practice" bei der Softwareentwicklung ist das Betreiben mehrerer Entwicklungsumgebungen, die für verschiedene Zwecke verwendet werden. Meist werden hierbei die Ebenen *Test*, *Staging* und *Production* unterschieden. Neue und unfertige Programmteile werden zunächst in der Test-Umgebung entwickelt und von den Softwareentwicklern selbst getestet. Sobald eine Implementierung durch die Programmierer als fertig angesehen wird, kann diese in den Staging-Bereich übertragen werden. Dessen Systemeinstellungen sind im Normalfall weitestgehend identisch mit denen des Produktivsystems.

Die Daten, mit denen gearbeitet wird, sind aber grundsätzlich nur Kopien von bestehenden Daten oder synthetisch erstellte Testdaten. So lassen sich darin realistische Systemtests durchführen, die aber keine Beeinträchtigung des in Betrieb befindlichen IT-Systems verursachen. Schließlich werden Programmteile, die in der Staging-Ebene alle Tests bestehen, in das Produktivsystem überführt. Bei der Realisierung einer WOA ist hierbei zu beachten, dass viele der eingesetzten Komponenten XaaS-Angebote sind, die meist keine Unterscheidung zwischen Test- und Produktivsystem vornehmen und nur auf einer Datenmenge arbeiten. Eine Unterscheidung zwischen Test- und Betriebsdaten oder gar Staging-Daten ist hierbei nicht ohne Weiteres möglich. Für die Durchführung der funktionalen Tests sowie des Systemtests sollte man in der Realisierungsphase einer WOA daher zwei mögliche Optionen in Betracht ziehen. Die Erste beinhaltet die Einrichtung von zwei Zugängen zu einem XaaS-Anbieter. Damit lassen sich explizite Testdaten auf dem einen Account und Produktivdaten auf dem anderen Account sauber trennen. Bei der Umschaltung von Test- auf Produktivsystem müssen so lediglich die Zugangsdaten zu dem jeweiligen Account innerhalb der Workflows geändert werden. Die zweite Option benötigt nur einen Account bei einem XaaS-Anbieter und arbeitet mit expliziten Testdatensätzen. Dabei besteht natürlich die Gefahr, dass man den Betrieb des Produktivsystems stört, indem man beispielsweise Produktivdaten durch Tests aus Versehen verändert.

Für die Nutzung von Test-, Staging- und Produktivebenen ist zusätzlich zu dieser Überlegung ein Controller notwendig, der diese Ebenen auch in den umzusetzenden Workflows in irgendeiner Form unterstützt. Der in Kapitel 5.8 vorgestellte XaaS-Anbieter *RunMyProcess* ermöglicht hierbei das Umschalten von drei Phasen: *Design*, *Test* und *Live* und bietet daher ideale Voraussetzungen für den Einsatz als Controller in einer WOA.

5.5 Betriebsphase

Die letzte Phase der hier vorgestellten WOA-Methode befasst sich mit notwendigen Maßnahmen im produktiven Betrieb des Systems. Dabei lassen sich drei Kernaufgaben identifizieren: die Überwachung von Prozessen und Services auf deren korrekte Arbeitsweise, die kontinuierliche Kostenkontrolle der eingesetzten XaaS-Angebote sowie die Wartung und Evolution einer WOA. Diese drei Aufgaben werden im Gegensatz zu den bisher vorgestellten Abläu-

fen der Methode nicht sequenziell, sondern parallel und kontinuierlich während des WOA-Betriebs ausgeführt.

5.5.1 Prozess- und Serviceüberwachung

Das Überwachen von Services und Prozessen bzw. Workflows, das dem Auffinden von Fehlern und Anomalien bei dem Betrieb dienen soll, ist ein wichtiger Baustein einer funktionierenden WOA. Schon der Ausfall eines einzelnen XaaS-Dienstes kann sich ohne Weiteres auf mehrere Prozesse auswirken und so ganze Teile eines Systems zum Ausfall bringen. Die folgenden drei Punkte sind hierbei die wichtigsten Ziele der Überwachung:

1. Es soll ein ständig funktionsfähiges IT-System gewährleistet sein.
2. Die vom XaaS-Anbieter zugesicherten SLAs sollen laufend überprüft werden.
3. Engpässe, Störungen oder Verbesserungspotenziale sollten aufgedeckt werden.

Überwachung

In diesem Schritt muss daher eine Möglichkeit geschaffen werden, um alle in der WOA verwendeten Services zu überwachen. Die Ausgangsbasis dafür bietet der in der Analyse- und Planungsphase entstandene Service-Katalog, der alle eingesetzten XaaS-Anbieter sowie die konkret vereinbarten SLAs enthält. Aus diesen SLAs lassen sich Metriken ableiten, die der Berechnung von Kennzahlen dienen. Das Prüfen fachlicher Korrektheit ist hierbei weniger relevant, da dafür konkrete Testfälle notwendig sind. Dieser Schritt wurde bereits in der Testphase während der Realisierung durchgeführt. Da es sich bei der WOA um ein verteiltes System mit vielen einzelnen Service-Komponenten handelt, sollte an dieser Stelle vor allem die Überprüfung der Verfügbarkeit der Services sowie deren Antwortverhalten getestet werden. Die Ergebnisse dieser ständigen Überprüfung sollten dann durch geeignete Maßnahmen, beispielsweise als Web-Oberfläche – als Dashboard oder Cockpit – oder als Berichte in Dokumentenform, den jeweiligen Stellen zur Verfügung gestellt werden. Bei dem Erkennen von Fehlern muss natürlich eine aktive Form der Benachrichtigung

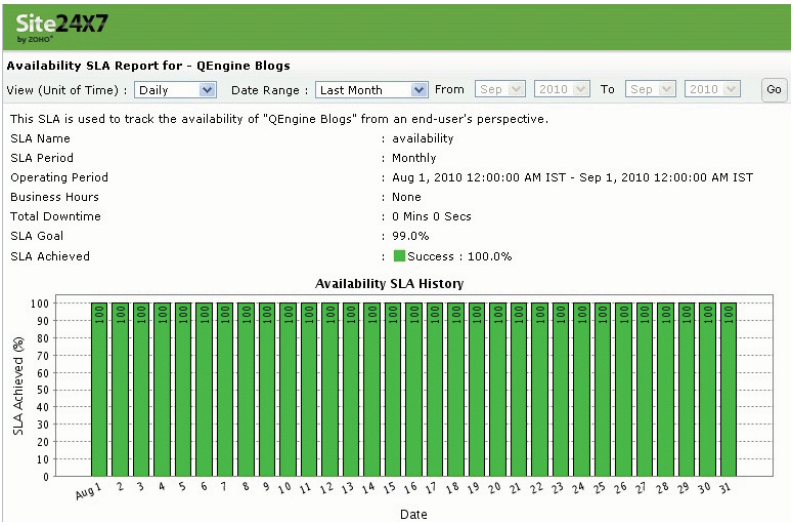
von IT-Personal erfolgen. Dies wird meist durch das Versenden von E-Mails und SMS durchgeführt.

KPIs, wie z. B. Verfügbarkeit eines Services, lassen sich durch entsprechende Metriken ermitteln. [DJI05] schlägt hierfür u. a. eine PING-Messung vor. Dieses Werkzeug überprüft, ob ein angesprochener Server unter einer angegebenen IP- oder Web-Adresse antwortet und misst dabei meist noch die Zeitspanne zwischen Absenden der Anfrage und Erhalten der Antwort, die *Round trip time*. Das PING-Programm ist sehr weit verbreitet, hat aber den Nachteil, dass sich damit nicht spezielle Services auf einem Server testen lassen, sondern nur die Existenz eines aktuell betriebsfähigen Servers. In dem hier vorliegenden Fall sind daher HTTP-Anfragen besser geeignet, da diese eine genaue Angabe des Service-Endpunkts ermöglichen. Dies schließt die Angabe von Subdomänen, Ports und HTTP-Methoden ebenso wie URL-Pfadangaben mit ein. Verwendet man darüber hinaus statt des gängigen `GET`-Befehls den HTTP-Befehl `HEAD`, wird die Antwort des Servers ohne den tatsächlichen Workload der Antwort zurückgesendet, wodurch sich zusätzliche Bandbreitenkapazität der Internetverbindung einsparen lässt. Diese Methode wird auch von vielen XaaS-Anbietern[55] verwendet, die Überwachungs-Dienste anbieten, so dass dies als „Best Practice" für den Test der Verfügbarkeit und des Antwortzeitverhaltens von WP angesehen werden kann.

Der eingesetzte Überwachungsmechanismus sollte darüber hinaus nicht nur eine Warnung für einen nicht verfügbaren Service anstoßen, sondern ebenfalls auch Statistiken über relevante SLA-Aspekte bereitstellen. Damit lassen sich beispielsweise Berichte anfertigen oder Dashboards aufbauen. Darin könnten auch weitere Kennzahlen wie MTTR und MTBF überwacht werden. In Abbildung 5.33 ist eine Überwachungsstatistik des Dienstes *Site24x7.com* abgebildet, der einen SLA-Bericht über die Verfügbarkeit darstellt. Dieser Dienst erlaubt hierbei die Angabe von speziellen SLA-Richtlinien, deren Einhaltung dann in solchen Diagrammen dargestellt werden kann.

Die Überwachung von Prozessen und Workflows lässt sich dagegen nicht durch einfache HTTP-Anfragen lösen. Hierbei kommt es auf den gewählten Anbieter bzw. die Umsetzung der Workflow-Engine an, wie im Falle eines Fehlers reagiert wird. Fehler in einem Prozess resultieren zumeist aber aus Störungen der angeschlossenen Services, so dass geeignete Maßnahmen getroffen werden können, bevor die Ausführung eines von dem Fehler betroffenen Pro-

[55] Beispielsweise www.serviceuptime.com oder http://site24x7.com.



Quelle: http://static.site24x7.com/images/sla-response-time.gif

Abbildung 5.33: Überwachungsstatistik der Verfügbarkeit eines Services mit Site24x7.com.

zesses stattfindet. Sollte dies nicht möglich sein oder es treten andere Ausführungsfehler auf, so kann die Einrichtung eines Dashboards hilfreich sein, welches die aktuell ausgeführten und ggf. fehlerhaften Workflow-Instanzen anzeigt. In diesem Zusammenhang sind auch automatische Benachrichtigungen über auftretende Fehler von Seiten des XaaS-Anbieters denkbar, die dem Unternehmen über den aktuellen Zustand eines Dienstes Auskunft geben und damit beispielsweise eine Fehlerbehandlung beim Unternehmen auslösen können.

Reaktion

Den Ausfall eines angeschlossenen XaaS-Dienstes nur zu erkennen, reicht nicht aus. Es müssen auch entsprechende Aktionen, im besten Fall automatisch, durchgeführt werden, um einen wirtschaftlichen Schaden abzuwenden oder zumindest gering zu halten. Die in der Notfall- und Alternativenplanung erstellten Maßnahmen sollten an dieser Stelle in den Systemablauf eingreifen. Beispielsweise werden alternative Services verwendet oder ganz andere Workflows angestoßen, die für diesen Zweck realisiert wurden. Auf jeden Fall muss der XaaS-Anbieter benachrichtigt werden, sobald es durch den Ausfall zu einer Verletzung der vereinbarten SLAs kommt. Somit liegt die Behebung eines fortwährenden Fehlers auf Seiten des Anbieters.

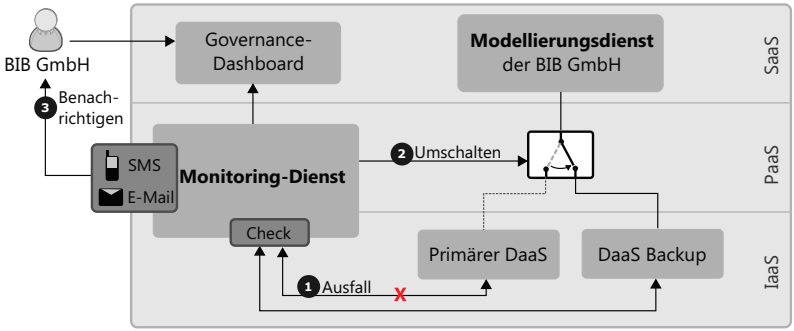


Abbildung 5.34: Ausfall eines DaaS-Anbieters bei der Service-Überwachung.

Um innerhalb der WOA mit einer Fehlerbenachrichtigung umzugehen, ist es sinnvoll, auf die Daten des Service-Katalogs zuzugreifen. Darin sind im technischen Teil die XaaS-Anbieter zu fachlichen Services zugeordnet und im fachlichen Teil die Services zu Prozessen zugeordnet. Damit lassen sich die wichtigsten Maßnahmen als Reaktion auf einen ausgefallenen Service durchführen:

- Die erste sinnvolle Maßnahme bei Ausfall eines XaaS-Dienstes ist zunächst die Überprüfung, durch welchen fachlichen Service der Dienst genutzt wird, um anschließend auf die Prozesse schließen zu können, die dadurch höchstwahrscheinlich beeinträchtigt werden. Dies ist der schnellste Weg, um eine erste automatische Auswertung durchzuführen.

- Im nächsten Schritt können die in diesen Prozessen hinterlegten Verantwortlichen darüber benachrichtigt werden, dass ein Prozess ihrer Zuständigkeit mit Problemen konfrontiert ist.

- Je nach Größe des Unternehmens und Anzahl an *data ownern*, deren Daten in einem Prozess verwendet werden, kann es sinnvoll sein, auch diese über eine Beeinträchtigung zu benachrichtigen.

- Als letzte Maßnahme können auch die in einem Prozess benötigten menschlichen Akteure, wie beispielsweise Sachbearbeiter, benachrichtigt werden, dass ein Prozess nicht korrekt funktioniert. Dies dient dann mehr als Information denn als Arbeitsanweisung für das Beheben des Fehlers.

Als Beispiel soll hier die *BBI GmbH* dienen, bei der der XaaS-Dienst für die Speicherung der eigens angebotenen Modellierungsplattform temporär ausfällt (Punkt 1). Durch die Überwachung des angeschlossenen DaaS-Dienstes ist innerhalb kurzer Zeit eine automatische Benachrichtigung per E-Mail und SMS an den zuständigen Mitarbeiter verschickt und gleichzeitig eine automatische Umleitung der Datenspeicherung auf einen bestehenden Backupdienst eingerichtet worden (Punkt 2 und 3). Damit können Kunden mit dem Modellierungsdienst weiterarbeiten und die *BBI GmbH* nach dem Fehler suchen. Über ein Web-Dashboard werden dem Unternehmen ebenfalls alle Workflows angezeigt, die infolge des Ausfalls nicht korrekt ausgeführt werden. In Abbildung 5.34 wird dieser Vorgang skizziert. Dieses Beispiel bildet lediglich eine

Möglichkeit ab, wie man mit dem Auftreten eines solchen Fehlers umgehen kann.

5.5.2 Kostenkontrolle

Durch das Auslagern von Services und die Nutzung von XaaS-Angeboten in der WOA lassen sich die durch die Nutzung entstehenden Kosten meist sehr genau berechnen. Bei Service-Angeboten im Festpreisbereich ist dies weniger handlungsrelevant, da man dort die Kosten bereits im Voraus berechnen kann. Anders verhält es sich dagegen mit XaaS-Diensten, die ein nutzungsabhängiges Kostenmodell (*pay-per-use*) verwenden. Hierbei kann eine ständig aktuelle Übersicht über die bereits angefallenen Kosten gepflegt werden. Dafür sind allerdings mehrere Bedingungen zu erfüllen. Zunächst muss jeder Aufruf eines solchen Services protokolliert werden. Dabei reicht es nicht, nur den Service und die aufrufende Rolle zu speichern, vielmehr müssen ebenfalls die für das Preismodell relevanten Parameter gespeichert werden. Diese können aber für jeden Anbieter und Service vollkommen unterschiedlich sein. Beispielsweise zahlt man bei Amazons Cloud Computing eine Gebühr für jede angefangene Stunde eines Servers im Betrieb. Hierbei müssen also Parameter wie Laufzeit des Servers, Serverart etc. protokolliert werden. Ein anderes Beispiel ist ein Zahlungsdienst wie PayPal, bei dem hauptsächlich die Höhe der Transaktion über die Kosten für den Service-Nutzer entscheidet. Zusätzlich zu diesen spezifischen Parametern muss für jeden Service eine Kostenberechnungsregel festgehalten werden. Auf Basis der protokollierten Parameter und dieser Regel lassen sich dann die für einen Service-Aufruf angefallenen Kosten berechnen.

Im Kontext einer SOA und dem Einsatz eines ESB für die Orchestrierung von Services bezeichnet man das Überwachen solcher Service-Aufrufe als *Business Activity Monitoring* (BAM) (vgl. [Jos08]). Für das korrekte Protokollieren muss hierbei also auch der fachliche Kontext verstanden worden sein, da beispielsweise ein reines Mitschreiben des Netzwerkverkehrs nicht ausreicht. Ob sich eine ständige Kostenkontrolle hierbei für eine WOA automatisieren lässt, hängt auch wieder von der eingesetzten Plattform (bzw. des Controllers) ab. In Kapitel 5.8 wird eine beispielhafte technologische Umsetzung eines Controllers mittels XaaS-Anbietern skizziert, auf dessen Basis eine derartige Kostenberechnung möglich ist. Eine WOA-Kostenabschätzung bezüglich der Planung, der Umsetzung und des Betriebs vor einer tatsächlichen Durchführung wird in Kapitel 6 ausführlich erläutert.

5.5.3 Wartung und Evolution

Die Wartung einer WOA hängt sehr eng mit der Service-Überwachung zusammen. Gerade in der Überwachung gefundene Fehlerquellen, wie oft ausfallende oder langsam antwortende Services, müssen zumindest verbessert oder sogar ausgetauscht werden. Daher lassen sich die Ergebnisse der Überwachung als eine der Grundlagen für die Wartung einer WOA auswerten. Eine Wartung im Sinne herkömmlicher IT-Systeme, bei denen es Hard- und Software zu kontrollieren und auszutauschen gilt, ist im Falle einer archetypischen WOA nicht notwendig. Diese Pflicht obliegt dann den XaaS-Anbietern. Natürlich lassen sich aber eigene Logik und Prozesse verbessern und warten, die auf einer Plattform im Internet abgelegt sind. Die innerhalb von IT-Systemen vorzunehmende Fehlerbehebung wird beispielsweise in ITIL in zwei Bereiche aufgeteilt, dem *Incident Management* und dem *Problem Management*. Ersteres befasst sich hauptsächlich mit der schnellen Behebung von Service-Ausfällen bei einer minimalen Störung des Geschäftsbetriebs. Das Problem Management ist eher als zweite Stufe anzusehen und versucht eine Ursachenforschung der aufgetretenen Probleme durchzuführen und so auf lange Sicht das Problem zu beheben (vgl. [Olb08]). Beide ITIL-Verfahren sind aber eher aus Sicht des XaaS-Anbieters relevant, da dieser den Service-Ausfall zu beheben hat.

Im Kontext einer WOA ist daher neben der natürlichen Weiterentwicklung von Geschäftsprozessen vor allem die Evolution der XaaS-Angebote ein zu beachtender Aspekt. Bei herkömmlichen Service-orientierten Systemen, die innerhalb eines Unternehmens liegen, unterliegen Services zwar auch einem gewissen evolutionären Wandel durch Weiterentwicklung, aber der Einsatz oder Wechsel auf neue Versionen lässt sich durch eine entsprechende IT-Organisation steuern. In einer WOA, in der ein Großteil der Services von Fremdanbietern bezogen wird, ist die Kontrolle über Weiterentwicklungen nicht gegeben. Je nach Professionalität oder Organisationskultur des Anbieters können für XaaS-Dienste plötzlich neue Schnittstellen eingeführt werden, die eine Anpassung auf Seiten der Service-Orchestrierung im WAC erfordern. Ebenso können Anbieter ihre Services einstellen oder deren Kostenstruktur ändern. Abhängig von der Art der vereinbarten SLAs hat man als Service-Nutzer mehr oder weniger Sicherheit vor solchen Vorfällen. Daher muss man selbst nach der Realisierung einer WOA den Markt der XaaS-Anbieter ständig im Auge behalten, um im Zweifelsfall auf alternative Dienste ausweichen zu können. Zwar wurde in der Notfall- und Alternativenplanung bereits nach diversen Aus-

weichmöglichkeiten gesucht, aber es können immer wieder neue Alternativen entstehen. Ein weiterer Grund für das wachsame Beobachten der existierenden XaaS-Angebote ist auch der generelle Preisdruck und die mögliche Reduzierung der eigenen IT-Kosten durch das Wechseln zu einem anderen Anbieter.

Die Wartung einer WOA ist daher selbst bei einem vollständig fehlerfrei funktionierenden System eine ständig anfallende Aufgabe. Der Zuwachs an registrierten APIs bei ProgrammableWeb, wie in Abbildung 5.21 dargelegt, ist hierbei nur ein Indiz für das Wachstum der Branche und der immer wieder neu entstehenden Service-Angebote.

5.6 Unterschiede im Vorgehen bei vorhandener IT-Infrastruktur

Das bis zu diesem Punkt beschriebene Vorgehensmodell zur Erstellung einer WOA beruht auf der Annahme, dass ein IT-System für ein Unternehmen erstellt wird, ohne auf ein ggf. vorhandenes Altsystem zu achten. An dieser Stelle soll explizit erläutert werden, welche zusätzlichen oder veränderten Schritte vorzunehmen sind, um das bestehende IT-System eines Unternehmens auf eine WOA zu migrieren. Sneed beschreibt die Migration von Software als das Überführen eines Softwaresystems in eine andere Zielumgebung (vgl. [SHT04]). Diese Aufgabe sei daher zunächst als rein technische Herausforderung zu sehen, die ein bestehendes Altsystem transformiert. Diese Aussage trifft auf die reine Transformation von Funktionalität eines Systems zu. Die Ist-Analyse eines bestehenden IT-Systems, die einer Migration vorausgehen muss, ist dagegen weniger technischer als vielmehr organisatorischer Natur. Die Analyse-Methoden von bestehenden IT-Systemen sind recht vielfältig und stehen nicht im Fokus dieser Arbeit, so dass im Folgenden nur eine kurze Einführung gegeben wird. Das Hauptaugenmerk liegt an dieser Stelle auf den in Teilen differierenden Vorgehensschritten der Methoden-Phasen, die herausgestellt werden.

Zu den bereits vorgestellten Phasen wird an dieser Stelle daher die Ist-Analyse hinzugefügt. Diese ist direkt nach der Phase der Geschäftsprozessmodellierung eingeordnet. An der Ausführungsanordnung der weiteren Phasen des Modells ändert sich nichts. In Abbildung 5.35 ist die geänderte Übersicht der Phasen als Teilmodell der Gesamtdarstellung abgebildet.

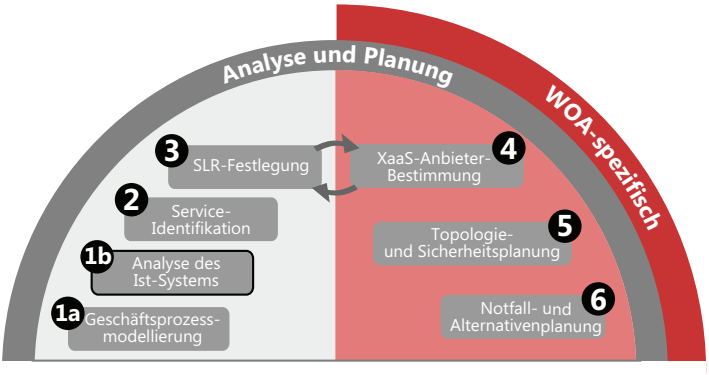


Abbildung 5.35: Die Phasen einer WOA-Umsetzung mit bestehendem IT-System.

5.6.1 Systemanalyse des Ist-Zustands

Eine Systemanalyse wird ausgeführt, wenn ein Unternehmen bereits ein bestehendes IT-System besitzt, welches in Teilen oder gänzlich auf eine WOA umgestellt werden soll. Dabei werden die Datenstrukturen, die Systemkomponenten, die Verknüpfungen zwischen den Teilen und ebenso bestehende Geschäftsprozesse untersucht und beschrieben. Je nach Zielsetzung der WOA-Strategie werden ggf. nur die Teile des IT-Systems untersucht, die ersetzt oder eingebunden werden sollen. Die resultierenden Dokumente aus dieser Phase umfassen einen Katalog bestehender IT-Komponenten, unabhängig davon, ob diese bereits als SOA-Komponenten existieren. Diese werden durch technische Details, Interfaces und Abhängigkeiten zu anderen Komponenten beschrieben. Darüber hinaus existiert eine Liste mit Zuordnung von IT-Komponenten zu bestehenden, konkreten fachlichen Services aus den fachlichen Prozessen, also die erste Version eines Service-Katalogs. Dies stellt eine Bestandsanalyse der bereits umgesetzten Prozesse durch IT innerhalb des Unternehmens dar.

Das Ziel einer Systemanalyse ist das Erfassen und Beschreiben eines bestehenden IT-Systems, um dieses für spätere Schritte im Planungsprozess zu berücksichtigen. Damit kann der Ist-Zustand eines Systems festgehalten werden. Diese Analyse wird daher auch als *Ist-Analyse* bezeichnet. Im Gegensatz dazu nennt man ein durch spezifizierte Anforderungen geplantes Sys-

tem *Soll-System*. Die Artefakte, die hierbei in Modellen dargestellt werden können, reichen von ERM-Diagrammen zur Darstellung von Datenmodellen, über Kollaborationsdiagramme zur Darstellung von Topologien und Interaktion von Komponenten bis hin zu Klassendiagrammen oder auch durch Pseudocode dargestellte Geschäftsregeln. Es stehen, je nach Anwendungsfall und Art des untersuchten Systems, unzählige Modellarten zur Verfügung. Im Kontext der Entwicklung verteilter Systeme sollen hierbei jedoch primär die einzelnen Komponenten des Systems und deren Interaktionen erfasst und dokumentiert werden. Es geht hierbei weniger um die tatsächliche Programmierung dieser Komponenten, so dass eine Systemanalyse für den Einsatz im WOA-Vorgehensmodell vor allem Datenmodelle, Services und Prozesse des bestehenden Systems erfassen sollte. Der Ablauf einer Systemanalyse lässt sich auf zwei generelle Methoden reduzieren: den *Top-down*- und den *Bottom-up*-Ansatz. Der Top-down-Ansatz startet mit dem Analysieren von Geschäftsprozessen, die durch ein bestehendes System umgesetzt sind, und arbeitet sich bis auf die technischen Schichten vor. Der Bottom-up-Ansatz geht in umgekehrter Reihenfolge vor. Dieser beginnt mit der Betrachtung der IT-Systemkomponenten und arbeitet sich nach oben bis auf Geschäftsprozessebene vor. Oft werden auch beide Methoden gleichzeitig durchgeführt, um eine konkrete Übereinstimmung von Geschäftsprozessen und umgesetzten Services zu erhalten. Dies wird dann als „Meet-in-the-middle"-Ansatz bezeichnet.

In einigen Vorgehensmodellen zur Erstellung von verteilten Systemen wird eine Ist-Analyse als integraler Bestandteil der Analyse- und Planungsphase genannt. Beispielsweise nennt es [Off08] die Bestandssystemanalyse, [LS08] dagegen nennt sie schlicht Systemanalyse und beschreibt damit ein Bottom-up-Vorgehen. Eine Methode, die lediglich die Identifizierung von Services in einem Altsystem beschreibt, ist in [ATM08] zu finden. Dort werden Systeme nach dem Vorkommen von Programmmustern untersucht, die sich zur Service-Implementierung anbieten. Der SOMA-Ansatz von IBM beschreibt eine Mischung aus beiden Methoden und sieht verschiedene Analyse-Schritte vor. Begonnen wird hierbei mit dem Identifizieren von fachlichen Dienstleistungen, den sogenannten *Business Components*, um anschließend durch eine Bottom-up-Analyse die identifizierten IT-Services mit den Komponenten in Verbindung zu bringen (vgl. [AA06, Gan06]). Da der Ansatz von IBM sehr gut dokumentiert ist und sich mit der ursprünglichen Zielsetzung – der Umsetzung einer SOA – hervorragend für die Analyse eines Altsystems innerhalb

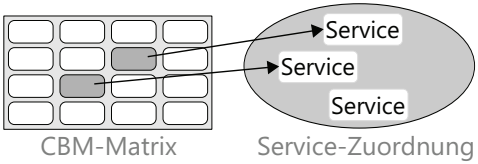
der Einführung einer WOA eignet, wird im Folgenden ein kurzer Einblick in die Vorgehensweise erläutert.

IBM-Ansatz zur Systemanalyse

Der Ansatz von IBM für die Analyse eines bestehenden Systems im Rahmen einer SOA-Umsetzung unterteilt sich in drei Teile: *Component Business Modeling* (CBM), Service-Zuordnung und Implementierung einer SOA. Im ersten Schritt, dem CBM, werden die Geschäftsprozesse des Unternehmens analysiert und als Komponenten (*Business Components*) zusammengefasst. Die Beschreibung einer solchen Komponente umfasst dabei den Geschäftszweck, die Aktivitäten dieser Komponente, die benötigten Ressourcen, die Steuerung der Komponente und die Verwendung weiterer Komponenten (Schnittstellen). Als Ergebnis dieser Analyse lässt sich eine Matrix aufbauen, die durch die zwei Dimensionen Verantwortlichkeit (*Accountability*) und fachliche Zuständigkeit (*Business Competency*) dargestellt wird (vgl. [PKR05]). Im zweiten Schritt des Vorgehensmodells von IBM werden IT-Services bestehender Systeme analysiert. In welcher Form diese dabei vorliegen, ob als fern aufrufbare Methoden mit definierten Schnittstellen (wie beispielsweise bei CORBA oder RPC) oder gar als monolithische Systeme, ist zunächst nicht von Belang. Informationen über bestehende IT-Systeme lassen sich beispielsweise durch vorhandene technische Dokumentationen, einen Bebauungsplan oder auch Mitarbeiterinterviews erfassen. Anschließend wird versucht, die identifizierten IT-Services den Komponenten der CBM-Analyse zuzuordnen. Diese Services beschreiben aber hier nicht zwangsläufig Services im Sinne von WP, sondern zunächst generelle IT-Komponenten. SOMA versteht sich dabei als Mischung aus Top-down- und Bottom-up-Vorgehensmodell und betrachtet sowohl die vorliegenden Geschäftsprozesse eines Unternehmens als auch das bereits vorhandene IT-System. Diese beiden Schritte sind in Abbildung 5.36 skizziert. Der dritte Schritt des Ansatzes von IBM liegt dann in der konkreten Umsetzung der SOA und wird hier nicht vertieft.

Unabhängig von dem gewählten Modell einer Ist-Analyse ist vor allem wichtig, die folgenden Informationen über ein bestehendes IT-System zu erhalten, damit die Einführung einer WOA gelingen kann:

- Katalog bestehender IT-Komponenten (seien es nun Services im Sinne einer WOA oder auch nicht) inklusive technischer Beschreibung, Interfaces und Abhängigkeiten zu anderen Komponenten.



Quelle: angelehnt an [TV08]

Abbildung 5.36: Die ersten zwei Schritte der SOMA-Methode von IBM.

- Katalog bestehender fachlicher Geschäftsprozesse, die durch das IT-System unterstützt werden.
- Zuordnung von den im Katalog aufgeführten IT-Komponenten zu bestehenden, konkreten Services aus den fachlichen Geschäftsprozessen. Also im Grunde eine Bestandsanalyse der bereits umgesetzten Prozesse durch IT innerhalb des Unternehmens. Somit lassen sich diese später besser durch entsprechende XaaS-Angebote ersetzen oder in die WOA einbinden.

Nach einer Ist-Analyse sollten **alle** IT-Komponenten (Services, Altsysteme, etc.) des Katalogs mindestens einem Prozess zugeordnet sein. Ist dies nicht der Fall, wird die Komponente augenscheinlich nicht verwendet und wird daher im weiteren Verlauf nicht mehr beachtet. Dies befreit natürlich nicht von der Pflicht, zu überprüfen, warum diese Komponente noch in Betrieb ist.

5.6.2 Anpassung der Ausführung der WOA-Methode

Für jede Phase des WOA-Vorgehensmodells wird beschrieben, welche zusätzlichen oder abgewandelten Schritte auszuführen sind, um die bestehende IT-Infrastruktur zu beachten.

Phase 1) Geschäftsprozessmodellierung + Ist-Analyse Bei den ersten beiden Phasen kommt es darauf an, welches Vorgehensmodell der Ist-Analyse gewählt wird. Ist darin bereits eine komplette Geschäftsprozessmodellierung enthalten, so kann die erste WOA-Phase entfallen. Ist dagegen nur eine partielle oder gar keine Betrachtung der fachlichen Prozesse vorgesehen, muss die in der WOA-Methode als erster Schritt vorgeschlagene Phase durchgeführt

werden. Es muss jedoch anschließend unbedingt ein Abgleich der Ergebnisse der Prozessmodellierung und der Erkenntnisse der Ist-Analyse geschehen, die unterschiedlich ausfallen kann. Wählt man die SOMA-Methode von IBM, so wird eine Geschäftsprozessmodellierung vorgenommen, die nach der Analyse des bestehenden Systems mit der CBM abgeglichen werden kann.

Phase 2) Service-Identifikation Die Identifikation von Services wird an dieser Stelle durch eine Typisierung der Aktivitäten in den Geschäftsprozessen durchgeführt, wie zuvor beschrieben. Der Aufbau des Service-Katalogs erfolgt dabei aber mit Unterstützung der durch die Ist-Analyse vorliegenden Daten. Hierbei lassen sich bereits identifizierte, bestehende IT-Komponenten bzw. Services zu den typisierten Aktivitäten der Geschäftsprozesse zuordnen, so dass der in der Analyse-Phase bereits begonnene Service-Katalog ergänzt bzw. erweitert wird. Services, die nicht bereits durch das bestehende IT-System abgedeckt werden, können in der beschriebenen Form aufgenommen werden. Diese werden später in jedem Fall durch XaaS-Dienste realisiert.

Phase 3) SLR-Festlegung Die SLRs für bereits bestehende Services können an dieser Stelle erweitert oder verändert werden, während neu identifizierte Services wie gehabt die funktionalen sowie nicht-funktionalen Beschreibungen erhalten. Der wichtigste Schritt dieser Phase ist nun gemäß der jeweiligen Outsourcing-Affinität des Unternehmens die Entscheidung, ob ein identifizierter IT-Service ausgelagert werden soll oder nicht. Sollte das Unternehmen eine archetypische WOA anstreben, so werden die SLRs so angelegt, dass versucht wird, für alle bestehenden Services des IT-Systems entsprechende Realisierungen als XaaS-Angebot zu finden. Eine andere Möglichkeit besteht in der teilweisen Ersetzung bestehender Services oder in dem kompletten Erhalt der bestehenden Services, die im eigenen Unternehmen verbleiben und in eine WOA eingebunden werden.

Phase 4) XaaS-Anbieter-Bestimmung Diese Phase wird ohne Änderungen des Vorgehens durchgeführt. Durch den ausführlichen Service-Katalog ist geklärt, welche IT-Services durch einen XaaS-Dienst realisiert werden sollen.

Phase 5) Topologie- und Sicherheitsplanung Für die Konkretisierung von Service-Interfaces und die Verfeinerung der Prozesse ergeben sich durch die

Nutzung von bestehenden IT-Komponenten einige Veränderungen. Die erste zu beachtende Anpassung besteht in der Verfeinerung der BPMN-Prozesse. Zusätzlich zu dem Logik-Pool muss ggf. ein zusätzlicher Pool für innerhalb des Unternehmens liegende Systeme erstellt werden. Damit lassen sich Abläufe definieren, die in einem ausgelagerten Controller platziert werden, und auf der anderen Seite Abläufe kenntlich machen, die beispielsweise auf einem internen ESB realisiert werden. So können bestehende Geschäftsprozesse durchaus an Komplexität zunehmen, wodurch analog die Komplexität des zu realisierenden Systems widergespiegelt wird. Auch der Aspekt der Datenlokalität ist hierbei verstärkt zu beachten, da ggf. vorhandene Daten und Systeme innerhalb des Unternehmens bestehen, die weiterverwendet werden sollen. Dabei kann häufig das Kopieren von internen Daten für deren Verarbeitung zu einem im Internet platzierten XaaS-Dienst auftreten.

Auch Architekturskizzen der Topologie können komplexer ausfallen, da darin nicht nur die Architektur auf Seite des Internets, sondern auch auf Seite des Unternehmens abgebildet werden muss. Was die Rechtestrukturen angeht, können diese ggf. aus bestehenden Systemen zur Nachbildung innerhalb einer WOA übernommen werden. Dies vereinfacht den Aufwand der Rechteerhebung. Bei einer Mischung von internen und externen Systemen, wie bei einer regulativen WOA, werden aber einerseits interne Rechte und Rollen mit externen zu realisierenden Rechten vermischt. Von Vorteil wäre hierbei eine bestehende PKI des Unternehmens, die sich durch Web-Technologien einer WOA weiterverwenden lassen. Ob und wie dies möglich ist, hängt aber von der möglichen Realisierung der genutzten XaaS-Anbieter und deren verwendeten Technologien für die Sicherheitsstruktur ab. In Abbildung 5.37 ist eine erweiterte Architekturskizze des zuvor beschriebenen Szenarios der Online-Bezahlung aus Abbildung 5.31 dargestellt. Hier wird davon ausgegangen, dass der Online-Shop des Unternehmens auf einem internen Server installiert ist und zusätzlich auch ein eigenes CRM-System in die WOA eingebunden werden soll. Ein WAC zur Überprüfung der Services, wie zuvor enthalten, wurde hier zur Übersichtlichkeit bewusst weggelassen.

Phase 6) Notfall- und Alternativenplanung Die Vorgehensweise bei der Planung von Notfällen oder alternativ einsetzbaren XaaS-Diensten bleibt auch bei dem Einsatz eines bestehenden IT-Systems im Grunde dieselbe. Es muss jedoch ein weiterer Aspekt beachtet werden. Neben den Auswirkungen von

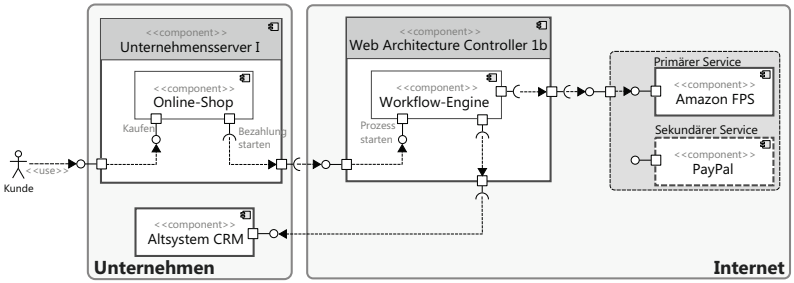


Abbildung 5.37: Architekturskizze mit System-Komponenten innerhalb des Unternehmens.

Ausfällen einzelner XaaS-Angebote muss auch die Gefahr des Ausfalls von internen Systemen untersucht werden. Sollten für die WOA bestehende IT-Infrastrukturen eingebunden werden, so muss man davon ausgehen können, dass dafür bereits Ausfallstrategien bestehen. Ebenso sollten redundante Systeme für wichtige IT-Komponenten bereits realisiert sein, so dass man hier keine weiteren Planungen im Sinne der Ausfallsicherheit vornehmen muss. Stellt sich hierbei heraus, dass diese Annahmen nicht zutreffend sind, so lassen sich Überlegungen von XaaS-Diensten auch auf interne IT-Strukturen übertragen: beispielsweise der redundante Aufbau von wichtigen IT-Komponenten, oder das Suchen nach möglichen XaaS-Diensten, die im Notfall als Überbrückungsdienst ausreichen. Für das Anstreben einer archetypischen WOA, in der auch alle internen IT-Systeme durch entsprechende XaaS-Dienste ersetzt werden, werden die Schritte dieser Phase ausgeführt, wie oben erläutert.

Phasen 7 und 8) Realisierung und Test sowie Systemtest Bei der Realisierung einer WOA unter Berücksichtigung bestehender IT-Systeme muss deren Einbindung auf technischer Ebene ermöglicht werden. Je nach Vorkommen von Altsystemen und den damit bereits vorhandenen Schnittstellen kann dies recht trivial bis extrem aufwendig sein. Die folgenden Szenarien beschreiben den Aufwand für das Einbinden verschiedener IT-Systeme:

- SOA-Implementierung: Liegt ein IT-System bereits als SOA vor, existieren die Funktionalitäten des Systems bereits als Web Services. Die Ge-

schäftsprozesse sind bereits darauf abgestimmt und ein Einbinden der einzelnen Services in die WOA kann ohne Umwege geschehen.

- IT-System mit Netzwerk-Schnittstellen: Bei einem noch nicht Service-orientierten Ansatz sind neben den technischen Maßnahmen zur Einbindung in eine WOA meist auch organisatorische Änderungen vorzunehmen. Denn in einer WOA, wie auch in einer SOA, sind IT-Services nach deren Fachlichkeit in der Organisationsstruktur verankert. Sie orientieren sich nicht an einer gegebenen Abteilungsstruktur. Neben den ggf. auftretenden organisatorischen Problemen müssen hier die Schnittstellen der Systeme für eine WOA angepasst werden. Die IT-Funktionalitäten müssen dabei über definierte Schnittstellen bereitgestellt werden. Dieses Erstellen von WP für bestehende Systeme ist mit größerem Aufwand verbunden. Bevor an dieser Stelle viel Arbeit in die Programmierung und Anpassung der bestehenden Systeme gesteckt wird, sollte man prüfen, ob sich diese nicht durch XaaS-Angebote aus dem Internet ersetzen lassen.

- IT-System mit hauptsächlich monolithischen Systemen: Eine solche Voraussetzung für die Migration auf WOA ist mit Sicherheit der ungünstigste Fall. Monolithische Systeme ohne Schnittstellen nach außen lassen sich nicht mit einfachen Mitteln in eine WOA (ebenso wenig in eine SOA) einbinden. Hierbei sollte man die Möglichkeit des Einsatzes neuer Produkte oder vorhandener XaaS-Anbieter in Betracht ziehen.

Der Umgang mit der Realisierung und dem Testen von bestehender IT-Infrastruktur im Zusammenhang mit einer WOA hängt, wie schon bei der Realisierungsphase genannt, stark vom gewählten Vorgehensmodell ab. Beispielsweise würde man bei agilem Vorgehen nach und nach die bestehenden Systeme mit den gewählten XaaS-Angeboten verbinden, die Anforderungen immer wieder prüfen und ggf. anpassen und wiederkehrend die Funktionalität testen. Traditionelle Methoden würden die Kopplung in der Planungsphase, wie oben beschrieben, durch Topologieskizzen und Beschreibung von Rechtestrukturen zuerst planen, um diese anschließend in der Realisierungsphase auszuführen.

Betriebsphasen Im Betrieb einer WOA mit bestehendem IT-System lassen sich ebenfalls einige zusätzliche Aspekte zum oben beschriebenen Vorgehensmodell nennen.

- Prozess- und Serviceüberwachung: Die unternehmensinternen Systeme, zumindest die als Service bereitgestellten, lassen sich im Normalfall durch dieselben XaaS-Dienste überwachen, die auch schon für die Überwachung der WOA-Services im Internet verwendet werden. Hierbei können daher alle Maßnahmen bei einem Ausfall oder einer Störung identisch verlaufen. Der einzige Unterschied ist, dass bei einem Ausfall interner Systeme das eigene IT-Personal die Problembehebung erledigen muss. Dies erfolgt bei einem XaaS-Dienst sonst durch den Anbieter.

- Kostenkontrolle: Die unternehmensinternen Dienste unterliegen anderen Kostentreibern als XaaS-Dienste. Normalerweise werden hierfür Server gekauft und nach und nach abgeschrieben oder vorhandene müssen zumindest gewartet und erneuert werden. Für Software fallen neben Wartungskosten zusätzlich ggf. noch Lizenzkosten an. Die hierfür aufgewendeten Mittel sind normalerweise nicht, wie bei einem XaaS-Dienst, durch ein einfaches nutzungsabhängiges Kostenmodell berechenbar. Dazu kommen noch Stromkosten für Betrieb und Kühlung etc. Eine Überwachung und ständige Aufsummierung der entstandenen Kosten für die unternehmensinternen Systeme ist damit schwer möglich. Der Teil einer WOA, der tatsächlich im Internet durch XaaS-Dienste realisiert ist, kann natürlich nach wie vor durch Protokollierung der Service-Nutzung mitverfolgt werden.

- Wartung und Evolution: Eine Wartung der im Unternehmen platzierten Hard- und Software muss entsprechend der bisher verfolgten Systemwartung erfolgen. Dies wird also zusätzlich zur WOA-spezifischen Wartungsphase durchgeführt. Die Evolution von Systemen und Services vollzieht sich dagegen ähnlich wie die einer reinen Internet-basierten WOA. Die ständige Suche nach alternativen XaaS-Anbietern in dieser Phase wird hierbei aber um die unternehmensinternen Services erweitert. Damit hält man sich die Option offen, in Zukunft ggf. bisher interne Services doch ins Internet zu einem Xaas-Dienst auszulagern.

5.7 Umsetzung einer WOA durch Feature Driven Development

Die Auswahl möglicher Vorgehensmodelle zur Umsetzung eines Softwareprojektes ist groß und es sollte versucht werden, das jeweils auf die Situation passendste Modell zu wählen. Eine Aussage in der Art „*Vorgehensmodell X ist für die Softwareentwicklung das geeignetste Modell*" lässt sich nicht treffen. Jede Vorgehensweise bringt spezifische Vor- und Nachteile mit sich, so dass man den Kontext des Projektes betrachten muss. Dieser setzt sich aus mehreren relevanten Komponenten zusammen. So ist die Art des umzusetzenden Softwareprojekts von Bedeutung. Hier kann es sich beispielsweise um die Entwicklung einer Desktop-Applikation, eines Web-basierten Systems oder eines komplexen verteilten Systems handeln. Ein weiterer wichtiger Aspekt ist der Auftraggeber und dessen Rolle im Projekt. Ist dieser eine staatliche Behörde, ein Unternehmen oder sogar eine interne Abteilung des eigenen Unternehmens, so lassen sich diverse Unterschiede in der Umsetzbarkeit mancher Methode aufzeigen. Als Beispiel seien hier die starke Einbindung des Kunden bei agilen Methoden und dessen schnelle Reaktion bei inhaltlichen Fragen zu nennen, die mancher Auftraggeber nicht erfüllen kann. Um sich nun im Rahmen der WOA für ein Vorgehensmodell zu entscheiden, lässt sich zunächst die folgende Frage formulieren:

> *Wenn man mit einer WOA eine Vereinfachung der Implementierung, Reduzierung der Komplexität und Reduktion der Kosten erreichen will, sollte man dann nicht auch ein Vorgehensmodell wählen, das möglichst einfach und „reduziert" ist?*

Im Kontext einer WOA kann diese Frage nur bejaht werden. Die Vorteile, die durch die Verwendung einfacher Technologien und geringer Komplexität innerhalb einer WOA entstehen, sollten durch ein einfaches und leichtgewichtiges Vorgehensmodell unterstützt werden. Im Folgenden wird daher auf die Umsetzung einer WOA durch die agile Vorgehensmethode *Feature Driven Development* eingegangen. Der Ansatz vieler agiler Vorgehensmodelle, die Planung zu Beginn des Projektes auf eine Sammlung von umzusetzenden Funktionalitäten (Stories oder Features) zu beschränken und dann direkt bereits mit der Programmierung zu starten, ist in Verbindung mit einem komplexen verteilten System wie der WOA zunächst als deutlicher Nachteil zu sehen. Hierbei

ist die Planung der IT-Architektur, der Berechtigungen auf Prozesse und der Abhängigkeiten zwischen einzelnen Programmteilen zu Beginn des Projekts, zumindest als Übersichtsmodell, von großer Relevanz. Dennoch bieten agile Vorgehensmodelle bzw. die darin vorgeschlagenen Methoden zur Ausführung des Projekts gerade durch die flexible Anpassung an Änderungen der Vorgaben und die häufigen und kurzen Release-Zyklen ideale Voraussetzungen für die Umsetzung eines verteilten Systems.

Die Eignung solcher Methoden für die Umsetzung einer WOA liegen auf der Hand, da diese aus vielen, teilweise unabhängigen, Modulen (Services) besteht, die Stück für Stück umgesetzt und eingeführt werden können. So lassen sich bereits früh im Projekt fertige Teile einer WOA testen und produktiv einsetzen. Die einzelnen Phasen des WOA-Vorgehensmodells, wie sie zuvor im Detail erläutert wurden, werden hierbei in teilweise abgewandelter Form und ggf. auch anderer Reihenfolge angewandt. Da es sich bei FDD um eine Vorgehensmethode für die Erstellung von IT-Systemen bzw. Software handelt, werden die während eines Betriebs einer WOA notwendigen Phasen nicht behandelt. Ebenso wird die Phase des Systemtests weggelassen, da diese unabhängig von der Systemerstellung durchgeführt werden kann.

Somit betrifft FDD die ersten sieben Phasen des Vorgehensmodells von der Geschäftsprozessmodellierung bis hin zur Realisierung. Die Entwicklung einer WOA mittels FDD ist für das Szenario der *BBI GmbH* sehr geeignet, da kein bestehendes IT-System erfasst und eingebunden werden muss. Die konkrete Anordnung der WOA-Phasen innerhalb des FDD-Modells ist in Abbildung 5.38 dargestellt und wird im Folgenden ausgeführt.

5.7.1 Erstellung des Gesamtmodells

In dieser ersten Phase von FDD werden grundlegende Modelle der umzusetzenden WOA angefertigt. Dabei werden die ersten Schritte der Geschäftsprozessmodellierungsphase durchlaufen und es entstehen Organisations- und Ablaufmodelle sowie weitere Dokumentationen, die eine gewisse Übersicht über den Umfang des zu realisierenden Systems gibt. Im Leitfaden zu FDD werden UML-Diagramme als Modellierungswerkzeug vorgeschlagen. Wie zuvor schon beschrieben, können dafür zunächst einfache Modelle in Form von Interaktionsdiagrammen verwendet werden. Da die Zeit für die Planungsphase bewusst stark begrenzt ist, lässt sich hier keine Modellierung bis auf eine fein-

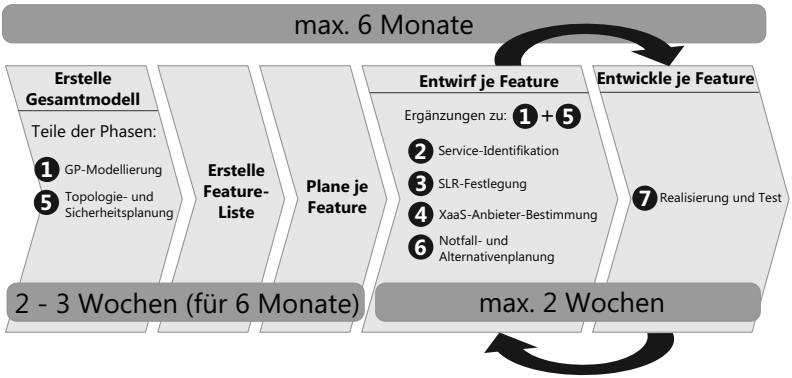


Abbildung 5.38: Durchführung der WOA-Phasen mittels der FDD-Methode.

granulare Ebene durchführen. Vielmehr dient die hier durchgeführte Modellierung in erster Linie einer Abbildung des gemeinsamen Verständnisses der fachlichen Zusammenhänge, die zu einer Konsensbildung zwischen Auftraggeber und Auftragnehmer führen soll. Von großer Bedeutung ist hierbei, dass in dieser Phase IT-Architekten direkt mit Domänenspezialisten zusammenarbeiten, um die fachlichen Zusammenhänge darzustellen. Zudem können, da es sich bei einer WOA um ein verteiltes System handelt, in dieser ersten Phase auch schon grobe Modelle einer Topologie, sowie Anforderungen an die Sicherheitsstrukturen besprochen und als erste Skizzen festgehalten werden. Verglichen mit dem ausführlichen Vorgehensmodell zur Erstellung einer WOA werden hierbei nur Teile der ersten und fünften Phase durchlaufen. In Anbetracht der zur Verfügung stehenden Zeit ist an dieser Stelle auch keine ausführliche Planung möglich.

Als Beispiel dient hier die Realisierung eines Seminarbuchungsprozesses der *BBI GmbH*. Hier wurde in der Grobplanung für die Domäne *Weiterbildung* das Fachgebiet *Seminargeschäft* dargestellt (siehe Abbildung 5.39).

Zusätzlich wurde zur fachlichen Klärung ein Datenmodell als ERM-Diagramm ermittelt, welches für eine spätere Umsetzung des Seminargeschäfts dienlich ist. Die darin vorkommenden Entitäten sind Dozent, Thema und Kunde. Da-

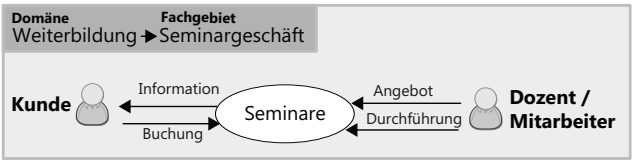


Abbildung 5.39: Übersichtsmodell des Fachgebiets *Seminargeschäft*.

bei setzen sich angebotene Seminare aus Dozenten und Themen zu einem speziellen Zeitpunkt zusammen, die, wenn sie durch einen Kunden gebucht werden, durch den Relationship-Typ Buchung repräsentiert werden (vgl. Abbildung 5.40).

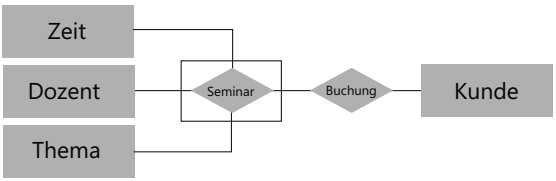


Abbildung 5.40: Datenmodell des Fachgebiets *Seminargeschäft*.

Die Dokumentation nach dieser ersten Phase fällt sehr viel geringer aus, als die, die durch einen kompletten Ablauf der ersten und fünften Phase eigentlich erstellt werden würde. Die in einer ausführlichen Planung erstellten Dokumente beschreiben fachliche Prozesse, Organisations- und Datenmodelle, Prozessverantwortliche sowie Geschäftsregeln aus der ersten Phase sowie Rollen, Rechte und technische Dokumentationen der Netzwerktopologie aus der fünften Phase des Vorgehensmodells. Durch den Charakter eines agilen Modells steht nun aber mehr die Kommunikation als die Dokumentation im Vordergrund, so dass Modelle nur insoweit erstellt werden, als sie zunächst für eine Übersicht über das System notwendig sind.

5.7.2 Erstellung einer Feature-Liste

Um in späteren Phasen eine Planung der Umsetzung und in Kapitel 6 auch die Berechnung der Umsetzungskosten zu ermöglichen, werden die einzelnen

Bereiche des umzusetzenden Systems in einer Dekompositionsphase bis auf Feature-Ebene verfeinert. Die einzelnen Ebenen sind hierbei: Domäne, Fachgebiet, Geschäftsaktivität und Schritt (*Feature*). Ein solches Feature ist möglichst kurz und beschreibt nicht das „Wie", sondern nur das „Was" und ggf. „Wer" eines Schrittes in einer Geschäftsaktivität. Daher werden an dieser Stelle noch keine Prozessmodelle in Form von BPMN erstellt, da diese sich hauptsächlich mit dem „Wie" auseinandersetzen. Im Normalfall sollte die Umsetzung eines Features nicht mehr als zwei Wochen benötigen. Dem Beispiel folgend lässt sich für das Fachgebiet Seminargeschäft die Geschäftsaktivität *Seminarbuchung* festhalten, die die folgenden drei Features enthält:

1. „Ein Kunde kann sich (online) über die angebotenen Seminare erkundigen."

2. „Ein Kunde kann angebotene Seminare (online) buchen."

3. „Ein Kunde bezahlt gebuchte Seminare mittels Online-Zahldienst."

Das Beschreiben aller Geschäftsaktivitäten durch Features erfordert auch an dieser Stelle neben den IT-Architekten die Domänenspezialisten, die das fachliche Wissen besitzen, um Anforderungen auf den Punkt genau zu definieren. Nachdem alle vorhandenen Geschäftsaktivitäten durch Features ausgedrückt wurden, wird eine Liste erstellt, die diese nach ihrer Relevanz für die zu erstellende WOA ordnet. Dies ermöglicht es im Verlauf eines Projektes auch, dass weniger relevante Features, beispielsweise aus Kostengründen, verworfen werden.

An dieser Stelle wird auch eine Aufwandsschätzung vorgenommen. Da das Umsetzungsziel eine WOA ist, lassen sich dafür zusätzliche Aspekte beachten, die zu einer Vereinfachung des Vorgehens führen können. Dazu gehören beispielsweise die Platzierung eines benötigten Services für ein Feature, der Kommunikationsaufwand in einem Feature aufgrund von einzubindenden Altsystemen oder die möglicherweise benötigte Realisierung als ausfallsichere Variante durch eine redundante Architektur. Die Durchführung eines solchen Schätzspiels wird in Kapitel 6.2.2 näher erläutert. Für die drei oben genannten Features lassen sich beispielhaft die folgenden Aufwandspunkte schätzen: Das erste Feature, bei dem der Kunde online Seminare betrachten kann, benötigt eine Art *Content-Management-System*, welches aber leicht zu der bestehenden Webseite der *BBI GmbH* hinzugefügt werden kann. Daher schätzen die

Experten auf fünf Aufwandspunkte. Das Online-Buchen sowie Online-Zahlen schätzen die Experten auf jeweils 10 Aufwandspunkte, da hierbei zum einen jeweils ein XaaS-Dienst eingebunden und zum anderen mehrere Kommunikationswege zwischen der Seminar-Plattform und den Diensten realisiert werden muss. Die Umrechnung von Aufwandspunkten in Personentage ist von FDD nicht vorgegeben und muss daher auf der Erfahrung von Experten oder einer neu getroffenen Einschätzung beruhen. An dieser Stelle wird einfach von einer direkten Entsprechung ausgegangen: Ein Aufwandspunkt entspricht einem Personentag.

Mit Abschluss des Schätzspiels sollten alle Features nicht nur in ihrer Reihenfolge, sondern zusätzlich mit ihrem Aufwand bekannt sein. Es ergibt sich dadurch eine Gesamtanzahl von umzusetzenden Aufwandspunkten für das Projekt. Diese lässt sich bei vorgegebener Laufzeit und der Prämisse einer kontinuierlichen Abarbeitung der Aufwandspunkte durch ein Feature-Burndown-Diagramm anschaulich darstellen. Dadurch kann der Projektfortschritt einer FDD-Umsetzung bildlich ausgedrückt werden und unterstützt die Projektleitung bei der Überprüfung und Anpassung der Projektplanung (siehe Abbildung 5.41). Das Diagramm zeigt einen beispielhaften Aufwand von 10.000 Punkten, der in einer Zeitspanne von 25 Kalenderwochen abzuarbeiten ist. Die rote Linie zeigt dabei die verbleibenden Aufwandspunkte je Kalenderwoche, wenn von einer gleichbleibenden Abarbeitung ausgegangen wird. Das Beispiel zeigt hier den Verlauf der Arbeiten bis zur 13. Kalenderwoche und teilt durch die blaue Linie mit, dass zu diesem Zeitpunkt zu wenige Aufwandspunkte abgearbeitet sind, da diese knapp über der roten Linie liegt.

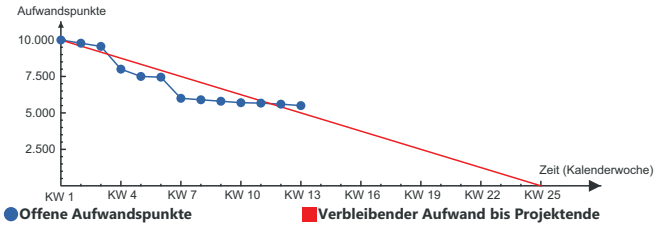


Abbildung 5.41: Feature-Burndown-Diagramm für FDD.

5.7.3 Planung von Features

Nach der reinen Listenform muss nun die tatsächlich durchzuführende Realisierungsreihenfolge der Features geplant werden. Für eine WOA ist hierbei zu beachten, dass XaaS-Angebote für die Bereitstellung von Infrastruktur eingebunden werden, bevor diese in Features Verwendung finden. Wie für den Ablauf von FDD vorgeschrieben, werden hier Zusammenhänge zwischen den Features überprüft und die Realisierungsabfolge darauf abgestimmt. Obwohl zu diesem Zeitpunkt noch immer keine Identifizierung von Services stattfand, sollten alleine aufgrund der Beschreibungen der Features und deren fachlicher Zugehörigkeit Zusammenhänge zwischen diesen zumindest abgeschätzt werden können. Jedes Feature bzw. jede Geschäftsaktivität bekommt im Verlauf der Planung ein Fertigstellungsdatum zugewiesen, welches durch Monat und Jahr einen spätesten Zeitpunkt festlegt, an dem das Feature realisiert sein muss. An dieser Stelle ist neben den Verzahnungen der einzelnen Features auch die Auslastung des beteiligten Personals zu beachten. Darüber hinaus können an dieser Stelle auch Meilensteine, Release-Zeitpunkte und Kontrollpunkte eingeplant werden.

Nach dieser Phase lässt sich zusätzlich zu Feature-Burndown-Diagrammen eine weitere Diagramm-Art für die Leitung des Projekts, und nicht zuletzt auch für die Programmierer, als Informationsquelle des aktuellen Fortschritts im Projekt einsetzen: die Parking-Lot-Diagramme. Diese Diagramme bündeln jeweils ein Feature-Set und stehen für einen Prozess der Planungsphase. Sie bilden auf kompaktem Raum die relevanten Planungs- und Fortschrittsdaten ab. Während des Projekts können diese Diagramme bei fortschreitender Entwicklung ausgedruckt werden. Sie erlauben den Beteiligten einen schnellen Überblick über den aktuellen Projektstatus.

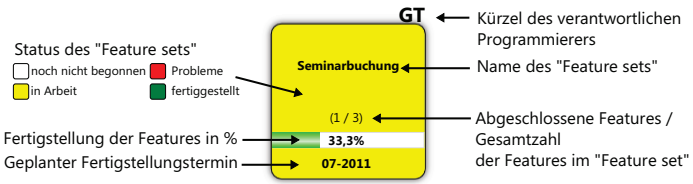


Abbildung 5.42: Parking-Lot-Diagramm für FDD.

In dieser Phase wird kein Domänenspezialist benötigt. Die Planung der Reihenfolge und der Belastung des IT-Personals wird durch die Projektleitung, die Entwicklungsmanager und die Chefprogrammierer durchgeführt. Nach Abschluss dieser Phase ist die Planung des Projekts beendet und ein grober zeitlicher Ablauf der Realisierung steht fest. Die daraus entstandenen Dokumentationen enthalten wenige Details und dienen vor allem als richtungsweisende Zielvorgabe für die Umsetzung, die es dann in jedem Feature zu verfeinern gilt. Darüber hinaus lässt sich damit die Aufwands- und Kostenplanung unterstützen.

5.7.4 Entwurf und Konstruktion von Features

Nach der recht kurzen Planungsphase folgt gemäß FDD-Methode nun eine Entwurfs- und Entwicklungsphase, die je Feature maximal zwei Wochen benötigen soll. Durch die fehlende Detailplanung muss nun innerhalb kurzer Zeit, also in wenigen Tagen, die Planung der Umsetzung des Features erfolgen. Inwieweit hierbei die Detailtiefe des zuvor beschriebenen Vorgehens erreicht werden kann, ist fraglich. Im Grunde ist dies aber auch nicht gefragt. Dem agilen Manifest folgend, ist funktionierender Code wichtiger als dessen Dokumentation. Daher muss die Dokumentation an dieser Stelle auch nicht weiter ausgeführt werden als unbedingt nötig. Inwieweit man daher den Vorgaben der zuvor beschriebenen WOA-Methode folgt und wie ausführlich die Dokumentation von Prozessen, Services und Anforderungen ausgestaltet wird, hängt u. a. von der Projektgröße und der Größe des Unternehmens ab. Während für ein größeres Unternehmen eine ausführliche Dokumentation von großer Wichtigkeit sein kann, genügt einem Start-up-Unternehmen ggf. ein funktionierendes System.

Entwirf je Feature

Innerhalb der kurzen Detailplanung wird ein Domänenspezialist für den jeweiligen Fachbereich des Features in Anspruch genommen, damit die Abläufe innerhalb des Features geklärt werden können. An dieser Stelle werden nun auch Teile der zuvor vernachlässigten Phasen 2, 3, 4 und 6 durchgeführt. Mit dem Fokus auf ein Start-up-Unternehmen, dem ein Service-Katalog in ITIL-Form in der Anfangsphase wenig Nutzen stiftet, werden hier alle zuvor beschriebenen und geforderten Dokumentationen reduziert. Die einzelnen Phasen laufen

zwar nach dem gleichen Muster ab, dienen aber primär dazu, die Funktionalität direkt im Anschluss umsetzen zu können.

Das Planungsteam verfeinert nun zunächst die vorliegenden Modelle der Geschäftsaktivitäten und erstellt Ablaufdiagramme in Form von BPMN-Prozessen. Gleichzeitig werden die darin genutzten Services identifiziert und die Anforderungen an diese festgelegt. Den Anforderungen von FDD genügt das folgende BPMN-Modell (siehe Abbildung 5.43). Hier wird aus der Domäne *Weiterbildung*, des Fachgebiets *Seminargeschäft* die Geschäftsaktivität *Seminarbuchung* dargestellt. Die hieraus ersichtlichen Informationen sind der chronologische Ablauf einer Buchung und die daran beteiligten Akteure.

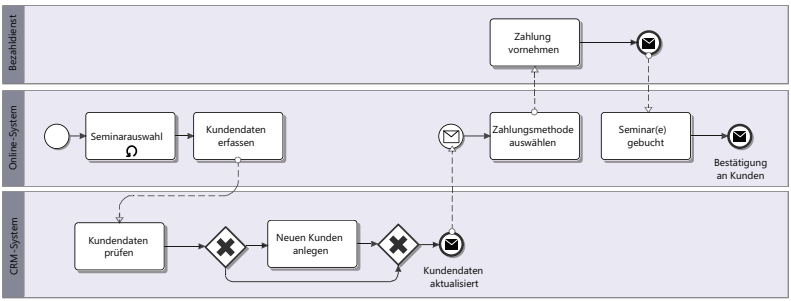


Abbildung 5.43: BPMN-Modell zur Seminarbuchung.

Der Service-Katalog mit fachlicher und technischer Sicht auf das Service-Portfolio eines Unternehmens wird hier beispielsweise als einfache Liste mit potenziellen XaaS-Angeboten geführt. Bei dem hier vorgestellten Beispiel wird der CRM-Anbieter salesforce.com sowie der Zahlungsdienst PayPal als XaaS-Anbieter ausgesucht. Für das Online-System soll dagegen eine eigene kleine Plattform auf Open-Source-Basis angepasst werden.

Bei der Notfall- und Alternativenplanung kann ad hoc entschieden werden, ob eine redundante Ausführung notwendig ist. Diese Entscheidung wird auch davon beeinflusst, welche Erfahrungen die Entwickler bei der Umsetzung des Features machen. Ggf. merkt man bereits bei der Implementierung, dass ein XaaS-Dienst nicht gut skaliert und man direkt erweiterte Strukturen oder sogar einen anderen XaaS-Anbieter benötigt. Sinnvoll wäre hierbei zunächst die Umsetzung eines ersten funktionierenden Features, um dieses anschließend

redundant aufzubauen. Agil bedeutet hierbei immer auch, dass man jederzeit auf geänderte Anforderungen oder Randbedingungen reagieren kann und soll. Im Gegensatz zu klassischen Vorgehensmodellen, in denen strikt nach der geplanten Struktur vorgegangen wird, ist bei agilen Methoden eine Veränderung jederzeit möglich und sogar notwendig. Dies wird auch durch den vierten Punkt des agilen Manifests „Reaktion auf Veränderung" ausgedrückt (siehe Kapitel 2.1).

Eine retrospektive Betrachtung der im Grundlagenteil vorgestellten Werte des eXtreme Programmings, wie Kommunikation und Mut, macht an dieser Stelle deren Notwendigkeit bei der Durchführung eines agilen Projekts klar. Als Beispiel sei die Auswahl an XaaS-Anbietern und die mit diesen gemachten Erfahrungen genannt, die von jedem Team unbedingt kommuniziert werden müssen, um mehrfache Arbeit zu reduzieren. Was bei traditionellen Vorgehensweisen durch SLRs oder Anforderungsspezifikationen klar geregelt ist, muss nun Stück für Stück aufgebaut werden. Eine stockende oder fehlerhafte Kommunikation zwischen den Teammitgliedern kann dabei negative Auswirkungen auf den Fortschritt des Projektes haben.

Entwickle je Feature

Die verbleibende Zeit der maximal zwei Wochen für die Umsetzung eines Features wird durch das tatsächliche Umsetzen der Feature-Funktionalität ausgefüllt. Hierbei gilt, dass zunächst Testfälle für die umzusetzenden Features geschrieben werden sollten, um deren korrekte Funktion zu gewährleisten. Gerade bei einem System, welches nun Stück für Stück wächst und keine komplette Planungsphase durchlief, sind Testfälle immens wichtig für das darauffolgende fehlerfreie Zusammenführen von Programmteilen. Die konkrete Umsetzung der einzelnen Features ist, wie bereits in Kapitel 5.4 ausgeführt, abhängig von der gewählten Ausprägung einer WOA, den zur Verfügung stehenden Programmiersprachen innerhalb eines Controllers und weiteren Randbedingungen. Die Umsetzung kann hierbei von der Modellierung von Workflows innerhalb eines Controllers bis hin zur konkreten Implementierung zusätzlich notwendiger Logik oder eigener WP reichen. Wenn ein Feature abgeschlossen ist, wird dieses im Normalfall von einem Chefprogrammierer abgenommen und das Programmierteam kann mit der Realisierung des nächsten Features beginnen.

Diese letzten zwei Phasen, der Detailentwurf und die konkrete Umsetzung von Features, werden für jedes geplante Feature-Set wiederholt, bis das System fertiggestellt ist. Durch eine kontinuierliche Integration der fertigen Features liegt dabei zu jedem Zeitpunkt ein funktionsfähiges IT-System vor.

5.8 Umsetzung eines WAC mittels XaaS

Der WAC bildet die Kernkomponente einer WOA und wurde als theoretischer Ansatz in Kapitel 3.2.4 vorgestellt. An dieser Stelle soll nun eine tatsächliche Umsetzung erläutert werden, die die Funktionalitäten eines Controllers bereitstellt. Die erste Herausforderung, mit der man hierbei konfrontiert wird, ist die Suche nach einem passenden XaaS-Angebot, welches im Idealfall alle Anforderungen an einen Controller erfüllt.

WAC-Architektur

Die an einen Controller gestellten Anforderungen werden an dieser Stelle in Kategorien zusammengefasst, um daraufhin möglichst passende XaaS-Angebote zu identifizieren. Wichtig ist hierbei vor allem auch eine umfassende API des Anbieters, damit die Funktionalitäten möglichst automatisiert miteinander zu koppeln sind. Zu nennen sind hier die folgenden Kernmodule eines Controllers, die durch potenzielle Anbieter aus dem Internet umzusetzen sind.

Überwachung und Protokollierung Das Überwachen betrifft hier nicht nur die Prozesse, die in einem Controller ablaufen sollen, sondern ebenso die SLAs der in einer WOA verwendeten Services. Dazu muss ein Anbieter ausgewählt werden, der gleichzeitig sinnvolle Messgrößen diverser Dienste überwacht und Unregelmäßigkeiten protokolliert. Im Idealfall bietet dieser auch noch eine sinnvolle Benachrichtigungsfunktionalität an, damit im Fehlerfall das zuständige IT-Personal unmittelbar alarmiert werden kann. Der in Kapitel 5.5.1 bereits erwähnte Anbieter *Site24x7.com* ist hierbei als gute Wahl zu betrachten, da dieser umfassende Überwachungsoptionen anbietet.

Service-Verzeichnis Ein Service-Katalog, der zum einen alle in der WOA verwendeten Services verzeichnet und darüber hinaus auch deren Verwendung in fachlichen Prozessen abbilden sollte, also einen Service-Katalog im ITIL-Sinne

abbildet, lässt sich, wenn man eine dynamisch anpassbare Struktur aufbauen möchte, am einfachsten mit DaaS-Angeboten realisieren. Hier bieten sich beispielsweise Amazon SimpleDB, Database.com (ein Angebot von Salesforce) oder Zoho Creator als günstige Anbieter mit einer umfassenden API an. Der im nächsten Schritt ausgewählte Anbieter der Workflow-Engine beinhaltet aber bereits einen Service-Katalog, der zwar keine echte API zur Abfrage der Dienste für die Überwachung derselbigen bereitstellt, jedoch über einen RSS-Feed abgefragt werden kann. Dadurch ist keine zusätzliche Einrichtung eines Service-Katalogs notwendig.

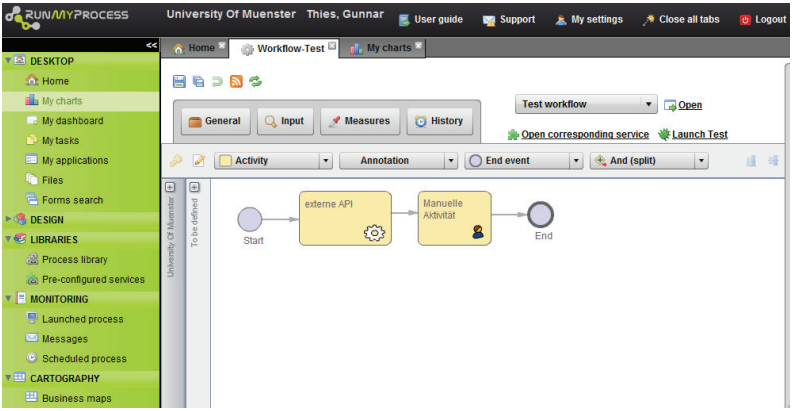
Sicherheit, Verbindungen, Transformation und Prozessablaufsteuerung Diese vier Anforderungen sind eng miteinander verbunden. Das Verwenden von WP jedweder Art und die ggf. notwendige Transformation von Nachrichten in andere Formate zur Verwendung mit Diensten sollten möglichst innerhalb der Prozessablaufsteuerung erfolgen können, da dort der Aufruf und die Verarbeitung von Daten stattfinden. Sind spezielle API-Schlüssel oder Zertifikate für die Nutzung von WP notwendig, so müssen diese auch innerhalb eines Workflows zur Verfügung stehen. Sollten diese Funktionalitäten nicht in einem der angebotenen XaaS-Dienste enthalten sein, so bleibt im Zweifel noch der Umweg über eigens implementierte Services, die beispielsweise eine Transformation durchführen und die transformierten Daten zurückgeben.

Bei der Wahl einer Variante ist u. a. darauf zu achten, ob bestehende Prozessmodelle existieren, die weiterverwendet werden sollen. Aktuell[56] sind nur wenige XaaS-Anbieter in der Lage, bestehende Prozessmodelle in deren Plattform zu übernehmen. So ist es beispielsweise möglich, BPMN-Prozesse in die Intalio BPMN-Software zu importieren. Diese bewirbt sich selbst als PaaS, muss aber heruntergeladen und auf einem eigenen Server installiert werden. Die meisten kommerziellen Anbieter verwenden eigene Workflow-Sprachen, die oft eine einfachere Struktur haben, als BPMN oder BPEL. Möchte man BPEL-Prozesse ausführen, so findet sich kein Anbieter, der dies innerhalb eines XaaS-Dienstes ermöglicht. Interessant ist dagegen, dass die meisten installierbaren Workflow-Engines, seien es kommerzielle Produkte von IBM, Oracle, Microsoft oder Open Source-Projekte wie Apache ODE, auf BPEL setzen.

Um eine Workflow-Steuerung im Controller zu ermöglichen, lassen sich diverse vorhandene PaaS-Anbieter identifizieren. Wie bereits erwähnt, ermögli-

[56] Stand: 10.01.2011

chen diese oft nur die Verwendung eigens definierter Workflow-Sprachen, die sich dafür aber meist sehr komfortabel im Browser bedienen lassen und auf die Fähigkeiten der Plattform abgestimmt sind. Anbieter wie *force.com*, *LongJump* oder *RunMyProcess* sind Beispiele hierfür. Die Plattform *force.com*, die von einem der größten SaaS-Betreiber salesforce.com bereitgestellt wird, ermöglicht mit dem eigenen *Visual Process Manager* eine Möglichkeit aus sechs verschiedenen Prozessteilen, *Form*, *Decision*, *Lookup*, *Question*, *Statement* und *Sub Process* einen Workflow zu erstellen. Dabei werden aber lediglich die Plattform-eigenen Aktionen unterstützt und keine einfache Möglichkeit zur Orchestrierung von externen Services angeboten[57]. Der Anbieter LongJump ermöglicht durch den eigenen *Workflow Designer* ebenfalls die Umsetzung von Geschäftsprozessen innerhalb der Plattform. Dieser ist allerdings sehr rudimentär und erlaubt nur die Definition von Status, Aktionen und Entscheidungen. Damit ist ein solcher Workflow nicht annähernd so ausdrucksstark wie ein BPEL- oder BPMN-Prozess.



Quelle: `http://www.runmyprocess.com`

Abbildung 5.44: Die Prozessmodellierungs-Ansicht von *RunMyProcess*.

Es gibt aber auch positive Beispiele: Der Anbieter *RunMyProcess* wurde speziell zur Umsetzung und Verwaltung von Geschäftsprozessen im Internet kon-

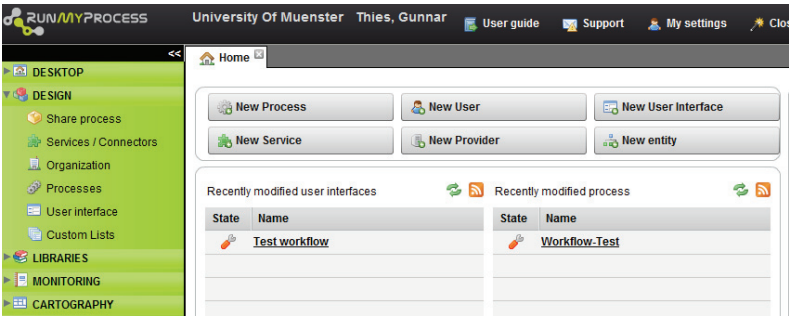
[57] Durch die Verwendung von Apex-Code lassen sich hier beispielsweise Web Services aufrufen und darüber einbinden.

zipiert und bietet eine Fülle an Funktionen, die wie geschaffen für eine WOA erscheinen. Neben dem Einbinden von externen Services wird auch eine Überwachungsoption (*Monitoring*) und das Erstellen von Auswertungen (*Reports*) über die Ausführung von Workflows ermöglicht. Darüber hinaus lassen sich bereits fertige Prozesse aus der Community mit einbeziehen, die bereits das Aufrufen der Grundfunktionalitäten von Google, Salesforce oder Zoho über die jeweilige API umsetzen. Auch verschiedene Möglichkeiten der Verifikation bei externen APIs, sei es über *Basic* oder *Digest Authentication*, dem Authentifizierungsmechanismus von Amazon oder Googles Implementierung von *OAuth* sind hierbei vorgesehen. Für manuelle Aktivitäten in Workflows lassen sich dann spezielle Benutzerschnittstellen in Form von HTML-Seiten erstellen oder durch API-Aufrufe von anderen Plattformen aus steuern. Die Workflow-Sprache orientiert sich bei RunMyProcess an BPMN 2.0, auch wenn nicht der gesamte Sprachumfang vorhanden ist. Wichtige Task-Typen, wie beispielsweise Service-Aufrufe oder manuelle Aufgaben, lassen sich hiermit aber umsetzen und ermöglichen so eine ganz konkrete Modellierung von Geschäftsprozessen mit menschlichen Akteuren. So kann hier beispielsweise bei dem Erreichen eines manuellen Tasks eine E-Mail an den entsprechenden Mitarbeiter versendet werden, damit dieser seine Task-Liste abarbeiten kann. Ein Überblick über den Workflow-Editor ist in Abbildung 5.44 dargestellt.

Für die Umsetzung eines Controllers, wie in Kapitel 3.2.4 vorgestellt, ist dieser Anbieter sehr gut geeignet. Mit Ausnahme des Moduls „Überwachung und Protokollierung", welches auch externe Services überprüfen soll, lassen sich alle Funktionalitäten eines WAC umsetzen. Auf der Startseite von *RunMyProcess* lassen sich übersichtlich Prozesse, Benutzer, Services, Service-Anbieter sowie Benutzerschnittstellen neu anlegen und zuletzt geänderte Prozesse oder Schnittstellen auf einen Blick feststellen (vgl. Abbildung 5.45).

Zusätzlich können Komponenten, wie beispielsweise einzelne Berichte über Workflow-Instanzen oder Statistiken, als HTML-Komponenten erstellt und dann als *iframe* in eine beliebige Webseite eingebaut werden. Dabei bietet *RunMyProcess* viele Widgets, die per Drag and Drop zu einer HTML-Komponente zusammengebaut werden können. Der Umfang und die eingebauten Funktionen der Plattform sind sehr vielseitig und wenig kostenintensiv.[58] Das Pariser Unternehmen betreibt ihr Angebot auf Amazons Cloud Computing Netzwerk, so dass dynamische Skalierbarkeit und Lastausgleich quasi

[58] Eine Einzel-Lizenz beginnt bei 30 € pro Jahr.



Quelle: `http://www.runmyprocess.com`

Abbildung 5.45: Der Startbildschirm des Anbieters *RunMyProcess*.

garantiert sind. Darüber hinaus gibt das Unternehmen an, den EU-Richtlinien der Datensicherheit zu entsprechen. So sollte dem tatsächlichen Einsatz in einer produktiven WOA nichts im Wege stehen.

Zusätzliche Steuerungskomponente Zusätzlich zu den genannten XaaS-Services für die Basisfunktionalitäten eines Controllers wird eine rudimentäre Steuerungskomponente für den Überwachungsdienst sowie die Prozesssteuerung in einem virtuellen Server selbst implementiert. Diese dient als Dashboard für den gesamten Controller und sollte bei einem kommerziellen Workflow-as-a-Service-Anbieter eine eigene API anbieten, durch die sich der Controller steuern lässt. In diesem beispielhaften Szenario wird von einem einzigen Controller für die umzusetzende WOA ausgegangen, so dass ausnahmsweise keine API bereitgestellt werden muss. Das Dashboard bezieht hierbei diverse Komponenten aus RunMyProcess und bietet so immer einen Überblick über aktuell laufenden Workflows.

Eine Programmierung in geringem Umfang ist an dieser Stelle dennoch erforderlich. Der RSS-Feed mit den von RunMyProcess verwendeten Services muss ausgewertet und an die API von Site24x7.com geschickt werden, um eine ständige Überwachung der in den Workflows verwendeten Services zu gewährleisten. Die Liste der in RunMyProcess verwendeten Services enthält dabei weitere Verknüpfungen auf die konkreten Service-Beschreibungen, denen man folgen kann, um einen einzelnen Service zu betrachten.

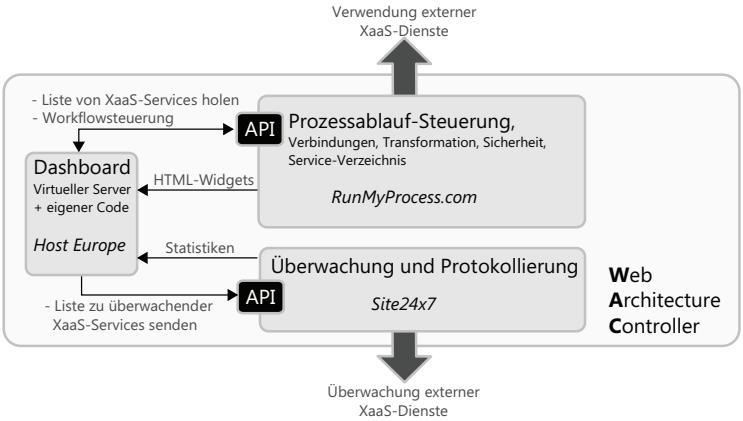


Abbildung 5.46: Beispielhafte Umsetzung eines WAC durch eine XaaS-Kombination.

Aus diesen XML-Daten lassen sich dann wichtige Informationen über die verwendeten Services erfahren und diese für die Überwachung mittels Site24x7.com verwenden. Möchte man darüber hinaus eine ständige Kostenkontrolle über die verwendete XaaS-Nutzung erstellen, lässt sich an dieser Stelle ein Programm realisieren, welches die Workflow-Historie als RSS-Feed aus RunMyProcess abruft, die verwendeten Services identifiziert und anhand von hinterlegten Kostenregeln die aktuell angefallenen Kosten berechnet. Die konkrete Ausführung des Codes wird jedoch an dieser Stelle nicht weiter vertieft. Eine Architekturskizze der hier beschriebenen Controller-Realisierung ist in Abbildung 5.46 dargestellt.

6 Kosten einer Web-orientierten Architektur

Die Betrachtung der Kosten eines Projektes vor dessen Durchführung und das Berechnen von differierenden Szenarios ist eine grundlegende Vorgehensweise in Wissenschaft und Wirtschaft zur Bewertung von Handlungsalternativen. Damit soll eine Entscheidungsgrundlage geschaffen werden, auf der man versucht, die Vor- und Nachteile eines Projektes klar zu definieren, eine *make-or-buy*-Entscheidung zu treffen und eine gewisse Planungssicherheit durch konkrete Abschätzungen von in der Zukunft liegenden Kosten zu erhalten. Im Folgenden wird zunächst der Aspekt von Kosten bei IKT-Systemen anhand der verbreiteten *TCO*-Berechnung aufgegriffen und näher betrachtet, um anschließend auf die damit verbundenen Möglichkeiten der Kostenberechnung einer WOA einzugehen. Ein konkretes Modell für die Berechnung der Umsetzungs- sowie Betriebskosten einer WOA wird zum Ende des Kapitels zunächst konzeptionell und daraufhin anhand eines Web-basierten Prototyps vorgestellt.

6.1 Die TCO-Berechnung

Für die Berechnung von Kosten existieren unzählige Verfahren in der Betriebs- und Volkswirtschaft. Jedes ist mit eigenen Vor- und Nachteilen behaftet. Kosten lassen sich dabei für unterschiedliche Anwendungsbereiche spezifizieren. Das TCO-Verfahren wird verwendet, um die Kosten von Investitionsprojekten zu beziffern. Die Grundidee ist hierbei, nicht nur die Anschaffungskosten eines Investitionsguts zu betrachten, sondern auch die zusätzlichen indirekten Kosten abzuschätzen, die im laufenden Betrieb anfallen. Diese können bei IT-Systemen tatsächlich bis zu 60 % der Gesamtkosten ausmachen (vgl. [Krc05]). Das ursprüngliche Modell der TCO-Berechnung geht zurück auf Bill Kirwin

(*Gartner Group*), der dieses Verfahren im Jahre 1987 für Microsoft entwickelte. Die folgende Definition von TCO wird durch die *Gartner Group* vertreten:[59]

> *"[TCO is] a comprehensive assessment of **information technology** (IT) or **other costs** across enterprise boundaries over time. For IT, TCO includes **hardware** and **software** acquisition, management and **support**, communications, end-user expenses, and the opportunity cost of **downtime**, training and other **productivity losses**."*

Nach dieser Auffassung ist die TCO-Berechnung als eine Bewertung über die Kosten von Informationstechnologie oder anderer Kosten über die Unternehmensgrenzen hinaus zu sehen. Speziell für den IT-Bereich, für den die Methode ursprünglich auch entwickelt wurde, werden laut der obigen Definition die folgenden Kostenbereiche betrachtet:

- Hard- und Softwarekosten
- Kosten der Administration und Hilfestellung für das IT-System
- Kosten für Kommunikationsbedarf
- Kosten für Endnutzer (beispielsweise Schulungen und Fortbildungen)
- Opportunitätskosten für Ausfälle, Training der Angestellten und weitere Produktivitätsminderungen

Während sich die Expertenmeinungen bei den Kostenbereichen der TCO weitgehend überschneiden, existieren für die konkrete Bewertung eines IT-Systems diverse Vorgehensmodelle. In einer vergleichenden Untersuchung von 20 Modellen zur allgemeinen TCO-Berechnung erfüllte das Modell der *Gartner Group* (vgl. [Gar03, Els05]) aus dem Jahre 2003 als Einziges 10 von 12 überprüften Qualitätskriterien (vgl. [GGW09]). Als Kriterien dienten hierbei u. a. der Umgang mit Kostenkategorien und -treibern sowie die möglichen Anwendungsbereiche. Dieses Modell wurde speziell für die Berechnung der Kosten verteilter Systeme konzipiert und während seiner Existenz immer wieder angepasst. Auch eine weitere Übersicht über TCO-Modelle in [TTS04] untersucht aktuelle Trends bei TCO-Verfahren und stellt die Rolle von Web Services bei der Berechnung von Kosten für IT-Systeme heraus. Die Konzepte der TCO-Berechnungsmethoden stammen dabei meistens aus der praktischen

[59] http://www.gartner.com/technology/research/it-glossary/#19_0

Anwendung und wurden von diversen großen Unternehmen der IT-Branche entwickelt und für deren Kunden umgesetzt (vgl. [WH00]). Daraus resultieren verschiedene Annahmen und Ausgestaltungen. In einem Aspekt stimmen jedoch alle Vorgehensmodelle überein: die Aufteilung in **direkte** und **indirekte** Kosten. Das TCO-Berechnungsverfahren dient hierbei vor allem den drei folgenden Zielen (vgl. [Gar03, GM05]):

1. Simulation der TCO-Berechnung für ein bestimmtes Unternehmen.
2. Aufstellung geplanter Kosten zur Gegenüberstellung mit den typischen (realen) Kosten.
3. Simulation für Kosten bei veränderten Szenarien, beispielsweise dem Einsatz von „Best practices" oder Komplexitätsreduktionen.

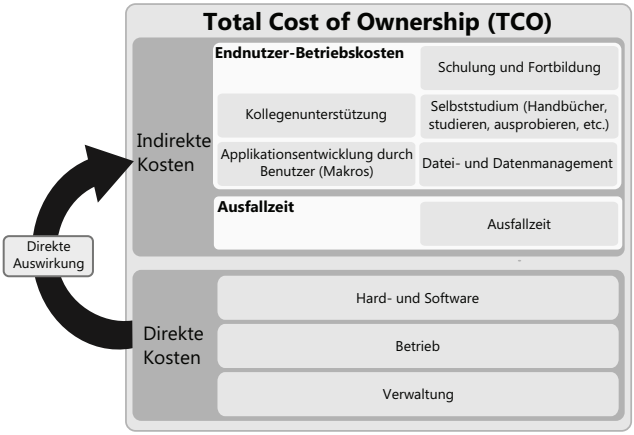
Dabei werden die Kosten im Vergleich zu manchen anderen Verfahren nicht nach Kostenstellen organisiert, sondern vielmehr vordefinierten Kategorien zugeordnet: den direkten und indirekten Kosten und deren einzelnen Kostenarten. Ein wichtiger Aspekt für die Vergleichbarkeit über Abteilungen oder sogar Unternehmen hinweg ist die Verwendung der gleichen Ein- und Zuteilung von Kosten. Ohne diese lassen sich keinerlei relevante Rückschlüsse oder Vergleiche aus den Berechnungsergebnissen ableiten. Außerdem sollte immer ein identischer Planungszeitraum angenommen werden, der beim TCO-Verfahren üblicherweise ein Jahr umfasst. Die Zusammensetzung der beiden Kostenarten wird aus Abbildung 6.1 ersichtlich.

6.1.1 Direkte Kosten

Direkte – budgetierte – Kosten fallen durch die Bereitstellung von IT-Systemen und IT-Dienstleistungen im Unternehmen an. Hierzu zählen typischerweise alle direkten Ausgaben für stationäre und mobile Endgeräte, Systemperipherie und Netzwerkkomponenten für die Systemumgebung sowie Ausgaben für die Bereitstellung des verteilten Systems. Diese Kosten lassen sich auf drei konkrete Teilbereiche aufteilen: *Hard- und Software*, *Betrieb* sowie *Verwaltung*.

Hard- und Software

Die Kosten für Hard- und Software verteilen sich auf vier Bereiche:



Quelle: Angelehnt an [Els05]

Abbildung 6.1: Übersicht der Kosten einer TCO.

- Hardware von Produktivsystemen
- Software im Sinne von Betriebssystemen, Applikationen, Datenbanken sowie weiteren Softwarewerkzeugen
- Nach- und Aufrüstung der (Produktiv-)Hardware
- Hard- und Software für die IT-Abteilung

Kosten für Hardware werden hierbei in Form von Ausgaben, Abschreibungen und auch als Leasing-Raten verzeichnet. Es ist zu beachten, dass neben Servern auch Client- und Netzwerk-Komponenten sowie Peripheriegeräte zur Hardware des Produktivsystems gezählt werden. Die Software-Kosten betreffen hierbei alle anfallenden Ausgaben, Leasing- oder Lizenzkosten für Betriebssysteme, Datenbank-Management-Systeme, Software-Werkzeuge sowie Groupware und Messaging-Software, die in direktem Zusammenhang mit dem verteilten System stehen. Die einzige Ausnahme bilden hier Eigenentwicklungen, die laut Gartner *nicht* zu den Softwarekosten gezählt werden. Im Gegensatz dazu werden im selben Modell jedoch Softwareanschaffungskosten als Abschreibungen oder Leasing-Kosten betrachtet. Diese Betrachtungsweise erscheint unlogisch, da so bei einer Entscheidung zwischen dem Kauf oder der eigenen Erstellung einer Software (*make-or-buy*-Entscheidung) die Eigenerstellung nicht, der Kauf jedoch voll auf die Kosten angerechnet wird. In einer

TCO-Betrachtung der SAP AG beispielsweise werden Implementierungskosten dagegen auch zu den direkten Kosten gezählt (vgl. [Sie04]). Personalkosten allerdings, die für Installation und Wartung der Software notwendig sind, werden nicht an dieser Stelle, sondern später den Betriebskosten zugerechnet.

Ein weiterer Kostenfaktor betrifft die Aufrüstung der Hardware von Produktivsystemen. Damit werden Kosten für Erweiterungen, Reparaturen oder Austausch von Hardware-Komponenten beziffert, die in einer ursprünglichen Planung des Systems nicht vorgesehen waren. Der letzte Kostenfaktor besteht aus Hard- und Software für die IT-Abteilung. Diese werden gesondert berechnet, da sie für das Garantieren des Ablaufs und die Wartung des verteilten Systems anfallen. Hierzu zählen Schulungssysteme und -software, Test- und Staging-Systeme sowie Werkzeuge zur Überwachung und Pflege des verteilten Systems.

Hard- und Software-Kosten werden üblicherweise als jährliche Planung vorgenommen und können aus diversen Quellen gesammelt werden. Dabei ist zu unterscheiden, ob eine Planung der zukünftigen Kosten oder eine Kostenkontrolle der vergangenen Perioden vorgenommen wird. Für die Planungsrechnung können beispielsweise Netzwerk-Diagramme, Preislisten von Komponenten, geplante Abschreibungspläne und die Budgetplanung eines IT-Systems herangezogen werden. In der Retrospektive können dann auch Leasing- und Kaufverträge, Kaufbelege, Rechnungen von Zulieferern sowie Inventurlisten und „Asset-Management"-Berichte als Informationsquellen dienen.

Betrieb

Die Kosten für den Betrieb eines verteilten Systems lassen sich in die folgenden vier Unterbereiche aufteilen, die alle den Personalkosten zuzuordnen sind. Dies betrifft Personal für:

- technische Dienstleistungen
- Prozesse und Planung
- Datenbank-Management und Administration
- Betrieb eines *Service-Desks*

Hierbei wird keine Unterscheidung zwischen internen und ausgelagerten Personalkosten getroffen. Die technischen Dienstleistungen umfassen ein großes Spektrum, welches in drei Teilbereiche untergliedert werden

kann: Client-seitige, Server-seitige sowie das Netzwerk betreffende Aufgaben. Im Folgenden ist eine Übersicht möglicher Dienstleistungen aufgelistet, wie sie in jedem der drei Teilbereiche auftreten können: technische Problemlösungen, Planung von Netzlasten, Leistungsregulierung, Benutzerverwaltung, Betriebssystem-Hilfe, Wartungsdienst, Software-Installation, Anwendungsverwaltung, Hardware-Konfiguration und -Installation, Festplatten- und Dateiverwaltung, Speicherkapazitätsplanung, Backup und Archivierung sowie Repository-Verwaltung.

Dem Bereich der Prozesse und Planung werden u. a. folgende Aktivitäten zugerechnet: Kundenverwaltung, Systemforschung, Planung und Produkt-Management, Evaluation und Beschaffung sowie Systemschutz. Diese Aktivitäten haben hierbei nichts mit der Planung im Erstellungsprozess eines Softwaresystems oder der Geschäftsprozessmodellierung des Vorgehensmodells zu tun, sondern sind Geschäftsaktivitäten des Unternehmens, beispielsweise zur Produktionsplanung.

Die Personalkosten für Management und Administration von Datenbanksystemen umfassen beispielsweise Indexverwaltung, Replikation, Log-Verwaltung, Daten-Wiederherstellung, Optimierung sowie weitere Wartungsaufgaben im Zusammenhang mit den Datenbanken des Unternehmens. Kosten der Entwicklung, Planung und Umsetzung von Datenmodellen für IT-Systeme werden hierbei je nach gewählter Methode entweder betrachtet oder vernachlässigt.

Ein Service-Desks, oft als Callcenter oder Hotline bezeichnet, hilft bei technischen Problemen, die dann meist durch den Nutzer selbst oder durch die Weiterleitung des Problems an das Personal der technischen Dienstleistungen behoben werden können. Dabei kann diese Hilfestellung per Telefon, E-Mail oder online geleistet werden. Die Überwachung der Effizienz eines Service-Desks wird hierbei durch Metriken vorgenommen. Üblich sind hierbei beispielsweise das Verhältnis von Anfragen zu Support-Mitarbeitern, die Anzahl monatlicher Anfragen, die Wartezeit pro Anfrage, die Dauer eines Vorgangs oder auch die Zeit bis zur Lösung des Problems (vgl. [Els05]).

Auch Personalkosten des Betriebs eines verteilten Systems sind jährlich zu planen. Als Quellen für die benötigten Informationen können prospektiv Organigramme der IT-Abteilung, Lohn- und Gehaltsdaten der Personalabteilung und Managementinformationen der IT-Abteilung verwendet werden. Für die retrospektive Betrachtung sind Zeitabrechnungen von erbrachten Dienst-

leistungen, Outsourcing- und Wartungsverträge sowie Metriken des Service-Desks hilfreich.

Verwaltung

Verwaltungskosten im Sinne der TCO-Berechnung bestehen aus drei Bestandteilen:

- Buchhaltung und Verwaltung
- Endnutzer-Schulungen
- IT-Abteilungs-Schulungen

Innerhalb der Buchhaltung und Verwaltung werden Dienstleistungen abgerechnet, die sich direkt mit dem verteilten System in Verbindung bringen lassen. Hierzu gehören beispielsweise die Assistenz für die Verwaltung der IT-Abteilung, die Anlagegüterverwaltung, die Budgetierung, das Durchführen von Audits, das Beschaffungswesen, die Vertragsverwaltung und vieles mehr. Die Schulungsaktivitäten werden getrennt nach IT-Abteilung und Endnutzer für die Erstellung sowie Durchführung von Schulungsmaßnahmen aufgenommen.

Die Daten zur Kostenermittlung lassen sich hierbei für die Planung vorausblickend aus Organigrammen der IT-Abteilung, Gehaltsdaten der Personalabteilung sowie Abteilungsbudgets schätzen. Rückblickend lassen sich zusätzlich dazu Unterlagen über unternehmensinterne Schulungen sowie externe Schulungsverträge verwenden.

6.1.2 Indirekte Kosten

Die indirekten – unbudgetierten – Kosten stehen im Zusammenhang mit den direkten Kosten, da sie nur anfallen können, weil ein IT-System vorhanden ist. Dabei werden im Grunde zwei Kostenarten unterschieden: die anfallenden Betriebskosten der Endnutzer und die Opportunitätskosten von Ausfallzeiten.

Die indirekten Kosten eines verteilten Systems sind somit eine Maßstab für Effizienz: Je weniger Kosten in die (Selbst-) Schulung, die Kollegenunterstützung, die Applikationsentwicklung und die Ausfallzeit fließen, desto effizienter ist die Bereitstellung des IT-Systems. Ein Problem der indirekten Kosten ist dabei, diese zu beziffern, da sie nur schwer messbar sind. Ein probates Mittel zur Bestimmung der indirekten Kosten ist daher die Befragung der Mitarbeiter

über deren jeweilige Zeitaufwände bei der Aufgabenerfüllung im Zusammenhang mit dem zu bewertenden IT-System (vgl. [Els05]).

Endnutzer-Betriebskosten

Wie auch alle vorhergehenden Arten der direkten Kosten lassen sich die Endnutzer-Betriebskosten in Teilbereiche untergliedern:

- Kollegenunterstützung
- Applikationsentwicklung (beispielsweise Makros)
- Schulung und Fortbildung
- Selbststudium
- Datei- und Datenmanagement

Kollegenunterstützung geschieht meist durch einen inoffiziellen Experten innerhalb einer Abteilung oder eines Teams, der tiefere Kenntnisse über einen bestimmten Bereich besitzt, und daher Kollegen helfen kann, Probleme mit dem IT-System zu lösen. Dies kann die Hilfe bei Problemen mit Anwendungen, aber auch die Wartung eines Endgeräts, Installation von Software, die Durchführung von Schulungen oder das Erstellen und Verwalten von Sicherheitskopien beinhalten.

Die Applikationsentwicklung ist in diesem Zusammenhang nicht als Softwareentwicklung im eigentlichen Sinne zu verstehen, sondern bedeutet das Erstellen von Hilfsmitteln für nicht geschäftskritische Anwendungen. Dies sind beispielsweise Makros für Excel oder Skripte für Office-Produkte, die die Arbeit erleichtern und von einem Nutzer eines Endgeräts selbst erstellt werden können.

Um Nutzer in der Bedienung eines IT-Systems zu unterweisen, werden intern oder extern Schulungen oder Fortbildungen durchgeführt, die die Nutzer besuchen. Im Gegensatz zu den Verwaltungskosten, in denen die Kosten der Endnutzer-Schulungen anhand der Ausgaben für den Dozenten und Räumlichkeiten bereits angesetzt werden, sind hier die Kosten für die Zeit des einzelnen Nutzers zu berechnen, in der dieser an einer Fortbildungsmaßnahme teilnimmt. Dies sind quasi die Opportunitätskosten des Nutzers, da dieser während der Schulungszeit nicht produktiv arbeiten kann.

Die Kosten für das Selbststudium gehören zur selben Kategorie von Kosten, da der Nutzer in dieser Zeit selbstständig Handbücher liest, im Internet nach

Anleitungen oder Problemlösungen sucht, oder auch durch „Trial and Error" versucht, ein IT-Problem zu lösen. Gerade diese Kostenverursacher lassen sich lediglich durch die Befragung der Nutzer ermitteln, da darüber keine Unterlagen oder Belege geführt werden können.

Der letzte Punkt der Endnutzer-Betriebskosten ist das Datei- und Datenmanagement, welches die Arbeitszeit beziffert, in denen der Nutzer Daten und Dateien anordnet, sortiert, sucht oder sichert.

Um die Daten zur Kostenermittlung zu erheben, können in dem Bereich der Endnutzer-Betriebskosten prospektiv lediglich die Gehaltsdaten der Personalabteilung und die pro Mitarbeiter geschätzte Zeit, die in Schulungen verbracht wird, betrachtet werden. Retrospektiv lassen sich diese Daten durch Endnutzer-Interviews und -Umfragen, sowie Nachweise besuchter Schulungen verifizieren und korrigieren.

Ausfallzeit

Die letzte Kostenkategorie bewertet die Ausfallzeit des betrachteten IT-Systems. Hierbei werden die Kosten ermittelt, die durch Systemausfälle und den Wiederanlauf des Systems auftreten. Es handelt sich also um die Zeit, in der ein Mitarbeiter nicht produktiv arbeiten kann, da eine Komponente oder das ganze System ausgefallen ist. Dies schließt Desktop-PCs, Server, Netzwerke, Drucker oder auch Anwendungen mit ein. Hierbei werden geplante und nicht geplante Ausfälle berechnet. Aber lediglich die in der Zeit auftretenden Personalkostenverluste werden betrachtet. Diese stellen den Teil des gezahlten Gehalts dar, der während der Ausfallzeit an die Mitarbeiter, die nicht produktiv tätig sein konnten, ausgezahlt wurde. Potenzielle Unternehmensverluste, die durch die Untätigkeit zusätzlich entstehen, werden hierbei außer Acht gelassen. Die Ausfallzeit wird ebenfalls als jährlicher Kostenblock gebucht.

Verlässliche Daten über die Ausfallzeit und vor allem die dadurch entstandenen Produktivitätsminderungen sind oft schwierig zu ermitteln. Beispielsweise kann es bei dem Ausfall eines Mailservers zwischen 20 und 23 Uhr passieren, dass kein Nutzer betroffen ist und somit keine Kosten der Unproduktivität entstehen. Daher werden Nutzerumfragen durchgeführt und Protokolle der IT-Abteilung über die überwachten Systemkomponenten sowie Protokolle des Service-Desks ausgewertet, um möglichst verlässliche Daten für die Kostenberechnung zu erhalten.

In Tabelle 6.1 werden alle zuvor vorgestellten Kostenkomponenten aufgelistet. Dazu werden Quellen für die Informationsgewinnung aus pro- und retrospektiver Sicht genannt, die der Kostenberechnung der TCO zu Gute kommen.

6.1.3 Kritik am TCO-Ansatz

Ein Nachteil des TCO-Ansatzes ist in der Nicht-Vergleichbarkeit der verschiedenen TCO-Ergebnisse zu sehen, wenn diese durch unterschiedliche Methoden ermittelt werden. Die Unterschiede resultieren aus den getroffenen Annahmen und den Kostenzurechnungen einzelner Methoden, die meist nicht übereinstimmen und somit relativ große Unterschiede in den Gesamtkosten errechnen. Teilweise ist auch die Eignung für eine bestimmte Branche generell infrage zu stellen (vgl. [GGW09]). Ein weiterer Nachteil ist in der reinen Kostenbetrachtung zu sehen, die keine Wertzuflüsse betrachtet und damit keinen echten Vergleichswert (wie beispielsweise der *Return on Investment* (ROI)) anbietet (vgl. [WH00]). Ansätze, um dieses Manko auszugleichen, sind in [Bro07] zu finden. Darin wird ein vollständiger Finanzplan (VOFI) (vgl. [Gro06]) vorgeschlagen, um den Lebenszyklus einer Investition in Perioden aufzuspalten und damit die TCO zu berechnen. Der Vorteil hierbei liegt in der Berücksichtigung der Einnahmen und Überschüsse sowie Steuern, die in einem VOFI dargelegt werden können. Damit werden einige Nachteile der TCO-Berechnung ausgeglichen. Diese Erweiterung ist aber für die spezielle TCO-Berechnung eines verteilten Systems kaum umsetzbar, da sich Schwierigkeiten bei der Zuordnung von Wertzuflüssen zu einem IT-System oder Teilen davon ergeben.

Bei dem Berechnen von Investitionsalternativen mit mehrperiodigem Planungshorizont – also auch der Entscheidung, zwischen einer SOA- oder WOA-Realisierung eines IT-Systems – dürften Zins- und Steuereffekte für einen ganz konkreten Vergleich auf monetärer Basis nicht ignoriert werden. Der VOFI bietet hier das geeignete Werkzeug, um die Berechnung möglichst stringent zu bewerten. Auf der Grundlage des VOFIs können die aus der TCO-Berechnung resultierenden Kosten dabei dann als spezieller (negativer) Totalgewinn analysiert werden (vgl. [GL04]). Eine Betrachtung der Zins- und Steuereffekte bei der hier vorzunehmenden Kostenberechnung ist aber nicht zielführend. Denn für die Betrachtung beider Effekte müssen aufgenommenes Kapital, ggf. Abschreibungsfolgen (degressiv, linear, etc.), sowie Einnahmen bekannt sein und berechnet werden. Zum einen sind dies Daten, die hier zu großen Teilen lediglich vermutet werden können und zum anderen bei einer Kostengegen-

Tabelle 6.1: Übersicht über alle Kostenkomponenten einer TCO-Berechnung verteilter Systeme.

Kostenart	Quellen für die Prospektive	Quellen für die Retrospektive
Direkte Kosten: *Hardware & Software*		
Hardware des Produktivsystems Updates der Hardware Hardware der IT-Abteilung Software Software der IT-Abteilung	Netzwerk-Diagramme, IT-Budgetplanung, Kostenschätzungen	Belege, Rechnungen, bei Eigenentwicklungen auch Stundennachweise
Direkte Kosten: *Betriebskosten (Personalkosten)*		
Technische Dienstleistungen DB-Management und Administration Prozesse und Planung Service-Desk	Organigramme der IT-Abteilung, Lohn- und Gehaltsdaten, Managementinformationen	Nachweise erbrachter Dienstleistungen, Outsourcing- und Wartungsverträge, Metriken des Service-Desks
Direkte Kosten: *Verwaltungskosten*		
Buchhaltung und Verwaltung Endnutzer-Schulungen Schulungen der IT-Abteilung	Organigramme der IT-Abteilung, Lohn- und Gehaltsdaten, Abteilungs-Budgets	Belege und Verträge unternehmensinterner und -externer Schulungen
Indirekte Kosten: *Endnutzer-Betriebskosten*		
Kollegenunterstützung Applikationsentwicklung (Makros) Schulungen Selbststudium Datei- und Daten-Management	Lohn- und Gehaltsdaten, eingeplante Zeit für Mitarbeiterschulungen	Endnutzer-Interviews und -Umfragen, Nachweise besuchter Schulungen
Indirekte Kosten: *Ausfallzeit*		
Ausfallzeit	Worst-case-, Best-case-Schätzungen	Aufzeichnungen der IT-Abteilung

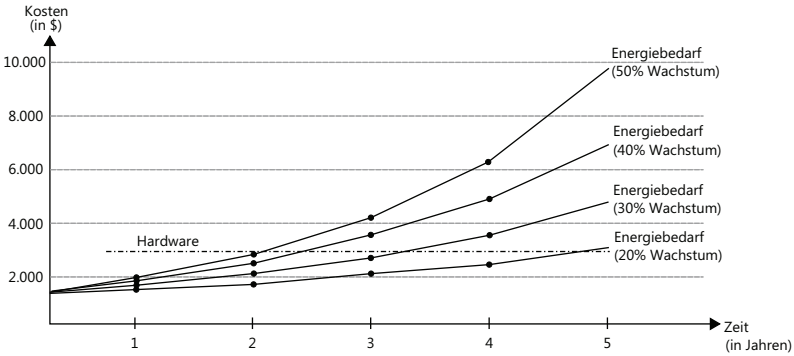
überstellung, die lediglich einen vergleichbaren Teilbereich der Kosten analysiert, nicht so relevant scheinen. Man sollte sich dabei bewusst sein, dass sich diese Effekte natürlich auf die Gesamtrentabilität einer Investition auswirken. Die Vergleichbarkeit ist hierbei aber generell schwierig, da man bei einer SOA-Investition die Anschaffungsauszahlung zu Beginn tätigt und die Hardware daraufhin über mehrere Jahre abschreibt. In einer archetypischen WOA, die hier zum Vergleich dienen soll, werden ausschließlich kontinuierliche, monatsbasierte Kosten erzeugt. Es wird daher im Folgenden ein relevanter Teil der TCO herauszugreifen, der zwar keine Gesamtkostenberechnung widerspiegelt, jedoch einen Vergleich von SOA- und WOA-Kosten zulässt. Dazu wird hier versucht, die Kosten der beiden Alternativen für den Zeitraum eines Jahres abzuschätzen und dabei miteinander zu vergleichen.

Dabei soll die Prämisse gelten, dass die umgesetzten Alternativen **gleichwertige Funktionalität** und dadurch **identische Einnahmen** generieren. So können Wertzuflüsse für die Betrachtung der Kosten vernachlässigt werden und der Vergleich von direkten Kosten stellt dennoch einen aussagekräftigen Wert dar. Diese Voraussetzungen sind bei gängigen Investitionsentscheidungen, bei denen beispielsweise zwei unterschiedlich produktive Maschinen miteinander verglichen werden sollen, meistens nicht gegeben.

Ebenfalls in den meisten TCO-Modellen nicht beachtet werden die Kosten für die Internetanbindung des Unternehmens. Unabhängig von der Realisierung eines IT-Systems sind diese Kosten ebenfalls den direkten Kosten des Betriebs zuzuordnen, da hierbei angenommen wird, dass das System auch aus dem Internet erreichbar sein soll. Im Falle einer Inhouse-SOA bedeutet dies, dass Kunden und Lieferanten auf die internen Server aus dem Internet zugreifen können. Sollte eine WOA erstellt werden, so ist das System bereits im Internet realisiert, aber die Mitarbeiter müssen auf das System für alltägliche Arbeitsvorgänge und Wartung zugreifen können.

Ein weiterer Kritikpunkt, der speziell für IT-Systeme und deren Auslagerung sehr relevant ist, sind die in gängigen TCO-Modellen nicht beachteten Kosten für den Energiebedarf jeglicher Hardware. Da gerade bei Hardware im Serverbetrieb im Normalfall eine 24-stündige Verfügbarkeit an jedem Tag des Jahres vorausgesetzt wird, ist der Stromverbrauch nicht zu vernachlässigen. In [Bar05] wird herausgestellt, dass der Kostenblock allein für die jährlichen Energiekosten eines Servers auf über 40 % des Anschaffungspreises des selbigen ansteigen kann. Für TCO-Berechnungen von großen Unternehmen oder Rechenzentren (die beispielsweise Cluster anbieten) lassen sich also auch diese

zu den relevanten direkten Kosten zählen. In Abbildung 6.2 sind dazu beispielhaft die geschätzten Energiekosten eines Servers in dessen Laufzeit unter der Annahme eines jährlich steigenden Bedarfs an Leistung dargestellt. Im moderaten Fall von 20 % Wachstum ist nach fünf Jahren der Anschaffungspreis eines Servers allein durch Stromkosten überschritten. Im schlimmsten hier gezeigten Fall, bei 50 % Wachstum des Leistungsbedarfs, hat man nach ca. vier Jahren bereits den doppelten Anschaffungspreis pro Jahr für die Energieversorgung bezahlt. Hierbei sind noch nicht einmal Strompreissteigerungen berücksichtigt.



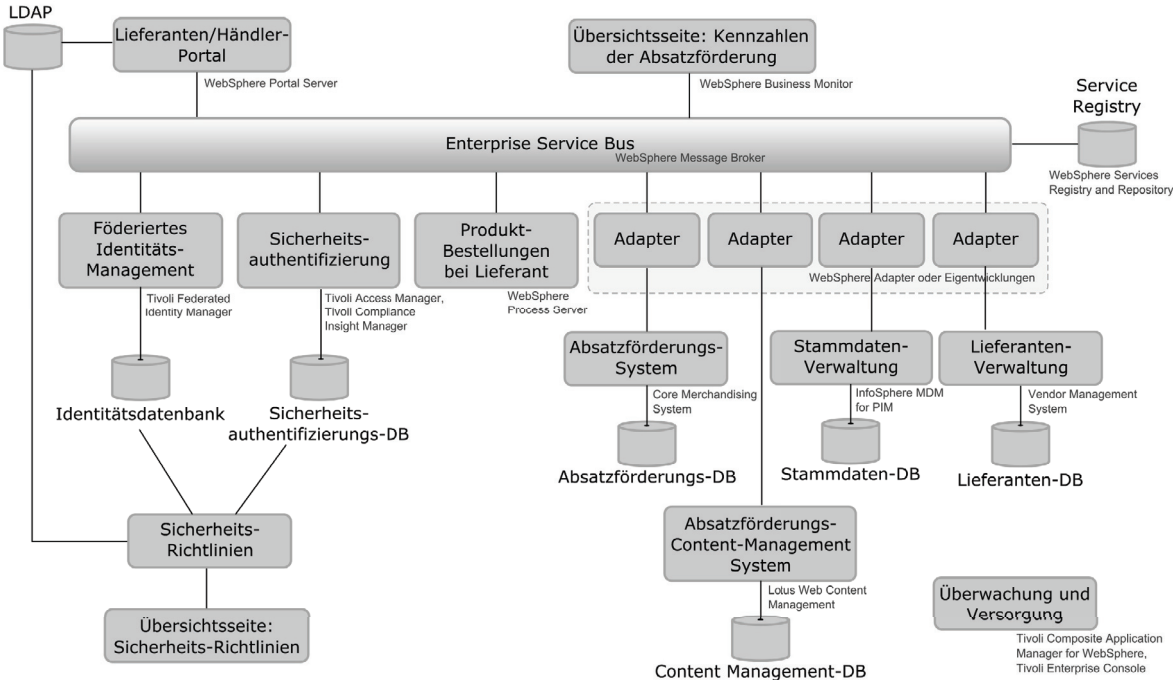
Quelle: nach [Bar05]

Abbildung 6.2: Abschätzung der Energiekosten während der Laufzeit eines Servers.

Bei der Bewertung der TCO eines verteilten Systems muss daher bedacht werden, dass die Stromkosten bei einer reinen WOA-Lösung bereits in den Service-Kosten, die dort den Software-Kosten zugerechnet werden, enthalten sind und somit bei dem Vergleich mit einem SOA-Ansatz als direkte Betriebskosten zusätzlich Beachtung finden müssen.

6.1.4 Exemplarische Beispielberechnung einer Teil-TCO

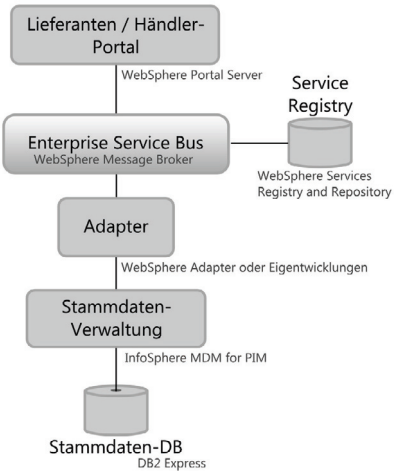
Als Grundlage für die Berechnung der TCO eines (SOA-) Projekts wird auf die Systemarchitektur der *JKHL Enterprises* eingegangen. Deren Zielarchitektur ist mit den von IBM vorgesehenen Komponenten in Abbildung 6.3 dargestellt.



Quelle: nach [KBD+08]

Abbildung 6.3: Umsetzung einer SOA für den Einzelhandel: Fallstudie von IBM.

Um einen Überblick über die TCO-Berechnung zu bekommen, werden hier Kostenelemente für einen vertikalen Ausschnitt aus der Fallstudie dargelegt. Für diesen werden dann die relevanten Kostentreiber in verschiedenen Ausprägungen dargelegt. Der Ausschnitt behandelt die Stammdatenpflege des Unternehmens und sieht dafür ein Portal, ein Stammdatenverwaltungssystem mit Datenbank sowie einen ESB vor. In Abbildung 6.4 ist eine Architekturskizze des gewählten Ausschnitts dargestellt.



Quelle: nach [KBD+08]

Abbildung 6.4: Ausschnitt aus der Referenz-Architektur.

Zuerst wird beispielhaft eine Aufstellung der direkten Kosten, mit Ausnahme der Verwaltungskosten, dieses Systemausschnittes erläutert. Später wird diesem Ergebnis eine Kostenbetrachtung des selben Systemteils als archetypische WOA gegenübergestellt. Eine Betrachtung der indirekten Kosten wird hier nicht vorgenommen, da diese zum einen hauptsächlich auf Schätzungen oder Befragungen in einer retrospektiven Informationsbeschaffungsphase beruhen (siehe Tabelle 6.1) und zum anderen in vielen Belangen den Kosten einer WOA in ähnlicher Höhe zuzurechnen sind und daher aus einem Vergleich gestrichen werden. Durch diese Annahme lässt sich eine tendenzielle Aussage über die direkten Kosten im Falle einer SOA gegenüber einer WOA treffen. Al-

le Beträge der folgenden TCO-Berechnungen werden zur Vereinfachung auf Tausender gerundet.

Hard- und Softwarekosten

Begonnen wird bei der Kostenbetrachtung zunächst mit der Auflistung der notwendigen Software, bevor dann die benötigte Hardware für die Systeme betrachtet wird. Die in dem Szenario dargestellten Softwarekomponenten einer SOA, die in diesem Fall, aufgrund des Fallstudienerstellers, alle aus dem Hause IBM stammen, sind in Tabelle 6.2 aufgelistet. Für den Zugriff auf die Daten wird ein Lieferanten/Händler-Portal benötigt, welches sich mittels des WebSphere-Portals umsetzen lässt. Um alle Komponenten der Teilarchitektur miteinander zu verbinden, wird an dieser Stelle ein IBM *WebSphere Message Broker* eingesetzt, der die Funktionalitäten eines ESB anbietet. Für die Stammdatenverwaltung wird ein spezielles Produkt von IBM verwendet, welches auf die Produktverwaltung für Handelsunternehmen zugeschnitten ist, das *InfoSphere MDM for Product Information Management*. Letztlich wird ein DBMS benötigt, welches für die Datenhaltung der Systeme zuständig ist. Dafür wird hier das Produkt *DB2 Express* ausgewählt. Die Höhe der Lizenzkosten für den Einsatz der IBM-Produkte berechnet sich mit Ausnahme des Portals auf Basis der verwendeten Server, auf denen das Produkt installiert wird. IBM rechnet hierbei mit der Einheit *Processor Value Units* (PVU)[60] und bietet eine Auflistung der möglichen Prozessortypen und deren jeweiliger Leistung. Hierbei gilt: je leistungsstärker ein Prozessor ist, desto teurer wird die Lizenz dafür. Für den später erläuterten Server, der für jedes der Softwareprodukte eingesetzt wird, ergibt sich aus der Liste ein PVU-Faktor von 50. Dabei wird hier die minimale Kostenschätzung angenommen, in der die Software abhängig von der verfügbaren Softwarelizenz für einen einzigen Nutzer bzw. einen einzelnen Server (Prozessor) lizenziert wird. Alles in allem ergibt sich hieraus eine Summe von 156.000,00 €.

Die hier genannten Preise für Softwarekomponenten der IBM sind lediglich Richtwerte, die aus einer Recherche auf den IBM-Webseiten gesammelt wurden. Inwiefern sich für Unternehmen bei einer direkten Kontaktaufnahme mit IBM Rabatte oder Vergünstigungen ergeben könnten, lässt sich nicht dar-

[60] Siehe : http://www-01.ibm.com/software/lotus/passportadvantage/pvu_licensing_for_customers.html.

Tabelle 6.2: Software-Kosten[a] für das Teilsystem der *JKHL Enterprises* als IBM-Lösung.

Funktion	**Komponente**	**Anz.**	**Lizenzkosten/Jahr**
Lieferanten/ Händler-Portal	IBM WebSphere Portal Express (20 Benutzer-Lizenz)	1	3.000,00 €
Enterprise Service Bus (Registry)	IBM WebSphere Message Broker 6.1 (PVU: 1.000,00 €)	50 PVU	50.000,00 €
Stammdaten-verwaltung	IBM InfoSphere Information Server for Foundation Master Data Management (PVU: 2.000,00 €)	50 PVU	100.000,00 €
Stammdaten-Datenbank	DB2 Express (PVU: 60 €)	50 PVU	3.000,00 €
			156.000,00 €

[a] Alle Preise auf dem Stand vom 22.01.2011.

stellen. Dass diese Preise oft eine Verhandlungsbasis darstellen, lässt sich aus Unternehmenskreisen vernehmen.

Durch die hohen Preise einer SOA-Lösung, die auf IBM-Produkten basiert, soll hier zusätzlich eine Variante erläutert werden, die die zuvor genannten Softwarearten komplett durch Open-Source-Produkte realisiert. Eine Diskussion der Qualität und Leistungsfähigkeit freier Software soll an dieser Stelle ausgeklammert werden. Zu bedenken bleibt allerdings, dass der professionelle Support bei den Kosten der IBM-Produkte oft inklusive ist und bei Open-Source-Software, wenn überhaupt, kostspielig dazu gekauft werden muss. Als Portal-Lösung bietet sich hier beispielsweise das *GateIn Portal* (ehemals JBoss Portal) an, welches eine ausgereifte Java-Plattform darstellt. Um eine möglichst nahtlose Integration der Komponenten zu erreichen, soll darüber hinaus der *JBoss ESB* verwendet werden. Beide Produkte sind als Community-Version erhältlich und daher kostenlos. Als Stammdatenverwaltung bietet sich ein Produkt der Firma Talend an, die bereits mit ihrer Open-Source-Lösung im Bereich Business Intelligence sehr weit verbreitet ist und auf viele Referenzkunden verweisen kann. Das *Talend Master Data Management* gibt es in der Community und der Enterprise Version. Während die Erste kostenlos verfügbar ist und an dieser Stelle Verwendung findet, ist die Enterprise-Version kostenpflichtig, bietet dafür aber auch besseren Support und erweiterte Funktionalität. Als DBMS soll hier *PostgreSQL* verwendet werden, welches sehr stabil ist und für eine frei verfügbare Datenbank sehr viel Funktionalität mit sich bringt. Die Summe der reinen Softwarekosten beläuft sich hier erwartungsgemäß auf 0,00 €.

Die Hardware, die für den Betrieb der oben aufgeführten Software notwendig ist, besteht für diesen Auszug des IT-Systems bei beiden Varianten aus vier 4-Core Dell-Rack-Servern (*PowerEdge R515 Rack, AMD Opteron 4130, 4C, 2.6 GHz*), auf denen auch die Berechnung der Software-Lizenzen der IBM-Produkte beruht. Der Anschaffungspreis für einen dieser Server liegt bei ca. 1.000,00 € je Server[61]. Das benötigte Rack für die Server, notwendige Switches und Hubs sowie die Verkabelung werden hierbei pauschal auf weitere 4.000,00 € geschätzt. Weitere Kosten für die Hardware, beispielsweise Laptops und Mobiltelefone für die Mitarbeiter des Unternehmens, werden nicht aufgenommen, da diese bei einer SOA und einer WOA als identisch angenommen werden und daher keinen Erkenntnisgewinn für den Kostenvergleich mit sich bringen. Da man bei der TCO-Berechnung meistens von einer einjähri-

[61] Stand: 17.01.2011. Siehe `www.dell.de`.

Tabelle 6.3: Software-Kosten[a] für das Teilsystem der *JKHL Enterprises* als Open-Source-Lösung.

Funktion	Komponente	Kosten/Jahr
Lieferanten/Händler-Portal	GateIn Portal	0,00 €
Enterprise Service Bus (Registry)	JBoss ESB	0,00 €
Stammdaten-verwaltung	Talend Master Data Management 4	0,00 €
Stammdaten-Datenbank	PostgreSQL 9	0,00 €
		0,00 €

gen Betrachtung der Kosten ausgeht, müssen (bei einer Abschreibung der IT-Komponenten innerhalb von 4 Jahren) die Hardware-Kosten geviertelt werden und liegen anschließend bei 2.000,00 € pro Jahr.

Betriebskosten

Energiekosten Die zuvor schon erwähnte Studie von [Bar05] zu Stromkosten von Servern entstand im Jahre 2005 und ging von 9 $ Cent für eine Kilowattstunde aus und verdoppelte zusätzlich den tatsächlichen Preis der Stromkosten aufgrund notwendiger Kühlung der Komponenten. Um diese Rechnung auf das Beispiel anzuwenden, wird der ungefähre Energieverbrauch der zuvor genannten Dell-Server geschätzt und die dadurch anfallenden Kosten berechnet. Das Netzteil des Dell-Servers wird mit 750 Watt angegeben. Dell bietet eine Web-Applikation an, die es ermöglicht die Konfiguration eines Rack-Servers im Browser zu testen[62]. Dabei wird auch der durchschnittliche Energieverbrauch der Server dargestellt, der für vergleichbare Server bei ca. 45 % der maximalen Netzteilleistung liegt. Daher wird an dieser Stelle von einem durchschnittlichen Verbrauch von ca. 340 Watt pro Stunde ausgegangen. Der somit zu kalkulierende Kilowatt-Verbrauch für einen Server liegt bei gerundeten 8 kWh pro Tag und summiert sich damit bei jährlichem Dauerbetrieb

[62] Siehe: `http://www.dell.com/html/us/products/rack_advisor_new/index.html`

von 365 Tagen auf einen Gesamtverbrauch von 2920 KWh. Bei einem durchschnittlichen Strompreis von 0,17 € pro KWh ergibt sich eine Summe von ca. 500,00 €, ohne eine ggf. notwendige Kühlung des Servers zu beachten. Nach einer Verdoppelung der Stromkosten, die in [Bar05] für Kühlungszwecke der Server ermittelt wurden, erreicht man Stromkosten von 1.000,00 € pro Server. Allein diese betragen damit bereits im Jahr der Anschaffung genau den Anschaffungspreis. Alle vier Dell-Server zusammen ergeben damit Stromkosten von geschätzten 4.000,00 € pro Jahr (ohne Strompreiserhöhungen oder Steigerung des Energiebedarfs).

Personal Eine Betrachtung der Personalkosten für technische Dienstleistungen, Datenbankmanagement sowie den Betrieb des Help Desks lassen sich lediglich durch Annahmen stützen. Die hier benannten Produkte von IBM beinhalten bereits die Support-Leistungen durch IBM. Dadurch wird der Zeitbedarf für Fehlersuche bei Hard- und Softwareproblemen mit Sicherheit reduziert. Auf Seiten des Unternehmens wird daher von einem Personalbedarf einer viertel Mitarbeiterstelle übers Jahr gerechnet. Die Open-Source-Variante muss dagegen vollständig autark mit eigenem Personal betreut werden. Aus diesem Grund wird hierfür eine ganze Stelle angesetzt. Die durchschnittlichen Personalkosten für eine ganze Mitarbeiterstelle sollen hier bei 55.000 € liegen, so dass die IBM-Lösung ca. 14.000,00 € und die Open-Source-Lösung 55.000,00 € pro Jahr an technisch bedingten Personalkosten benötigt. Verwiesen sei noch darauf, dass hierbei keinerlei Entwicklungskosten enthalten sind. Die Realisierung von Adaptern für die Vernetzung der Komponenten mittels ESB oder des Aufbaus von Datenbanken würde hierbei den direkten Kosten zu Beginn eines Projektes zugerechnet werden.

Internetverbindung Eine Verbindung zum Internet wird bei jeder hier vorgebrachten IT-System-Realisierung benötigt, da es sich um ein globales IT-System handelt, welches das Handelssystem mit Filialen verbindet und natürlich durch Kunden und Lieferanten zugänglich sein muss. Die Server, auf denen die hier betrachteten SOA-Lösungen realisiert sind, stehen bei einer Inhouse-Lösung im Unternehmen, so dass die Internetanbindung extrem wichtig ist. Der Ausfall der Verbindung hat zur Folge, dass die Mitarbeiter intern zwar noch auf das System zugreifen können, aber keinerlei Zugriff von außen durch Filialen, Kunden oder Lieferanten erfolgen kann. Aus diesem Grund muss bei

Tabelle 6.4: Direkte Kosten der IBM-Lösung.

Kostenart	Kosten/Jahr
Software	156.000,00 €
Hardware	2.000,00 €
Betriebskosten: Energie	4.000,00 €
Betriebskosten: Personal	14.000,00 €
Betriebskosten: Internet	12.000,00 €
	188.000,00 €

einer SOA-Realisierung im eigenen Unternehmen die verwendete Internetverbindung eine möglichst hohe Datenübertragungskapazität bieten. Für dieses Rechenbeispiel wird von einem symmetrischen DSL-Anschluss (SDSL) ausgegangen, der identische Up- und Downloadraten ermöglicht und theoretische Übertragungsraten von bis zu 20.000 kbit/s bietet. Ein möglicher Anbieter ist hier die *QSC AG* mit Sitz in Köln. Die Kosten einer solchen Leitung liegen für das Produkt *Q-DSLmax 20.000 sym.* bei einer angenommenen Laufzeit von mindestens 3 Jahren bei ca. 1.000,00 € pro Monat.

Die hier ermittelten Kosten für die beiden Alternativen beziehen sich also auf die Investitionsauszahlungen und die Betriebskosten für ein Projekt, dessen Laufzeit auf 4 Jahre angesetzt wird. Für einen späteren Vergleich mit einer WOA muss berücksichtigt werden, dass dort keine Server-Hardware benötigt wird. In Tabelle 6.4 ist die Summe der direkten Kosten (ohne Verwaltungskosten) der Teilarchitektur des Systems in der Umsetzungsvariante mit IBM-Produkten aufgeschlüsselt. In Tabelle 6.5 werden die Kosten der Open-Source-Variante gezeigt. Beide Ergebnisse beziffern die jährlichen Kosten für den Betrieb der Teilarchitektur und werden für einen späteren Vergleich mit einer WOA-Umsetzung derselben Architektur herangezogen.

6.2 Kostenberechnung einer WOA

Nach dem Eingehen auf die Besonderheiten der Kostenberechnung einer WOA mittels TCO wird die zuvor vorgenommene Beispielberechnung der TCO mit einer WOA-Lösung dargestellt, um die Vorteile bei den direkten Kosten darzustellen. Über das durch viele Annahmen gestützte Beispiel einer TCO-

Tabelle 6.5: Direkte Kosten der Open-Source-Lösung.

Kostenart	Kosten/Jahr
Software	0 €
Hardware	2.000,00 €
Betriebskosten: Energie	4.000,00 €
Betriebskosten: Personal	55.000,00 €
Betriebskosten: Internet	12.000,00 €
	73.000,00 €

Rechnung hinaus soll eine Berechnung der gesamten Realisierungs- und Betriebskosten einer WOA ermöglicht werden. Dies wird im Anschluss durch die Definition eines mathematischen Kostenmodells vorgenommen. Auf der Annahme einer mittels FDD durchgeführten WOA-Realisierung beruhend, ermöglicht es eine Schätzung der Umsetzungskosten. Darüber hinaus lassen sich damit auch die Betriebskosten der WOA auf Grundlage der verwendeten Services und deren Kostenmodelle berechnen.

6.2.1 TCO-Berechnung einer WOA

Im Gegensatz zu einem gängigen verteilten System, welches in einem Unternehmen mittels Hard- und Software umgesetzt wird, basiert eine WOA auf der Nutzung von XaaS-Angeboten des Internets und muss daher bei einer TCO-Berechnung in einigen Fällen anders behandelt werden. Auf die jeweiligen Kostenarten und deren Bedeutung für die Berechnung einer WOA wird im Folgenden eingegangen.

Soft- und Hardware Für die konkrete Umsetzung eines verteilten Systems als WOA lassen sich bei diesen Kosten einige Unterschiede feststellen. Innerhalb einer WOA sind für die Kosten der Hardware des Produktivsystems, dem TCO-Modell folgend, zum einen die Endgeräte der Nutzer zuzurechnen. Dies können sowohl mobile als auch stationäre Geräte sein, die eine Internetverbindung herstellen. Zum anderen werden die notwendigen Netzwerkkomponenten zur Ermöglichung des Internetzugangs ebenfalls hierzu gezählt. Weitere Hardware, die sich bei einem Inhouse-System vor allem in Server-

Hardware niederschlägt, gibt es in einer archetypischen WOA nicht. Updates für die Hardware beschränken sich demnach auch lediglich auf die wenigen vorhandenen Geräte und Netzwerkkomponenten. Spezielle Hardware für die IT-Abteilung, neben den normalen Endgeräten, existiert innerhalb einer WOA ebenfalls nicht. Alle notwendigen Funktionalitäten der IT-Abteilung, die der Kostenart „Software der IT-Abteilung" zugerechnet werden und beispielsweise bei der Überwachung der Systeme und Tests der Infrastruktur Verwendung finden, werden durch Services aus dem Internet abgewickelt. Hierbei findet auch der Einsatz von WACs sinnvolle Verwendung. Der für den Vergleich einer Inhouse-Lösung mit einer WOA interessanteste Aspekt liegt in den Software-Kosten. Zu diesen wird bei einer WOA-Lösung außer den notwendigen Betriebssystemen und Browser-Software auf den Endgeräten keine weitere interne Software zugerechnet. Dafür werden aber alle in Anspruch genommenen XaaS-Angebote dazu gezählt. Dieser Kostenblock ist daher von Seiten einer WOA der interessanteste und wird im später vorgestellten Prototyp des Kostenmodells simuliert.

Betrieb In der Zurechnung von Kosten zu einer WOA sind auch bei den Betriebskosten Unterschiede zu einer Inhouse-Lösung zu erkennen. Die technischen Dienstleistungen bestehen dabei nur aus der Wartung und Pflege der Endgeräte sowie des Internetzugangs. Weitere Dienstleistungen werden hier nicht in Anspruch genommen und fallen daher geringer aus als bei einer Inhouse-Lösung. Die Verwaltung und Administration von Datenbanken fällt sogar vollständig weg, da keine internen Datenbanksysteme verwendet werden. Alle normalerweise anfallenden Arbeiten werden hierbei von den Service-Anbietern vorgenommen, die bereits in den direkten Software-Kosten enthalten sind. Die Personalkosten für den Bereich der Prozesse und der Planung sowie für den Betrieb eines Service-Desks sind bei beiden Ausprägungen eines verteilten Systems ungefähr gleich zu bewerten. Ein Unterschied könnte in der Beschaffenheit des Service-Desks bestehen, der bei einer WOA u. a. auch externe Service-Hotlines einbezieht, die durch einen XaaS-Anbieter bereitgestellt werden. Die Stromkosten, die bei einer SOA-Lösung den direkten Betriebskosten zugerechnet werden, fallen hierbei ebenfalls weg bzw. sind in den Kosten für XaaS-Dienste enthalten. Ein interessanter Punkt sind hierbei noch die Internetkosten, die für die TCO-Betrachtung verteilter Systeme, die mit dem Internet verbunden sind, auch den direkten Betriebskosten zuge-

rechnet werden können. Bei oberflächlicher Betrachtung einer WOA kann man zu dem Schluss kommen, dass durch ein innerhalb des Internets realisierten Systems die Internetverbindung eines Unternehmens einen sehr viel höheren Stellenwert besitzt als in einer Inhouse-Lösung. Einerseits können die Mitarbeiter des Unternehmens bei Ausfall der Internetverbindung zwischen dem Betriebsstandort und dem Serverstandort des XaaS-Betreibers nicht mehr produktiv mit der WOA-Realisierung des Systems arbeiten. Andererseits ist das System in einem solchen Fall von Kunden, Lieferanten und Filialen weiterhin erreichbar, da die Realisierung im Internet liegt und im Normalfall durch redundant ausgelegte Systeme bei den XaaS-Anbietern weiterhin funktioniert. Im Gegensatz dazu können bei einer Inhouse-Lösung in einem solchen Fall die Mitarbeiter zwar weiterarbeiten, jedoch kann kein Zugriff von außen mehr erfolgen. Darüber hinaus muss auch die Verbindungskapazität für eine Inhouse-Lösung deutlich höher sein, da Kunden und Filialen von außen auf die internen Systeme zugreifen. Bei einer WOA-Realisierung werden lediglich die Zugriffe von den Mitarbeitern auf die ausgelagerten XaaS-Angebote über die Leitung gesendet. Je nach Zweck eines IT-Systems und der Einbindung der Kunden ist die Internetverbindung im WOA-Fall sogar weniger relevant und kann beispielsweise durch den kurzfristigen Einsatz von *Universal Mobile Telecommunications System* (UMTS)-Verbindungen über Smartphones oder Richtfunk-Internetverbindungen überbrückt werden. Entgegen einer oberflächlichen Betrachtung lassen sich hierbei also Kosten für die Internetverbindung einsparen.

Verwaltung Die direkten Kosten für die Verwaltung einer WOA weisen keine große Abweichung von denen eines Inhouse-Systems auf, da auch dabei Buchhaltung sowie Schulungen für Endnutzer als auch für die IT-Abteilung auftreten.

Endnutzer-Betriebskosten In einer WOA tauchen ebenfalls alle indirekten Kostenaspekte auf. Ein Kostenvorteil oder sogar höhere Ausgaben sind hier nicht zu erwarten, da sich das anfallende Tagesgeschäft der Mitarbeiter durch den Einsatz einer WOA im Vergleich zur reinen Inhouse-Lösung nicht grundlegend verändert.

Ausfallzeit Im Vergleich zur Inhouse-Lösung ist der Ausfallzeit bei einer WOA-Lösung sicherlich etwas mehr Gewicht beizumessen. Abhängig von

der System-Architektur kann der Ausfall eines einzelnen Servers oder XaaS-Anbieters zwar in beiden Realisierungsformen ähnliche Auswirkungen auf die Produktivität der Mitarbeiter haben, für die Fehlerbehebung gibt man im WOA-Fall aber die Verantwortung an den XaaS-Anbieter weiter. Daher ist es unabdingbar, konkrete Vorkehrungen für den Ausfall von WOA-Komponenten vorzuhalten, damit die entstehenden Kosten bei Ausfällen nicht zu hoch werden. Der Ausfall einer Internetverbindung bedeutet in einer WOA vor allem, dass die Mitarbeiter nicht mehr auf normalem Wege auf das WOA-System zugreifen können, die Kunden jedoch schon. Kurzfristig lassen sich auch auf anderem Wege Internetverbindungen aufbauen. Je nachdem, ob es sich bei dem System um eine Kunden-orientierte Plattform oder eher um ein durch das Unternehmen genutztes IT-System handelt, sind hier mehr oder weniger Opportunitätskosten zu erwarten. Bei dem Ausfall von einzelnen Services innerhalb der WOA können, je nach vereinbartem Service-Level, ggf. entstandene Kosten an den Anbieter weitergeleitet werden. Zwar wirbt eine Vielzahl der Service-Anbieter mit einer Verfügbarkeit von über 99,99 %, im Falle der Nichterfüllung dieser Zusicherung werden aber über das „Kleingedruckte" der SLAs meist höhere Haftungssummen ausgeschlossen. Daher ist es sehr wichtig, sich bei geschäftskritischen Anwendungen und Services gerade über den Ausfall der Systemkomponenten Gedanken zu machen und im Zweifelsfall Redundanzen vorzusehen.

Zusammenfassend lässt sich festhalten, dass an dieser Stelle von der Annahme ausgegangen wird, dass bei allen indirekten Kosten und den direkten Verwaltungskosten eine (geschätzte) Kostengleichheit auftritt. D. h. die Höhe dieser Kostenarten ist in beiden Realisierungen eines verteilten Systems zumindest ähnlich und nicht von der konkreten Umsetzung abhängig. Die direkten Kosten für Hard- und Software verteilen sich unterschiedlich auf die einzelnen Aspekte. Ebenso verhält es sich mit den Betriebskosten für technische Dienstleistungen und das Management von Datenbanken sowie bei den Stromkosten.

Exemplarische Beispielberechnung der Teil-TCO einer WOA

Legt man die zuvor getroffene Annahme zugrunde, so lassen sich auch Teilberechnungen der TCO, die lediglich die direkten Hard- und Softwarekosten sowie die Betriebskosten betrachten, für den Vergleich der Kosten einer SOA und einer WOA-Realisierung heranziehen. Die in Kapitel 6.1.4 ausgeführ-

te Beispielberechnung der TCO eines Teilsystems der *JKHL Enterprises* soll nun durch eine WOA-Realisierung umgesetzt werden. Daher gilt es zunächst, die vier Hauptelemente der Architektur aus Abbildung 6.4 durch XaaS-Angebote abzudecken. Das Lieferanten/Händler-Portal lässt sich dabei durch den weitverbreiteten SaaS-Anbieter *salesforce.com* und dessen Produkt *SalesCloud 2* in der Enterprise-Version realisieren. Diese ermöglicht die Entwicklung eigener Oberflächen und Applikationen, so dass sich ein Portal für die Kunden und Lieferanten entwickeln lässt. Oft sind darüber hinaus auf dem Marktplatz von salesforce.com bereits vorgefertigte Applikationen verfügbar, die sofort eingesetzt werden können. Das verbindende Element, welches in einer SOA durch den ESB repräsentiert wird, kann hierbei mit dem bereits bekannten PaaS-Angebot *RunMyProcess* umgesetzt werden, in dem die Definition diverser Workflows und Prozesse ermöglicht wird, um Teile des Systems miteinander zu verbinden. Der Preis von 30,00 € im Jahr für diesen Dienst wird in der Berechnung vernachlässigt. Die Stammdatenverwaltung und die dafür benötigte Datenbank lassen sich durch ein einzelnes Angebot ersetzen: die XaaS-Lösung *Data Scout - Enterprise Edition* des deutschen Anbieters *Cloud Factory*. Dieser ermöglicht die zentrale Verwaltung von Stammdaten in der Cloud und erlaubt dabei die Konsolidierung von Stammdaten aus SAP-, Oracle- oder Microsoft-Systemen. Die Energiekosten können bei einer WOA-Lösung entfallen, da die angebotenen Dienste allesamt von den XaaS-Anbietern betrieben werden. Ebenso verhält es sich mit den Personalkosten für Wartung und Betrieb der Systeme. Denn auch diese Aufgabe wird durch den jeweiligen Betreiber wahrgenommen. Wie bei der IBM- oder Open-Source-Lösung fallen auch hier Kosten für eine Internetverbindung an. Jedoch ist dabei die hohe Bandbreite nicht, eine stabile Verbindung aber sehr wohl erforderlich. In einer WOA-Realisierung genügt dasselbe Produkt der *QSC AG* wie zuvor, jedoch mit einer geringeren Bandbreite von maximal 10.000 kbit/s. Der monatliche Preis für diese Internetverbindung beläuft sich auf 599,00 € und damit ca. 7.000,00 € im Jahr. Eine Kostenübersicht über diese Komponenten ist in Tabelle 6.6 dargelegt.

Eine abschließende Gegenüberstellung der drei Realisierungs-Arten der Teilarchitektur zeigt eine Abschätzung der direkten Kosten (siehe Tabelle 6.7). Dabei wird ersichtlich, dass beide SOA-Lösungen deutlich teurer ausfallen als die Realisierung einer WOA. Selbst eine Variante mit Open-Source-Software ist noch über das Dreifache teurer als eine WOA-Umsetzung. Somit lässt sich ein Einsparpotential der Kosten bei einer WOA-Umsetzung von ca. 90 % gegen-

Tabelle 6.6: WOA-Realisierung des Teilsystems der *JKHL Enterprises*.

Funktion	**Komponente**	**Kosten/Jahr**
Lieferanten/Händler-Portal	Sales Cloud 2 Enterprise	1.500,00 €
Enterprise Service Bus (Registry)	RunMyProcess	-
Stammdatenverwaltung und DB	Data Scout - Enterprise Edition 4	10.000,00 €
Betriebskosten: Energie und Personal	-	0,00 €
Betriebskosten: Internetverbindung	Q-DSLmax (QSC AG)	7.000,00 €
		18.500,00 €

Tabelle 6.7: Übersicht der ermittelten direkten TCO-Kosten verschiedener Lösungen.

IT-System-Realisierung	**Kosten/Jahr**	**Einsparpotential**
SOA-Lösung mit IBM-Produkten	188.000,00 €	0 %
SOA-Lösung mit Open-Source-Produkten	73.000,00 €	60 %
WOA-Lösung	18.500,00 €	90 % bzw. 75 %

über der IBM-Variante und 75 % gegenüber der Open-Source-Variante feststellen.

Bei dieser TCO-Berechnung fließen keine strategischen Überlegungen und deren monetäre Bewertung mit ein, wie beispielsweise eine Risikobewertung der Auslagerung von Daten oder dem Verlust der Kontrolle über das System. Darüber hinaus wird zu diesem Zeitpunkt noch keinerlei Aussage über den zusätzlichen Implementierungsaufwand eines solchen Systems getroffen. Als erstes Ergebnis des Vergleichs der direkten Kosten soll diese Fallstudie aber an dieser Stelle als Beispiel ausreichen.

Eine ausführliche TCO-Berechnung für ein komplettes IT-System ist sehr viel komplexer und lässt sich ohne IT-Unterstützung kaum durchführen. Eine in-

teressante Entwicklung ist in diesem Zusammenhang das TCO-Tool von *softEnvironment*[63], welches im Auftrag des *Schweizer Informatikstrategieorgan Bund* erstellt wurde und als Open-Source-Software zur Verfügung steht. Damit lassen sich speziell für den IT-Bereich die Elemente für die TCO-Berechnung einzeln eingeben, diverse grafische Übersichten über einzelne Kostenblöcke oder eine gesamte TCO-Berechnung ausgeben und damit zu einem Vergleichswert kommen. Nach weiteren freien Produkten für die TCO-Berechnung sucht man allerdings vergebens, so dass die drei zuvor verglichenen Teil-TCO-Rechnungen mit dem TCO-Tool erstellt wurden. Eine Auswahl grafischer sowie tabellarischer Auswertungen zu diesem TCO-Beispiel befinden sich in Anhang B.

6.2.2 Kostenschätzung der WOA-Erstellung bei Verwendung von FDD

Neben der Abschätzung von direkten Hard- und Softwarekosten, die sich aus der Architektur-Planung ableiten lassen und bei einer WOA-Umsetzung durch den Einsatz von XaaS-Angebote bestimmt werden, sind die Personalkosten für die Planung und Implementierung bei der Durchführung eines IT-Projektes ein relevanter Faktor. Die Berechnung der TCO wird hierfür originär nicht verwendet, da Softwareerstellungskosten nicht berücksichtigt werden. Neben der zuvor behandelten TCO als Untersuchung von Investitionsalternativen wird eine Abschätzung der WOA-Kosten auf Basis der FDD-Methode erläutert. Durch die damit vorgegebene Struktur für die Planung und Umsetzung eines IT-Systems lässt sich hierfür ein mathematisches Modell verwenden. Ein solches Vorgehen ist zugeschnitten auf das Szenario der *BBI GmbH*, die eine WOA als Grundlage des eigenen Unternehmens einführen möchte, dazu FDD verwendet und nun eine Abschätzung der Kosten für die Realisierung sowie für den anschließenden Betrieb der WOA vornehmen möchte. Im Folgenden werden dafür drei Bestandteile des Kostenmodells erläutert. Dabei werden zunächst die Möglichkeiten zu einer Abschätzung in der Planungsphase sowie der Realisierungsphase vorgestellt. Daraufhin folgt die Erläuterung eines Modells der Berech-

[63] http://www.softenvironment.ch

nung von monatlichen Betriebskosten einer WOA-Realisierung unter der Berücksichtigung unterschiedlicher Geschäftsmodelle der XaaS-Anbieter.[64]

Planungsphase

Die Planungsphase eines IT-Systems soll laut FDD maximal zwei bis drei Wochen dauern. Da die teilnehmenden Akteure festgelegt sind, beispielsweise Domänenexperten, Chefprogrammierer sowie Chefarchitekten des Software-Herstellers, ist der Personalkostenaufwand für die Planungsphase auf Basis der für die Personen angesetzten Stundenlöhne bestimmbar. Dazu werden für jeden Tag der Planungsphase die Stundenlöhne der beteiligten Personen mit dem Faktor der jeweiligen Beteiligung aufsummiert. Die folgende Formel ermöglicht hierbei eine Berechnung der Planungskosten (siehe Formel 6.1).

$$PK = \sum_{t=1}^{T} \underbrace{\sum_{p \in P_{plan}} (k_p * b_{pt})}_{\text{Kosten pro Tag}} \tag{6.1}$$

$T =$ Anzahl der Tage in der Planungsperiode
$t =$ Tagesindex der Planungsperiode
$k_p =$ Kostensatz der Person p
$b_{pt} =$ Beteiligung (in Prozent) der Person p am Tag t
$P_{plan} =$ Beteiligte Personen des Planungsteams

Als Minimalbeispiel sei hier die Umsetzung einer WOA für das operative System der *BBI GmbH* angenommen. Bei einer angenommenen Gesamtlaufzeit von sechs Monaten werden drei Wochen für die Planungsphase angesetzt ($T = 15$ Tage). An dem Planungsprozess nehmen drei Mitarbeiter der *BBI GmbH* teil, die die Rollen der Domänenexperten durch eine Person sowie der IT-Architekten durch zwei Personen selbst ausfüllen und alle einen moderaten Kostensatz von 500 €[65] ausweisen ($k_p = 500$ €/Tag). Die beiden IT-Architekten des Teams werden während der Planungsphase komplett ausgelastet, woraus

[64] Das Kostenmodell ist Teil des angenommenen Journal-Beitrags [TV13], der nach Planung des Editor-in-Chief aber erst im Jahre 2013 erscheinen wird.

[65] Die Opportunitätskosten für den Ausfall der drei Mitarbeiter für das Alltagsgeschäft sind bei einer ausgelasteten Unternehmensberatung durchaus doppelt so hoch anzusetzen.

Tabelle 6.8: Minimalbeispiel der Planungskosten bei FDD für die *BBI GmbH*.

Mitarbeiter	**Auslastung**	**Kosten**
IT-Architekt I	100 % × 15 × 500 €	7.500,00 €
IT-Architekt II	100 % × 15 × 500 €	7.500,00 €
Domänenspezialist	((100 % × 10) + (25 % × 5)) × 500 €	ca. 5.000,00 €
		20.000,00 €

sich eine Beteiligung b_{pt} von 100 % an jedem Tag der Planungsphase ergibt. Der dritte Mitarbeiter wird als Domänenexperte in der dritten Woche der Planung lediglich zu 25 % seiner Arbeitszeit belastet. Eine Abschätzung der Kosten für diese Planungsphase ergibt hierbei eine Summe von ca. 20.000,00 €, deren Rechnung in Tabelle 6.8 dargestellt ist.

Implementierungsphase

Von zentraler Bedeutung für die Kostenplanung einer Umsetzung ist die Implementierungsphase. Hierbei werden zum einen meist mehr Personen beschäftigt und zum anderen wird sehr viel mehr Zeit für die Erstellung eines Systems gegenüber der Anfangsplanung benötigt. Es wird die innerhalb der Planungsphase entstandene Feature-Liste umgesetzt, die alle Anforderungen des Systems beschreibt. Die in dieser Phase durchgeführte Detailplanung und deren Umsetzung sollen dabei pro Feature maximal zwei Wochen benötigen. Eine Möglichkeit der Abschätzung von Aufwand eines Features ist das Durchführen eines Schätzspiels mittels der Komplexität-Größe-Matrix (siehe Abbildung 6.5). Dabei gibt jeder Teilnehmer des Planungsteams eine Meinung zum benötigten Aufwand eines vorliegenden Features ab. So soll ein Konsens über den geschätzten Umfang jedes Features erzielt werden. Bei dieser Methode werden für die einzelnen Ausprägungen der Matrix, meist aufgrund von Projekterfahrung, ganz konkrete Werte hinterlegt, in diesem Fall also Aufwandspunkte bzw. Personentage. Durch diese frühe Planung lassen sich die für das gesamte Projekt notwendigen Aufwände an Personalkosten schätzen.[66]

[66] Die Schätzung beruht hierbei, wie bei allen Projektaufwandsschätzungen, auf der Expertenmeinung der Projektleitung und der Chefprogrammierer und kann daher in der Genauigkeit stark variieren.

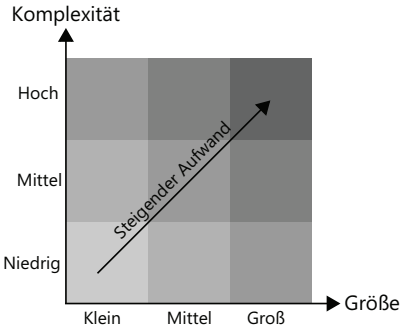


Abbildung 6.5: Aufwandschätzung von Features im FDD.

Die Schätzung basiert dabei auf den gemittelten Stundenlöhnen der einzusetzenden Entwickler, Domänenexperten und Chefprogrammierer multipliziert mit der ermittelten Anzahl an Personentagen, die für die Umsetzung des gesamten Projekts angesetzt sind. Im Gegensatz zu einer Aufwandsschätzung wie beispielsweise der Function-Point-Methode (vgl. [Bal01]), ist dieses Verfahren sehr viel einfacher durchzuführen. Außerdem kann diese Schätzung direkt nach der in den ersten drei Wochen durchgeführten Planungsphase vorgenommen werden, während bei klassischen Verfahren nach dieser Zeit bestenfalls das Lastenheft festgelegt ist. Ein Nebeneffekt dieser Schätzung ist auch die Preisermittlung einzelner Features. Damit lassen sich ggf. aus Gründen einer drohenden Budgetüberschreitung einzelne Features aus der Realisierung anhand deren antizipierten Kosten entfernen. Die folgende Formel beschreibt die Kostenabschätzung der Implementierungsphase:

Tabelle 6.9: Kosten eines Features in der Implementierungsphase.

Mitarbeiter	**Auslastung**	**Kosten**
Feature: Callcenter mit Twilio		
Entwickler I	100 % × 5 × 350 €	1.750,00 €
Entwickler II	100 % × 5 × 350 €	1.750,00 €
Domänenexperte	75 % × 1 × 500 €	375,00 €
		3.875,00 €

$$IK = \sum_{f=1}^{F} \underbrace{u_f * \sum_{p \in P_{impl}} (k_p * b_{pf})}_{\text{Kosten pro Feature.}} \tag{6.2}$$

F = Featureanzahl des Projektes
f = Featureindex
u_f = Umfang des Features f in Personentagen
k_p = Kostensatz der Person p
b_{p_f} = Beteiligung (in Prozent) der Person p an Feature f
P_{impl} = Beteiligte Personen des Realisierungsteams

Eine beispielhafte Ausführung dieser Kostenschätzung für ein Feature gestaltet sich wie folgt: Aus der Planungsphase ist zum einen der Aufwand in Personentagen geschätzt, der für das Feature zur Verfügung steht. Ebenso steht auch das jeweilige Team fest, welches das Feature umsetzen soll. Hierzu zählen jeweils auch zu einem kleinen Teil Domänenspezialisten, die zu Beginn der Detailplanung benötigt werden. Als Beispiel sei hier weiterhin die *BBI GmbH* und deren Umsetzung einer WOA angeführt. Für das erste beispielhafte Feature „Callcenter mit Twilio" wurde eine Aufwandsschätzung von fünf Punkten (Personentagen) festgelegt, wobei sich zwei Entwickler die gesamte Zeit sowie ein Domänenexperte am ersten Tag zu 75 % für die Umsetzung des Features verantwortlich zeichnen. Für die Entwickler wird hierbei ein Tageskostensatz von 350 € angesetzt. Um die Gesamtsumme der Implementierungskosten zu errechnen, werden alle einzelnen Featurekosten aufaddiert (siehe Tabelle 6.9).

Die Gesamtkosten des Erstellungsprozesses einer WOA lassen sich somit komprimiert durch die folgende Formel darstellen (siehe Gleichung 6.3):

$$GK = \underbrace{\sum_{t=1}^{T} \sum_{p \in P_{plan}} (k_p * b_{pt})}_{\text{Kosten der Planungsphase}} + \underbrace{\sum_{f=1}^{F} u_f * \sum_{p \in P_{impl}} (k_p * b_{pf})}_{\text{Kosten der Implementierungsphase}} \tag{6.3}$$

Betriebskosten einer WOA

Bei der Umsetzung einer archetypischen WOA wird davon ausgegangen, dass ausschließlich XaaS-Angebote zur Umsetzung verwendet werden. Dabei existieren viele verschiedene Geschäftsmodelle und ebenso viele Kostenmodelle. Die einfachste Variante ist hierbei eine monatlich anfallende Gebühr für die Nutzung eines Dienstes unabhängig von der Anzahl der Nutzer oder der Intensität der Nutzung. Bei solchen Bezahlungsmodellen lassen sich, wie zuvor geschehen, die TCO für direkte Kosten auch recht trivial berechnen. Wenn jedoch weitere Parameter für die Kostenberechnung relevant sind, wie beispielsweise die Zahl der Nutzer oder die Zeitspanne der Nutzung, sogenannte „Pay-per-use"-Geschäftsmodelle, so lässt sich eine TCO-Rechnung nicht mehr einfach durchführen. Es ergeben sich daher komplette Berechnungsregeln für die Kosten eines XaaS-Dienstes, die auf beliebig vielen Parametern beruhen können. Eine Berechnung der Betriebskosten unter diesen Voraussetzungen erfordert daher zweierlei: Zum einen ist es notwendig, für jeden in der WOA verwendeten XaaS-Dienst eine Berechnungsregel zu erstellen, die das Bezahlungsmodell des Angebots widerspiegelt. Zum anderen müssen Eingabe-Parameter für diese Regeln geschätzt werden, um konkrete Kosten zu ermitteln. Unabhängig von dem Geschäftsmodell eines XaaS-Dienstes, sei es ein Pay-per-use-Modell oder eine monatliche Nutzungsgebühr, rechnen die meisten XaaS-Dienste die Kosten auf monatlicher Basis ab. Daher werden alle folgenden Annahmen und Ausgestaltungen monatsweise vorgenommen. Um dieses Konstrukt der Kostenberechnung mathematisch auszudrücken, wird die folgende Formel verwendet (siehe Formel 6.4). Im Grunde findet hierbei eine Aufsummierung der Kosten jedes einzelnen Services über eine beliebige Laufzeit (im Normalfall 12 Monate) statt. Die Eingabeparameter für die jeweilige Service-Kostenfunktion f_s werden dazu als Eingabevektor n_{ms} definiert.

$$BK = \sum_{m=1}^{M} \underbrace{\sum_{s=1}^{S} f_s(n_{ms})}_{\text{Service-Kosten pro Monat}} \tag{6.4}$$

$M =$ Anzahl betrachteter Monate (12)
$m =$ Monatsindex
$S =$ Anzahl verwendeter Services in der Implementierung
$s =$ Serviceindex
$f_s =$ Kostenfunktion des Services s in Abhängigkeit des Vektors n_{ms}
$n_{ms} =$ Vektor mit Nutzungsparametern des Services im Monat m des Services s für die Funktion f_s

Als Beispiel einer Berechnung der Kosten wird hier der bereits zuvor verwendete Bezahlungsdienst PayPal verwendet. Für diesen lässt sich eine einfache Berechnungsregel auf den Informationen über die Kostenstruktur erstellen. Hierbei entstehen einem Unternehmen keine fixen Kosten. Es wird eine geringe Gebühr pro Zahlungsvorgang über PayPal fällig, die von der gezahlten Währung, dem Land des Zahlenden und der Höhe des Gesamtvolumens der Transaktionen innerhalb eines Monats abhängig ist. Unterschieden wird hierbei noch zwischen Zahlungen innerhalb der EU[67] und Zahlungen aus den übrigen Ländern. Als Parameter werden die Anzahl an *Transaktionen*, die Höhe des *Umsatzes* und der *Anteil*, der vom Umsatz über PayPal abgerechnet wird, benötigt. PayPal erhebt auf jede Transaktion eine Gebühr von 0,35 € zuzüglich eines Anteils zwischen 1,2 % - 1,9 % der Transaktionshöhe. Der prozentuale Anteil wird hierbei abhängig vom Monatsumsatz in Tabelle 6.10 dargestellt.

Daraus ergibt sich die folgende Berechnungsformel für die Kosten von Zahlungen, die per PayPal in EU-Ländern getätigt wurden. Die Formel erwartet hierbei die Eingabe der drei Parameter **Transaktionen**, **Umsatz** und **Anteil** (siehe Listing 6.1).

Durch die Verwendung dieser Formel lässt sich der Kostenanteil des Services PayPal pro Monat bei vorliegenden Eingabeparametern direkt berechnen. In dieser Form müssen daher Berechnungsregeln für alle XaaS-Angebote

[67] Außerdem aus Norwegen, Island und Liechtenstein.

Tabelle 6.10: Gebühren für die Verwendung des PayPal-Dienstes.[a]

Monatsumsatz	€-Zahlungen	Andere Zahlungen
bis 5.000 €	1,9 % × TS[b] + 0,35 €	3,9 % × TS + 0,35 €
5.001 - 25.000 €	1,7 % × TS + 0,35 €	3,7 % × TS + 0,35 €
25.000 - 50.000 €	1,5 % × TS + 0,35 €	3,5 % × TS + 0,35 €
> 50.000 €	1,2 % × TS + 0,35 €	3,2 % × TS + 0,35 €

[a] Stand: 14.02.2011
[b] Transaktionssumme

angelegt werden. Das Berechnen der direkten Softwarekosten für eine spätere Verwendung in einer TCO-Berechnung wird durch die Aggregation aller Service-Kosten einer WOA-Realisierung für ein Jahr erreicht. Dabei wird vorausgesetzt, dass für jedes XaaS-Angebot passende Werte angegeben werden können.

6.3 Betriebskostenberechnung mittels Web-Applikation

Das manuelle Berechnen der Betriebskosten einer WOA ist durch die Handhabung von Rechenregeln sowie Parametern für jeden XaaS-Dienst recht aufwendig. Ideal wäre eine IT-Unterstützung, die das einfache Konstruieren von Berechnungsregeln ermöglicht, die Eingabe von Nutzungsparametern vereinfacht und eine Auswertung von Betriebskosten einer WOA für unterschiedliche Nutzungsszenarien bereitstellt. Aus diesem Grund wurde im Rahmen dieser Arbeit eine prototypische Web-Applikation entwickelt, die die zu Anfang des Kapitels genannten Ziele der TCO-Methode verfolgt: Simulation der direkten Softwarekosten eines Unternehmens, Aufstellung von Kosten zur Gegenüberstellung mit typischen Kosten und die Möglichkeit, unterschiedliche Szenarien zu berechnen. Dem letzten Punkt wird hierbei besondere Aufmerksamkeit zuteil, da gerade das Variieren der möglichen XaaS-Service-Kombinationen und ihrer Parameter einen Überblick über die Kosten einer WOA verschafft. Der schematische Aufbau des Kernkonzepts ist in Abbildung 6.6 dargestellt. Hieraus wird ersichtlich, dass durch die Erstellung eines Service-Katalogs aus diversen XaaS-Angeboten des Internets mit deren Berechnungsregeln im Zu-

```
# Berechnen des Gesamtvolumens
Volumen = Umsatz * (Anteil / 100);
# Festlegen des prozentualen Anteils
if(Volumen < 5000)
{
  EURO-Prozentsatz = 0.019;
}
if((5001 < Volumen) AND (Volumen < 25000))
{
  EURO-Prozentsatz = 0.017;
}
if((25001 < Volumen) AND (Volumen < 50000))
{
  EURO-Prozentsatz = 0.015;
}
if(Volumen > 50000)
{
  EURO-Prozentsatz = 0.012;
}

#Berechnung der tatsächlichen Kosten für EURO-Zahlungen
Kosten-EU = (Volumen * EURO-Prozentsatz) + (Transaktionen *
    0.35);
```

Listing 6.1: Berechnungsregel für PayPal.

sammenspiel mit der Eingabe von Szenario-Daten die direkten Softwarekosten im Sinne der TCO ermittelt werden können.

In einer Entscheidungssituation, in der jede Variable (*hier:* umzusetzender fachlicher Service) diverse Ausprägungen (*hier:* XaaS-Anbieter) annehmen kann, steigt die Anzahl möglicher Kombinationen exponentiell zur Anzahl an Variablen. Bei der Umsetzung einer WOA lassen sich so diverse Service-Anbieter in beinahe beliebiger Kombination miteinander verbinden. In Abbildung 6.7 ist eine Auswahl an Service-Anbietern und Kategorien als morphologischer Kasten dargestellt. Die hier dargestellten Kategorien zeigen nur einen Ausschnitt der Anbieter. Ein exemplarischer Pfad durch die Anbietervielfalt ist eingezeichnet, der in diesem Beispiel lediglich eine aus $5^7 = 78.125$ Möglichkeiten darstellt. Aus dieser Vielzahl wird schon ersichtlich, dass sich eine Abschätzung der Betriebskosten einer WOA, deren XaaS-Angebote komplexe Berechnungsregeln voraussetzen, ohne Programmunterstützung sehr schwer

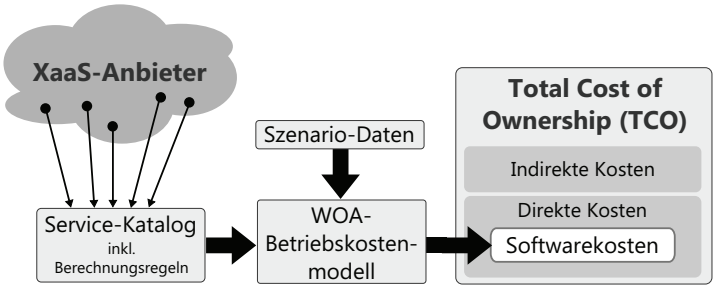


Abbildung 6.6: Konzeption des WOA-Betriebskosten-Modells.

umsetzen lassen würde. Daher wird im Folgenden das Konzept sowie die prototypische Umsetzung der Web-Applikation zur WOA-Kostenberechnung vorgestellt.

			Anbieter					
Service-Kategorien	Infrastruktur	Datenbank	Amazon SimpleDB	Microsoft Azure	Zoho Creator	NextDB	MySQL*	...
		Hosting	Host Europe	domainFACTORY	STRATO	1 & 1	Hetzner	...
		Cloud-Comp.	Google App Engine	Amazon EC2	GoGrid	GridLayer	Elastic Server	...
		Groupware (Wikis)	Springnote	Zoho Wiki	Google Sites	MindTouch	Media Wiki*	...
		...	...	...	...	...	...	...
	Prozesse	CRM	Salesforce	Zoho CRM	LongJump	Aria	Highrise	...
		Bezahlung	Amazon FPS	PayPal	PayLeap	PaySimple	moneybookers	...
		PaaS	BungeeConnect	RunMyProcess	Heroku	Etelos	Force.com	...
	:	...	...	...	...	...	...	...

*) Software, die auf einem eigenen Server lauffähig ist.

Abbildung 6.7: Ausschnitt der möglichen Service-Kombinationen.

6.3.1 Konzept

Als Voraussetzung für den Prototyp muss als erster Teil ein Service-Katalog erstellt werden, der möglichst viele potenziell nutzbare Services beinhaltet, damit verschiedene Szenarien getestet werden können. Ein solches Portfolio muss einige Informationen bereitstellen:

- **Welche Services gibt es?** Hier muss ein Service-Katalog aufgebaut werden, der es ermöglicht, aufgrund von Detailinformationen einen Dienstleister auszuwählen. Dabei sollten die Services in Kategorien eingeteilt werden, damit deren Auffinden und Vergleichen effektiv geschehen kann. Dazu lässt sich als oberste Ebene einer Hierarchie die Kategorisierung in die drei funktionalen Service-Typen Infrastruktur, Geschäftslogik und Präsentation verwenden, wie in Kapitel 2.3.5 vorgeschlagen. Die nächsttiefere Ebene greift dann eine Typisierung nach Vorbild des Service-Verzeichnisses *ProgrammableWeb* auf.

- **Auf welchem Kostenmodell beruht die Abrechnung des angebotenen Services?** Es existieren beinahe ebenso viele Abrechnungsmodelle wie es Services gibt. Hier reicht die Spanne von monatlichen Entgelten über Gesamtnutzungsdauer oder Übertragungsvolumen bis hin zu Rechnerkapazitäten pro Stunde. Oft gibt es dabei diverse Parameter, die angepasst werden können, um eine Kostenberechnung vorzunehmen.

Der zweite Teil des Prototyps ermöglicht das Anlegen von Szenarien und die Eingabe von Nutzungsdaten für ein solches. Dabei wird eine mehrperiodische Betrachtung aufgegriffen, indem im Modell zwölf Perioden (hier: Monate) als Planungsdaten für eigene Szenarien hinterlegt werden können, um die Kostenberechnung nach einem typischen TCO-Modell aufzustellen.

Um diese beiden Teile in einer Web-Applikation zu vereinen, wird ein Datenbankmodell benötigt, welches sich in drei Bereiche untergliedert: die *Abbildung und Definition von Kostenmodellen*, das *Anlegen von Szenarien* und *das Speichern von Plandaten* (siehe Abbildung 6.8). Eine kurze Erläuterung zu jeder der Entitäten ist im Folgenden gegeben:

- **Service**: Hier werden einzelne XaaS-Dienste gespeichert. Jeder Dienst verweist dabei auf genau einen Anbieter. Daten, die hier gespeichert werden, sind u. a. Fixkosten des Services, eine URL zum Service-Angebot sowie ein Verweis auf die Kategorie, zu der dieser Service zugeordnet wird.

- **Kategorie**: Die Dienstlandschaft wird in diverse Kategorien eingeteilt, die hier gespeichert werden. Dabei wird jede genau einem Service-Typ zugeordnet.

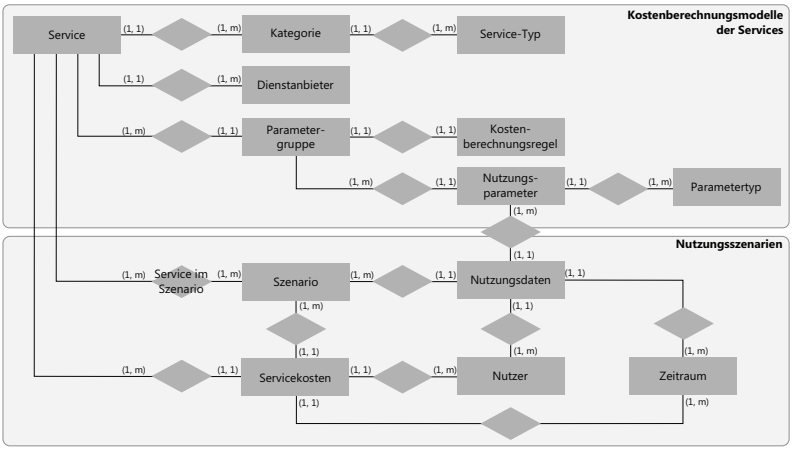


Abbildung 6.8: Datenbankmodell für die Kostenberechnung.

- **Service-Typ**: Die drei existierenden Service-Typen (Infrastruktur, Geschäftslogik und Präsentation) werden hier hinterlegt, damit einzelne Kategorien zugeordnet werden können.
- **Dienstanbieter**: Die grundlegenden Daten eines XaaS-Anbieters, wie Name des Unternehmens und dessen Domain werden angelegt.
- **Parametergruppe**: Jedem Service können beliebig viele Parametergruppen zugeordnet werden. Diese dienen hierbei als Container für eine beliebige Anzahl an Parametern.
- **Nutzungsparameter**: Ein Nutzungsparameter beschreibt eine Komponente, die bei der Verwendung eines Dienstes zur Kostenberechnung verwendet wird. Er ist entweder ein numerischer Wert oder ein Wert aus einer Liste von Optionen.
- **Parametertyp**: Es gibt zwei Typen: numerische Werte (*slider*) und Optionslisten (*options*).
- **Kostenberechnungsregel**: Hier können die Berechnungsregeln zur Kostenermittlung eines XaaS-Dienstes gespeichert werden. Jede defi-

nierte Regel ist genau einer Parametergruppe zugeordnet und kann dadurch alle dieser Gruppe zugeordneten Nutzungsparameter verwenden. Diese können daher vom Prototyp als Eingabewerte für die Kostenberechnungsregel verwendet werden.

- **Szenario**: Die Namen von erstellten Szenarien für die Berechnung von Kosten einer WOA werden hier gespeichert.

- **Service im Szenario**: Dieser Relationship-Typ enthält Informationen über die Zuordnung von XaaS-Services zu einem Szenario. Dadurch lassen sich beliebige Service-Portfolios für Szenarien zusammenstellen, die dann durch den Nutzer angepasst oder unverändert verwendet werden.

- **Nutzungsdaten**: Einzelne Plandaten, die für die Nutzung von Services angegeben werden müssen, können hier benutzerspezifisch gespeichert werden. Ein Datensatz verweist dabei auf ein Szenario, einen Nutzer sowie auf einen spezifischen Nutzungsparameter eines XaaS-Services und speichert dafür einen Wert sowie einen Zeitraum (Periode) in dem die angegebene Nutzung stattfindet.

- **Nutzer**: Benutzer der Berechnungsanwendung werden hier hinterlegt. So können die eingegebenen Daten für die Szenarien unter einer eindeutigen Zuordnung zu Nutzern gespeichert und somit wiederverwendet werden.

- **Servicekosten**: Die Kosten eines Services in einem Zeitraum (Periode), die unter Verwendung einer Kostenberechnungsregel durch den Prototyp berechnet wurden, werden hier gespeichert. Ein Datensatz verweist dabei auf den zugehörigen Service, ein Szenario, den Nutzer sowie einen Zeitraum. Die so berechneten Kosten werden für die verschiedenen Darstellungen der Kostenaggregation im Dashboard verwendet.

Das hier beschriebene Datenbankschema wird innerhalb des Prototyps nur für die Planung von Service-Nutzungen und der darauf beruhenden Betriebskostenberechnung für ein Szenario verwendet. Es ist aber denkbar, dieses Datenbankmodell bzw. den Prototyp um eine Überwachungsfunktionalität zu erweitern, um die Service-Nutzung einer tatsächlichen WOA-Umsetzung zu

protokollieren. Damit würde der Prototyp nicht nur für Planungszwecke, sondern zusätzlich für die Kostenkontrolle einer WOA dienen. Da die Kostenberechnungsregeln bereits bekannt sind, müssten hier lediglich die Nutzungsparameter kontinuierlich fortgeschrieben werden. Anhand dieser Daten ließen sich bereits entstandene Kosten tagesaktuell ermitteln. Aufgrund der Struktur des Datenbankschemas, welches in Teilen bereits die Sternstruktur eines Datawarehouse aufweist, könnten durch geringfügige Erweiterungen am Prototyp auch typische Aufgaben wie beispielsweise *Reporting* oder *Online Analytical Processing* (OLAP) ausgeführt werden. Durch eine solche Erweiterung wäre dann die Verwendung als zusätzliches Modul in einem WAC für die kontinuierliche Kostenkontrolle möglich.

6.3.2 Implementierung

Für die Implementierung des Prototyps wurde auf Server-Seite für die Programmlogik PHP und für die Datenbank MySQL verwendet. Die Client-Seite basiert auf dynamisch generierten HTML-Seiten, die durch den Einsatz von JavaScript zu einer Web-Applikation ausgebaut wurden. Der Einsatz von AJAX hilft das Nachladen der kompletten Seite zu unterbinden und ermöglicht eine dynamische Arbeitsweise mit der Anwendung. So werden beispielsweise die Diagramme des Dashboards der Kostenübersicht per AJAX angefordert, auf dem Server die Kosten berechnet und daraufhin per JavaScript innerhalb der HTML-Seite aktualisiert.

Der Prototyp gliedert sich in zwei Bereiche: der Eingabe der Services sowie der Bearbeitung von Szenarien und die damit verbundene Eingabe von Plandaten. Bei der Eingabe von Services wird auf der linken Seite des Bildschirms eine Baumübersicht über alle Services des Katalogs angezeigt, aus denen ein Service zur Bearbeitung ausgewählt werden kann. Alternativ lassen sich neue Services anlegen. Nach der Eingabe von Grunddaten und der Spezifikation einzelner Parameter lässt sich in Form von PHP-Code eine Berechnungsregel eingeben und testen. In Abbildung 6.9 ist hier beispielhaft die Ansicht der Berechnungsregel für das PayPal Basiskonto dargestellt.

Nach der Eingabe relevanter XaaS-Services und deren Regeln für die Berechnung von Kosten kann in der Szenario-Übersicht ein Service-Mix aus den verfügbaren Diensten zusammengestellt und die Parameter per Schieberegler für jeden Monat eines zwölfperiodigen Zeitraums spezifiziert werden. Hierbei

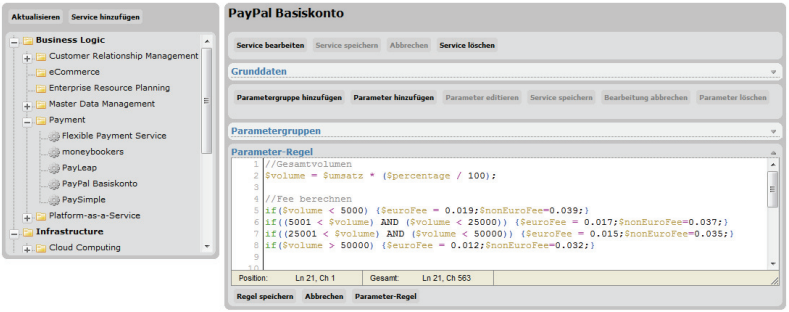


Abbildung 6.9: Eingabe einer Regel für den PayPal-Service.

lassen sich dann auch verschiedene Szenarien für die Nutzung der gewählten XaaS-Angebote durchspielen, beispielsweise eine gesteigerte Nutzung des Abrechnungsdienstes während der Vorweihnachtszeit. Die ständig aktualisierte Detailübersicht gibt dabei jederzeit Aufschluss über die Aufteilung der Kosten. Diagramme zeigen diese dabei aufgeschlüsselt nach einzelnen Monaten, nach Kostenarten und nach Diensten an. In Abbildung 6.10 sind die XaaS-Angebote des ausgeführten TCO-Beispiels einer WOA ausgewählt und für eine steigende Nutzeranzahl im Verlauf des Jahres konfiguriert worden. Die Fixkosten sind hierbei durch die Stammdatenverwaltung des *Data Scout*-Angebots bestimmt, die sich nicht nach der Anzahl der Nutzer richtet. Die variablen Kosten beruhen auf den eingegebenen Planungsdaten für die Anzahl der Nutzer bei *Sales Cloud 2* und *RunMyProcess*.

In Anhang C sind weitere Screenshots des Prototyps dargestellt, die zeigen, wie sich Service-Parameter konfigurieren und Nutzungsparameter in den Szenarien festlegen lassen.

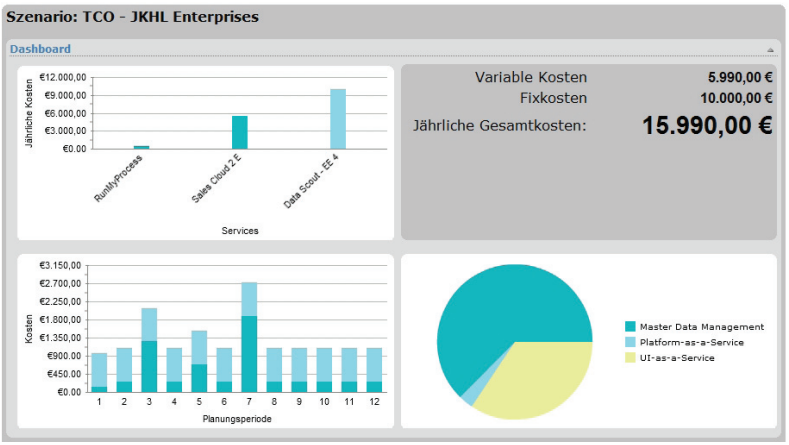


Abbildung 6.10: Auswertung des WOA-Szenarios der TCO-Berechnung mit Änderungen der Nutzerzahlen.

7 Schlussbetrachtung

Im Folgenden werden zunächst die Inhalte der Arbeit zusammengefasst. Daraufhin wird eine Handlungsempfehlung für den Einsatz einer WOA in Unternehmen formuliert, die sich auf die Chancen, die Gefahren, die Schwächen und die Stärken einer WOA stützt. Abschließend wird die Arbeit durch einen Ausblick auf zukünftige Entwicklungen und weitere Forschungsfragen abgerundet.

7.1 Zusammenfassung

Den Startpunkt dieser Arbeit bildeten zwei Beobachtungen: zum einen stellen sich IT-Systemumsetzungen mittels Service-orientierter Architekturen als zu komplex und zu teuer heraus und erreichen nicht die ursprünglich versprochenen Ziele wie Flexibilität, Kosten- und Komplexitätsreduzierung. Zum anderen zeigt sich in neuartigen Applikationen des Web 2.0 wie beispielsweise Mashups, dass die Verknüpfung von unterschiedlichen Datenquellen und die Nutzung von Schnittstellen durch die Verwendung einfacher Web-Standards relativ leicht umgesetzt werden können. Dabei sind die beiden Systemarten im Kern identisch: Es handelt sich um verteilte IT-Systeme, die über das Internet oder ein Intranet kommunizieren. Für ein System, welches konsequent den HTTP-Standard für die Schnittstellendefinition und Kommunikation nutzt, wird bereits der Name Ressourcen-orientierte Architektur verwendet. Auch der Begriff der Web-orientierten Architektur taucht bereits in einigen Quellen auf, definiert dabei aber meistens ein System, welches ausschließlich auf dem REST-Standard beruht.

Aus diesen Beobachtungen wurde hier ein eigenes Konzept einer Web-orientierten Architektur abgeleitet, das sich dem Ziel einer einfachen Umsetzung eines IT-Systems für Unternehmen verschreibt. Dabei gilt zum einen die Prämisse, dass grundsätzlich möglichst einfache Web-Standards für die Realisierung eingesetzt werden sollen, im Bedarfsfall aber durchaus auch komplexere Lösungen wie WS-Standards verwendet werden dürfen. Zum anderen

sollen, wenn möglich, alle fachbezogenen Systemkomponenten durch bereits vorhandene Everything-as-a-Service-Angebote realisiert werden. Dadurch lassen sich in Abhängigkeit von dem umzusetzenden fachlichen Fokus eines Unternehmens IT-Komponenten im besten Fall vollständig auslagern. Sollten durch Regularien bestimmte Daten innerhalb des Unternehmens verbleiben müssen, so lassen sich hybride Ausprägungen einer WOA konstruieren.

Um sich der Umsetzung einer WOA methodisch zu nähern, wurden aufgrund der architektonischen und technischen Nähe Vorgehensmodelle aus dem SOA-Bereich betrachtet. Seit der ersten Veröffentlichung des SOA-Konzepts im Jahr 1996 sind viele Ansätze und Vorgehensmodelle zur Beschreibung, Planung und Umsetzung einer SOA entstanden. In diesem Bereich existieren ebenso konzeptionelle Modelle, die sich der Thematik hauptsächlich aus wissenschaftlicher Sichtweise nähern, wie auch in der Praxis verwendete Modelle, die von Praktikern oft direkt an realen Umsetzungsprojekten entwickelt wurden. Einerseits befinden sich hierunter sehr ausführliche Vorgehensmodelle, die sich über den gesamten Systems Development Life Cycle erstrecken und damit die Analyse, die Planung, die Umsetzung und den Betrieb einer SOA abdecken. Andererseits gibt es aber auch partielle Modelle, in denen einzelne Teilbereiche der SOA-Entwicklung oder SOA-Planung analysiert und betrachtet werden. Alle diese Ansätze haben jedoch den gemeinsamen Nachteil, dass die Aspekte der Einbindung externer XaaS-Angebote und die Nutzung einfacher Web-Standards nicht beachtet werden. Daher ließen sich nur einzelne Teile von Methoden für die dann folgende Beschreibung eines WOA-Vorgehensmodells verwenden. Dadurch wurde die Motivation der Entwicklung einer eigenen Methode bestärkt.

Aus den vorangegangenen Überlegungen zur Architektur und dem Konzept einer WOA und unter Berücksichtigung einiger Anregungen aus den verwandten Arbeiten wurde ein Vorgehensmodell entwickelt, welches die Analyse und Planung, die konkrete Umsetzung sowie den Betrieb einer WOA vollständig beschreibt. Nach der anfänglichen Betrachtung notwendiger Modellierungssprachen und einem Einblick in die zu treffenden organisatorischen Entscheidungen wurde zunächst die Analyse- und Planungsphase erläutert.

Der Start der Analyse ist hier durch eine Modellierung der Geschäftsprozesse eines Unternehmens gekennzeichnet, die als essenziell für eine Systemumsetzung angesehen wird. Diesem Schritt schließen sich die Service-Identifikation auf fachlicher Ebene sowie die Festlegung von SLRs an. Darauf werden die drei folgenden Schritte als WOA-spezifisch herausgestellt. Darin werden zu-

nächst geeignete XaaS-Anbieter identifiziert, danach die Topologie der WOA und eine Sicherheits- und Rechtestruktur definiert, um daraufhin die Analyse- und Planungsphase mit der Festlegung von Notfall- und Alternativplänen als Ausfallstrategie zu beenden. Die in diesen ersten sechs Schritten entstandene Dokumentation beschreibt zu diesem Zeitpunkt die Geschäftsprozesse, die Architektur sowie die zu verwendenden Services und XaaS-Anbieter, so dass darauf in der Realisierungsphase zurückgegriffen werden kann.

Der zweite Teil der WOA-Methode beschreibt die Schritte zur Realisierung und das Konzept für Tests einzelner Workflows sowie des gesamten Systems. Dabei wird bewusst auf die Ausführung konkreter Implementierungs- bzw. Modellierungsschritte verzichtet, da sich diese nicht als generelle Vorgehensmethode beschreiben lassen. Der Grund dafür liegt darin, dass die tatsächliche Realisierung eine WOA zu sehr von den Details der Auswahl von XaaS-Services und der jeweils umzusetzenden Topologie abhängig ist.

Obwohl viele Vorgehensmodelle zur Umsetzung von IT-Systemen den Betrieb eines verteilten Systems eher nachrangig behandeln, wird in dieser Arbeit auch auf die Betriebsphase einer WOA viel Wert gelegt. Erläutert werden daher drei weitere Phasen, die sich im Betrieb parallel zueinander ausführen lassen. Begonnen wird bei der Prozess- und Serviceüberwachung, die bei einem verteilten System unabdingbar ist, um beispielsweise Fehler früh zu erkennen und zu behandeln. Außerdem wird auf die Kostenkontrolle einer WOA eingegangen, die durch die Nutzung von XaaS-Diensten eine kontinuierliche Berechnung und Aufarbeitung der laufenden Betriebskosten ermöglicht. Abschließend wird die Wartung und die Evolution einer WOA beschrieben, da diese gerade durch den Einsatz volatiler XaaS-Angebote einem ständigen Wandel unterliegt.

Ein Unternehmen besitzt oft bereits Altsysteme, die auch in ein neues IT-System eingebunden werden müssen. Die daraus resultierenden Unterschiede für die Umsetzung einer WOA wurden gesondert betrachtet. Dabei wurden methodische Ansätze zur Ist-Systemanalyse beschrieben und daraufhin die notwendigen Änderungen in den einzelnen Phasen des WOA-Vorgehensmodells erläutert.

Getrieben durch die einfache Architektur und das Konzept einer WOA und mit dem Ziel, auch die konkrete Umsetzung möglichst einfach zu gestalten, wurde eine adaptierte Version des Vorgehensmodells beschrieben, die sich an der agilen Methode des Feature Driven Development orientiert. Hierbei wurde herausgestellt, inwieweit sich die Umsetzung einer WOA steuern und

organisieren lässt, die die zuvor beschriebenen Phasen durchläuft und dabei zusätzlich den Vorteil der einfachen Plan- und Berechenbarkeit mit sich bringt.

Um zu zeigen, dass sich das Konzept einer WOA auch tatsächlich als umsetzbar erweist, wurde eine konkrete Realisierung des *Web Architecture Controllers* vorgenommen. Durch die Verknüpfung bestehender XaaS-Angebote konnte so die geforderte Funktionalität dieser Steuerungskomponente weitgehend umgesetzt werden.

Zum Ende der Arbeit wurde darüber hinaus eine wirtschaftliche Betrachtung einer WOA anhand einer Totalkostenberechnung vorgenommen. In dieser wurde nach der Vorstellung der TCO-Berechnungsmethode ein Ausschnitt eines verteilten IT-Systems herausgestellt und dessen Kosten berechnet. Dies geschah zunächst anhand zweier SOA-Lösungen mit der Verwendung von einerseits IBM- und andererseits Open-Source-Produkten. Anschließend wurde dieselbe Architektur als WOA-Lösung betrachtet. So konnte gezeigt werden, dass eine WOA-Realisierung bei einem Vergleich der direkten Kosten bis zu 90 % Kostenersparnis gegenüber einer SOA-Lösung aufweist. Um auch die Kosten des Softwareerstellungsprozesses einer WOA berechnen zu können, wurde im Anschluss ein mathematisches Kostenmodell aufgestellt. Dieses basiert auf dem Einsatz der agilen Methode FDD für die WOA-Umsetzung und ermöglicht dabei, die Kosten des Planungs- sowie des Umsetzungsprozesses vor bzw. in den ersten Wochen nach dem Projektstart abzuschätzen. Die Betriebskosten einer WOA sind daraufhin als weiterer Kostenfaktor identifiziert worden. Diese lassen sich für XaaS-Dienste, die oft unterschiedliche Geschäftsmodelle und damit Kostenberechnungsfunktionen besitzen, mittels einer weiteren mathematischen Funktion berechnen. Im Rahmen dieser Arbeit wurde eine Web-Applikation erstellt, die es ermöglicht die Betriebskosten für beliebige Service-Kombination zu berechnen und für unterschiedliche Nutzungsszenarien miteinander zu vergleichen.

7.2 Handlungsempfehlung

Um eine Handlungsempfehlung für die Einführung einer WOA geben zu können, werden hier zunächst die innerhalb der Arbeit identifizierten Erkenntnisse zusammengefasst. Diese werden für eine übersichtliche Darstellung innerhalb einer *Strength, Weakness, Opportunities and Threats* (SWOT)-Matrix in Stärken, Schwächen, Chancen und Gefahren eingestuft. Dabei werden Schwächen und

Stärken als interne, Chancen und Risiken dagegen als externe Einflussfaktoren wahrgenommen. Da an dieser Stelle das SWOT-Modell lediglich als Visualisierung der erarbeiteten Erkenntnisse über eine WOA eingesetzt wird (vgl. [NDH02, SVOK11]), erfolgt keine Analyse bzw. Erstellung von Übergangsstrategien zwischen den einzelnen Matrixkategorien (vgl. [Sei06]).

Gefahren Die größte Gefahr, die der Einsatz einer WOA mit sich bringt, ist in der Sicherheit der ausgelagerten Daten zu sehen. Das Konzept der WOA sieht vor, diese möglichst vollständig auszulagern. Das führt dazu, dass zumindest in einer archetypischen WOA potenziell unternehmenskritische Daten außerhalb des eigenen Unternehmens gespeichert werden. Durch die Nutzung vieler unterschiedlicher XaaS-Anbieter und durch den Austausch der Daten zwischen diversen Services für die Erfüllung eines Workflows fließen viele Daten durch eine WOA. Um hierbei dem unsachgemäßen Umgang mit Daten entgegenzuwirken, sind umfangreiche rechtliche Regelungen zwischen dem Unternehmen und den XaaS-Anbietern notwendig, so dass möglichst eine breite Vertrauensbasis hergestellt wird. Eine weitere Gefahr kann die sich teilweise sehr schnell ändernde Anbieterstruktur darstellen, bei der XaaS-Dienste manchmal so schnell wie sie erscheinen auch wieder vom Markt verschwinden. Dieses Problem lässt sich jedoch abschwächen, indem bei der Auswahl von Anbietern und der Planung einer WOA ein gesundes Maß an Vorsicht an den Tag gelegt wird. So sollte man sich nicht unbedingt für einen XaaS-Anbieter entscheiden, nur weil dieser das günstigste Angebot unterbreitet, der Anbieter selbst aber noch kein Renommee besitzt. Als weitere Gefahr kann sich der durch das Web 2.0 bekannte Beta-Status eines XaaS-Dienstes herausstellen. Dies kann unter Umständen eine Umschreibung für einen unfertigen und sich häufig ändernden Dienst sein, dessen Schnittstellensignaturen einem permanenten Änderungsvorgang unterliegen und dabei möglicherweise weder eine ordentliche Dokumentation noch eine korrekte Versionierung der API vorliegt. Daher gilt hier, den XaaS-Anbieter in jedem Fall genauer zu analysieren und zu bewerten.

Schwächen Neben den Gefahren, die als externe Einflussfaktoren wahrgenommen werden, existieren auch Schwächen einer WOA, die eher als interne Faktoren eingestuft werden. Die größte Schwäche liegt im Vergleich zu einer SOA in der Abhängigkeit von einem XaaS-Anbieter. Sollten bei diesem Pro-

bleme auftreten, die die Funktionsfähigkeit des angebotenen Dienstes beeinträchtigen, wirkt sich dies direkt auf den Betrieb des Systems aus. Dies kann im schlimmsten Fall zum Ausfall der gesamten WOA führen. Zudem kann diese ohne funktionierenden Internetanschluss nicht von den Mitarbeitern eines Unternehmens genutzt werden. Möglichkeiten, letzteres zu überbrücken, bestehen beispielsweise in der vorübergehenden Nutzung mobiler Zugriffswege wie UMTS oder Richtfunkantennen. Als weitere Schwäche sind die zum Teil noch zu wenig spezialisierten XaaS-Angebote anzuführen. Zwar existieren viele Lösungen für Standardaufgaben wie CRM, Wissensmanagement, Bezahlungsvorgänge oder E-Commerce, es mangelt jedoch an Lösungen für fachspezifische Aufgaben eines Unternehmens. Daher wird auch in einer WOA-Lösung ein gewisses Maß an Programmieraufwand notwendig sein, um ein vollständiges IT-System zu realisieren. Analog zu der zuvor genannten Gefahr der Sicherheit der Daten ist derselbe Punkt auch als Schwäche aufzuführen: das Sicherheitskonzept für die Auslagerung von Daten. Dieses kann zusätzlich als interner Einflussfaktor angesehen werden, da durch das Outsourcing viel Aufwand in geeignete Sicherheits- und Rechtestrukturen fließt, der in einer reinen SOA-Lösung zwar ebenfalls auftritt, aber nicht in dem Maße wie bei einer WOA. Zuletzt muss auch die Datenlokalität als Schwäche aufgezählt werden. Für Anwendungsfälle, bei denen viele Daten für die Ausführung eines Services von einem Server auf den anderen übertragen werden müssen, ist eine WOA grundsätzlich nicht geeignet. Ein Unternehmen, welches viele datenintensive Workflows benötigt, sollte möglichst lokale Lösungen für den Datentransport bevorzugen.

Chancen Eine WOA bietet aber auch Chancen, die den Gefahren gegenüberstehen. Zunächst ist hierzu die Angebotsvielfalt der XaaS-Dienste zu zählen. Obwohl vor allem Aufgaben von Standardsoftware und Web-lastigen Themen abgedeckt werden, ist die Auswahl meist so groß, dass man sich nicht zwangsläufig an einen einzigen Anbieter binden muss. Dazu kommt, dass u. a. durch die Entwicklungen des Cloud Computing ein ständiger Anbieterzuwachs zu verzeichnen ist. Dies lässt darauf hoffen, dass in Zukunft genügend Dienste bereitstehen, um auch WOA-Lösungen zu realisieren, die über Standardaufgaben hinausgehen. Der zuvor als Gefahr eingestufte Beta-Status eines XaaS-Angebots kann gleichzeitig als eine Chance wahrgenommen werden. Durch die ständige Verbesserung eines Dienstes kann durch professionellen Um-

gang mit Schnittstellen seitens des Anbieters eine kontinuierliche Fehlerbehebung und Erweiterung eines Services stattfinden, die dem Endnutzer zugutekommen. Die mit Abstand größte Chance hinsichtlich der Wirtschaftlichkeit eines IT-Systems ist in der Möglichkeit der Umsetzung einer kostengünstigen Software-Infrastruktur zu sehen, die durch die potenziell vollständige Auslagerung von Hard- und Software zu erreichen ist.

Stärken Als Stärke einer WOA ist hier zunächst der einfache Aufbau der Architektur hinsichtlich der Hard- und Software im Vergleich zu einer SOA zu nennen. Vor allem die Nutzung von einfachen Web-Standards für die Vernetzung sowie die Reduzierung bzw. der Verzicht auf eigene Server-Hardware bringen hierbei Vorteile. Damit ist generell weniger Hardware für das IT-System eines Unternehmens anzuschaffen. Eine große Stärke ist analog zur Chance der Kostenreduzierung in der Kostentransparenz zu sehen. Durch die Nutzung der XaaS-Dienste lassen sich, wie im Betriebskostenmodell gezeigt, die laufend entstehenden Kosten für die Nutzung von Services zum einen im Voraus simulieren und zum anderen im Betrieb kontinuierlich protokollieren und kontrollieren. Mit dem modularen Aufbau aus verschiedenen XaaS-Diensten wird eine flexible Prozessstruktur aufgebaut, die leicht an sich ändernde Gegebenheiten der zugrunde liegenden fachlichen Geschäftsprozesse anzupassen ist. Folgt man hierbei dem REST-Paradigma für die Beschreibung und Nutzung von Services, ist die Verwendung oder das Tauschen von konkreten XaaS-Services aufgrund der selbstbeschreibenden Service-Schnittstellen leicht zu bewerkstelligen. Die Ergebnisse der Arbeit sind in Tabelle 7.1 dargestellt.

Da sich sämtliche Web-orientierten Dienste, auf die sich das Konzept einer WOA stützt, bereits „in der Cloud" befinden, lässt sich das hier entwickelte WOA-Vorgehensmodell offensichtlich auch im Kontext des derzeit stark diskutierten Cloud Computing verwenden. Viele der in der SWOT-Matrix genannten Aspekte lassen sich ebenfalls auf das Cloud Computing übertragen.

Die Kernfrage einer Handlungsempfehlung kann an dieser Stelle folgendermaßen lauten: „*Unter welchen Gesichtspunkten kann die Umsetzung eines verteilten IT-Systems eines Unternehmens durch eine WOA empfohlen werden?*". Diese Frage lässt sich nicht eindeutig durch Wenn-Dann-Regeln beantworten, aber durch die folgenden Aspekte ist zumindest eine Einschätzung möglich, ob eine Systemumsetzung durch eine WOA sinnvoll sein kann, oder ob es Gründe

Tabelle 7.1: SWOT-Matrix für eine WOA.

Stärken	**Schwächen**
• Einfacher Aufbau der Architektur • Wenig Bedarf an eigenen Infrastrukturkomponenten • Kostentransparenz • Flexible Prozessstruktur	• Abhängigkeit von Dienstleistern und Internetanschluss • Mangel an spezialisierten Diensten • Sicherheitskonzept • Datenlokalität
Chancen	**Gefahren**
• Service-Vielfalt • Beta-Status von Services • Ständiges Anbieterwachstum • Kostengünstige/-transparente Software-Infrastruktur	• Datensicherheit • Beta-Status von Services • Volatile Anbieterstruktur

gibt, die diese als ungeeignet erscheinen lassen. Die folgenden beispielhaften Situationen sprechen jeweils einzeln betrachtet für den Einsatz einer WOA:

- Das Kerngeschäft des Unternehmens ist in weiten Teilen durch Standardsoftware (CRM-, ERP-Systeme, Office-Software, E-Commerce-Bestandteile) geprägt.
- Es existieren Service-orientierte Ansätze bzw. Software mit Web-basierten Schnittstellen für fachlogische Komponenten, die sich durch SOA-Standards als schwer wartbar herausstellen und teuer sind.
- Das Unternehmen besitzt bisher kein umfassendes IT-System (oder möchte ein Neues für einen spezifischen Bereich aufbauen) und eine Systemumsetzung kann ohne Altlasten realisiert werden.
- Es handelt sich um ein Start-up-Unternehmen, welches ein dynamisch wachsendes System als Herausforderung ansieht und dieses Stück für Stück aufbauen möchte.
- Das Unternehmen will Kosten sparen und dafür möglichst viel Hard- und Software auslagern.
- Es soll lediglich ein Teil der vorhandenen IT-Landschaft auf eine WOA migriert bzw. realisiert werden, für dessen Umsetzung etablierte XaaS-Anbieter existieren. Darüber hinaus besteht die Möglichkeit und die Bereitschaft, das interne Netzwerk für die Nutzung von Schnittstellen nach außen hin zu öffnen.

Diese Punkte beschreiben dabei oft Gründe, aus denen man generell über das Outsourcing und die Konzentration auf das Kerngeschäft eines Unternehmens nachdenkt. Dem gegenüber stehen Situationen, die sich für eine WOA-Umsetzung eher als hinderlich erweisen können, da darin entweder die Ziele einer WOA nicht unterstützt oder die damit erreichbaren Chancen nicht genutzt werden. Liegt eines der folgenden Szenarien vor, so sollte die Umsetzung einer WOA nicht in Betracht gezogen werden.

- Es handelt sich um ein Großunternehmen mit dem Anspruch einer hochgradig skalierbaren Server-Farm, welches Dutzende eigener Server betreibt und eine Vielzahl technischen Personals dafür beschäftigt.

- Das Unternehmen hat seine IT-Systeme bereits in eine „private Cloud" ausgelagert und es sind keine weitreichenden Vorteile durch den Einsatz einer WOA zu erwarten.
- Das Unternehmen besitzt viele Altsysteme, die innerhalb des Unternehmens verbleiben und in ein neues IT-System eingebunden werden sollen. Dabei wird das neue IT-System hauptsächlich für die Kommunikation der Altsysteme verwendet und es werden wenige, bis keine XaaS-Funktionalitäten benötigt.
- Das Unternehmen ist nicht bereit, Daten in irgendeiner Form auszulagern.
- Die durch XaaS-Dienste umzusetzenden Funktionalitäten benötigen für die Arbeit häufig eine große Menge an Daten, die dann über das Internet übertragen werden müssen. Dies tritt bei der Umsetzung von Data Warehouse-Aufgaben auf, die wegen des hohen Datenaufkommens keine geeigneten Kandidaten für eine WOA-Verwendung sind.

Eine konkrete Beurteilung der Eignung einer WOA-Lösung für eine gegebene Situation ist daher immer ein Entscheidungsprozess, in dem viele Faktoren zu beachten sind. Generell müssen dabei immer die Größe des Unternehmens, die bereits eingesetzten IT-Komponenten eines bestehenden Systems, die Zielvorgabe für das neu zu erstellende System sowie die Möglichkeit der Auslagerung von Daten beachtet und untersucht werden. Unbenommen davon existieren Stärken und Chancen, die in dieser Arbeit aufgezeigt wurden, die sich beim sinnvollen Einsatz einer WOA für ein Unternehmen auszahlen können.

7.3 Fazit und Ausblick

Einige Fragen blieben in der vorliegenden Arbeit unbeantwortet und zeichnen sich daher als interessante zukünftige Forschungsgebiete ab. Ein Aspekt der WOA, der in dieser Arbeit nicht betrachtet wurde, sind die Vorgänge auf der Seite des Anbieters eines XaaS-Dienstes. Hier sind beispielsweise Fragen wie die Wartbarkeit und Versionierung von Schnittstellen, das Konzeptionieren von XaaS-Angeboten für den Kunden oder der auch auf Seiten des Anbieters

kostengünstige Betrieb der eigenen XaaS-Dienste von Interesse. Wie einfach sich eine moderne Web API mit Hilfe von bestehenden Frameworks programmieren lässt, wurde in dieser Arbeit daher ebenfalls nicht betrachtet. Der Fokus lag in dieser Arbeit nur auf dem Anwender einer WOA, nicht dem Anbieter von XaaS-Diensten. Eine weitere Herausforderung für Anbieter wie auch Anwender stellt die Sicherstellung der Datenkonsistenz über mehrere Data Center hinweg dar. In diesem Zusammenhang spielt auch der Energieverbrauch von IT-Komponenten eine große Rolle. Dieser wurde bei der TCO-Berechnung einer WOA betrachtet, eine ausführliche Auseinandersetzung mit dem Energiebedarf einer WOA-Umsetzung und allgemein von XaaS-Diensten im Kontext des Cloud Computing ist jedoch eine Forschungsfrage, die vor dem Hintergrund stromsparender IT-Hardware und der „Green-IT" sehr interessant ist.

Auch der WAC als Kernkomponente wurde zwar erläutert und in seiner Rolle definiert, es wurde jedoch keine Implementierung eines idealen WAC beschrieben. Zwar lassen sich, wie am Beispiel des Dienstes *RunMyProcess* gezeigt, bestehende Anbieter für die Kontrolle einer WOA verwenden, diese setzen aber nicht alle geforderten Eigenschaften in idealer Weise um. Hier lässt sich in Zukunft über eine Implementierung nachdenken.

Die Datensicherheit kann bisher noch als ein großer Mangel des WOA-Konzepts benannt werden. In diesem Bereich werden bereits Verschlüsselungsmethoden erforscht, die das Speichern verschlüsselter Daten bei einem DaaS-Anbieter ermöglichen könnten. Ein wichtiges Ziel wäre hierbei aus Sicht der WOA, dass eine ausgelagerte Datenbank die Daten komplett verschlüsselt ablegt und dennoch die wichtigen Eigenschaften einer relationalen Datenbank bereitstellt, wie Indizes, Fremdschlüsselbeziehungen und Integritätsbedingungen. Sollte hier ein Durchbruch gelingen, stünde der großflächigen Nutzung externer Speicherdienste und damit dem Beheben eines großen Nachteils einer WOA, gerade in Deutschland, nichts mehr im Weg.

Betrachtet man die Entwicklungen des Cloud Computing, die Beliebtheit einiger XaaS-Anbieter wie beispielsweise *Salesforce.com* und zusätzlich die Bestrebungen der Unternehmen und Hersteller von Hardware, für die Zukunft energiesparende Rechenzentren zu bauen, könnte sich das Architekturkonzept einer WOA als tragfähig und realitätsnah erweisen. Es wird sich zeigen, in welche Richtung sich die Möglichkeiten von XaaS-Angeboten und der Bereich des Cloud Computing technisch wie auch rechtlich weiterentwickeln und ob dadurch die Realisierung einer WOA weiter vereinfacht werden wird. Eine WOA ist zwar aktuell für verteilte IT-Systeme nicht universell einsetzbar

und kein vollständiger Ersatz für SOA. Sie eignet sich aber gut für einige IT-Herausforderungen sowie manche Unternehmensstrukturen und bietet dabei große wirtschaftliche Vorteile.

Literaturverzeichnis

[AA06] Arsanjani, Ali ; Allam, Abdul: Service-Oriented Modeling and Architecture for Realization of an SOA. In: *2006 IEEE International Conference on Services Computing (SCC 2006), 18.-22. September 2006, Chicago, Illinois, USA*. 2006, S. 521

[ACKM04] Alonso, Gustavo ; Casati, Fabio ; Kuno, Harumi ; Machiraju, Vijay: *Web services: Concepts, architectures and applications*. Berlin Heidelberg : Springer, 2004

[AES08] Abuosba, Khalil A. ; El-Sheikh, Asim A.: Formalizing Service-Oriented Architectures. In: *IT Professional* 10 (2008), Juli, S. 34–38

[Afs07] Afshar, Mohamad: *SOA Governance: Framework and Best Practices – An Oracle White Paper*. Internet. http://www.oracle.com/us/technologies/soa/oracle-soa-governance-best-practice-066427.pdf. Version: Mai 2007, Abruf: 13. Februar 2010

[AGA+08] Arsanjani, A. ; Ghosh, S. ; Allam, A. ; Abdollah, T. ; Ganapathy, S. ; Holley, K.: SOMA: A method for developing service-oriented solutions. In: *IBM Systems Journal* 47 (2008), Nr. 3, S. 377–396

[Alb08] Alby, Tom: *Web 2.0: Konzepte, Anwendungen, Technologien*. München : Hanser Fachbuchverlag, 2008

[AST07] Andrew S. Tanenbaum, Maarten van S.: *Distributed Systems - Principles and Paradigms*. 2. Auflage. Upper Saddle River, New Jersey, USA : Pearson Prentice Hall, 2007

[ATM08] Arcelli, F. ; Tosi, C. ; M., Zanoni: Can Design Pattern Detection be Useful for Legacy System Migration towards SOA? In: *SDSOA 08:*

Proceedings of the 2nd international workshop on Systems development in SOA environments. New York, USA, 2008, S. 63–68

[AZE+07a] Arsanjani, Ali ; Zhang, Liang-Jie ; Ellis, Michael ; Allam, Abdul ; Channabasavaiah, Kishore: *Design an SOA solution using a reference architecture*. Internet. http://www.ibm.com/developerworks/library/ar-archtemp/. Version: März 2007, Abruf: 26.02.2011

[AZE+07b] Arsanjani, Ali ; Zhang, Liang-Jie ; Ellis, Michael ; Allam, Abdul ; Channabasavaiah, Kishore: S3: A Service–Oriented Reference Architecture. In: *IT Professional* 9 (2007), Nr. 3, S. 10–17

[Bal98] Balzert, Helmut: *Lehrbuch der Software–Technik: Software-Management, Software Qualitätssicherung, Unternehmensmodellierung*. 1. Auflage. Heidelberg : Spektrum-Akademischer Verlag, 1998

[Bal01] Balzert, Helmut: *Lehrbuch der Software–Technik: Software–Entwicklung*. 2. Auflage. Heidelberg : Spektrum-Akademischer Verlag, 2001

[Bar03] Barry, Douglas K.: *Web Services and Service-Oriented Architecture: Your Road Map to Emerging IT*. Burlington, MA, USA : Morgan Kaufman, 2003

[Bar05] Barroso, Luiz A.: The Price of Performance. In: *ACM Queue* 3 (2005), Nr. 7, S. 48–53

[BBF05] Bieberstein, Nobert ; Bose, Sanjay ; Fiammante, Marc: *Service-Oriented Architecture Compass*. Prentice Hall International, 2005

[BBM+01] Ballinger, Keith ; Brittenham, Peter ; Malhotra, Ashok ; Nagy, William A. ; Pharies, Stefan: *Web Services Inspection Language (WS-Inspection) 1.0*. Internet. http://public.dhe.ibm.com/software/dw/specs/ws-wsilspec/ws-wsilspec.pdf. Version: November 2001, Abruf: 11. Dezember 2010

[Bec04] Beck, Kent: *Extreme Programming Explained: Embrace Change*. 2. Auflage. Amsterdam, Niederlande : Addison-Wesley Longman, 2004

[Bel08] Bell, Michael: *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*. 1. Auflage. John Wiley & Sons, 2008

[Ben83] Benington, Herbert D.: Production of Large Computer Programs. In: *IEEE Annals of the History of Computing* 5 (1983), Nr. 4, S. 350–361

[BF08] Blanchard, Benjamin S. ; Fabrycky, Wolter J.: *Systems Engineering and Analysis*. 4. Auflage. Upper Saddle River, NJ, USA : Prentice Hall, 2008

[BHBL09] Bizer, Christian ; Heath, Tom ; Berners-Lee, Tim: Linked Data - The Story So Far. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 5 (2009), Nr. 3, S. 1–22

[BK08] Becker, Jörg ; Kahn, Dieter: Der Prozess im Fokus. In: Becker, Michael; Rosemann M. Jörg; Kugeler K. Jörg; Kugeler (Hrsg.): *Prozessmanagement. Ein Leitfaden zur prozessorientierten Organisationsgestaltung*. 6. Auflage. Berlin Heidelberg : Springer, 2008, S. 3–16

[BL98] Berners-Lee, T.: *Cool URIs don't change*. Internet. http://www.w3.org/Provider/Style/URI.html. Version: 1998, Abruf: 12. Februar 2011

[BLCC+06] Berners-Lee, Tim ; Chen, Yuhsin ; Chilton, Lydia ; Connolly, Dan ; Dhanaraj, Ruth ; Hollenbach, James ; Lerer, Adam ; Sheets, David: Tabulator: Exploring and analyzing linked data on the semantic web. In: *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006

[BLFF96] Berners-Lee, T. ; Fielding, R. ; Frystyk, H.: *Hypertext Transfer Protocol – HTTP/1.0 - RFC 1945*. Internet. http://tools.ietf.org/html/rfc1945. Version: May 1996, Abruf: 13. Februrar 2011

[BLH08] Buxmann, P. ; Lehmann, S. ; Hess, T.: Software as a Service. In: *Wirtschaftsinformatik* 6 (2008), S. 500–503

[BMM09] Becker, Jörg ; Matzner, Martin ; Müller, Oliver: Comparing Architectural Styles for Service-Oriented Architectures - a REST vs. SOAP Case Study. In: *Information Systems Development*. Springer US, 2009, S. 207–215

[BMT06] Brown, William A. ; Moore, Garry ; Tegan, William: *SOA governance – IBM's approach*. Internet. ftp://ftp.software.ibm.com/software/soa/pdf/SOA_Gov_Process_Overview.pdf. Version: August 2006, Abruf: 13. Februar 2011

[BN84] Birrell, Andrew D. ; Nelson, Bruce J.: Implementing Remote Procedure Calls. In: *ACM Transactions on Computer Systems (TOCS)* 2 (1984), Februar, Nr. 1, S. 39–59

[BPSM+08] Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Internet. http://www.w3.org/TR/xml/. Version: November 2008, Abruf: 13. Februar 2011

[Bro07] Brocke, Jan vom: *Serviceorientiertes Prozesscontrolling, Gestaltung von Organisations- und Informationssystemen bei Serviceorientierten Architekturen*, Westfälische Wilhelms-Universität Münster, Habilitationsschrift, 2007

[BS04] Becker, Jörg ; Schütte, Reinhard: *Handelsinformationssysteme. Domänenorientierte Einführung in die Wirtschaftsinformatik*. 2. Auflage. Frankfurt am Main : Moderne Industrie, 2004

[BSW95] Beutelspacher, A. ; Schwenk, J. ; Wolfenstetter, K.-D.: *Moderne Verfahren der Kryptographie*. vieweg, 1995

[BT03] Boehm, Barry ; Turner, Richard: *Balancing Agility and Discipline: A Guide for the Perplexed*. 1. Auflage. Boston, MA, USA : Addison-Wesley Professional, 2003

[BW08] Bleek, Wolf-Gideon ; Wolf, Henning: *Agile Softwareentwicklung - Werte, Konzepte und Methoden*. 1. Auflage. Heidelberg : dpunkt.verlag, 2008

[CDK02] Coulouris, George ; Dollimore, Jean ; Kindberg, Tim ; Studium, Pearson (Hrsg.): *Verteilte Systeme - Konzepte und Design*. 3. Auflage. Addison-Wesley, 2002

[CDNDP+05] Colombo, Massimiliano ; Di Nitto, Elisabetta ; Di Penta, Massimiliano ; Distante, Damiano ; Zuccalà, Maurilio: Speaking a Common language: A conceptual Model for Describing Service-Oriented Systems. In: *ICSOC 2005: International Conference on Service Oriented Computing, Amsterdam*. Berlin Heidelberg : Springer, 2005, S. 48–60

[CDOML08] Cavalli, A.R. ; De Oca, E.M. ; Mallouli, W. ; Lallali, M.: Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints. In: *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. Vancouver, BC, Canada, 2008, S. 315–318

[CFP05] Cearley, David W. ; Fenn, Jackie ; Plummer, Daryl C.: *Gartner's Position on the Five Hottest IT Topics and Trends in 2005*. Internet. http://www.gartner.com/DisplayDocument?id=480912. Version: Mai 2005, Abruf: 13. Februar 2011

[Cha04] Chappell, David A.: *Enterprise Service Bus*. 1. Auflage. Peking, China : O'Reilly, 2004

[Che76] Chen, Peter: The Entity-Relationship Model – Toward a Unified View of Data. In: *ACM Transactions on Database Systems* 1 (1976), Nr. 1, S. 9–36

[CHKT05] Conrad, Stefan ; Hasselbring, Wilhelm ; Koschel, Arne ; Tritsch, Roland: *Enterprise Application Integration*. 1. Auflage. Heidelberg : Spektrum-Akademischer Verlag, 2005

[Cof09] Coffee, Peter: Emergence of the Platform as a Service for Enterprise Technology Alignment. In: *Information Management und Consulting* 24 (2009), Nr. 2, S. 24–33

[Cro06a] Crockford, D.: *The application/json Media Type for JavaScript Object Notation (JSON)*. Internet. http://tools.ietf.org/html/rfc4627. Version: Juli 2006, Abruf: 13. Februar 2011

[Cro06b] Crockford, Douglas: JSON: The fat-free alternative to XML. In: *XML 2006*. Boston, Dezember 2006

[Cus10] Cusumano, Michael: Cloud Computing and SaaS as New Computing Platforms. In: *Communications of the ACM* 53 (2010), April, Nr. 4, S. 27–29

[DA99] Dierks, T. ; Allen, C.: *The TLS Protocol Version 1.0 - RFC 2246*. Internet. http://www.ietf.org/rfc/rfc2246.txt. Version: January 1999, Abruf: 14. Februar 2011

[DFPS07] De Capitani di Vimercati, S. ; Foresti, S. ; Paraboschi, S. ; Samarati, P.: Privacy of Outsourced Data. In: *Digital Privacy: Theory, Technologies and Practices*. Auerbach Publications (Taylor and Francis Group), Dezember 2007

[DH76] Diffie, W. ; Hellman, M.: New Directions in Cryptography. In: *IEEE Transactions on Information Theory* IT-22 (1976), November, Nr. 6, S. 644–654

[DJI05] Dostal, Wolfgang ; Jeckle, Mario ; Ingo, Melzer: *Service–orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis*. 1. Auflage. München : Elsevier Spektrum Akad. Verl., 2005

[EAK06] Erradi, Abdelkarim ; Anand, Sriram ; Kulkarni, Naveen: SOAF: An Architectural Framework for Service Definition and Realization. In: *IEEE International Conference on Services Computing 2008, July 8-11, 2008, Honolulu, Hawaii, USA* (2006), S. 151–158

[Eck09] Eckert, Claudia: *IT-Sicherheit: Konzepte – Verfahren – Protokolle*. 6. Auflage. München : Oldenbourg, 2009

[Els05] Elsener, Markus: *Kostenmanagement in der IT*. Bonn : mitp, 2005

[Els06] Elsässer, Wolgang: *ITIL einführen und umsetzen: Leitfaden für effizientes IT-Management durch Prozessorientierung*. 2., erw. Auflage. München : Hanser Fachbuchverlag, 2006

[Erl04] Erl, Thomas: *Service-Oriented Architecture: A field guide to integrating XML and Web services*. 9. Auflage. Upper Saddle River, NJ, USA : Prentice Hall International, 2004

[Erl05] Erl, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design*. 1. Auflage. Upper Saddle River, NJ, USA : Prentice Hall International, 2005

[Erl07] Erl, Thomas: *Service-Oriented Architecture: Principles of service design*. 1. Auflage. Upper Saddle River, NJ, USA : Prentice Hall International, 2007

[Eur09] European Computer Manufacturers Association: *ECMA-Script Language Specification 5th Edition (ECMA-262)*. Internet. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf. Version: Dezember 2009, Abruf: 14. Februar 2011

[FGM+99] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1 - RFC2616*. Internet. http://www.ietf.org/rfc/rfc2616.txt. Version: Juni 1999, Abruf: 14. Februar 2011

[FH07] Friedrichsen, Uwe ; Hackel, Dirk: Wege zu einer SOA. In: *Information Management & Consulting* 3 (2007), Nr. 22, S. 50–57

[FHBH+99] Franks, J. ; Hallam-Baker, P. ; Hostetler, J. ; Lawrence, S. ; Leach, P. ; Luotonen, A. ; Sink, E. ; Stewart, L.: *HTTP Authentication: Basic and Digest Access Authentication*. Internet. http://tools.ietf.org/html/rfc2617. Version: Juni 1999, Abruf: 14. Februar 2011

[Fie00] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, USA, Dissertationsschrift, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[FZ09] Finger, Patrick ; Zeppenfeld, Klaus: *SOA und WebServices*. Berlin Heidelberg : Springer, 2009

[Gal08] Gall, Nick: *WOA: Putting the Web Back in Web Services*. Internet. http://blogs.gartner.com/nick_gall/2008/11/19/woa-putting-the-web-back-in-web-services/. Version: November 2008, Abruf: 14. Februar 2011

[Gan06] Ganci, John: *Patterns: SOA foundation service creation scenario*. 1. Auflage. Poughkeepsie, NY, USA : International Technical Support Organization, 2006 (IBM Redbooks)

[Gar03] Gartner: *Distributed Computing - Chart of Accounts*. Internet. http://www.gartner.com/4_decision_tools/modeling_tools/costcat.pdf. Version: Juni 2003, Abruf: 14. Februar 2010

[GGW09] Geissdörfer, Klaus ; Gleich, Ronald ; Wald, Andreas: Standardisierungspotentiale lebenszyklusbasierter Modelle des strategischen Kostenmanagements. In: *Zeitschrift für Betriebswirtschaft* 6 (2009), S. 693–716

[GHN09] Governor, James ; Hinchcliffe, Dion ; Nickull, Duane: *Web 2.0 Architectures – What entrepreneurs and information architects need to know*. O'Reilly Media, 2009

[GHR06] Gudgin, Martin ; Hadley, Marc ; Rogers, Tony: *Web Services Addressing 1.0 - Core*. Internet. http://www.w3.org/TR/ws-addr-core/. Version: Mai 2006, Abruf: 14. Februar 2011

[GL04] Grob, Heinz L. ; Lahme, Norman: Total Cost of Ownership-Analyse mit vollständigen Finanzplänen. In: *Controlling* 16 (2004), Nr. 3, S. 157–164

[GM05] Gadatsch, Andreas ; Mayer, Elmar: *Masterkurs IT-Controlling*. Wiesbaden : Vieweg Friedr. + Sohn Verlag, 2005

[Gra03] Grabowski, Jens: *Specification Based Testing of Real-Time Distributed Systems: Languages, Tools and Applications*. Berlin : Logos, 2003

[Gro06] Grob, Heinz L.: *Einführung in die Investitionsrechnung: Eine Fallstudiengeschichte*. 5., vollst. überarb. u. erw. Auflage. München : Vahlen, 2006

[GS06] Grässle, Patrick ; Schacher, Markus: *Agile Unternehmen durch Business Rules - Der Business Rules Ansatz*. Berlin Heidelberg : Springer, 2006

[Haa05] Haas, H.: Reconciling Web services and REST services (Keynote). Version: 2005. http://www.w3.org/2005/Talks/1115-hh-k-ecows/#%281%29, Abruf: 02.03.2011. In: *Proc. of the 3rd IEEE European Conference on Web Services (ECOWS 2005)*. Växjö, Sweden, 2005

[Had06] Hadley, Marc J. ; Inc., Sun M. (Hrsg.): *Web Application Description Language (WADL)*. Internet. https://wadl.dev.java.net/wadl20061109.pdf. Version: 2006, Abruf: 14. Februar 2011

[Hed09] Hedley, Marc: *Web Application Description Language*. Internet. http://www.w3.org/Submission/wadl/. Version: August 2009, Abruf: 14. Februar 2011

[Hen08] Henning, Michi: The Rise and Fall of CORBA. In: *Communications of the ACM* 51 (2008), August, Nr. 8, S. 52–57

[Hia00] Hiatt, Charlotte J.: *A Primer for Disaster Recovery Planning in an IT Environment*. Hershey, PA, USA : IGI Publishing, 2000

[Hin08] Hinchcliffe, Dion: *What Is WOA? It's The Future of Service-Oriented Architecture*. Internet. http://hinchcliffe.org/archive/2008/02/27/16617.aspx. Version: Februar 2008, Abruf: 14. Februar 2011

[HKKR05] Hitz, Martin ; Kappel, Gerti ; Kapsammer, Elisabeth ; Retschitzegger, Werner: *UML @ Work. Objektorientierte Modellierung mit UML 2*. 3., aktualisierte und überarbeitete Auflage. Heidelberg : dpunkt.verlag, 2005

[HL07] Hack, Stefan ; Lindemann, Markus A.: *Enterprise SOA einführen*. 1. Auflage. Bonn : SAP PRESS, 2007

[HLs06] Heutschi, Roger ; Legner, Christine ; Österle, Hubert: Serviceorientierte Architekturen: Vom Konzept zum Einsatz in der Praxis. In: *Integration, Informationslogistik und Architektur, DW2006, 21.-22. Sept. 2006, Friedrichshafen* (2006), S. 361–382

[HLV07] Hagemann, Stephan ; Letz, Carolin ; Vossen, Gottfried: Web Service Discovery – Reality Check 2.0. In: *Proc. 3rd International Conference on Next Generation Web Services Practices (NWeSP), Seoul, Korea* (2007), S. 113–118

[HMPR04] Hevner, Alan R. ; March, Salvatore T. ; Park, Jinsoo ; Ram, Sudha: Design science in information systems research. In: *Management Information Systems Quarterly* 28 (2004), Nr. 1, S. 75–106

[HPFS02] Housley, R. ; Polk, W. ; Ford, W. ; Solo, D.: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 3280)*. Internet. http://tools.ietf.org/html/rfc3280. Version: 2002, Abruf: 14. Februar 2011

[HSSJS08] Hoyer, Volker ; Stanoesvka-Slabeva, Katarina ; Janner, Till ; Schroth, Christoph: Enterprise Mashups: Design Principles towards the Long Tail of User Needs. In: *IEEE International Conference on Services Computing 2008, July 8-11, 2008, Honolulu, Hawaii, USA* Vol. 2 (2008), S. 601–602

[HTV10] Haselmann, Till ; Thies, Gunnar ; Vossen, Gottfried: Looking into a REST-based API for Database-as-a-Service Systems. In: *12th IEEE International Conference on Commerce and Enterprise Computing, Shanghai, China* (2010), S. 17–24

[HV10] Haselmann, Till ; Vossen, Gottfried: Database-as-a-Service für kleine und mittlere Unternehmen. In: *Working Paper No. 3, Förderkreis der Angewandten Informatik, IHK-Bericht* (2010), November

[HW03] Hohpe, Gregor ; Woolf, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003

[IEE01] IEEE Standards Association: *Recommended Practice for Architectural Description of Software-Intensive Systems, ANSI/IEEE Std 1471 :: ISO/IEC 42010*. August 2001

[Int89] International Organization for Standardization: *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture ISO 7498/2*. Januar 1989

[JM05] Jones, S ; Morris, M: *A methodology for service architectures*. http://www.oasis-open.org/committees/download.php/15071/A%20methodology%20for%20Service%20Architectures%201%202%204%20-%20OASIS%20Contribution.pdf. Version: Oktober 2005, Abruf: 14. Februar 2011

[Jos08] Josuttis, Nicolai: *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*. Heidelberg : dpunkt.verlag, 2008

[Jos09] Josuttis, Nicolai: *Das SOA-Manifest*. 1. Auflage. Heidelberg : dpunkt.verlag, 2009

[KAB+04] Keen, Martin ; Acharya, Amit ; Bishop, Susan ; Hopkins, Alan ; Milinski, Sven ; Nott, Chris ; Robinson, Rick ; Adams, Jonathan ; Verschueren, Paul: *Patterns: Implementing an SOA Using an Enterprise Service Bus*. Internet. http://www.redbooks.ibm.com/redbooks/pdfs/sg246346.pdf. Version: Juli 2004, Abruf: 14. Februar 2011

[KBD+08] Keen, Martin ; Bhogal, Kulvir S. ; Dube, Sunil ; Kaushik, Rashmi ; Putte, Geert Van d. ; Wong, Albert ; Yallapragada, Sravan: *Case study: SOA Retail Business Pattern*. Internet. http://www.redbooks.ibm.com/redpapers/pdfs/redp4459.pdf. Version: September 2008, Abruf: 14. Februar 2011

[KBS07] Krafzig, Dirk ; Banke, Karl ; Slama, Dirk: *Enterprise SOA: Best Practices für Serviceorientierte Architekturen–Einführung, Umsetzung, Praxis*. 1. Auflage. Bonn : mitp, 2007

[Kee04] Keen, Martin: *Patterns: Implementing an SOA using an Enterprise Service Bus*. 1. Auflage. Research Triangle Park, NC, USA : IBM International Technical Support Organization, 2004

[KKB07] Klose, K. ; Knackstedt, R. ; Beverungen, D.: Identification of Services - A Stakeholder-Based Approach to SOA Development and its Application in the Area of Production Planning. In: H., Winter R. Schelp J. J. Schelp J. (Hrsg.): *Proceedings of the Fifteenth European Conference on Information Systems*. Universität St. Gallen, St. Gallen, 2007, S. 1802–1814

[Klü07] Klückmann, Jörg: *10 Steps to Business-Driven SOA*. Internet. http://www.ids-scheer.com/set/5020/ARIS_Expert_Paper_-_10_Steps_to_SOA_Klueckmann_2007-03_en.pdf. Version: Juni 2007, Abruf: 14. Februar 2011

[KP09] Kätker, Stefan ; Patig, Susanne: Model-driven Development of Serviceoriented Business Application Systems. In: *Business Services: Konzepte, Technologien, Anwendungen. 9. Internationale Tagung Wirtschaftsinformatik 25.-27. Februar 2009, Wien* (2009), S. 171–180

[Krc05] Krcmar, Helmut: *Informationsmanagement*. 4., überarb. und erw. Auflage. Berlin Heidelberg : Springer, 2005

[Let07] Letz, Carolin: *Web Service Detection in Service-oriented Software Development: A Semantic Syntactic Approach*, Westfälische Wilhelms-Universität, Münster, Dissertationsschrift, 2007

[Ley01] Leymann, Frank ; IBM (Hrsg.): *Web Services Flow Language*. Internet. http://xml.coverpages.org/WSFL-Guide-200110.pdf. Version: Mai 2001, Abruf: 14. Februar 2011

[LS08] Lamparter, Steffen ; Sure, York: An Interdisciplinary Methodology for Building Service-oriented Systems on the Web. In: *IEEE International Conference on Services Computing 2008, July 8-11, 2008, Honolulu, Hawaii, USA* Vol. 2 (2008), S. 475–478

[LSJAnCM08] López-Sanza, Marcos ; J. Acuñaa, César ; Cuestaa, Carlos E. ; Marcosa, Esperanza: Modelling of Service-Oriented Architectures

with UML. In: *Electronic Notes in Theoretical Computer Science* 194 (2008), April, Nr. 4, S. 23–37

[Mal06] Malinverno, Paolo: *Service-Oriented Architecture Craves Governance*. Internet. http://www.gartner.com/DisplayDocument?id=488180. Version: Januar 2006, Abruf: 14. Februar 2011

[Man08] Manes, Anne T.: *WOA, ROA, SOA: When will it stop?* Internet. http://apsblog.burtongroup.com/2008/05/woa-roa-soa-whe.html. Version: Mai 2008, Abruf: 14. Februar 2011

[Mat07] Mathas, Christoph: *SOA intern: Praxiswissen zu serviceorientierten IT-Systemen*. 1. Auflage. München : Hanser Fachbuchverlag, 2007

[MBH+06] Martin, D. ; Burstein, M. ; Hobbs, J. ; Lassila, O. ; McDermott, D. ; McIlraith, S. ; Narayanan, S. ; Paolucci, M. ; Parsia, B. ; Payne, T. ; Sirin, E. ; Srinivasan, N. ; Sycara, K.: *OWL-S: Semantic Markup for Web Services. OWL-S Coalition*. Internet. http://www.ai.sri.com/daml/services/owl-s/1.2/. Version: 2006, Abruf: 14. Februar 2011

[MBQM08] Mos, Adrian ; Boulze, Alain ; Quaireach, Samuel ; Meynier, Claude: Multi-Layer Perspectives and Spaces in SOA. In: *In SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments, Leipzig* (2008), S. 69–74

[McA05] McAfee, Andrew: Will Web Services Really Transform Collaboration? In: *MIT Sloan Management Review* 46 (2005), Nr. 2

[McB02] McBreen, Pete: *Questioning Extreme Programming*. 1. Auflage. Pearson Education, 2002

[MELT08] McCabe, F. G. ; Estefan, J. A. ; Laskey, K. ; Thornton, D.: *Reference Architecture for Service Oriented Architecture Version 1.0*. Internet. http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf. Version: April 2008, Abruf: 14. Februar 2011

[MG09] Mell, Peter ; Grance, Tim: Effectively and Securely Using the Cloud Computing Paradigm. In: *FISSEA 2009 Conference, 24.-26. März, NIST, Gaithersburg MD, USA* (2009)

[Min02] Mintert, Stefan: *XML & Co. Die W3C-Spezifikationen für Dokumenten- und Datenarchitektur*. Addison-Wesley, 2002

[Mit05] Mitra, Tilak: *Business-driven development*. Internet. http://www.ibm.com/developerworks/webservices/library/ws-bdd/. Version: Dezember 2005, Abruf: 14. Februar 2011

[MLM+06] MacKenzie, C. M. ; Laskey, K. ; McCabe, F. ; Brown, P. F. ; Metz, R: *Reference Model for Service Oriented Architecture 1.0*. Internet. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf. Version: April 2006, Abruf: 14. Februar 2011

[MNS05] Muehlen, Michael zur ; Nickerson, Jeffrey V. ; Swenson, Keith D.: Developing web services choreography standards – the case of REST vs. SOAP. In: *SDSOA '08, May 11, 2008, Leipzig, Germany* Bd. 40. Amsterdam, Niederlande : Elsevier Science Publishers B. V., Juli 2005, S. 9–29

[Mus10] Musser, John: Open APIs: State of the market. In: *Glue Conference 2010, 26.-27. April, Denver, CO, USA*, 2010

[MW08] Miers, Derek ; White, Stephen A.: *BPMN Modeling and Reference Guide – Understanding and Using BPMN*. Future Strategies Inc, 2008

[Nad04] Nadhan, E. G.: Seven Steps to a Service-Oriented Evolution. In: *Business Integration Journal* (2004), S. 41–44

[Nau07] Naumann, Felix: Datenqualität. In: *Informatik Spektrum* 30 (2007), Nr. 1, S. 27–31

[NDH02] Nieschlag, Robert ; Dichtl, Erwin ; Hörschgen, Hans: *Marketing*. 19. Auflage. Berlin : Duncker & Humblot, 2002

[Net88] Network Working Group: *RPC: Remote Procedure Call – Protocol Specification Version 2*. Internet. http://tools.ietf.org/html/rfc1057. Version: Juni 1988, Abruf: 14. Februar 2011

[nG08] Ünay, Ozan ; Gündem, Taflan I.: A survey on querying encrypted XML documents for databases as a service. In: *SIGMOD Record* 37 (2008), Nr. 1, S. 12–20

[NL04] Newcomer, Eric ; Lomow, Greg A.: *Understanding SOA with Web Services*. Addison-Wesley Longman, 2004

[OADH06] Ouyang, Chun ; Aalst, Wil M. d. ; Dumas, Marlon ; Hofstede, Arthur H. ; BPMcenter.org (Hrsg.): *From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way*. Internet. http://eprints.qut.edu.au/5266/1/5266.pdf. Version: 2006, Abruf: 14. Februar 2011

[OAS05] OASIS: *Web Service Implementation Methodology - Public Review Draft 1.0*. Internet. http://www.oasis-open.org/committees/download.php/21354/fwsi-im-1.0-guidelines-doc-wd-PublicReviewDraft1.0.pdf. Version: September 2005, Abruf: 14. Februar 2011

[OAS07] OASIS ; OASIS (Hrsg.): *Web Services Business Process Execution Language Version 2.0*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. Version: 2007, Abruf: 14. Februar 2011

[OAS09a] OASIS: *Web Services Dynamic Discovery (WS-Discovery)*. Internet. http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01. Version: Juli 2009, Abruf: 14. Februar 2011

[OAS09b] OASIS: *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2*. Internet. http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html. Version: Februar 2009, Abruf: 14. Februar 2011

[Obj05] Object Management Group: *Unified Modeling Language – Version 2.0*. Internet. http://www.omg.org/spec/UML/2.0/. Version: Juli 2005, Abruf: 14. Februar 2011

[Obj09] Object Management Group: *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*. http://www.omg.org/spec/

SoaML/1.0/Beta2/PDF. Version: Dezember 2009, Abruf: 14. Februar 2011

[Obj10] Object Management Group: *Business Process Model and Notation (BPMN) – Version 2.0*. Internet. http://www.omg.org/spec/BPMN/2.0/PDF. Version: Juni 2010, Abruf: 14. Februar 2011

[Off08] Offermann, Philipp: SOAM – Eine Methode zur Konzeption betrieblicher Software mit einer Serviceorientierten Architektur. In: *Wirtschaftsinformatik* 6 (2008), S. 461–471

[Olb08] Olbrich, Alfred: *ITIL kompakt und verständlich: Effizientes IT Service Management - Den Standard für IT-Prozesse kennenlernen, verstehen und erfolgreich in der Praxis umsetzen*. 4., erw. und verb. Auflage. Vieweg+Teubner, 2008

[O'S06] O'Sullivan, J.: *Towards a Precise Understanding of Service Properties*, Faculty of Information Technology, Queensland University of Technology, Australia, Dissertationsschrift, 2006

[Oul09] Oulette, Jason: *Development with the Force.com Platform: Building Business Applications in the Cloud*. Addison-Wesley Longman, 2009

[Ove07a] Overdick, Hagen: The Resource-Oriented Architecture. In: *2007 IEEE Congress on Services, 9-13 Juli, Salt Lake City, Utah, USA* (2007), S. 340–347

[Ove07b] Overdick, Hagen: Towards Resource–Oriented BPEL. In: *Proceedings of the 2nd ECOWS07 Workshop on Emerging Web Services Technology, 26.-28. November, Halle (Saale)* (2007), S. 129–140

[Pap07] Papazoglou, Michael P.: *Web services: Principles and technology*. Harlow : Pearson/Prentice Hall, 2007

[Pau08] Pautasso, Cesare: BPEL for REST. In: *Proc. of the 6th International Conference on Business Process Management (BPM 2008), Milan, Italy* (2008), September, S. 278–293

[PD10] Pedrinaci, C. ; Domingue, J.: Toward the Next Wave of Services: Linked Services for the Web of Data. In: *Journal of Universal Computer Science, 2010* 16 (2010), Juli, Nr. 13, 1694–1719. http://www.jucs.org/jucs_16_13/toward_the_next_wave/jucs_16_13_1694_1719_pedrinaci.pdf

[PFP02] Palmer, Stephen R. ; Felsing, Mac ; Palmer, Steve: *A Practical Guide to Feature-Driven Development*. Prentice Hall International, 2002 (The Coad Series)

[PH06] Papazoglou, Michael P. ; Heuvel, Willem-Jan van d.: Service-oriented design and development methodology. In: *Int. J. Web Eng. Technol.* 2 (2006), Nr. 4, S. 412–442

[PKR05] Pohle, George ; Korsten, Peter ; Ramamurthy, Shanker ; Business Value study, IBM I. (Hrsg.): *Component business models: Making specialization real*. Internet. http://www-935.ibm.com/services/us/imc/pdf/g510-6163-component-busines%s-models.pdf. Version: 2005, Abruf: 14. Februar 2011

[PKS02] Pol, Martin ; Koomen, Tim ; Spillner, Andreas: *Management und Optimierung des Testprozesses*. 2., aktualisierte Auflage. Heidelberg : dpunkt.verlag, 2002

[Por85] Porter, Michael E.: *Competitive Advantage*. New York, USA : The Free Press, 1985

[PR10] Pohl, Klaus ; Rupp, Chris: *Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering – Foundation Level*. Heidelberg : dpunkt.verlag, 2010

[PVM09] Pietschmann, Stefan ; Voigt, Martin ; Meißner, Klaus: Dynamic Composition of Service-Oriented Web User Interfaces. In: *Proceedings of the 4th International Conference on Internet and Web Applications and Services (ICIW 2009), Mestre/Venice, Italy* (2009), S. 217–222

[PZL08] Pautasso, Cesare ; Zimmermann, Olaf ; Leymann, Frank: Restful web services vs. "big" web services: making the right architectural decision. In: *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA, 2008, S. 805–814

[QSPS07] Quartel, Dick A. ; Steen, Maarten W. A. ; Pokraev, Stanislav ; Sinderen, Marten J.: COSMO: A conceptual framework for service modelling and refinement. In: *Information Systems Frontiers* 9 (2007), Nr. 2–3, S. 225–244

[RD00] Rahm, Erhard ; Do, Hong H.: Data Cleaning: Problems and Current Approaches. In: *IEEE Data Eng. Bull.* 23 (2000), Nr. 4, S. 3–13

[RH08] Reussner, Ralf ; Hasselbring, Wilhelm: *Handbuch der Software-Architektur*. 2. Auflage. Heidelberg : dpunkt.verlag, 2008

[Rod08] Rodriguez, Alex: *RESTful Web services: The basics*. Internet. http://www.ibm.com/developerworks/webservices/library/ws-restful/. Version: November 2008, Abruf: 14. Februar 2011

[Ros03] Ross, Ronald G.: *Principles of the Business Rule Approach*. Addison-Wesley Longman, 2003

[Roy70] Royce, Winston W.: Managing the development of large software systems. In: *Technical Papers of Western Electronic Show and Convention (WesCon), Los Angeles, USA* (1970), August, S. 25–28

[RQZ07] Rupp, Chris ; Queins, Stefan ; Zengler, Barbara: *UML 2 glasklar. Praxiswissen für die UML-Modellierung*. 3., aktualisierte Auflage. München : Hanser Fachbuchverlag, 2007

[RR07] Richardson, Leonard ; Ruby, Sam: *RESTful Web Services*. O'Reilly Media, Inc., 2007

[RSA78] Rivest, R. ; Shamir, A. ; Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In: *Communications of the ACM* 21 (1978), Nr. 2, S. 120–126

[RT10] Reimers, Stefan ; Thies, Gunnar: *PHP 5.3 und MySQL 5.5*. 3. Auflage. Bonn : Galileo Press, 2010.

[SBS08] Sneed, Harry M. ; Baumgartner, Manfred ; Seidl, Richard: *Der Systemtest. Anforderungsbasiertes Testen von Software-Systemen*. 1. Auflage. München : Hanser Fachbuchverlag, 2008

[Sch96] Schulte, W. R.: *Service Oriented Architectures - Part 2, SSA Research Note SPA-00-7426*. Internet. http://www.gartner.com/DisplayDocument?doc_cd=29202&ref=g_fromdoc. Version: April 1996, Abruf: 18.02.2011

[Sch01] Scheer, August-Wilhelm: *ARIS-Modellierungs-Methoden, Metamodelle, Anwendungen*. 4. Auflage. Berlin Heidelberg : Springer, 2001

[Sch02] Schulte, W. R.: *Predicts 2003: Enterprise Service Buses Emerge*. Internet. http://www.gartner.com/DisplayDocument?doc_cd=111977. Version: Dezember 2002, Abruf: 14. Februar 2011

[Sei06] Seibold, Holger: *IT-Risikomanagement*. München : Oldenbourg, 2006

[SHT04] Sneed, Harry M. ; Hasitschka, Martin ; Teichmann, Maria-Therese: *Software-Produktmanagement: Wartung und Weiterentwicklung bestehender Anwendungssysteme*. 1. Auflage. Heidelberg : dpunkt.verlag, 2004

[SI07a] Stein, S. ; Ivanov, K.: EPK nach BPEL Transformation als Voraussetzung für praktische Umsetzung einer SOA. In: *Software Engineering 2007 – Lecture Notes in Informatics (LNI)* 105 (2007), März, S. 75–80

[SI07b] Stein, Sebastian ; Ivanov, Konstantin: Vorgehensmodell zur Entwicklung von Geschäftsservicen. In: K.-P. Fähnrich, M. T. (Hrsg.): *Integration Engineering – Motivation, Begriffe, Methoden und Anwendungsfälle*. Eigenverlag Leipziger Informatik-Verbund (LIV), 2007

[Sie04] Siemers, H.-H.: Was kostet ein Kunde? TCO-Betrachtung im Umfeld von Customer Relationship Management. In: *SAP INFO* (2004), April, Nr. 115, S. 30–33

[SIVE08] Schepers, T. G. J. ; Iacob, M. E. ; Van Eck, P. A. T.: A lifecycle approach to SOA governance. In: *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA, 2008, S. 1055–1061

[SK07] Schroth, Christoph ; Kirchhoff, Lars: Web 2.0 und SOA - verwandte Konzepte? In: *Praxis der Wirtschaftsinformatik, HMD* 255 (2007), S. 49–57

[SN96] Schulte, W. R. ; Natis, Yefim V.: *Service Oriented Architectures - Part 1, SSA Research Note SPA-00-7425*. Internet. http://www.gartner.com/DisplayDocument?doc_cd=29201&ref=g_fromdoc. Version: April 1996, Abruf: 18.02.2011

[Sne04] Snell, J: *Resource-oriented vs. activity-oriented Web services*. http://www.ibm.com/developerworks/xml/library/ws-restvsoap/. Version: October 2004, Abruf: 14. Februar 2011

[Sta05] Staud, Josef L.: *Datenmodellierung und Datenbankentwurf. Ein Vergleich aktueller Methoden.* Berlin Heidelberg : Springer, 2005

[Sta06] Staud, Josef L.: *Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für betriebswirtschaftliche Standardsoftware*. Berlin Heidelberg : Springer, 2006

[SVOK11] Schönthaler, Frank ; Vossen, Gottfried ; Oberweis, Andreas ; Karle, Thomas: *Geschäftsprozesse für Business Communities - Modellierungssprachen, Methoden, Werkzeuge*. München : Oldenbourg, 2011

[SZ08] Stych, Christof ; Zeppenfeld, Klaus: *ITIL® (Informatik Im Fokus)*. Berlin Heidelberg : Springer, 2008

[THW05] Ten-Hove, Ron ; Walker, Peter: *Java Business Integration 1.0: Final Release*. Internet. http://jcp.org/aboutJava/

communityprocess/final/jsr208/index.html. Version: August 2005, Abruf: 14. Februar 2011

[TMK+08] Takase, Toshiro ; Makino, Satoshi ; Kawanaka, Satoshi ; Ueno, Ken ; Ferris, Christopher ; Ryman, Arthur ; IBM (Hrsg.): *Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0*. Internet. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-wadlwsd%l/WADLWSDLpaper20080621.pdf. Version: 2008, Abruf: 14. Februar 2011

[Toi02] Toigo, Jon W.: *Disaster Recovery Planning: Preparing for the Unthinkable*. 3. Auflage. Prentice Hall International, 2002

[Tor03] Torra, Vicenç: *Information Fusion in Data Mining*. 1. Auflage. Berlin Heidelberg : Springer, 2003

[TPC05] Todd, Stephen ; Parr, Francis ; Conner, Michael: *A Primer for HTTPR*. Internet. http://www.ibm.com/developerworks/webservices/library/ws-phtt/. Version: März 2005, Abruf: 14. Februar 2011

[Tre09] Trembly, Ara C.: Just What Is SOA, Anyway? In: *National Underwriter / Property & Casualty Risk & Benefits Management* 113 (2009), Nr. 1, S. 20–23

[TS07] Tanenbaum, Andrew. S. ; Steen, Marten van: *Verteilte Systeme*. 2. Auflage. Pearson Studium, 2007

[TTS04] Treber, Udo ; Teipel, Philip ; Schwickert, Axel C.: Total Cost of Ownership – Stand und Entwicklungstendenzen 2003. In: *Arbeitspapiere WI, Nr. 1/2004, Hrsg.: Professur BWL – Wirtschaftsinformatik, Justus-Liebig-Universität Gießen* (2004)

[TV08] Thies, Gunnar ; Vossen, Gottfried: Web-Oriented Architectures: On the Impact of Web 2.0 on Service-Oriented Architectures. In: *IEEE Asia-Pacific Services Computing Conference 2008, Dez. 9.-12., Yilan, Taiwan* (2008), S. 1075–1082

[TV09a] Thies, Gunnar ; Vossen, Gottfried: Governance in Web-Oriented Architectures. In: *IEEE Asia-Pacific Services Computing Conference 2009, Dez. 7.-11., Biopolis, Singapore* (2009), S. 180–186

[TV09b] Thies, Gunnar ; Vossen, Gottfried: Modelling Web-Oriented Architectures. In: *Proc. 6th Asia-Pacific Conference on Conceptual Modelling (APCCM), 20.-23. Januar, Wellington, New Zealand* Conferences in Research and Practice in Information Technology, Volume 96, Australian Computer Science Communications, Volume 31, Number 6 (2009), S. 97–105

[TV13] Thies, Gunnar ; Vossen, Gottfried: Agile Development of Web-Oriented Architectures. In: *International Journal of Information Systems in the Service Sector (IJISSS) - Advanced Research Topics in Services Computing* 4 (erscheint 2013), Nr. 3

[Uni03] United States Department of Justice: *Information Resources Management*. Internet. http://www.justice.gov/jmd/irm/lifecycle/table.htm. Version: 2003, Abruf: 18.02.2011

[VH07] Vossen, Gottfried ; Hagemann, Stephan: *Unleashing Web 2.0: From concepts to creativity*. Burlington, MA, USA : Elsevier/Morgan Kaufmann, 2007

[Vos06] Vossen, Gottfried: Have Service–Oriented Architectures Taken a Wrong Turn Already? In: *IFIP TC 8 International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006), Vienna, Austria*. 2006, S. 23–24

[Vos08] Vossen, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbank- Management- Systeme*. 5. überarbeitete und erweiterte Auflage. München : Oldenbourg, 2008

[VZT08] Vossen, Gottfried ; Zengerling, Knut ; Thies, Gunnar: Serviceorientierung: Vereinfachung von Softwareentwicklung. In: Brocke, J. vom (Hrsg.) ; Becker, J. (Hrsg.): *Einfachheit in Wirtschaftsinformatik und Controlling - Festschrift für Heinz Lothar Grob*. München : Verlag Franz Vahlen, 2008, S. 135–153

[W3C04a] W3C: *Resource Description Framework (RDF)*. Internet. http://www.w3.org/RDF/. Version: Februar 2004, Abruf: 14. Februar 2011

[W3C04b] W3C ; W3C (Hrsg.): *Semantic Markup for Web Services*. Internet. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/. Version: 2004, Abruf: 14. Februar 2011

[W3C04c] W3C ; W3C (Hrsg.): *Web Service Architecture*. Internet. http://www.w3.org/TR/ws-arch/wsa.pdf. Version: 2004, Abruf: 14. Februar 2011

[W3C04d] W3C: *Web Services Glossary*. Internet. http://www.w3.org/TR/ws-gloss/. Version: Februar 2004, Abruf: 14. Februar 2011

[W3C07a] W3C: *Semantic Annotations for WSDL and XML Schema*. Internet. http://www.w3.org/TR/sawsdl/. Version: August 2007, Abruf: 14. Februar 2011

[W3C07b] W3C: *Semantic Annotations for WSDL and XML Schema (SAWSDL)*. Internet. http://www.w3.org/TR/sawsdl/. Version: August 2007, Abruf: 29. Januar 2011

[W3C07c] W3C: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Internet. http://www.w3.org/TR/soap12-part1/. Version: April 2007, Abruf: 29. Januar 2011

[W3C07d] W3C ; W3C (Hrsg.): *Web Services Description Language 2.0*. Internet. http://www.w3.org/TR/wsdl20-primer/. Version: 2007, Abruf: 14. Februar 2011

[WAB+07] Wahli, Ueli ; Ackerman, Lee ; Bari, Alessandro D. ; Hodgkinson, Gregory ; Kesterton, Anthony ; Olson, Laura ; Portier, Bertrand: *Building SOA Solutions Using the Rational SDP - IBM Redbook*. Internet. http://www.redbooks.ibm.com/redbooks/pdfs/sg247356.pdf. Version: April 2007, Abruf: 14. Februar 2011

[WCL+05] Weerawarana, Sanjiva ; Curbera, Francisco ; Leymann, Frank ; Ferguson, Donald F. ; Storey, Tony: *Web Services Platform Architecture*. Prentice Hall International, 2005

[Wes07] Weske, Mathias: *Business Process Management: Concepts, Languages, Architectures*. 1. Auflage. Berlin Heidelberg : Springer, 2007

[WH00] Wild, Martin ; Herges, Sascha: Total Cost of Ownership (TCO) - Ein Überblick. In: *Arbeitspapiere WI, Nr. 1/2000, Hrsg.: Professur BWL - Wirtschaftsinformatik, Justus-Liebig-Universität Gießen* 1 (2000), Nr. 1

[WH06] Wilde, Thomas ; Hess, Thomas: *Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung*. Arbeitsbericht Nr. 2/2006 des Instituts für Wirtschaftsinformatik und Neue Medien der Ludwig-Maximilians-Universität München. Internet. http://www.en.wim.bwl.uni-muenchen.de/download/epub/ab_2006_02.pdf. Version: 2006, Abruf: 19.02.2011

[WH07] Wilde, Thomas ; Hess, Thomas: Forschungsmethoden der Wirtschaftsinformatik - Eine empirische Untersuchung. In: *Wirtschaftsinformatik* 49 (2007), Nr. 4, S. 280–287

[Whi76] White, James E.: *A High-Level Framework for Network-Based Resource Sharing*. Internet. http://portal.acm.org/citation.cfm?id=1499878. Version: Januar 1976, Abruf: 14. Februar 2011

[WSM06] WSMO Working Group: *Web Service Modeling Ontology*. Internet. http://www.wsmo.org/TR/d2/v1.3/20061021/. Version: 2006, Abruf: 14. Februar 2011

[WWRL09] Wang, Cong ; Wang, Qian ; Ren, Kui ; Lou, Wenjing: Ensuring data storage security in Cloud Computing. In: *Proc. 17th International Workshop on Quality of Service (IWQoS)*. Charleston, SC, USA, Juli 2009, S. 1–9

[Zac05] Zacharias, Roger: Serviceorientierung – Der OO-König ist tot, es lebe der SOA-König. In: *OBJEKTspektrum* 2 (2005), April, Nr. 2, S. 43–52

[Zim80] Zimmermann, Hubert: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. In: *IEEE Transactions on Communications* 28 (1980), Nr. 4, S. 425–432

[ZKG04] Zimmermann, Olaf ; Krogdahl, Pal ; Gee, Clive: *Elements of Service-Oriented Analysis and Design*. Internet. http://www.ibm.com/developerworks/webservices/library/ws-soad1/. Version: Juni 2004, Abruf: 14. Februar 2011

[ZLC08] Zhang, Jin ; Li, Feng-lin ; Chi, Chi-Hung: On Web Service Construction Based on REpresentation State Transfer. In: *ICEBE '08: Proceedings of the 2008 IEEE International Conference on e-Business Engineering*. Washington, DC, USA, 2008, S. 665–668

[ZYCJ06] Zhang, Tao ; Ying, Shi ; Cao, Sheng ; Jia, Xiangyang: A Modeling Framework for Service-Oriented Architecture. In: *Sixth International Conference on Quality 2006, 27.-28. Oktober, Peking, China*, 2006, S. 219–226

[ZZC+08] Zhang, L.-J. ; Zhou, N. ; Chee, Y.-M. ; Jalaldeen, A. ; Ponnalagu, K. ; Sindhgatta, R. R. ; Arsanjani, A. ; Bernardini, F.: SOMA-ME: A platform for the model-driven design of SOA solutions. In: *IBM Systems Journal* 47 (2008), Nr. 3, S. 397–413

A XSD-Beschreibung von WADL

Das folgende XSD-Dokument beschreibt die Struktur eines WADL-Dokuments zur Beschreibung einer Web-basierten Service-Schnittstelle.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://wadl.dev.java.net/2009/02"
  xmlns:tns="http://wadl.dev.java.net/2009/02"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  elementFormDefault="qualified">

  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <xs:element name="application">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
           unbounded"/>
        <xs:element ref="tns:grammars" minOccurs="0"/>
        <xs:element ref="tns:resources" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="tns:resource_type"/>
          <xs:element ref="tns:method"/>
          <xs:element ref="tns:representation"/>
          <xs:element ref="tns:param"/>
        </xs:choice>
        <xs:any namespace="##other" processContents="lax"
           minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="doc">
    <xs:complexType mixed="true">
      <xs:sequence>
```

```
33          <xs:any namespace="##other" processContents="lax"
               minOccurs="0"
34            maxOccurs="unbounded"/>
35        </xs:sequence>
36        <xs:attribute name="title" type="xs:string"/>
37        <xs:attribute ref="xml:lang"/>
38        <xs:anyAttribute namespace="##other" processContents="lax
             "/>
39      </xs:complexType>
40    </xs:element>
41
42    <xs:element name="grammars">
43      <xs:complexType>
44        <xs:sequence>
45          <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
               unbounded"/>
46          <xs:element minOccurs="0" maxOccurs="unbounded" ref="
               tns:include"/>
47          <xs:any namespace="##other" processContents="lax"
               minOccurs="0"
48            maxOccurs="unbounded"/>
49        </xs:sequence>
50      </xs:complexType>
51    </xs:element>
52
53    <xs:element name="resources">
54      <xs:complexType>
55        <xs:sequence>
56          <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
               unbounded"/>
57          <xs:element ref="tns:resource" maxOccurs="unbounded"/>
58          <xs:any namespace="##other" processContents="lax"
               minOccurs="0"
59            maxOccurs="unbounded"/>
60        </xs:sequence>
61        <xs:attribute name="base" type="xs:anyURI"/>
62        <xs:anyAttribute namespace="##other" processContents="lax
             "/>
63      </xs:complexType>
64    </xs:element>
65
66    <xs:element name="resource">
67      <xs:complexType>
68        <xs:sequence>
```

```
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
           unbounded"/>
        <xs:element ref="tns:param" minOccurs="0" maxOccurs="
           unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="tns:method"/>
          <xs:element ref="tns:resource"/>
        </xs:choice>
        <xs:any minOccurs="0" maxOccurs="unbounded" namespace="
           ##other"
          processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="type" type="tns:resource_type_list"/>
      <xs:attribute name="queryType" type="xs:string"
        default="application/x-www-form-urlencoded"/>
      <xs:attribute name="path" type="xs:string"/>
      <xs:anyAttribute namespace="##other" processContents="lax
         "/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="resource_type_list">
    <xs:list itemType="xs:anyURI"/>
  </xs:simpleType>

  <xs:element name="resource_type">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
           unbounded"/>
        <xs:element ref="tns:param" minOccurs="0" maxOccurs="
           unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="tns:method"/>
          <xs:element ref="tns:resource"/>
        </xs:choice>
        <xs:any minOccurs="0" maxOccurs="unbounded" namespace="
           ##other"
          processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:anyAttribute namespace="##other" processContents="lax
         "/>
```

```
105     </xs:complexType>
106   </xs:element>
107
108   <xs:element name="method">
109     <xs:complexType>
110       <xs:sequence>
111         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
              unbounded"/>
112         <xs:element ref="tns:request" minOccurs="0"/>
113         <xs:element ref="tns:response" minOccurs="0"
114           maxOccurs="unbounded"/>
115         <xs:any namespace="##other" processContents="lax"
              minOccurs="0"
116           maxOccurs="unbounded"/>
117       </xs:sequence>
118       <xs:attribute name="id" type="xs:ID"/>
119       <xs:attribute name="name" type="tns:Method"/>
120       <xs:attribute name="href" type="xs:anyURI"/>
121       <xs:anyAttribute namespace="##other" processContents="lax
            "/>
122     </xs:complexType>
123   </xs:element>
124
125   <xs:simpleType name="Method">
126     <xs:union memberTypes="tns:HTTPMethods xs:NMTOKEN"/>
127   </xs:simpleType>
128
129   <xs:simpleType name="HTTPMethods">
130     <xs:restriction base="xs:NMTOKEN">
131       <xs:enumeration value="GET"/>
132       <xs:enumeration value="POST"/>
133       <xs:enumeration value="PUT"/>
134       <xs:enumeration value="HEAD"/>
135       <xs:enumeration value="DELETE"/>
136     </xs:restriction>
137   </xs:simpleType>
138
139   <xs:element name="include">
140     <xs:complexType>
141       <xs:sequence>
142         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
              unbounded"/>
143       </xs:sequence>
144       <xs:attribute name="href" type="xs:anyURI"/>
```

```
145       <xs:anyAttribute namespace="##other" processContents="lax
              "/>
146     </xs:complexType>
147   </xs:element>
148
149   <xs:element name="request">
150     <xs:complexType>
151       <xs:sequence>
152         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
              unbounded"/>
153         <xs:element ref="tns:param" minOccurs="0" maxOccurs="
              unbounded"/>
154         <xs:element ref="tns:representation" minOccurs="0"
155           maxOccurs="unbounded"/>
156         <xs:any namespace="##other" processContents="lax"
              minOccurs="0"
157           maxOccurs="unbounded"/>
158       </xs:sequence>
159       <xs:anyAttribute namespace="##other" processContents="lax
              "/>
160     </xs:complexType>
161   </xs:element>
162
163   <xs:element name="response">
164     <xs:complexType>
165       <xs:sequence>
166         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
              unbounded"/>
167         <xs:element ref="tns:param" minOccurs="0" maxOccurs="
              unbounded"/>
168         <xs:element ref="tns:representation" minOccurs="0"
169           maxOccurs="unbounded"/>
170         <xs:any namespace="##other" processContents="lax"
              minOccurs="0"
171           maxOccurs="unbounded"/>
172       </xs:sequence>
173       <xs:attribute name="status" type="tns:statusCodeList"/>
174       <xs:anyAttribute namespace="##other" processContents="lax
              "/>
175     </xs:complexType>
176   </xs:element>
177
178   <xs:simpleType name="uriList">
179     <xs:list itemType="xs:anyURI"/>
```

```
180   </xs:simpleType>
181
182   <xs:element name="representation">
183     <xs:complexType>
184       <xs:sequence>
185         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
              unbounded"/>
186         <xs:element ref="tns:param" minOccurs="0" maxOccurs="
              unbounded"/>
187         <xs:any namespace="##other" processContents="lax"
              minOccurs="0"
188           maxOccurs="unbounded"/>
189       </xs:sequence>
190       <xs:attribute name="id" type="xs:ID"/>
191       <xs:attribute name="element" type="xs:QName"/>
192       <xs:attribute name="mediaType" type="xs:string"/>
193       <xs:attribute name="href" type="xs:anyURI"/>
194       <xs:attribute name="profile" type="tns:uriList"/>
195       <xs:anyAttribute namespace="##other" processContents="lax
            "/>
196     </xs:complexType>
197   </xs:element>
198
199   <xs:simpleType name="statusCodeList">
200     <xs:list itemType="xs:unsignedInt"/>
201   </xs:simpleType>
202
203   <xs:simpleType name="ParamStyle">
204     <xs:restriction base="xs:string">
205       <xs:enumeration value="plain"/>
206       <xs:enumeration value="query"/>
207       <xs:enumeration value="matrix"/>
208       <xs:enumeration value="header"/>
209       <xs:enumeration value="template"/>
210     </xs:restriction>
211   </xs:simpleType>
212
213   <xs:element name="param">
214     <xs:complexType>
215       <xs:sequence>
216         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
              unbounded"/>
217         <xs:element ref="tns:option" minOccurs="0" maxOccurs="
              unbounded"/>
```

```
        <xs:element ref="tns:link" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax"
           minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="href" type="xs:anyURI"/>
      <xs:attribute name="name" type="xs:NMTOKEN"/>
      <xs:attribute name="style" type="tns:ParamStyle"/>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="type" type="xs:QName" default="
         xs:string"/>
      <xs:attribute name="default" type="xs:string"/>
      <xs:attribute name="required" type="xs:boolean" default="
         false"/>
      <xs:attribute name="repeating" type="xs:boolean" default=
         "false"/>
      <xs:attribute name="fixed" type="xs:string"/>
      <xs:attribute name="path" type="xs:string"/>
      <xs:anyAttribute namespace="##other" processContents="lax
         "/>
    </xs:complexType>
  </xs:element>

  <xs:element name="option">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
           unbounded"/>
        <xs:any namespace="##other" processContents="lax"
           minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="value" type="xs:string" use="required
         "/>
      <xs:attribute name="mediaType" type="xs:string"/>
      <xs:anyAttribute namespace="##other" processContents="lax
         "/>
    </xs:complexType>
  </xs:element>

  <xs:element name="link">
    <xs:complexType>
      <xs:sequence>
```

```
252         <xs:element ref="tns:doc" minOccurs="0" maxOccurs="
               unbounded"/>
253         <xs:any namespace="##other" processContents="lax"
               minOccurs="0"
254           maxOccurs="unbounded"/>
255       </xs:sequence>
256       <xs:attribute name="resource_type" type="xs:anyURI"/>
257       <xs:attribute name="rel" type="xs:token"/>
258       <xs:attribute name="rev" type="xs:token"/>
259       <xs:anyAttribute namespace="##other" processContents="lax
             "/>
260     </xs:complexType>
261   </xs:element>
262
263 </xs:schema>
```

Listing A.1: XSD-Dokument für WADL.

B Auswertungen der Beispielberechnung einer TCO

Die folgenden Auswertungen wurden im Rahmen der Berechnung einer Teil-TCO aus Kapitel 6 durch das Open-Source-Werkzeug TCO-Tool von *softEnvironment* erstellt. Die Abbildungen zeigen dabei Auswertungen aus der Software für die drei vorgestellten Lösungen des Umsetzungsszenarios: eine IBM-, eine Open-Source- sowie eine WOA-Lösung.

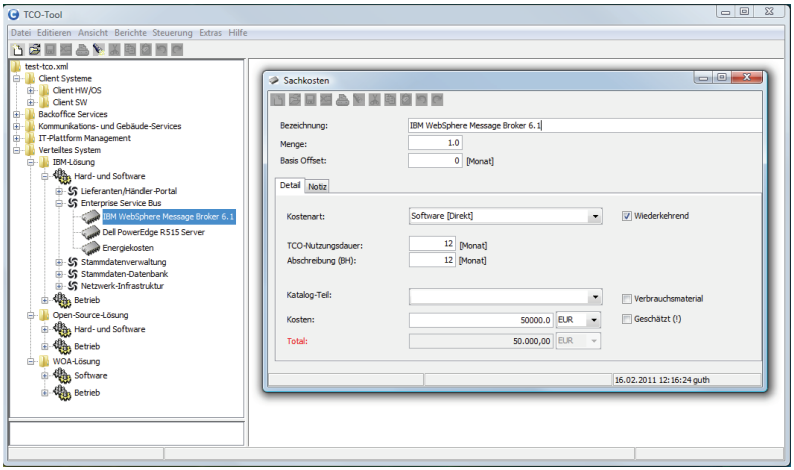


Abbildung 2.1: TCO-Tool von *softEnvironment*.

1 IBM-Lösung

(Kosten inklusive Untergruppen resp. deren Dienste.)

Nach <Kostenart>:

Kostenart	TCO 1. Jahr	TCO 2. Jahr	TCO 3. Jahr	TCO 4. Jahr	TCO-Kosten über gesamte Nutzungsdauer
Betrieb (allg.) [Direkt]	14.000,00	14.000,00	14.000,00	14.000,00	**56.000,02**
Betrieb (Energiekosten) [Direkt]	4.000,00	4.000,00	4.000,00	4.000,00	**16.000,00**
Betrieb (Hardware) [Direkt]	2.000,00	2.000,00	2.000,00	2.000,00	**8.000,00**
Betrieb (Internetkosten) [Direkt]	12.000,00	12.000,00	12.000,00	12.000,00	**48.000,00**
Betrieb (Software) [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Dienstleistung [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Gegenseitige Nutzerunterstützung [Indirekt]	0,00	0,00	0,00	0,00	**0,00**
Hardware [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Infrastruktur [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Installation [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Integration [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Schulung [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Software [Direkt]	156.000,00	156.000,00	156.000,00	156.000,00	**624.000,00**
Storage [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Unproduktive Zeit [Indirekt]	0,00	0,00	0,00	0,00	**0,00**
<Undefiniert>	0,00	0,00	0,00	0,00	**0,00**
Total	**188.000,00**	**188.000,00**	**188.000,00**	**188.000,00**	**752.000,02**

Alle Kosten in [EUR]

Abbildung 2.2: Kostenübersicht über die TCO der IBM-Lösung.

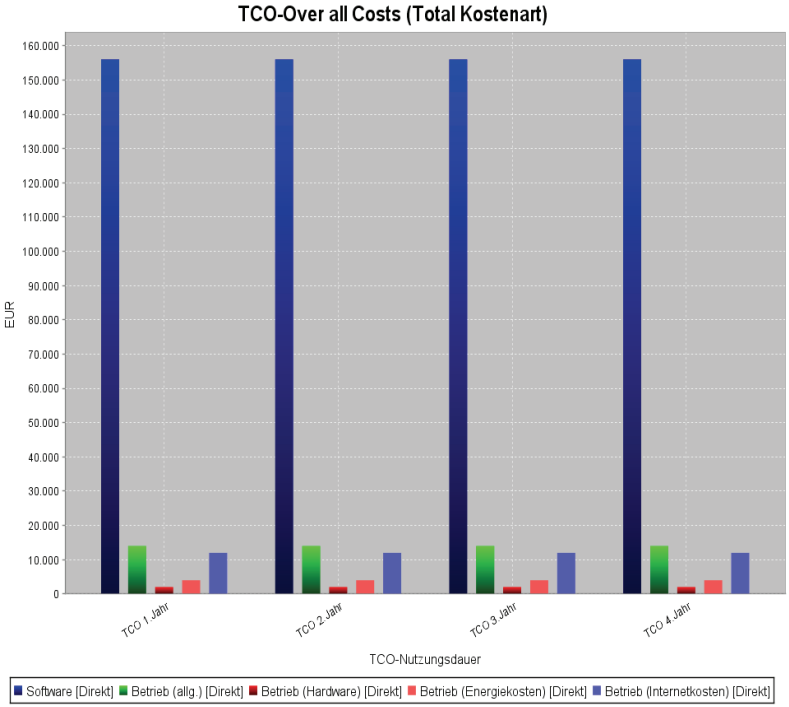


Abbildung 2.3: Kostenübersicht nach Kostenarten über die TCO der IBM-Lösung.

1 Open-Source-Lösung

(Kosten inklusive Untergruppen resp. deren Dienste.)

Nach <Kostenart>:

Kostenart	TCO 1. Jahr	TCO 2. Jahr	TCO 3. Jahr	TCO 4. Jahr	TCO-Kosten über gesamte Nutzungsdauer
Betrieb (allg.) [Direkt]	55.000,00	55.000,00	55.000,00	55.000,00	**220.000,00**
Betrieb (Energiekosten) [Direkt]	4.000,00	4.000,00	4.000,00	4.000,00	**16.000,00**
Betrieb (Hardware) [Direkt]	2.000,00	2.000,00	2.000,00	2.000,00	**8.000,00**
Betrieb (Internetkosten) [Direkt]	12.000,00	12.000,00	12.000,00	12.000,00	**48.000,00**
Betrieb (Software) [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Dienstleistung [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Gegenseitige Nutzerunterstützung [Indirekt]	0,00	0,00	0,00	0,00	**0,00**
Hardware [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Infrastruktur [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Installation [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Integration [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Schulung [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Software [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Storage [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Unproduktive Zeit [Indirekt]	0,00	0,00	0,00	0,00	**0,00**
<Undefiniert>	0,00	0,00	0,00	0,00	**0,00**
Total	**73.000,00**	**73.000,00**	**73.000,00**	**73.000,00**	**292.000,00**

Alle Kosten in [EUR]

Abbildung 2.4: Kostenübersicht über die TCO der Open-Source-Lösung.

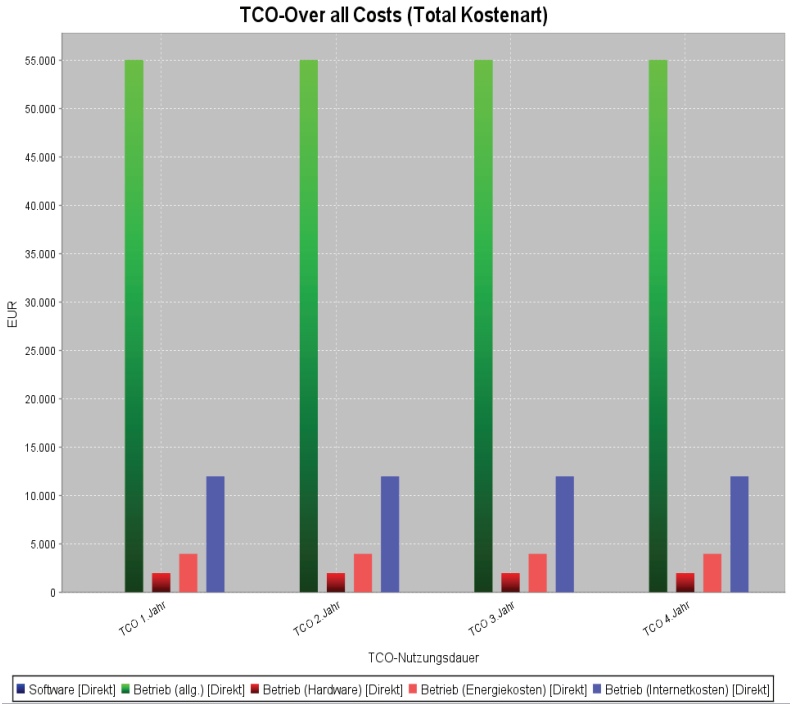


Abbildung 2.5: Kostenübersicht nach Kostenarten über die TCO der Open-Source-Lösung.

1 WOA-Lösung

(Kosten inklusive Untergruppen resp. deren Dienste.)

Nach <Kostenart>:

Kostenart	TCO 1. Jahr	TCO 2. Jahr	TCO 3. Jahr	TCO 4. Jahr	TCO-Kosten über gesamte Nutzungsdauer
Betrieb (allg.) [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Betrieb (Energiekosten) [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Betrieb (Hardware) [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Betrieb (Internetkosten) [Direkt]	7.000,00	7.000,00	7.000,00	7.000,00	**28.000,00**
Betrieb (Software) [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Dienstleistung [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Gegenseitige Nutzerunterstützung [Indirekt]	0,00	0,00	0,00	0,00	**0,00**
Hardware [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Infrastruktur [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Installation [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Integration [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Schulung [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Software [Direkt]	11.500,00	11.500,00	11.500,00	11.500,00	**46.000,00**
Storage [Direkt]	0,00	0,00	0,00	0,00	**0,00**
Unproduktive Zeit [Indirekt]	0,00	0,00	0,00	0,00	**0,00**
<Undefiniert>	0,00	0,00	0,00	0,00	**0,00**
Total	**18.500,00**	**18.500,00**	**18.500,00**	**18.500,00**	**74.000,00**

Alle Kosten in [EUR]

Abbildung 2.6: Kostenübersicht über die TCO der WOA-Lösung.

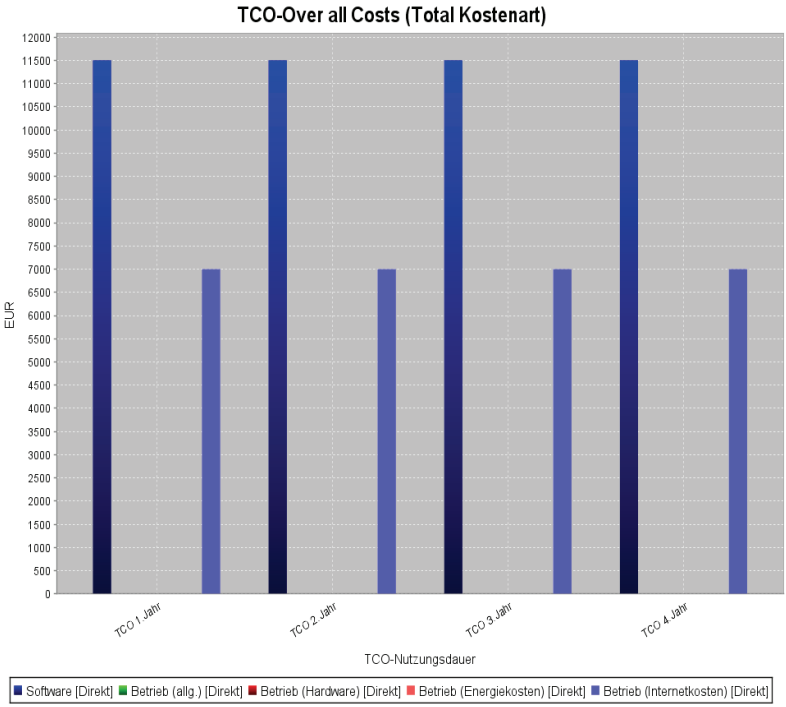


Abbildung 2.7: Kostenübersicht nach Kostenarten über die TCO der WOA-Lösung.

C Prototyp zur Bestimmung von Betriebskosten einer WOA

Die folgenden Bildschirmfotos stammen aus dem Prototypen zur WOA-Betriebskostenberechnung und zeigen die Eingabe des Services *PayPal Basiskonto* und dessen Parameter, die Bearbeitung einer Berechnungsregel der Kosten sowie die Darstellung der Konfigurationsmöglichkeiten durch Schieberegler in einem WOA-Szenario.

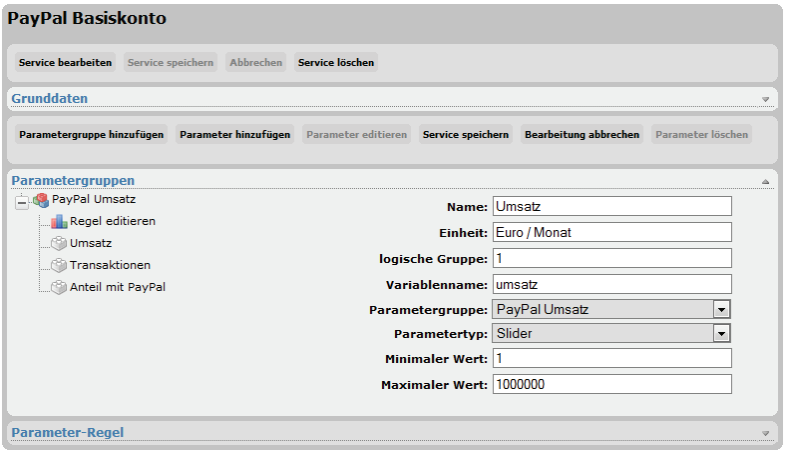


Abbildung 3.1: Eingabe der Parameter für den Service PayPal Basiskonto.

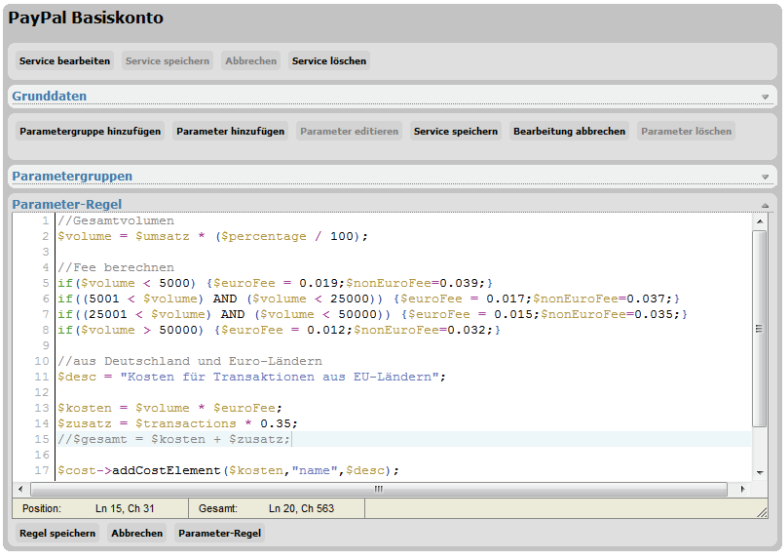


Abbildung 3.2: Bearbeitung einer Kostenberechnungsregel für den Service PayPal Basiskonto.

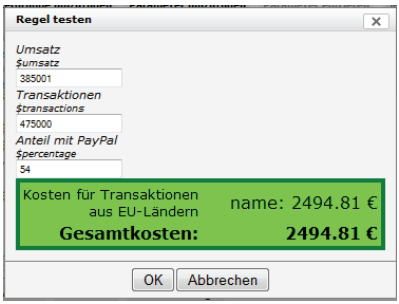


Abbildung 3.3: Testen der Kostenberechnungsregel des Services PayPal Basiskonto.

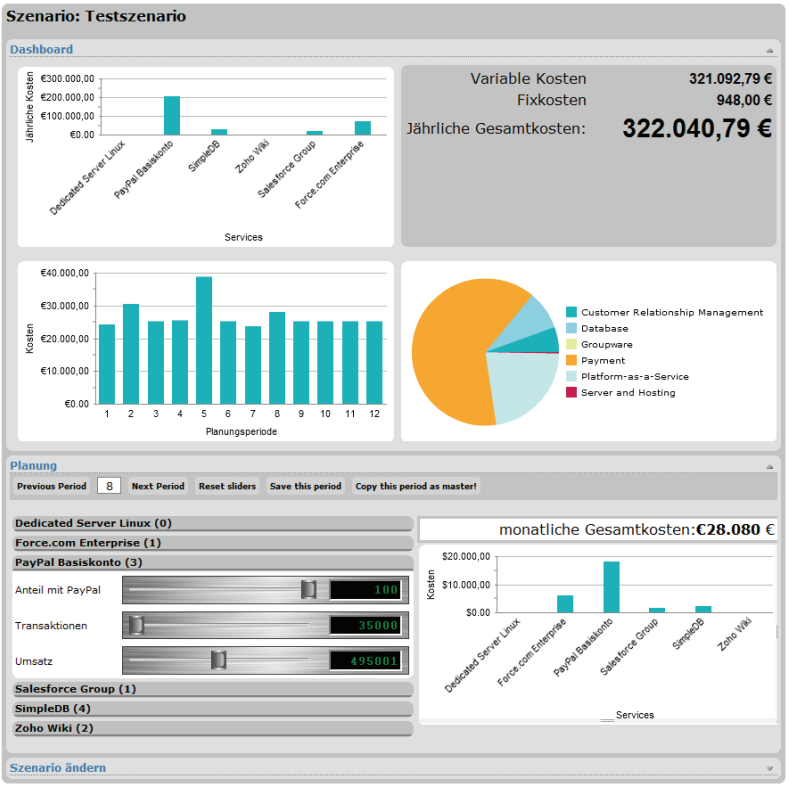


Abbildung 3.4: Konfiguration der Nutzung des Services PayPal Basiskonto in einer Periode.

Web-orientierte Architekturen

Gunnar Thies

Das Konzept einer Web-orientierten Architektur (WOA) beschreibt die Umsetzung eines verteilten Systems nach dem Vorbild des Internets. Das Ziel ist hierbei, ein kostengünstiges, flexibles und wartbares IT-System für Unternehmen umzusetzen, welches die Vorzüge der im Internet entstehenden Service-Landschaft ausnutzt. Motiviert wird das Thema einerseits durch die bestehenden Probleme Service-orientierter Architekturen und andererseits durch die scheinbare Leichtigkeit der Vernetzung von Everything-as-a-Service-Diensten im Internet durch Technologien und Ansätze des sogenannten Web 2.0. Die vorliegende Arbeit stellt das Architekturkonzept sowie die Prinzipien einer WOA dar, um anschließend ein ausführliches Entwicklungsmodell zur Analyse, Planung und Umsetzung zu beschreiben. Darüber hinaus wird eine Kostenbetrachtung vorgenommen, die neben der Bestimmung von Planungs- und Umsetzungskosten auch eine Berechnung diverser Betriebskostenszenarien ermöglicht.

ISBN 978-3-8405-0042-8 EUR 24,00
02400
9 783840 500428